

| | |
|---|-----|
| 1. Welcome | 18 |
| 1.1 Introducing Ignition | 21 |
| 1.2 Startup Guide | 27 |
| 1.2.1 Startup - Perspective Session | 58 |
| 1.2.2 Startup - Vision Client | 73 |
| 1.3 Installing and Upgrading Ignition | 93 |
| 1.3.1 Command Line Installations | 100 |
| 1.3.2 ZIP File Installations | 102 |
| 1.3.3 Installing or Upgrading a Module | 110 |
| 1.3.4 Ignition 8 Upgrade Guide | 114 |
| 1.4 Support, Feedback, and Resources | 122 |
| 1.5 Perspective and Vision — Which Visualization System Is Best for Me? | 124 |
| 1.6 A Vision-Oriented Guide to Perspective | 127 |
| 1.7 Licensing and Activation | 131 |
| 1.7.1 Emergency Activation | 140 |
| 1.8 Maker Edition | 142 |
| 1.9 Ignition Edge | 144 |
| 1.9.1 Edge Gateway | 148 |
| 1.9.2 Edge Panel | 154 |
| 1.9.3 Edge IIoT | 158 |
| 1.9.4 Edge Compute | 163 |
| 1.9.5 Edge Sync Services | 165 |
| 1.9.6 Edge EAM | 170 |
| 1.10 Launchers and Workstation | 171 |
| 1.10.1 Designer Launcher | 183 |
| 1.10.2 Perspective Workstation | 185 |
| 1.10.3 Vision Client Launcher | 190 |
| 1.10.4 Configuring Pre-Deployed Launchers | 195 |
| 2. Ignition Modules | 200 |
| 2.1 Vision | 203 |
| 2.1.1 Vision Designer Interface | 207 |
| 2.1.2 Vision Windows | 216 |
| 2.1.2.1 Window Types | 224 |
| 2.1.2.2 Popup Windows | 227 |
| 2.1.2.2.1 Parameterized Popup Windows | 234 |
| 2.1.2.3 Navigation Strategies in Vision | 240 |
| 2.1.2.3.1 Navigation - Tab Strip | 243 |
| 2.1.2.3.2 Navigation - Two Tier | 245 |
| 2.1.2.3.3 Navigation - Tree View | 249 |
| 2.1.2.3.4 Navigation - Forward and Back Buttons | 252 |
| 2.1.2.3.5 Navigation - Drill Down | 256 |
| 2.1.2.3.6 Navigation - Menubar | 260 |
| 2.1.2.3.7 Navigation - Retargeting | 263 |
| 2.1.3 Working with Vision Components | 265 |
| 2.1.3.1 Creating Vision Components | 277 |
| 2.1.3.2 Vision Component Customizers | 280 |
| 2.1.3.3 Drawing Tools | 291 |
| 2.1.3.3.1 Shape Geometry | 296 |
| 2.1.3.3.2 Fill and Stroke | 301 |
| 2.1.3.4 Images and SVGs in Vision | 306 |
| 2.1.3.5 Comparison Charts | 312 |
| 2.1.3.6 HTML in Vision | 326 |
| 2.1.3.7 Localization in Vision | 328 |
| 2.1.4 Binding Types in Vision | 330 |
| 2.1.4.1 Property Bindings in Vision | 334 |
| 2.1.4.2 Tag Bindings in Vision | 336 |
| 2.1.4.3 Indirect Tag Bindings in Vision | 339 |
| 2.1.4.4 Tag History Bindings in Vision | 343 |
| 2.1.4.5 Expression Binding in Vision | 349 |
| 2.1.4.6 Named Query Bindings | 351 |
| 2.1.4.7 DB Browse Bindings | 354 |
| 2.1.4.8 SQL Query Bindings in Vision | 359 |
| 2.1.4.9 Cell Update Bindings | 363 |
| 2.1.4.10 Function Bindings | 373 |
| 2.1.4.11 Color Animation in Vision | 375 |
| 2.1.5 Vision Templates | 382 |
| 2.1.5.1 Creating a Template | 388 |
| 2.1.5.2 Template Indirection | 395 |
| 2.1.5.3 Using the Template Repeater | 400 |
| 2.1.5.4 Using the Template Canvas | 408 |
| 2.1.6 Security in Vision | 418 |
| 2.1.6.1 Login Security | 421 |
| 2.1.6.2 Component and Window Security | 422 |
| 2.1.6.3 Security in Scripting | 424 |
| 2.1.7 Scripting in Vision | 428 |
| 2.1.7.1 Script Builders in Vision | 432 |
| 2.1.7.2 Component Events | 438 |
| 2.1.7.3 Extension Functions | 447 |
| 2.1.7.4 Custom Component Methods | 454 |

| | |
|--|-----|
| 2.1.7.5 Focus Manipulation | 456 |
| 2.1.7.6 Client Event Scripts | 460 |
| 2.1.7.7 Read a Cell from a Table | 466 |
| 2.1.8 Historian in Vision | 470 |
| 2.1.8.1 Using the Vision Easy Chart | 473 |
| 2.1.8.1.1 Easy Chart - Axes | 478 |
| 2.1.8.1.2 Easy Chart - Subplots | 485 |
| 2.1.8.1.3 Easy Chart - Pen Names and Groups | 489 |
| 2.1.8.1.4 Easy Chart - Pen Renderer | 493 |
| 2.1.8.1.5 Easy Chart - Digital Offset | 496 |
| 2.1.8.1.6 Easy Chart - Calculated Pens | 499 |
| 2.1.8.1.7 Using the Tag Browse Tree for Charting | 507 |
| 2.1.8.1.8 Indirect Easy Chart | 510 |
| 2.1.8.1.9 Charting - Right Click Menu | 515 |
| 2.1.8.1.10 Easy Chart - Database Pens | 517 |
| 2.1.8.2 Using the Classic Chart | 520 |
| 2.1.8.3 Other Vision Trending Charts | 528 |
| 2.1.9 Alarming in Vision | 531 |
| 2.1.9.1 Vision Alarm Status Table - Common Tasks | 536 |
| 2.1.9.1.1 Vision Alarm Status - General Filtering | 538 |
| 2.1.9.1.2 Vision Alarm Status - Filter on Associated Data | 543 |
| 2.1.9.1.3 Vision Alarm Status - Row Styles | 545 |
| 2.1.9.1.4 Vision Alarm Status - Marquee Mode | 548 |
| 2.1.9.1.5 Vision Alarm Status - Acknowledgement | 550 |
| 2.1.9.1.6 Vision Alarm Status - Shelving | 552 |
| 2.1.9.1.7 Vision Alarm Status - Tag History and Alarm History | 555 |
| 2.1.9.1.8 Using Alarm Status Tags in Vision | 556 |
| 2.1.9.2 Vision Alarm Journal Table - Common Tasks | 561 |
| 2.1.9.2.1 Vision Alarm Journal - General Filtering | 562 |
| 2.1.9.2.2 Vision Alarm Journal - Filter on Associated Data | 566 |
| 2.1.9.2.3 Vision Alarm Journal - Filter on Date Range | 569 |
| 2.1.9.2.4 Vision Alarm Journal - Focusing on Alarms | 572 |
| 2.1.9.2.5 Vision Alarm Journal - Row Styles | 575 |
| 2.1.9.2.6 Vision Alarm Journal - Searching | 578 |
| 2.1.10 Reporting in Vision | 581 |
| 2.1.11 Vision Client Tags | 597 |
| 2.1.12 Vision Project Properties | 602 |
| 2.1.12.1 Client Update Modes | 613 |
| 2.1.12.2 Setting Up Auto Login | 617 |
| 2.1.12.3 Using Touch Screen Mode | 619 |
| 2.1.13 Common Tasks in Vision | 623 |
| 2.1.13.1 Component Animation | 624 |
| 2.1.13.2 Custom Input Template | 629 |
| 2.1.13.3 Client Tags for Indirection | 632 |
| 2.1.13.4 High Performance HMI Techniques | 644 |
| 2.1.13.5 Open Dynamic Windows on Startup | 647 |
| 2.1.13.6 Tank Cutaway | 649 |
| 2.1.13.7 Dropdown List Example | 652 |
| 2.1.13.8 Multi-Monitor Clients | 659 |
| 2.1.14 Local Client Fallback | 662 |
| 2.1.15 Client Launchers Commands | 664 |
| 2.1.16 Deploying Vision Client Launchers | 666 |
| 2.1.17 Vision Client Launcher Settings | 671 |
| 2.2 Perspective | 679 |
| 2.2.1 Perspective Sessions | 685 |
| 2.2.1.1 Ignition Perspective App | 695 |
| 2.2.1.2 Session Properties | 709 |
| 2.2.2 Perspective Design Principles | 715 |
| 2.2.2.1 Navigation Strategies in Perspective | 718 |
| 2.2.3 Pages in Perspective | 727 |
| 2.2.4 Views in Perspective | 738 |
| 2.2.4.1 Embedded Views | 742 |
| 2.2.4.2 Popup Views | 745 |
| 2.2.5 Containers in Perspective | 750 |
| 2.2.5.1 Coordinate Containers | 754 |
| 2.2.5.2 Column Containers | 761 |
| 2.2.5.3 Tab Containers | 770 |
| 2.2.5.4 Breakpoint Containers | 774 |
| 2.2.5.5 Flex Containers | 780 |
| 2.2.5.6 Test Your Responsive Design Using Chrome's Developer Tools | 792 |
| 2.2.6 Perspective Designer Interface | 795 |
| 2.2.7 Working with Perspective Components | 808 |
| 2.2.7.1 Perspective Component Properties | 823 |
| 2.2.7.2 Images, SVGs, and Icons in Perspective | 839 |
| 2.2.7.3 Localization in Perspective | 850 |
| 2.2.7.4 Component Events and Actions | 854 |
| 2.2.8 Perspective Project Properties | 873 |
| 2.2.9 Styles | 880 |
| 2.2.9.1 Perspective Themes | 888 |

| | |
|---|------|
| 2.2.9.2 Style Classes | 900 |
| 2.2.10 Bindings in Perspective | 917 |
| 2.2.10.1 Tag Bindings in Perspective | 931 |
| 2.2.10.1.1 Drop Configuration | 942 |
| 2.2.10.2 Property Bindings in Perspective | 953 |
| 2.2.10.3 Expression Bindings in Perspective | 958 |
| 2.2.10.4 Expression Structure Bindings in Perspective | 962 |
| 2.2.10.5 Query Bindings in Perspective | 967 |
| 2.2.10.6 Tag History Bindings in Perspective | 969 |
| 2.2.10.7 HTTP Bindings in Perspective | 975 |
| 2.2.10.8 Transforms | 977 |
| 2.2.10.8.1 Map Transform | 978 |
| 2.2.10.8.2 Format Transform | 983 |
| 2.2.10.8.3 Script Transform | 986 |
| 2.2.10.8.4 Expression Transform | 989 |
| 2.2.11 Scripting in Perspective | 990 |
| 2.2.11.1 Perspective Component Methods | 992 |
| 2.2.11.2 Component Message Handlers | 997 |
| 2.2.11.3 Perspective Property Change Scripts | 1002 |
| 2.2.11.4 Perspective Session Event Scripts | 1006 |
| 2.2.12 Security in Perspective | 1022 |
| 2.2.13 Alarming in Perspective | 1027 |
| 2.2.13.1 Perspective Alarm Status Table - Common Tasks | 1029 |
| 2.2.13.1.1 Perspective Alarm Status - User Interaction | 1030 |
| 2.2.13.1.2 Perspective Alarm Status - Configuring Properties in Designer | 1034 |
| 2.2.13.1.3 Perspective Alarm Status - Filtering | 1040 |
| 2.2.13.1.4 Perspective Alarm Status - Acknowledgement | 1044 |
| 2.2.13.1.5 Perspective Alarm Status - Shelving | 1049 |
| 2.2.13.1.6 Perspective Alarm Status - Row Styles | 1054 |
| 2.2.13.2 Perspective Alarm Journal Table - Common Tasks | 1063 |
| 2.2.13.2.1 Perspective Alarm Journal - User Interaction | 1064 |
| 2.2.13.2.2 Perspective Alarm Journal - Configuring Properties in Designer | 1066 |
| 2.2.13.2.3 Perspective Alarm Journal - Filtering | 1070 |
| 2.2.13.2.4 Perspective Alarm Journal - Row Styles | 1079 |
| 2.2.14 Reporting in Perspective | 1085 |
| 2.2.15 Common Tasks in Perspective | 1090 |
| 2.2.15.1 Self-Hiding Navigation Drawer | 1092 |
| 2.3 OPC UA | 1102 |
| 2.3.1 OPC UA Client Connection Settings | 1106 |
| 2.3.2 OPC UA Server Settings | 1111 |
| 2.3.3 Third Party OPC Servers | 1113 |
| 2.3.3.1 Connecting to Kepware OPC UA | 1114 |
| 2.3.3.2 OPC COM | 1118 |
| 2.3.4 OPC UA Security | 1122 |
| 2.4 OPC UA Drivers | 1130 |
| 2.4.1 Allen Bradley Ethernet | 1132 |
| 2.4.1.1 Connecting to CompactLogix | 1134 |
| 2.4.1.2 Connecting to ControlLogix | 1136 |
| 2.4.1.3 Connecting to Logix | 1138 |
| 2.4.1.4 Connecting to MicroLogix | 1141 |
| 2.4.1.5 Connecting to PLC5 | 1143 |
| 2.4.1.6 Connecting to SLC | 1146 |
| 2.4.1.7 Allen-Bradley Connection Paths | 1149 |
| 2.4.2 Modbus | 1155 |
| 2.4.2.1 Connecting to Modbus Device | 1156 |
| 2.4.2.2 Modbus Addressing | 1160 |
| 2.4.2.3 Modbus Address Mapping | 1163 |
| 2.4.3 Siemens | 1169 |
| 2.4.4 UDP and TCP Driver | 1173 |
| 2.4.5 DNP3 | 1176 |
| 2.4.6 Omron NJ Driver | 1180 |
| 2.4.7 Omron FINS Driver | 1186 |
| 2.4.8 Programmable Device Simulator | 1191 |
| 2.4.9 BACnet | 1198 |
| 2.4.9.1 BACnet Local Device | 1200 |
| 2.4.9.2 BACnet Remote Device | 1202 |
| 2.5 Tag Historian | 1204 |
| 2.5.1 Tag History Gateway Settings | 1206 |
| 2.5.2 Configuring Tag History | 1212 |
| 2.5.3 Data Partitioning and Pruning | 1217 |
| 2.5.4 Custom Tag History Aggregates | 1220 |
| 2.5.5 Tag History Splitter | 1223 |
| 2.5.6 Historian Simulator | 1227 |
| 2.5.7 DB Table Historian Provider | 1232 |
| 2.6 SQL Bridge (Transaction Groups) | 1238 |
| 2.6.1 Understanding Transaction Groups | 1243 |
| 2.6.1.1 Types of Groups | 1253 |
| 2.6.1.2 Item Types | 1259 |
| 2.6.1.3 Hour and Event Meters | 1266 |

| | |
|---|------|
| 2.6.2 Transaction Group Examples | 1272 |
| 2.6.2.1 Block Group | 1275 |
| 2.6.2.2 Recipe Group | 1282 |
| 2.6.2.3 Update or Insert Group | 1286 |
| 2.6.2.4 Trigger Options | 1289 |
| 2.6.2.5 Transaction Group Update Modes | 1293 |
| 2.6.2.6 OPC to OPC Transaction Group | 1298 |
| 2.7 Reporting | 1299 |
| 2.7.1 Report Designer Interface | 1303 |
| 2.7.2 Report Data | 1310 |
| 2.7.2.1 Report Parameters | 1313 |
| 2.7.2.2 Named Query Data Source | 1317 |
| 2.7.2.3 SQL Query Data Source | 1319 |
| 2.7.2.4 Basic SQL Query | 1321 |
| 2.7.2.5 Tag Historian Query | 1324 |
| 2.7.2.6 Tag Calculation Query | 1326 |
| 2.7.2.7 Alarm Journal Query | 1328 |
| 2.7.2.8 Scripting Data Source | 1330 |
| 2.7.2.9 Static CSV | 1336 |
| 2.7.2.10 Nested Queries | 1337 |
| 2.7.3 Report Design | 1341 |
| 2.7.3.1 Report Design Tools | 1345 |
| 2.7.3.2 Stroke and Fill Properties for Reports | 1349 |
| 2.7.3.3 Data Keys | 1357 |
| 2.7.3.3.1 Keychain Expressions | 1367 |
| 2.7.3.4 Report Charts | 1370 |
| 2.7.3.4.1 Getting Started with Report Charts | 1375 |
| 2.7.3.5 Report Tables | 1380 |
| 2.7.3.5.1 Getting Started with the Report Table | 1386 |
| 2.7.3.5.2 Table Rows | 1390 |
| 2.7.3.5.3 Table Row Versioning | 1395 |
| 2.7.3.5.4 Charts Inside of Tables | 1399 |
| 2.7.3.5.5 Grouping Data Inside of Tables | 1409 |
| 2.7.3.5.6 Table Groups | 1416 |
| 2.7.3.6 Images in Reports | 1422 |
| 2.7.3.7 CrossTab and Simple Tables | 1428 |
| 2.7.4 Report Schedules | 1433 |
| 2.7.4.1 Scheduling Actions | 1437 |
| 2.7.5 Common Reporting Tasks | 1444 |
| 2.7.5.1 Tutorial: The Report Workflow | 1445 |
| 2.7.5.2 Labels with Embedded Barcodes | 1457 |
| 2.7.5.3 Converting Legacy Reports | 1462 |
| 2.8 Alarm Notification | 1464 |
| 2.8.1 Notification Contact Info | 1467 |
| 2.8.2 On-Call Rosters | 1470 |
| 2.8.3 Notification Profile Types | 1474 |
| 2.8.3.1 Email Notification Profile | 1475 |
| 2.8.3.1.1 Email Alarm Acknowledgement | 1480 |
| 2.8.3.2 Simple One-Way Email Notification Profile | 1482 |
| 2.8.3.3 SMS Notification Profile | 1485 |
| 2.8.3.4 Twilio SMS Notification | 1488 |
| 2.8.3.5 Voice Notification Profile | 1491 |
| 2.8.3.5.1 Notification Security PIN | 1496 |
| 2.8.3.5.2 Voice Notification Scripts | 1499 |
| 2.8.3.5.3 Registering Voice Alarm Module to CUCM | 1504 |
| 2.8.3.6 Remote Gateway Notification | 1507 |
| 2.8.4 Alarm Notification Pipelines | 1512 |
| 2.8.4.1 Alarm Pipeline Designer Interface | 1516 |
| 2.8.4.2 Simple Pipeline | 1519 |
| 2.8.4.3 Pipeline Blocks | 1524 |
| 2.8.4.3.1 Notification Block | 1532 |
| 2.8.4.4 Pipeline - Filter on Alarm Priority | 1544 |
| 2.8.4.5 Pipeline - Filter on Alarm Associated Data | 1547 |
| 2.8.4.6 Pipeline - Simple Loop | 1551 |
| 2.8.4.7 Pipeline - Escalation | 1553 |
| 2.8.4.8 Pipeline - Notification Block Consolidation | 1557 |
| 2.8.4.9 Pipeline Status | 1558 |
| 2.8.5 Adding Pipelines to Tags | 1559 |
| 2.9 Enterprise Administration | 1562 |
| 2.9.1 Creating a Controller | 1565 |
| 2.9.2 Adding an Agent | 1568 |
| 2.9.3 Agent Tasks | 1572 |
| 2.9.3.1 Agent Task - Backup and Restore | 1576 |
| 2.9.3.2 Agent Task - Licensing | 1582 |
| 2.9.3.3 Agent Task - Send Project | 1586 |
| 2.9.3.4 Agent Task - Scheduling | 1594 |
| 2.9.4 Agent Management | 1597 |
| 2.9.4.1 Agent Recovery | 1600 |
| 2.9.4.2 Automated Agent Installation | 1603 |

| | |
|--|------|
| 2.9.4.3 License Management | 1605 |
| 2.9.4.4 Remote Upgrade | 1607 |
| 2.9.5 Redundant EAM Configuration | 1612 |
| 2.9.6 Event Threshold Settings | 1613 |
| 2.9.7 EAM in the Designer | 1615 |
| 2.10 Sequential Function Charts | 1619 |
| 2.10.1 SFC Designer Interface | 1625 |
| 2.10.2 SFC Basics | 1632 |
| 2.10.2.1 Chart Flow and Rules | 1634 |
| 2.10.2.2 Chart Scope and Variables | 1641 |
| 2.10.2.3 Chart Properties | 1644 |
| 2.10.3 SFC Elements | 1647 |
| 2.10.4 SFCs in Action | 1659 |
| 2.10.4.1 Monitoring and Debugging Charts | 1663 |
| 2.10.4.2 Pause, Resume, and Cancel | 1665 |
| 2.10.4.3 Action Step Best Practices | 1667 |
| 2.11 Symbol Factory | 1668 |
| 2.12 SECS/GEM | 1671 |
| 2.12.1 Using the SECS/GEM Module | 1673 |
| 2.12.2 SECS/GEM Equipment Connections | 1680 |
| 2.12.3 SECS Definition Language (SDL) File | 1685 |
| 2.12.4 SECS/GEM Messages | 1718 |
| 2.12.5 SECS/GEM Simulator | 1725 |
| 2.12.5.1 Additional SECS/GEM Simulator Features | 1728 |
| 2.12.6 Basic SECS/GEM Troubleshooting | 1733 |
| 2.13 WebDev | 1735 |
| 2.13.1 httpPost | 1743 |
| 3. Ignition Platform | 1745 |
| 3.1 Gateway | 1747 |
| 3.1.1 Home | 1754 |
| 3.1.2 Status | 1757 |
| 3.1.2.1 Systems | 1761 |
| 3.1.2.2 Connections | 1776 |
| 3.1.2.2.1 Connections - EAM Agents | 1777 |
| 3.1.2.2.2 Connections - Databases | 1779 |
| 3.1.2.2.3 Connections - Designers | 1782 |
| 3.1.2.2.4 Connections - Devices | 1785 |
| 3.1.2.2.5 Connections - Gateway Network | 1788 |
| 3.1.2.2.6 Connections - Store & Forward | 1795 |
| 3.1.2.2.7 Connections - OPC Connections | 1798 |
| 3.1.2.2.8 Connections - SECS/GEM Equipment | 1803 |
| 3.1.2.2.9 Connections - Perspective Sessions | 1806 |
| 3.1.2.2.10 Connections - Vision Clients | 1811 |
| 3.1.2.3 Diagnostics - Execution | 1814 |
| 3.1.2.4 Diagnostics - Logs | 1815 |
| 3.1.2.5 Diagnostics - Running Scripts | 1819 |
| 3.1.2.6 Diagnostics - Threads | 1820 |
| 3.1.3 Config | 1823 |
| 3.1.3.1 Gateway Settings | 1828 |
| 3.1.3.2 Email Settings | 1832 |
| 3.1.3.3 Web Server Settings | 1834 |
| 3.1.3.3.1 Secure Communication (SSL / TLS) | 1839 |
| 3.1.4 Gateway Backup and Restore | 1845 |
| 3.1.4.1 Gateway Scheduled Backups | 1848 |
| 3.1.4.2 Project Export and Import | 1851 |
| 3.1.5 Ignition Exchange | 1856 |
| 3.1.6 Gateway Command-line Utility - gwcmd | 1862 |
| 3.2 Projects | 1864 |
| 3.2.1 Project Inheritance | 1868 |
| 3.2.2 Project Templates | 1877 |
| 3.2.3 Project Settings | 1887 |
| 3.3 Ignition Redundancy | 1892 |
| 3.3.1 Setting Up Redundancy | 1895 |
| 3.3.2 Database Considerations | 1901 |
| 3.3.3 Redundant Licensing | 1903 |
| 3.4 Gateway Network | 1905 |
| 3.5 Database Connections | 1916 |
| 3.5.1 Installing Databases | 1921 |
| 3.5.1.1 Installing MySQL | 1923 |
| 3.5.1.2 Installing Microsoft SQL Server Express | 1932 |
| 3.5.1.3 Installing PostgreSQL | 1934 |
| 3.5.2 Connecting to Databases | 1935 |
| 3.5.2.1 Connecting to MySQL | 1937 |
| 3.5.2.2 Connecting to Microsoft SQL Server Express | 1941 |
| 3.5.2.3 Connecting to Oracle Express | 1952 |
| 3.5.2.4 Connecting to PostgreSQL | 1956 |
| 3.5.2.5 Connecting to SQLite | 1958 |
| 3.5.2.6 JDBC Drivers and Translators | 1961 |
| 3.5.3 Store and Forward | 1971 |

| | |
|---|------|
| 3.5.3.1 Using Store and Forward | 1973 |
| 3.5.3.2 Configuring Store and Forward | 1976 |
| 3.5.3.3 Controlling Quarantine Data | 1978 |
| 3.6 Security | 1980 |
| 3.6.1 Gateway General Security Settings | 1981 |
| 3.6.2 Classic Authentication Strategy | 1984 |
| 3.6.2.1 Managing Users and Roles | 1988 |
| 3.6.2.1.1 User Schedules | 1993 |
| 3.6.2.2 Internal Authentication | 1997 |
| 3.6.2.3 Database Authentication | 1999 |
| 3.6.2.4 Active Directory Authentication | 2003 |
| 3.6.2.5 AD Internal Hybrid | 2008 |
| 3.6.2.6 AD Database Hybrid | 2012 |
| 3.6.2.7 Fallback Cache Authentication | 2016 |
| 3.6.2.8 Verify a User on a User Source | 2018 |
| 3.6.3 Identity Provider Authentication Strategy | 2020 |
| 3.6.3.1 Configuring Identity Providers | 2023 |
| 3.6.3.1.1 User Attribute Mapping | 2032 |
| 3.6.3.1.2 OpenID Connect 1.0 Example | 2038 |
| 3.6.3.1.3 SAML Example | 2046 |
| 3.6.3.2 User Grants | 2053 |
| 3.6.3.3 Test Login and Logout | 2055 |
| 3.6.3.4 Security Levels | 2059 |
| 3.6.3.5 Security Level Rules | 2064 |
| 3.6.3.6 Troubleshooting Identity Providers | 2069 |
| 3.6.3.7 Referencing User Information | 2072 |
| 3.6.4 Service Security | 2076 |
| 3.6.5 Security Zones | 2079 |
| 3.6.6 Project Security in the Designer | 2082 |
| 3.7 Designer | 2085 |
| 3.7.1 General Designer Interface | 2096 |
| 3.7.2 Designer Tools | 2106 |
| 3.7.2.1 Image Management Tool | 2109 |
| 3.7.2.2 Script Console | 2112 |
| 3.7.2.3 Database Query Browser | 2118 |
| 3.7.2.4 Output Console | 2122 |
| 3.7.3 Project Properties | 2124 |
| 3.7.4 Find and Replace | 2132 |
| 3.7.5 Keyboard Shortcuts | 2137 |
| 3.7.6 Saving Projects | 2140 |
| 3.7.7 Designer Diagnostics | 2143 |
| 3.8 Tags | 2150 |
| 3.8.1 Tag Browser | 2153 |
| 3.8.2 Types of Tags | 2163 |
| 3.8.2.1 System Tags | 2169 |
| 3.8.3 Creating Tags | 2178 |
| 3.8.4 User Defined Types - UDTs | 2191 |
| 3.8.4.1 UDT Parameters | 2207 |
| 3.8.4.2 UDT Multi-Instance Wizard | 2215 |
| 3.8.4.3 UDT Inheritance | 2219 |
| 3.8.4.4 UDT Nesting | 2227 |
| 3.8.5 Tag Groups | 2231 |
| 3.8.6 Tag Providers | 2238 |
| 3.8.7 Tag Event Scripts | 2241 |
| 3.8.8 Tag Properties | 2246 |
| 3.8.8.1 Tag Alarm Properties | 2259 |
| 3.8.8.2 Tag Scaling Properties | 2266 |
| 3.8.8.3 Tag Security Properties | 2269 |
| 3.8.9 Tag Data Types | 2273 |
| 3.8.10 Tag Paths | 2278 |
| 3.8.11 Tag Quality and Overlays | 2282 |
| 3.8.12 Exporting and Importing Tags | 2294 |
| 3.9 Alarming | 2301 |
| 3.9.1 Alarm Journal | 2307 |
| 3.9.2 Configuring Alarms | 2317 |
| 3.9.2.1 Dynamic Alarm Attributes | 2325 |
| 3.9.2.2 Alarms in UDTs | 2330 |
| 3.9.2.3 Alarm Associated Data | 2334 |
| 3.10 Localization and Languages | 2338 |
| 3.10.1 Creating Translation Lists | 2342 |
| 3.10.2 Switching the Current Language | 2353 |
| 3.10.3 Localization Best Practices | 2354 |
| 3.10.4 Translating Built-in Terms | 2360 |
| 3.11 Expression Language and Syntax | 2364 |
| 3.12 SQL in Ignition | 2371 |
| 3.12.1 Writing SQL Queries | 2378 |
| 3.12.1.1 SQL Select Statements | 2383 |
| 3.12.1.2 SQL Where Clauses | 2386 |
| 3.12.1.3 SQL Table Joins | 2391 |

| | |
|--|------|
| 3.12.1.4 SQL Common Functions | 2397 |
| 3.12.1.5 SQL Stored Procedures | 2402 |
| 3.12.2 Query Builder | 2406 |
| 3.12.3 Named Queries | 2413 |
| 3.12.3.1 Named Query Workspace | 2417 |
| 3.12.3.2 Named Query Parameters | 2424 |
| 3.12.3.3 Named Query Caching | 2427 |
| 3.12.3.4 Named Query Conversions | 2429 |
| 3.12.3.5 Using Named Queries - Example | 2433 |
| 3.12.4 Queries in Scripting | 2437 |
| 3.12.5 Common SQL Tasks | 2440 |
| 3.12.5.1 Filter Rows in a Table | 2442 |
| 3.12.5.2 Inserting Data into a Database | 2445 |
| 3.12.5.3 Updating the Database through the Power Table | 2448 |
| 3.12.5.4 Refreshing a SQL Query | 2452 |
| 3.12.5.5 Editing Multi-Selected Rows from a Table | 2454 |
| 3.12.5.6 Storing Files in a Database | 2456 |
| 3.12.5.7 Simple Database Editor | 2459 |
| 3.12.6 Basic SQL Troubleshooting | 2468 |
| 3.12.6.1 Slow Queries | 2471 |
| 3.12.6.2 SQL Query Volume Optimization | 2473 |
| 3.13 Scripting | 2477 |
| 3.13.1 Python Scripting | 2479 |
| 3.13.1.1 Variables, Data Types, and Objects | 2483 |
| 3.13.1.1.1 Numeric Types | 2490 |
| 3.13.1.1.2 Strings | 2492 |
| 3.13.1.1.3 Lists and Tuples | 2500 |
| 3.13.1.1.4 Dictionaries | 2505 |
| 3.13.1.1.5 Datasets | 2508 |
| 3.13.1.1.6 Dates | 2513 |
| 3.13.1.2 Conditions and Loops | 2518 |
| 3.13.1.3 Error Handling | 2523 |
| 3.13.1.4 Built-In Functions | 2527 |
| 3.13.1.5 Libraries | 2531 |
| 3.13.1.6 User Defined Functions | 2534 |
| 3.13.2 Scripting in Ignition | 2537 |
| 3.13.2.1 Getting Started with Scripting in Ignition | 2544 |
| 3.13.2.2 Gateway Event Scripts | 2550 |
| 3.13.2.3 Project Library | 2559 |
| 3.13.2.4 Web Services, SUDS, and REST | 2564 |
| 3.13.2.4.1 HTTP Methods | 2565 |
| 3.13.2.4.2 SUDS - Library Overview | 2569 |
| 3.13.2.5 JSON Format | 2574 |
| 3.13.4 Basic Python Troubleshooting | 2578 |
| 3.13.4.1 Reading Error Messages | 2582 |
| 3.13.4.2 Troubleshooting - Nothing Happened | 2586 |
| 3.13.4.3 Troubleshooting Workflow | 2588 |
| 3.13.4.4 Scripting Vs. SQL Vs. Expressions | 2594 |
| 3.13.5 Scripting Examples | 2599 |
| 3.13.5.1 Location Based Vision Startup Scripts | 2600 |
| 3.13.5.2 Reading and Writing to Tags | 2603 |
| 3.13.5.3 Exporting and Importing a CSV | 2606 |
| 3.13.5.4 Adding a Delay to a Script | 2612 |
| 3.13.5.5 Export Tag Historian to CSV | 2617 |
| 3.13.5.6 Parsing XML with the Etree Library | 2620 |
| 3.14 Audit Log and Profiles | 2625 |
| 3.14.1 Alarm Notification Auditing | 2632 |
| 3.14.2 Audit Log Display | 2633 |
| 3.14.3 Auditing Actions Reference | 2636 |
| 4. Components | 2641 |
| 4.1 Perspective Components | 2642 |
| 4.1.1 Perspective - Chart Palette | 2643 |
| 4.1.1.1 Perspective - Chart Range Selector | 2644 |
| 4.1.1.2 Perspective - Pie Chart | 2653 |
| 4.1.1.3 Perspective - Power Chart | 2658 |
| 4.1.1.4 Perspective - Simple Gauge | 2679 |
| 4.1.1.5 Perspective - Time Series Chart | 2682 |
| 4.1.1.6 Perspective - XY Chart | 2697 |
| 4.1.2 Perspective - Container Palette | 2727 |
| 4.1.2.1 Perspective - Breakpoint Container | 2728 |
| 4.1.2.2 Perspective - Column Container | 2730 |
| 4.1.2.3 Perspective - Coordinate Container | 2732 |
| 4.1.2.4 Perspective - Flex Container | 2735 |
| 4.1.2.5 Perspective - Tab Container | 2742 |
| 4.1.3 Perspective - Display Palette | 2748 |
| 4.1.3.1 Perspective - Alarm Journal Table | 2749 |
| 4.1.3.2 Perspective - Alarm Status Table | 2761 |
| 4.1.3.3 Perspective - Barcode | 2780 |
| 4.1.3.4 Perspective - Cylindrical Tank | 2784 |

| | |
|---|------|
| 4.1.3.5 Perspective - Dashboard | 2786 |
| 4.1.3.5.1 Configuring a Dashboard | 2793 |
| 4.1.3.6 Perspective - Icon | 2803 |
| 4.1.3.7 Perspective - Image | 2805 |
| 4.1.3.8 Perspective - Inline Frame | 2808 |
| 4.1.3.9 Perspective - Label | 2810 |
| 4.1.3.10 Perspective - LED Display | 2812 |
| 4.1.3.11 Perspective - Linear Scale | 2814 |
| 4.1.3.12 Perspective - Map | 2818 |
| 4.1.3.13 Perspective - Markdown | 2839 |
| 4.1.3.14 Perspective - Moving Analog Indicator | 2843 |
| 4.1.3.15 Perspective - PDF Viewer | 2847 |
| 4.1.3.16 Perspective - Progress | 2849 |
| 4.1.3.17 Perspective - Sparkline | 2853 |
| 4.1.3.18 Perspective - Table | 2860 |
| 4.1.3.18.1 Displaying a SubView in a Table | 2875 |
| 4.1.3.18.2 Table Column Configurations | 2884 |
| 4.1.3.19 Perspective - Thermometer | 2892 |
| 4.1.3.20 Perspective - Tree | 2895 |
| 4.1.3.21 Perspective - Video Player | 2899 |
| 4.1.4 Perspective - Embedding Palette | 2903 |
| 4.1.4.1 Perspective - Accordion | 2904 |
| 4.1.4.2 Perspective - Carousel | 2909 |
| 4.1.4.2.1 Carousel Component Example | 2915 |
| 4.1.4.3 Perspective - Embedded View | 2926 |
| 4.1.4.4 Perspective - Flex Repeater | 2929 |
| 4.1.4.5 Perspective - View Canvas | 2936 |
| 4.1.5 Perspective - Input Palette | 2940 |
| 4.1.5.1 Perspective - Barcode Scanner Input | 2941 |
| 4.1.5.2 Perspective - Button | 2943 |
| 4.1.5.3 Perspective - Checkbox | 2946 |
| 4.1.5.4 Perspective - DateTime Input | 2949 |
| 4.1.5.5 Perspective - DateTime Picker | 2951 |
| 4.1.5.6 Perspective - Dropdown | 2954 |
| 4.1.5.7 Perspective - File Upload | 2960 |
| 4.1.5.7.1 Download and Upload Files | 2963 |
| 4.1.5.8 Perspective - Multi-State Button | 2967 |
| 4.1.5.9 Perspective - Numeric Entry Field | 2970 |
| 4.1.5.10 Perspective - One-Shot Button | 2973 |
| 4.1.5.11 Perspective - Password Field | 2976 |
| 4.1.5.12 Perspective - Radio Group | 2978 |
| 4.1.5.13 Perspective - Signature Pad | 2981 |
| 4.1.5.14 Perspective - Slider | 2988 |
| 4.1.5.15 Perspective - Text Area | 2990 |
| 4.1.5.16 Perspective - Text Field | 2992 |
| 4.1.5.17 Perspective - Toggle Switch | 2994 |
| 4.1.6 Perspective - Navigation Palette | 2996 |
| 4.1.6.1 Perspective - Horizontal Menu | 2997 |
| 4.1.6.1.1 Navigating with the Horizontal Menu Component | 3004 |
| 4.1.6.2 Perspective - Link | 3013 |
| 4.1.6.3 Perspective - Menu Tree | 3014 |
| 4.1.7 Perspective - Report Viewer | 3024 |
| 4.1.8 Perspective - View Object | 3028 |
| 4.1.9 Perspective - Symbols Palette | 3031 |
| 4.1.9.1 Perspective - Motor | 3032 |
| 4.1.9.2 Perspective - Pump | 3035 |
| 4.1.9.3 Perspective - Sensor | 3038 |
| 4.1.9.4 Perspective - Valve | 3040 |
| 4.1.9.5 Perspective - Vessel | 3042 |
| 4.2 Vision Components | 3046 |
| 4.2.1 Vision - Input Palette | 3047 |
| 4.2.1.1 Vision - Text Field | 3048 |
| 4.2.1.2 Vision - Numeric Text Field | 3055 |
| 4.2.1.3 Vision - Spinner | 3063 |
| 4.2.1.4 Vision - Formatted Text Field | 3066 |
| 4.2.1.5 Vision - Password Field | 3074 |
| 4.2.1.6 Vision - Text Area | 3081 |
| 4.2.1.7 Vision - Dropdown List | 3088 |
| 4.2.1.8 Vision - Slider | 3096 |
| 4.2.1.9 Vision - Language Selector | 3104 |
| 4.2.2 Vision - Buttons Palette | 3109 |
| 4.2.2.1 Vision - Button | 3110 |
| 4.2.2.2 Vision - 2 State Toggle | 3118 |
| 4.2.2.3 Vision - Multi-State Button | 3126 |
| 4.2.2.4 Vision - One-Shot Button | 3134 |
| 4.2.2.5 Vision - Momentary Button | 3142 |
| 4.2.2.6 Vision - Toggle Button | 3149 |
| 4.2.2.7 Vision - Check Box | 3156 |
| 4.2.2.8 Vision - Radio Button | 3163 |

| | |
|--|------|
| 4.2.2.9 Vision - Tab Strip | 3170 |
| 4.2.3 Vision - Display Palette | 3177 |
| 4.2.3.1 Vision - Label | 3178 |
| 4.2.3.2 Vision - Numeric Label | 3183 |
| 4.2.3.3 Vision - Multi-State Indicator | 3188 |
| 4.2.3.4 Vision - LED Display | 3194 |
| 4.2.3.5 Vision - Moving Analog Indicator | 3200 |
| 4.2.3.6 Vision - Image | 3206 |
| 4.2.3.7 Vision - Progress Bar | 3212 |
| 4.2.3.8 Vision - Cylindrical Tank | 3217 |
| 4.2.3.9 Vision - Level Indicator | 3222 |
| 4.2.3.10 Vision - Linear Scale | 3228 |
| 4.2.3.11 Vision - Barcode | 3235 |
| 4.2.3.12 Vision - Meter | 3240 |
| 4.2.3.13 Vision - Compass | 3247 |
| 4.2.3.14 Vision - Thermometer | 3253 |
| 4.2.3.15 Vision - IP Camera Viewer | 3259 |
| 4.2.4 Vision - Tables Palette | 3266 |
| 4.2.4.1 Vision - Table | 3267 |
| 4.2.4.1.1 Vision - Table Customizer | 3281 |
| 4.2.4.2 Vision - Power Table | 3286 |
| 4.2.4.2.1 Vision - Power Table Customizer | 3298 |
| 4.2.4.3 Vision - List | 3301 |
| 4.2.4.4 Vision - Tree View | 3310 |
| 4.2.4.4.1 Vision - Tree View Customizer | 3317 |
| 4.2.4.5 Vision - Comments Panel | 3319 |
| 4.2.4.6 Vision - Tag Browse Tree | 3329 |
| 4.2.5 Vision - Charts Palette | 3335 |
| 4.2.5.1 Vision - Easy Chart | 3336 |
| 4.2.5.1.1 Vision - Easy Chart Customizer | 3348 |
| 4.2.5.2 Vision - Chart | 3355 |
| 4.2.5.2.1 Vision - Chart Customizer | 3362 |
| 4.2.5.3 Vision - Sparkline Chart | 3382 |
| 4.2.5.4 Vision - Bar Chart | 3388 |
| 4.2.5.5 Vision - Radar Chart | 3395 |
| 4.2.5.6 Vision - Status Chart | 3401 |
| 4.2.5.7 Vision - Pie Chart | 3411 |
| 4.2.5.8 Vision - Box and Whisker Chart | 3417 |
| 4.2.5.9 Vision - Equipment Schedule | 3426 |
| 4.2.5.10 Vision - Gantt Chart | 3438 |
| 4.2.6 Vision - Calendar Palette | 3444 |
| 4.2.6.1 Vision - Calendar | 3445 |
| 4.2.6.2 Vision - Popup Calendar | 3450 |
| 4.2.6.3 Vision - Date Range | 3455 |
| 4.2.6.4 Vision - Day View | 3462 |
| 4.2.6.5 Vision - Week View | 3468 |
| 4.2.6.6 Vision - Month View | 3474 |
| 4.2.7 Vision - Admin Palette | 3479 |
| 4.2.7.1 Vision - User Management | 3480 |
| 4.2.7.2 Vision - Schedule Management | 3488 |
| 4.2.7.3 Vision - Roster Management | 3496 |
| 4.2.7.4 Vision - SFC Monitor | 3502 |
| 4.2.8 Vision - Alarming Palette | 3507 |
| 4.2.8.1 Vision - Alarm Status Table | 3508 |
| 4.2.8.1.1 Vision - Alarm Row Style Customizer | 3515 |
| 4.2.8.2 Vision - Alarm Journal Table | 3519 |
| 4.2.9 Vision - Containers Palette | 3528 |
| 4.2.9.1 Vision - Container | 3529 |
| 4.2.9.2 Vision - Template Repeater | 3534 |
| 4.2.9.3 Vision - Template Canvas | 3537 |
| 4.2.10 Vision - Misc Palette | 3542 |
| 4.2.10.1 Vision - Paintable Canvas | 3543 |
| 4.2.10.2 Vision - Line | 3549 |
| 4.2.10.3 Vision - Pipe Segment | 3554 |
| 4.2.10.4 Vision - Pipe Joint | 3559 |
| 4.2.10.5 Vision - Sound Player | 3564 |
| 4.2.10.6 Vision - Timer | 3568 |
| 4.2.10.7 Vision - Signal Generator | 3570 |
| 4.2.11 Vision - Reporting Palette | 3572 |
| 4.2.11.1 Vision - Report Viewer | 3573 |
| 4.2.11.2 Vision - Row Selector | 3581 |
| 4.2.11.3 Vision - Column Selector | 3587 |
| 4.2.11.4 Vision - File Explorer | 3592 |
| 4.2.11.5 Vision - PDF Viewer | 3597 |
| 4.2.12 Vision - Web Browser Palette | 3604 |
| 4.2.12.1 Vision - Web Browser Component | 3605 |
| 4.2.13 Vision - The Window Object | 3610 |
| 4.2.14 Copy of -- WIP Page just with frequently used notes | 3614 |
| 4.3 Report Design Components | 3615 |

| | |
|---------------------------------------|------|
| 4.3.1 Report Design - Chart Palette | 3616 |
| 4.3.1.1 Report - Bar Chart | 3617 |
| 4.3.1.2 Report - Pie Chart | 3622 |
| 4.3.1.3 Report - Timeseries Chart | 3626 |
| 4.3.1.4 Report - XY Chart | 3632 |
| 4.3.2 Report Design - Display Palette | 3637 |
| 4.3.2.1 Report - Barcode | 3638 |
| 4.3.2.2 Report - Images | 3642 |
| 4.3.2.3 Report - Labels | 3645 |
| 4.3.3 Report Design - Object Palette | 3650 |
| 4.3.3.1 Report - Page Object | 3651 |
| 4.3.3.2 Report - Report Object | 3653 |
| 4.3.4 Report Design - Shape Palette | 3655 |
| 4.3.4.1 Report - Ellipse Shape | 3656 |
| 4.3.4.2 Report - Line Shape | 3658 |
| 4.3.4.3 Report - Pencil Shape | 3660 |
| 4.3.4.4 Report - Polygon Shape | 3662 |
| 4.3.4.5 Report - Rectangle Shape | 3664 |
| 4.3.4.6 Report - Star Shape | 3667 |
| 4.3.4.7 Report - Text Shape | 3670 |
| 4.3.5 Report Design - Table Palette | 3674 |
| 4.3.5.1 Report - Crosstab Table | 3675 |
| 4.3.5.2 Report - Simple Table | 3680 |
| 4.3.5.3 Report - Table | 3685 |
| 5. Expression Functions | 3692 |
| 5.1 Advanced | 3693 |
| 5.1.1 columnRearrange | 3694 |
| 5.1.2 columnRename | 3695 |
| 5.1.3 forceQuality | 3696 |
| 5.1.4 property | 3697 |
| 5.1.5 qualifiedValue | 3698 |
| 5.1.6 runScript | 3699 |
| 5.1.7 sortDataset | 3701 |
| 5.1.8 tag | 3703 |
| 5.1.9 typeOf | 3704 |
| 5.2 Aggregates | 3705 |
| 5.2.1 groupConcat | 3706 |
| 5.2.2 max | 3708 |
| 5.2.3 maxDate | 3710 |
| 5.2.4 mean | 3712 |
| 5.2.5 median | 3714 |
| 5.2.6 min | 3716 |
| 5.2.7 minDate | 3718 |
| 5.2.8 stdDev | 3720 |
| 5.2.9 sum | 3722 |
| 5.3 Alarming Expressions | 3724 |
| 5.3.1 isAlarmActive | 3725 |
| 5.3.2 isAlarmActiveFiltered | 3726 |
| 5.4 Colors | 3727 |
| 5.4.1 brighter | 3728 |
| 5.4.2 color | 3729 |
| 5.4.3 darker | 3730 |
| 5.4.4 gradient | 3731 |
| 5.5 Date and Time | 3732 |
| 5.5.1 add* | 3733 |
| 5.5.2 dateArithmetic | 3734 |
| 5.5.3 dateDiff | 3735 |
| 5.5.4 dateExtract | 3736 |
| 5.5.5 dateFormat | 3737 |
| 5.5.6 datelsAfter | 3738 |
| 5.5.7 datelsBefore | 3739 |
| 5.5.8 datelsBetween | 3740 |
| 5.5.9 datelsDaylight | 3741 |
| 5.5.10 *Between | 3742 |
| 5.5.11 fromMillis | 3743 |
| 5.5.12 get* | 3744 |
| 5.5.13 getDate | 3745 |
| 5.5.14 getTimezone | 3746 |
| 5.5.15 getTimezoneOffset | 3755 |
| 5.5.16 getTimezoneRawOffset | 3756 |
| 5.5.17 midnight | 3757 |
| 5.5.18 now | 3758 |
| 5.5.19 setTime | 3759 |
| 5.5.20 timeBetween | 3760 |
| 5.5.21 toMillis | 3761 |
| 5.6 Identity Provider | 3762 |
| 5.6.1 containsAll | 3763 |
| 5.6.2 containsAny | 3764 |
| 5.7 JSON | 3765 |

| | |
|------------------------|------|
| 5.7.1 jsonFormat | 3766 |
| 5.7.2 jsonGet | 3767 |
| 5.7.3 jsonSet | 3768 |
| 5.8 Logic | 3769 |
| 5.8.1 binEnc | 3770 |
| 5.8.2 binEnum | 3771 |
| 5.8.3 case | 3772 |
| 5.8.4 coalesce | 3773 |
| 5.8.5 getBit | 3774 |
| 5.8.6 hasChanged | 3775 |
| 5.8.7 if | 3776 |
| 5.8.8 isGood | 3777 |
| 5.8.9isNull | 3778 |
| 5.8.10 lookup | 3779 |
| 5.8.11 switch | 3781 |
| 5.8.12 try | 3782 |
| 5.9 Math | 3783 |
| 5.9.1 abs | 3784 |
| 5.9.2 acos | 3785 |
| 5.9.3 asin | 3786 |
| 5.9.4 atan | 3787 |
| 5.9.5 ceil | 3788 |
| 5.9.6 cos | 3789 |
| 5.9.7 exp | 3790 |
| 5.9.8 floor | 3791 |
| 5.9.9 log | 3792 |
| 5.9.10 log10 | 3793 |
| 5.9.11 pow | 3794 |
| 5.9.12 round | 3795 |
| 5.9.13 sin | 3796 |
| 5.9.14 sqrt | 3797 |
| 5.9.15 tan | 3798 |
| 5.9.16 toDegrees | 3799 |
| 5.9.17 toRadians | 3800 |
| 5.10 String | 3801 |
| 5.10.1 concat | 3802 |
| 5.10.2 escapeSQL | 3803 |
| 5.10.3 escapeXML | 3804 |
| 5.10.4 fromBinary | 3805 |
| 5.10.5 fromHex | 3806 |
| 5.10.6 fromOctal | 3807 |
| 5.10.7 indexOf | 3808 |
| 5.10.8 lastIndexOf | 3809 |
| 5.10.9 left | 3810 |
| 5.10.10 len | 3811 |
| 5.10.11 lower | 3812 |
| 5.10.12 numberFormat | 3813 |
| 5.10.13 repeat | 3814 |
| 5.10.14 replace | 3815 |
| 5.10.15 right | 3816 |
| 5.10.16 split | 3817 |
| 5.10.17 substring | 3818 |
| 5.10.18 toBinary | 3819 |
| 5.10.19 toHex | 3820 |
| 5.10.20 toOctal | 3821 |
| 5.10.21 trim | 3822 |
| 5.10.22 upper | 3823 |
| 5.10.23 stringFormat | 3824 |
| 5.10.24 urlEncode | 3825 |
| 5.11 Translation | 3828 |
| 5.11.1 translate | 3829 |
| 5.12 Type Casting | 3830 |
| 5.12.1 toBoolean | 3831 |
| 5.12.2 toBorder | 3832 |
| 5.12.3 toColor | 3835 |
| 5.12.4 toDataSet | 3840 |
| 5.12.5 toDate | 3841 |
| 5.12.6 toDouble | 3842 |
| 5.12.7 toFloat | 3843 |
| 5.12.8 toFont | 3844 |
| 5.12.9 toInt | 3845 |
| 5.12.10 toInteger | 3846 |
| 5.12.11 toLong | 3847 |
| 5.12.12 toStr | 3848 |
| 5.12.13 toString | 3849 |
| 5.13 Users | 3850 |
| 5.13.1 hasRole | 3851 |
| 5.13.2 isAuthorized | 3852 |
| 6. Scripting Functions | 3853 |

| | |
|--|------|
| 6.1 system.alarm | 3854 |
| 6.1.1 system.alarm.acknowledge | 3855 |
| 6.1.2 system.alarm.cancel | 3857 |
| 6.1.3 system.alarm.createRoster | 3858 |
| 6.1.4 system.alarm.getRosters | 3859 |
| 6.1.5 system.alarm.getShelvedPaths | 3861 |
| 6.1.6 system.alarm.listPipelines | 3862 |
| 6.1.7 system.alarm.queryJournal | 3863 |
| 6.1.8 system.alarm.queryStatus | 3865 |
| 6.1.9 system.alarm.shelve | 3868 |
| 6.1.10 system.alarm.unshelve | 3869 |
| 6.2 system.bacnet | 3870 |
| 6.2.1 system.bacnet.synchronizeTime | 3871 |
| 6.2.2 system.bacnet.synchronizeTimeUtc | 3872 |
| 6.2.3 system.bacnet.writeWithPriority | 3873 |
| 6.3 system.dataset | 3875 |
| 6.3.1 system.dataset.addColumn | 3876 |
| 6.3.2 system.dataset.addRow | 3878 |
| 6.3.3 system.dataset.addRows | 3880 |
| 6.3.4 system.dataset.appendDataset | 3882 |
| 6.3.5 system.dataset.clearDataset | 3883 |
| 6.3.6 system.dataset.dataSetToHTML | 3884 |
| 6.3.7 system.dataset.deleteRow | 3885 |
| 6.3.8 system.dataset.deleteRows | 3886 |
| 6.3.9 system.dataset.exportCSV | 3887 |
| 6.3.10 system.dataset.exportExcel | 3888 |
| 6.3.11 system.dataset.exportHTML | 3889 |
| 6.3.12 system.dataset.filterColumns | 3890 |
| 6.3.13 system.dataset.formatDates | 3892 |
| 6.3.14 system.dataset.fromCSV | 3894 |
| 6.3.15 system.dataset.getColumnHeaders | 3896 |
| 6.3.16 system.dataset.setValue | 3897 |
| 6.3.17 system.dataset.sort | 3898 |
| 6.3.18 system.dataset.toCSV | 3900 |
| 6.3.19 system.dataset.toDataSet | 3901 |
| 6.3.20 system.dataset.toExcel | 3903 |
| 6.3.21 system.dataset.toPyDataSet | 3904 |
| 6.3.22 system.dataset.updateRow | 3905 |
| 6.4 system.date | 3907 |
| 6.4.1 system.date.*Between | 3908 |
| 6.4.2 system.date.add* | 3910 |
| 6.4.3 system.date.format | 3912 |
| 6.4.4 system.date.fromMillis | 3914 |
| 6.4.5 system.date.get* | 3915 |
| 6.4.6 system.date.getDate | 3917 |
| 6.4.7 system.date.getTimezone | 3918 |
| 6.4.8 system.date.getTimezoneOffset | 3927 |
| 6.4.9 system.date.getTimezoneRawOffset | 3928 |
| 6.4.10 system.date.isAfter | 3929 |
| 6.4.11 system.date.isBefore | 3930 |
| 6.4.12 system.date.isBetween | 3931 |
| 6.4.13 system.date.isDaylightTime | 3932 |
| 6.4.14 system.date.midnight | 3933 |
| 6.4.15 system.date.now | 3934 |
| 6.4.16 system.date.parse | 3935 |
| 6.4.17 system.date.setTime | 3936 |
| 6.4.18 system.date.toMillis | 3937 |
| 6.5 system.db | 3938 |
| 6.5.1 system.db.addDatasource | 3939 |
| 6.5.2 system.db.beginNamedQueryTransaction | 3940 |
| 6.5.3 system.db.beginTransaction | 3942 |
| 6.5.4 system.db.clearAllNamedQueryCaches | 3944 |
| 6.5.5 system.db.clearNamedQueryCache | 3946 |
| 6.5.6 system.db.closeTransaction | 3948 |
| 6.5.7 system.db.commitTransaction | 3949 |
| 6.5.8 system.db.createSProcCall | 3950 |
| 6.5.9 system.db.dateFormat | 3953 |
| 6.5.10 system.db.execSProcCall | 3955 |
| 6.5.11 system.db.getConnectionInfo | 3956 |
| 6.5.12 system.db.getConnections | 3957 |
| 6.5.13 system.db.refresh | 3958 |
| 6.5.14 system.db.removeDatasource | 3959 |
| 6.5.15 system.db.rollbackTransaction | 3960 |
| 6.5.16 system.db.runNamedQuery | 3962 |
| 6.5.17 system.db.runPrepQuery | 3964 |
| 6.5.18 system.db.runPrepUpdate | 3966 |
| 6.5.19 system.db.runQuery | 3969 |
| 6.5.20 system.db.runScalarPrepQuery | 3972 |
| 6.5.21 system.db.runScalarQuery | 3973 |

| | |
|---|------|
| 6.5.22 system.db.runSFNamedQuery | 3974 |
| 6.5.23 system.db.runSFPrepUpdate | 3976 |
| 6.5.24 system.db.runSFUpdateQuery | 3978 |
| 6.5.25 system.db.runUpdateQuery | 3979 |
| 6.5.26 system.db.setDatasourceConnectURL | 3981 |
| 6.5.27 system.db.setDatasourceEnabled | 3982 |
| 6.5.28 system.db.setDatasourceMaxConnections | 3983 |
| 6.6 system.device | 3984 |
| 6.6.1 system.device.addDevice | 3985 |
| 6.6.1.1 system.device.addDevice - deviceProps Listing | 3988 |
| 6.6.2 system.device.listDevices | 3994 |
| 6.6.3 system.device.refreshBrowse | 3995 |
| 6.6.4 system.device.removeDevice | 3996 |
| 6.6.5 system.device.setDeviceEnabled | 3997 |
| 6.6.6 system.device.setDeviceHostname | 3998 |
| 6.7 system.dnp3 | 3999 |
| 6.7.1 system.dnp3.directOperateAnalog | 4001 |
| 6.7.2 system.dnp3.directOperateBinary | 4003 |
| 6.7.3 system.dnp3.freezeAnalogs | 4005 |
| 6.7.4 system.dnp3.freezeAnalogsAtTime | 4006 |
| 6.7.5 system.dnp3.freezeCounters | 4007 |
| 6.7.6 system.dnp3.freezeCountersAtTime | 4008 |
| 6.7.7 system.dnp3.selectOperateAnalog | 4009 |
| 6.7.8 system.dnp3.selectOperateBinary | 4011 |
| 6.8 system.eam | 4013 |
| 6.8.1 system.eam.getGroups | 4014 |
| 6.8.2 system.eam.queryAgentHistory | 4015 |
| 6.8.3 system.eam.queryAgentStatus | 4017 |
| 6.8.4 system.eam.runTask | 4019 |
| 6.9 system.file | 4021 |
| 6.9.1 system.file.fileExists | 4022 |
| 6.9.2 system.file.getTempFile | 4023 |
| 6.9.3 system.file.openFile | 4024 |
| 6.9.4 system.file.openFiles | 4025 |
| 6.9.5 system.file.readFileAsBytes | 4027 |
| 6.9.6 system.file.readFileAsString | 4028 |
| 6.9.7 system.file.saveFile | 4029 |
| 6.9.8 system.file.writeFile | 4031 |
| 6.10 system.groups | 4033 |
| 6.10.1 system.groups.loadFromFile | 4034 |
| 6.10.2 system.groups.removeGroups | 4035 |
| 6.11 system.gui | 4036 |
| 6.11.1 system.gui.chooseColor | 4037 |
| 6.11.2 system.gui.closeDesktop | 4038 |
| 6.11.3 system.gui.color | 4039 |
| 6.11.4 system.gui.confirm | 4041 |
| 6.11.5 system.gui.convertPointToScreen | 4042 |
| 6.11.6 system.gui.createPopupMenu | 4043 |
| 6.11.7 system.gui.desktop | 4045 |
| 6.11.8 system.gui.errorBox | 4046 |
| 6.11.9 system.gui.findWindow | 4047 |
| 6.11.10 system.gui.getCurrentDesktop | 4048 |
| 6.11.11 system.gui.getScreenIndex | 4049 |
| 6.11.12 system.gui.getDesktopHandles | 4050 |
| 6.11.13 system.gui.getOpenedWindowNames | 4052 |
| 6.11.14 system.gui.getOpenedWindows | 4053 |
| 6.11.15 system.gui.getParentWindow | 4054 |
| 6.11.16 system.gui.getQuality | 4055 |
| 6.11.17 system.gui.getScreens | 4056 |
| 6.11.18 system.gui.getSibling | 4057 |
| 6.11.19 system.gui.getWindow | 4058 |
| 6.11.20 system.gui.getWindowNames | 4060 |
| 6.11.21 system.gui.inputBox | 4061 |
| 6.11.22 system.gui.isTouchscreenModeEnabled | 4062 |
| 6.11.23 system.gui.messageBox | 4063 |
| 6.11.24 system.gui.openDesktop | 4064 |
| 6.11.25 system.gui.openDiagnostics | 4066 |
| 6.11.26 system.gui.passwordBox | 4067 |
| 6.11.27 system.gui.setScreenIndex | 4068 |
| 6.11.28 system.gui.setTouchscreenModeEnabled | 4069 |
| 6.11.29 system.gui.showNumericKeypad | 4070 |
| 6.11.30 system.gui.showTouchscreenKeyboard | 4072 |
| 6.11.31 system.gui.transform | 4073 |
| 6.11.32 system.gui.warningBox | 4075 |
| 6.12 system.math | 4076 |
| 6.12.1 system.math.geometricMean | 4077 |
| 6.12.2 system.math.kurtosis | 4078 |
| 6.12.3 system.math.max | 4079 |
| 6.12.4 system.math.mean | 4080 |

| | |
|--|------|
| 6.12.5 system.math.meanDifference | 4081 |
| 6.12.6 system.math.median | 4082 |
| 6.12.7 system.math.min | 4083 |
| 6.12.8 system.math.mode | 4084 |
| 6.12.9 system.math.normalize | 4085 |
| 6.12.10 system.math.percentile | 4086 |
| 6.12.11 system.math.populationVariance | 4087 |
| 6.12.12 system.math.product | 4088 |
| 6.12.13 system.math.skewness | 4089 |
| 6.12.14 system.math.standardDeviation | 4090 |
| 6.12.15 system.math.sum | 4091 |
| 6.12.16 system.math.sumDifference | 4092 |
| 6.12.17 system.math.sumLog | 4093 |
| 6.12.18 system.math.sumSquares | 4094 |
| 6.12.19 system.math.variance | 4095 |
| 6.13 system.nav | 4096 |
| 6.13.1 system.nav.centerWindow | 4097 |
| 6.13.2 system.nav.closeParentWindow | 4098 |
| 6.13.3 system.nav.closeWindow | 4099 |
| 6.13.4 system.nav.desktop | 4101 |
| 6.13.5 system.nav.getCurrentWindow | 4102 |
| 6.13.6 system.nav.goBack | 4103 |
| 6.13.7 system.nav.goForward | 4104 |
| 6.13.8 system.nav.goHome | 4105 |
| 6.13.9 system.nav.openWindow | 4106 |
| 6.13.10 system.nav.openWindowInstance | 4107 |
| 6.13.11 system.nav.swapTo | 4108 |
| 6.13.12 system.nav.swapWindow | 4110 |
| 6.14 system.net | 4112 |
| 6.14.1 system.net.getExternalIpAddress | 4113 |
| 6.14.2 system.net.getHostName | 4114 |
| 6.14.3 system.net.getIpAddress | 4115 |
| 6.14.4 system.net.getRemoteServers | 4116 |
| 6.14.5 system.net.httpClient | 4117 |
| 6.14.6 system.net.httpDelete | 4124 |
| 6.14.7 system.net.httpGet | 4125 |
| 6.14.8 system.net.httpPost | 4127 |
| 6.14.9 system.net.httpPut | 4129 |
| 6.14.10 system.net.openURL | 4131 |
| 6.14.11 system.net.sendEmail | 4133 |
| 6.15 system.opc | 4135 |
| 6.15.1 system.opc.browse | 4136 |
| 6.15.2 system.opc.browseServer | 4138 |
| 6.15.3 system.opc.browseSimple | 4140 |
| 6.15.4 system.opc.getServers | 4142 |
| 6.15.5 system.opc.getServerState | 4143 |
| 6.15.6 system.opc.isServerEnabled | 4145 |
| 6.15.7 system.opc.readValue | 4146 |
| 6.15.8 system.opc.readValues | 4147 |
| 6.15.9 system.opc.setServerEnabled | 4148 |
| 6.15.10 system.opc.writeValue | 4150 |
| 6.15.11 system.opc.writeValues | 4151 |
| 6.16 system.opchda | 4152 |
| 6.16.1 system.opchda/browse | 4153 |
| 6.16.2 system.opchda/getAggregates | 4154 |
| 6.16.3 system.opchda/getAttributes | 4155 |
| 6.16.4 system.opchda/getServers | 4156 |
| 6.16.5 system.opchda/insert | 4157 |
| 6.16.6 system.opchda/insertReplace | 4158 |
| 6.16.7 system.opchda/isServerAvailable | 4159 |
| 6.16.8 system.opchda/readAttributes | 4160 |
| 6.16.9 system.opchda/readProcessed | 4161 |
| 6.16.10 system.opchda/readRaw | 4162 |
| 6.16.11 system.opchda/replace | 4163 |
| 6.17 system.opcua | 4164 |
| 6.17.1 system.opcua/callMethod | 4165 |
| 6.18 system.perspective | 4167 |
| 6.18.1 system.perspective.alterLogging | 4168 |
| 6.18.2 system.perspective.closeDock | 4170 |
| 6.18.3 system.perspective.closePage | 4171 |
| 6.18.4 system.perspective.closePopup | 4172 |
| 6.18.5 system.perspective.closeSession | 4173 |
| 6.18.6 system.perspective.download | 4174 |
| 6.18.7 system.perspective/getProjectInfo | 4175 |
| 6.18.8 system.perspective/getSessionInfo | 4176 |
| 6.18.9 system.perspective/isAuthorized | 4178 |
| 6.18.10 system.perspective/login | 4179 |
| 6.18.11 system.perspective/logout | 4180 |
| 6.18.12 system.perspective.navigate | 4181 |

| | |
|---|------|
| 6.18.13 system.perspective.navigateBack | 4183 |
| 6.18.14 system.perspective.navigateForward | 4184 |
| 6.18.15 system.perspective.openDock | 4185 |
| 6.18.16 system.perspective.openPopup | 4186 |
| 6.18.17 system.perspective.print | 4188 |
| 6.18.18 system.perspective.refresh | 4189 |
| 6.18.19 system.perspective.sendMessage | 4190 |
| 6.18.20 system.perspective.setTheme | 4191 |
| 6.18.21 system.perspective.toggleDock | 4192 |
| 6.18.22 system.perspective.togglePopup | 4193 |
| 6.18.23 system.perspective.vibrateDevice | 4195 |
| 6.18.24 system.perspective.workstation | 4196 |
| 6.18.24.1 system.perspective.workstation.exit | 4197 |
| 6.18.24.2 system.perspective.workstation.toKiosk | 4198 |
| 6.18.24.3 system.perspective.workstation.toWindowed | 4199 |
| 6.19 system.print | 4200 |
| 6.19.1 system.print.createImage | 4201 |
| 6.19.2 system.print.createPrintJob | 4202 |
| 6.19.3 system.print.printToImage | 4204 |
| 6.20 system.project | 4206 |
| 6.20.1 system.project.getProjectName | 4207 |
| 6.20.2 system.project.getProjectNames | 4208 |
| 6.21 system.report | 4209 |
| 6.21.1 system.report.executeAndDistribute | 4210 |
| 6.21.2 system.report.executeReport | 4213 |
| 6.21.3 system.report.getReportNamesAsDataset | 4215 |
| 6.21.4 system.report.getReportNamesAsList | 4216 |
| 6.22 system.roster | 4217 |
| 6.22.1 system.roster.addUsers | 4218 |
| 6.22.2 system.roster.createRoster | 4219 |
| 6.22.3 system.roster.deleteRoster | 4220 |
| 6.22.4 system.roster.getRosters | 4221 |
| 6.22.5 system.roster.removeUsers | 4223 |
| 6.23 system.secsgem | 4224 |
| 6.23.1 system.secsgem.copyEquipment | 4225 |
| 6.23.2 system.secsgem.deleteToolProgram | 4227 |
| 6.23.3 system.secsgem.enableDisableEquipment | 4228 |
| 6.23.4 system.secsgem.getResponse | 4229 |
| 6.23.5 system.secsgem.getToolProgram | 4231 |
| 6.23.6 system.secsgem.getToolProgramDataset | 4232 |
| 6.23.7 system.secsgem.sendRequest | 4233 |
| 6.23.8 system.secsgem.startSimEventRun | 4234 |
| 6.23.9 system.secsgem.toDataSet | 4235 |
| 6.23.10 system.secsgem.toTreeDataSet | 4237 |
| 6.23.11 system.secsgem.sendResponse | 4238 |
| 6.24 system.security | 4240 |
| 6.24.1 system.security.getRoles | 4241 |
| 6.24.2 system.security.getUsername | 4242 |
| 6.24.3 system.security.getUserRoles | 4243 |
| 6.24.4 system.security.isScreenLocked | 4245 |
| 6.24.5 system.security.lockScreen | 4246 |
| 6.24.6 system.security.logout | 4247 |
| 6.24.7 system.security.switchUser | 4248 |
| 6.24.8 system.security.unlockScreen | 4250 |
| 6.24.9 system.security.validateUser | 4251 |
| 6.25 system.serial | 4253 |
| 6.25.1 system.serial.closeSerialPort | 4254 |
| 6.25.2 system.serial.configureSerialPort | 4255 |
| 6.25.3 system.serial.openSerialPort | 4257 |
| 6.25.4 system.serial.port | 4258 |
| 6.25.5 system.serial.readBytes | 4260 |
| 6.25.6 system.serial.readBytesAsString | 4261 |
| 6.25.7 system.serial.readLine | 4262 |
| 6.25.8 system.serial.readUntil | 4263 |
| 6.25.9 system.serial.sendBreak | 4264 |
| 6.25.10 system.serial.write | 4265 |
| 6.25.11 system.serial.writeBytes | 4266 |
| 6.26 system.sfc | 4267 |
| 6.26.1 system.sfc.cancelChart | 4268 |
| 6.26.2 system.sfc.getRunningCharts | 4269 |
| 6.26.3 system.sfc.getVariables | 4271 |
| 6.26.4 system.sfc.pauseChart | 4274 |
| 6.26.5 system.sfc.redundantCheckpoint | 4275 |
| 6.26.6 system.sfc.resumeChart | 4276 |
| 6.26.7 system.sfc.setVariable | 4277 |
| 6.26.8 system.sfc.setVariables | 4279 |
| 6.26.9 system.sfc.startChart | 4281 |
| 6.27 system.tag | 4282 |
| 6.27.1 system.tag.browse | 4283 |

| | |
|---|------|
| 6.27.2 system.tag/browseHistoricalTags | 4287 |
| 6.27.3 system.tag/configure | 4289 |
| 6.27.4 system.tag/copy | 4294 |
| 6.27.5 system.tag/deleteAnnotations | 4296 |
| 6.27.6 system.tag/deleteTags | 4297 |
| 6.27.7 system.tag/exists | 4299 |
| 6.27.8 system.tag/exportTags | 4300 |
| 6.27.9 system.tag/getConfiguration | 4301 |
| 6.27.10 system.tag/importTags | 4303 |
| 6.27.11 system.tag/isOverlaysEnabled | 4304 |
| 6.27.12 system.tag/move | 4305 |
| 6.27.13 system.tag/queryAnnotations | 4307 |
| 6.27.14 system.tag/queryTagCalculations | 4309 |
| 6.27.15 system.tag/queryTagDensity | 4311 |
| 6.27.16 system.tag/queryTagHistory | 4313 |
| 6.27.17 system.tag/readAsync | 4316 |
| 6.27.18 system.tag/readBlocking | 4318 |
| 6.27.19 system.tag/requestGroupExecution | 4319 |
| 6.27.20 system.tag/setOverlaysEnabled | 4320 |
| 6.27.21 system.tag/storeAnnotations | 4321 |
| 6.27.22 system.tag/writeAsync | 4323 |
| 6.27.23 system.tag/writeBlocking | 4324 |
| 6.28 system.twilio | 4325 |
| 6.28.1 system.twilio/getAccounts | 4326 |
| 6.28.2 system.twilio/getAccountsDataset | 4327 |
| 6.28.3 system.twilio/getPhoneNumbers | 4328 |
| 6.28.4 system.twilio/getPhoneNumbersDataset | 4329 |
| 6.28.5 system.twilio/sendSms | 4330 |
| 6.29 system.user | 4332 |
| 6.29.1 system.user/addCompositeSchedule | 4333 |
| 6.29.2 system.user/addHoliday | 4334 |
| 6.29.3 system.user/addRole | 4336 |
| 6.29.4 system.user/addSchedule | 4337 |
| 6.29.5 system.user/addUser | 4339 |
| 6.29.6 system.user/createScheduleAdjustment | 4340 |
| 6.29.7 system.user/editHoliday | 4342 |
| 6.29.8 system.user/editRole | 4344 |
| 6.29.9 system.user/editSchedule | 4345 |
| 6.29.10 system.user/editUser | 4347 |
| 6.29.11 system.user/getHoliday | 4348 |
| 6.29.12 system.user/getHolidayNames | 4349 |
| 6.29.13 system.user/getHolidays | 4350 |
| 6.29.14 system.user/getNewUser | 4351 |
| 6.29.15 system.user/getRoles | 4353 |
| 6.29.16 system.user/getSchedule | 4354 |
| 6.29.17 system.user/getScheduledUsers | 4356 |
| 6.29.18 system.user/getScheduleNames | 4357 |
| 6.29.19 system.user/getSchedules | 4358 |
| 6.29.20 system.user/getUser | 4360 |
| 6.29.21 system.user/getUsers | 4362 |
| 6.29.22 system.user/isUserScheduled | 4364 |
| 6.29.23 system.user/removeHoliday | 4365 |
| 6.29.24 system.user/removeRole | 4367 |
| 6.29.25 system.user/removeSchedule | 4368 |
| 6.29.26 system.user/removeUser | 4370 |
| 6.30 system.util | 4371 |
| 6.30.1 system.util/audit | 4372 |
| 6.30.2 system.util/beep | 4374 |
| 6.30.3 system.util/execute | 4375 |
| 6.30.4 system.util/exit | 4376 |
| 6.30.5 system.util/getAvailableLocales | 4377 |
| 6.30.6 system.util/getAvailableTerms | 4378 |
| 6.30.7 system.util/getClientId | 4379 |
| 6.30.8 system.util/getConnectionMode | 4380 |
| 6.30.9 system.util/getConnectTimeout | 4381 |
| 6.30.10 system.util/getEdition | 4382 |
| 6.30.11 system.util/getGatewayAddress | 4383 |
| 6.30.12 system.util/getGatewayStatus | 4384 |
| 6.30.13 system.util/getGlobals | 4385 |
| 6.30.14 system.util/getInactivitySeconds | 4386 |
| 6.30.15 system.util/getLocale | 4387 |
| 6.30.16 system.util/getLogger | 4388 |
| 6.30.17 system.util/getProjectName | 4390 |
| 6.30.18 system.util/getProperty | 4391 |
| 6.30.19 system.util/getReadTimeout | 4392 |
| 6.30.20 system.util/getSessionInfo | 4393 |
| 6.30.21 system.util/getSystemFlags | 4395 |
| 6.30.22 system.util/getVersion | 4397 |
| 6.30.23 system.util/InvokeAsynchronous | 4399 |

| | |
|---|------|
| 6.30.24 system.util.invokeLater | 4401 |
| 6.30.25 system.util.jsonDecode | 4403 |
| 6.30.26 system.util.jsonEncode | 4405 |
| 6.30.27 system.util.modifyTranslation | 4407 |
| 6.30.28 system.util.playSoundClip | 4408 |
| 6.30.29 system.util.queryAuditLog | 4410 |
| 6.30.30 system.util.retarget | 4411 |
| 6.30.31 system.util.sendMessage | 4413 |
| 6.30.32 system.util.sendRequest | 4415 |
| 6.30.33 system.util.sendRequestAsync | 4416 |
| 6.30.34 system.util.setConnectionMode | 4418 |
| 6.30.35 system.util.setConnectTimeout | 4419 |
| 6.30.36 system.util.setLocale | 4420 |
| 6.30.37 system.util.setLevel | 4421 |
| 6.30.38 system.util.setReadTimeout | 4422 |
| 6.30.39 system.util.threadDump | 4423 |
| 6.30.40 system.util.translate | 4424 |
| 7. Reference Pages | 4426 |
| 7.1 Color Selector Reference | 4427 |
| 7.2 Gateway Configuration File Reference | 4431 |
| 7.3 Gateway Port Reference | 4437 |
| 7.4 Ignition Auto-Generated SQL Tables | 4439 |
| 7.5 Perspective Event Types Reference | 4443 |
| 7.6 Style Reference | 4448 |
| 7.7 Symbol Reference | 4461 |
| 7.7.1 Tag Browser Symbol Reference | 4462 |
| 7.7.2 Project Browser Symbol Reference | 4464 |
| 7.7.3 Designer Symbol Reference | 4466 |
| 7.8 Ignition Gateway File Reference | 4485 |
| 7.9 Data Type Formatting Reference | 4486 |
| 7.10 Scripting Object Reference | 4489 |
| 7.11 Alarm Event Properties Reference | 4493 |
| 8. Tutorials & Helpful Tricks | 4497 |
| 8.1 Mapping a Network Drive | 4498 |
| 8.2 Troubleshooting Guides | 4500 |
| 8.2.1 Troubleshooting Guide - Alarm Notification | 4503 |
| 8.2.2 Troubleshooting Guide - Databases | 4505 |
| 8.2.3 Troubleshooting Guide - Gateway Network | 4507 |
| 8.2.4 Troubleshooting Guide - Gateway Scripts | 4508 |
| 8.2.5 Troubleshooting Guide - Modules | 4509 |
| 8.2.6 Troubleshooting Guides - Devices | 4510 |
| 8.2.7 Troubleshooting Guide - Store and Forward | 4511 |
| 8.2.8 Troubleshooting Guide - Tag History | 4512 |
| 8.3 Development Server Best Practices | 4513 |
| 8.4 Adding Security Certificates into Keystores | 4515 |
| 8.5 Mass Edits on Large Amount of Tags | 4516 |
| 8.6 Storing image in database as blob | 4517 |
| 8.7 Ignition Advanced Configurations | 4518 |
| 8.8 Setting Vision Application to Launch When Server Boots Up | 4519 |

Welcome



Welcome to Ignition by Inductive Automation

Ignition is not like any other HMI / SCADA system you've seen before! Ignition is a single install, runs from a single location, is server based, and is sold by the server not by the client. It uses current web-based technology and allows you to bring the IT department and the plant floor closer than ever. Ignition uses Java for fast secure runtime clients, and that means it can be used on any system, any version. Windows 10, Mac OSX, Linux or Cloud servers. You can switch between systems or use one type for the server and a combination of types for your runtime clients. If an IT department wants to use Windows servers to host and the Operations groups want to use their new MacBooks, Ignition can handle it.

What can Ignition do for you? It has built-in, drag and drop functionality for just about anything you can imagine. Ignition does HMI/SCADA controls, dashboards, historical trending, database access, reporting, alarming, security, Sequential Function Charts, redundancy, failover control, Enterprise Administration, and more. Ignition does all this well because of its *modular architecture*. You choose the features and functionality you need. Want to try some of the other features? You always have a full version of Ignition with a resettable timer to try new things. Play around as much as you like and decide for yourself what you want.

About this User Manual

This [Welcome](#) section provides a [broad overview](#) and information relating to [modules](#), [architectures](#), [installation](#), and an [Upgrade Guide](#). The sidebar on the left is a table of contents organized so you can navigate easily and quickly through Ignition's features, modules, functions, and so forth.

The [Glossary](#) defines common terms and acronyms found in Ignition and in the industry. Finally, in the [Appendix](#), you'll find an exhaustive list of all the [Components](#), [Expressions](#), and [Scripting Functions](#) in Ignition.

Take a look at the navigation section on the left and poke around, or use the search bar at the top of the navigation.

Ignition 8.1

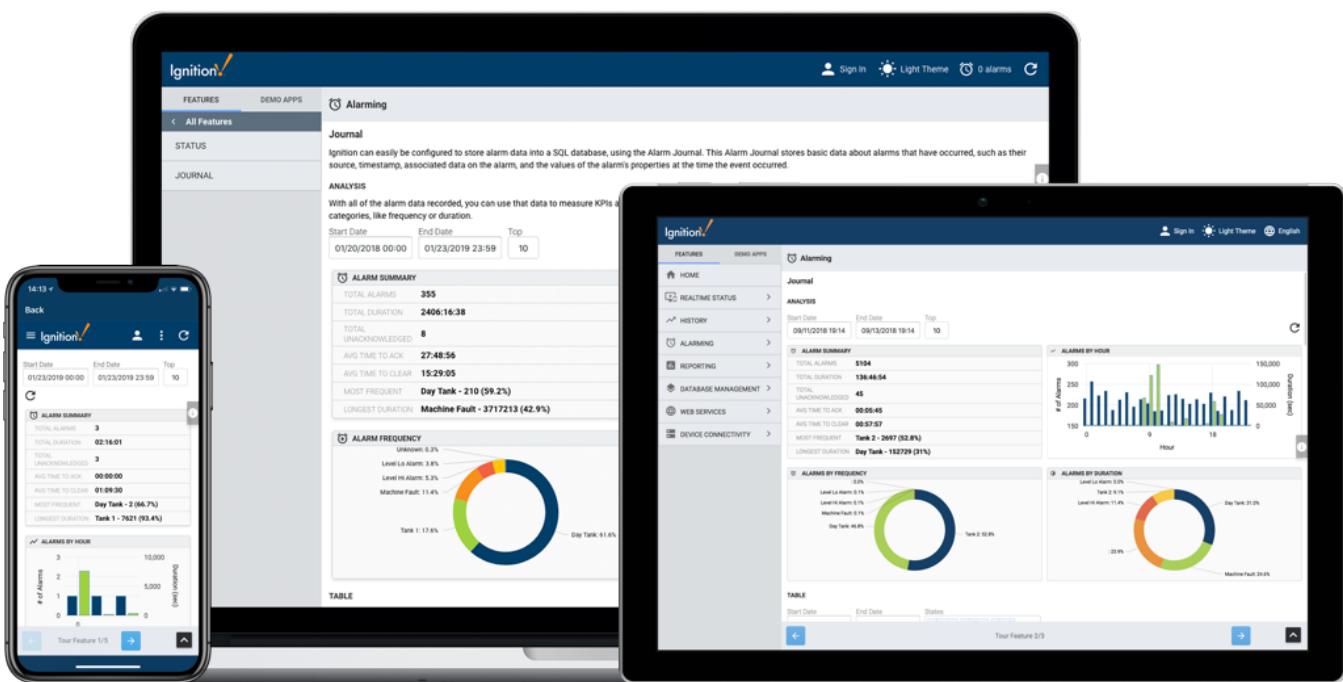
This User Manual is specifically for Ignition 8.1. Newer major versions of the software will have separate user manuals that focus on newer features.

Inductive University

[Inductive University](#) is a free online learning platform designed to help you master Ignition. You can watch over 500 training videos and participate in our credential program to test your knowledge. It's the first step to joining our ranks of Certified Integrators.

This User Manual section is closely tied with Inductive University so you can have a conceptual understanding from this manual and a visual walk-through of the feature in the videos. Throughout this online manual, you will find links to the various Inductive University videos on the right side of the page. These videos correspond to the information in that page or section. This way, you can get a conceptual understanding from the information provided here, and a visual walk-through of the feature by watching the videos. The video links will look like the one to the right of this paragraph.

The Inductive University logo is located at the top left of the section. It features a blue hexagon with the letters "IU" in white, surrounded by a green laurel wreath. To the right of the logo, the words "INDUCTIVE UNIVERSITY" are written in a bold, serif font. Below this, a section titled "What's New in 8.1" is shown in bold black text, with a link "Watch the Video" in blue text below it.



Ignition Basics

Ignition is completely different from what you are used to using. See what Ignition is and some of the many things it can do for you. Learn the basics and how it can work for you.

[Introducing Ignition](#)

System Architectures

Ignition can support many different types of architectures from single standalone systems to vast enterprise level structures. Learn about the types of architectures and how they work.

[System Architectures](#)

Partner Programs

Inductive Automation is proud to have a Partner Program that allows others to create modules to enhance Ignition's functionality. You can learn more about [Sepasoft's MES modules](#) and [Cirrus Link's MQTT modules](#) here.

Startup Guide

New to Ignition? Get started by getting Ignition running on your computer in minutes with our Startup Guide. Download Ignition and evaluate the full version right away.

[Startup Guide](#)

Modules

Are you ready to *really* get to know Ignition? Modules are the building blocks of Ignition. They are the reason it can do so much and be so flexible. Adding the functionality you need is simple with Ignition Modules.

[Modules Overview](#)

SDK Documentation

Documentation for the [Ignition Module SDK](#). Anyone looking to create a new module should start here.

Installing and Upgrading

You can install Ignition in under 3 minutes; staying current is just as easy. No hassles, no secret handshakes, only one lightweight install and you can have clients running everywhere.

[Installing and Upgrading Ignition](#)

New in this Version

Are you looking for something that has changed? Or maybe you want to see the things we have been adding to Ignition? Check out the New in this Version page for a list of improvements for each version.

[New in this Version](#)

Ignition™, Ignition Edge™, and Inductive Automation® are trademarks of Inductive Automation.

All other product or company names that may be mentioned in this publication are tradenames, trademarks, or registered trademarks of their respective owners.

Ignition User Manual © 2015-2021 Inductive Automation

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

All Inductive Automation products are trademarks or registered trademarks of Inductive Automation. Other brand and product names are trademarks

or registered trademarks of their respective holders.

Every effort has been made to make this user manual as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. Inductive Automation shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in his manual.

Introducing Ignition

What Is Ignition?

Ignition SCADA by Inductive Automation® combines an unlimited licensing model, with instant web-based deployment, and the industry-leading toolset for supervisory control and data acquisition (SCADA) — all on one open and scalable universal platform. Ignition is The New SCADA because it solves all the major pain points of old SCADA. Ignition empowers your business to easily control your processes, and track, display, and analyze all your data, without limits. Ignition is designed from the ground up to be approachable and easy to get started with while it is flexible and capable of scaling up to the largest projects. Ignition is unlimited, so for the low cost of one server license you can seamlessly connect all your data, design any kind of industrial application with ease, and instantly web-deploy unlimited clients to anyone, anywhere — all from one universal platform for industrial automation.

Ignition has a core focused on everything you need and nothing that you don't.

| | |
|---|--|
|  | Server Based One computer controls everything in Ignition. That means you only install in one place and connect everything else to that server. No separate install for the Designer. You can have as many Clients (runtimes) and Designers as you want, no installation necessary. |
|  | Simple Licensing One license for the server gives you unlimited access. Unlimited Clients, unlimited Tags, unlimited projects, and more. With unlimited run-time clients at no additional cost, you can get your important data and analytics to your entire team, across your whole company. Ignition allows you to create and use as many tags as you need for devices, OPC servers, and anything else, without limits. |
|  | Web-based Deployment Ignition is installed, deployed, and managed using standard web technologies. Connect and change settings, update projects, create new Tags, and more, all from any computer on the network. |
|  | Web-launched Designer and Clients Ignition's Designer tool and runtime clients and sessions are launched using the web. Run a large amount of clients across your enterprise at the same time. In initial benchmark testing, Ignition Perspective can simultaneously launch 200 clients with ease. |
|  | Built-in Security Security is built in to every aspect of Ignition. You can manage users, edit screen permissions down to the individual Tag or component level, and even secure certain assets in your projects. |
|  | Scalable Modular Architecture Create a scalable and centrally managed SCADA system using Ignition's web-based server architecture. Do more with unlimited connections to PLCs, databases, web-based deployment, and web-launched clients. Learn more about Ignition's System Architectures. Projects can be made small and expanded out indefinitely as your business grows, or you can have smaller installations connected to and interacting with other larger systems. |
|  | Cross Platform Ignition works with any major operating system, even iOS and Android. Ignition is built on Java, which means you can install and run it on any modern operating system. Ignition will run on Windows, Mac OS X, and Linux. Backups of your system are even cross platform, you can load a backup from any system. |

On this page ...

- [What Is Ignition?](#)
- [New in This Release](#)
- [What Can it Do?](#)
 - [Web-Based HMI and SCADA Applications Deployment](#)
 - [High Performance Historian](#)
 - [Reporting Engine](#)
 - [Alarming](#)
- [Communications Hub](#)
 - [Mix and Match](#)
- [Understanding Ignition's Flow](#)



What is Ignition?

[Watch the Video](#)

New in This Release

- Ignition 8 features a new poll-free protocol that updates tag data instantly for both Vision and Perspective clients. As a result, Vision clients in Ignition 8 are drastically faster while using dramatically less CPU power.
- For the new Vision Module, we've created new native Vision client launchers for Windows, Mac, and Linux, so you can easily launch Vision clients on any major operating system without needing to install and update Java.
- Ignition Vision is perfect for dedicated plant-floor displays and HMIs, while Ignition Perspective is perfect for mobile-first industrial applications. You can use them on their own or together to form the industry's most powerful, cutting-edge visualization system.
- Vision client screens will look even better in Ignition 8, thanks to a new design and new support for high-resolution displays.
- Fonts have also been improved with more choices and better rendering so your screens look sharper and more stylish than ever.
- Ignition 8 has an improved concurrent editing system which allows multiple people to work on the same project at the same time without locking each other out of resources. This will help your team work faster and more collaboratively.
- In Ignition Perspective, you can use powerful and flexible CSS3 styles to change the appearance and position of anything in your application. By combining styles into themes, you'll be able to apply and edit styles across multiple applications in an instant.

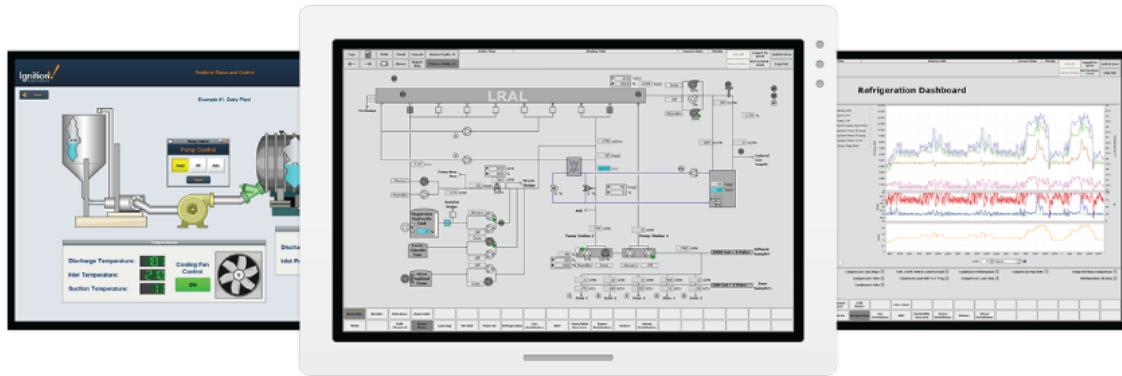
What Can it Do?

Because of its modular nature, Ignition can do a lot or a little and be completely customized to your needs. A lot of the basic functionality is shared between modules and stored in the Gateway to make everything as smooth and seamless as possible. Some of these basics are database connections, device connections, Tags, Security profiles, Alarming, and more. Because these are all shared resources, you don't have to worry about managing multiple connections for the different modules in Ignition: it just works. You can build [database](#) applications using Ignition. You can create POS, CRM, and inventory tracking systems. You can bring in data from any SQL database in your plant and interact with other existing ERP and Access systems

Modules provide the core functionality for Ignition where the Gateway provides the backbone of shared resources. There are no limitations to Ignition, here are **just a few** of the things Ignition can do for you:

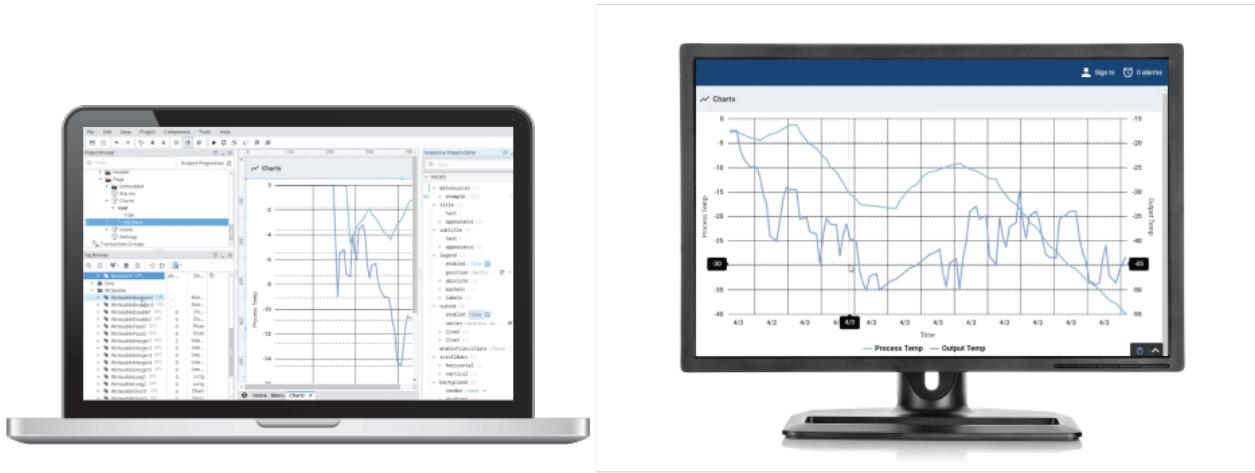
Web-Based HMI and SCADA Applications Deployment

Using web technologies, you can access PLCs and connect to SQL databases on any platform. Ignition provides the common HMI/SCADA capabilities, but in an easier and more efficient way. Create runtime clients and sessions that show current or past data, interact with your other systems and devices, create displays and controls for anything wired into your system. See and manage your entire system from one place.



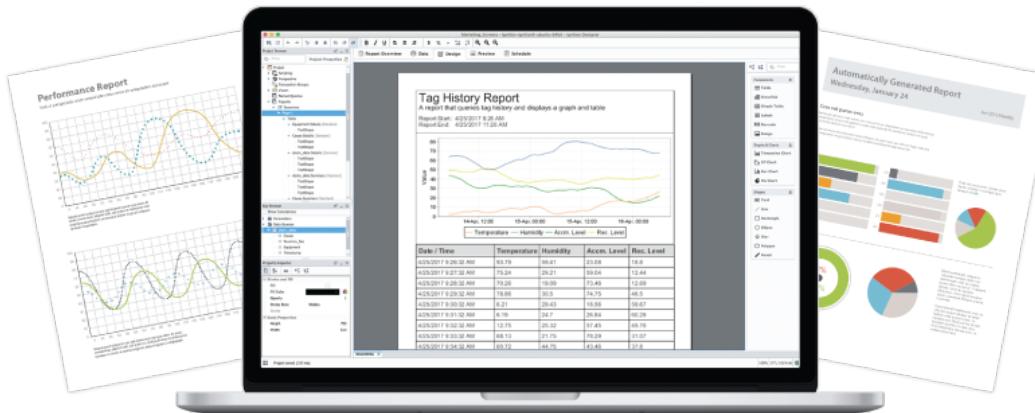
High Performance Historian

You can use any SQL database with Ignition's [high-performance historian](#). It's simple to set up and Ignition will monitor and control the data for you. The Historian includes partitioning and uses compression algorithms for fast data storage and retrieval.



Reporting Engine

You can [create dynamic PDF reports](#) that users can see and change with on the fly. Give users the ability to change date ranges or select past reports without any changes to the system. Create schedules to email or save compliance reports when you need them.



Alarming

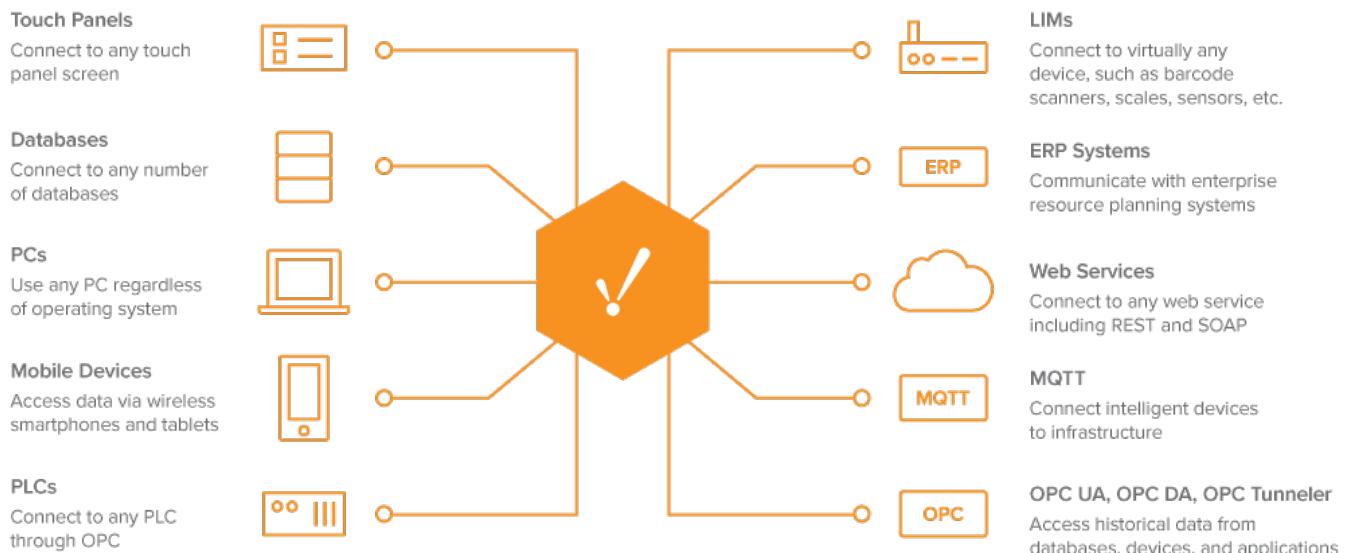
You can use Ignition's state-of-the-art alarming engine to [create alarms](#) for any condition you can imagine. Alarms can be recorded, acknowledged, or dealt with by sending out notifications. Email, SMS and voice messages can be sent to individuals or whole lists of users based on any logic you need.



Communications Hub

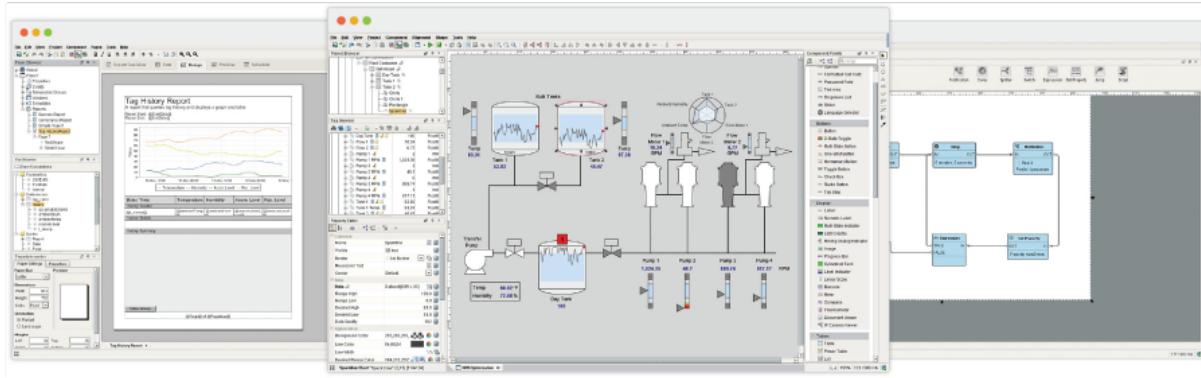
You can use Ignition as an effective communication hub in your network. You can use some or all of these features to make Ignition move data back and forth throughout your facility. Connect everything together with:

- OPC-based and communicates with virtually [any OPC server](#).
- [SQL-based](#) (JDBC) and connects to any number of Databases.
- Built-in [PLC drivers](#) and connects to any PLC (Allen-Bradley, Siemens, Modbus, etc.) through OPC.
- Supports any web services and launches anywhere.
- Connects with [other enterprise systems](#) (ERP).
- [Connects to devices](#) such as barcode scanners, scales, and sensors.
- Connects to any mobile device such as wireless smartphones and tablets.
- Connects to any [touch panel screen](#).



Mix and Match

You can mix and match all of these features and more in any way you want!



Understanding Ignition's Flow

Ignition is simple to set up and use, unlike anything else you've encountered in the HMI/SCADA world before. Getting installed and running is easy, and once you have your project set up it's extremely simple to start using. After installation, setting up and using a project follows this simple set of steps:

- **Launch the Gateway Webpage**

The Gateway Webpage is where all of your information is stored, and where all of the general configuration happens. Here you can set up your Device connections, your Database Connections, alter your security, make backups, set up redundancy, and much more.

- **Launch the Designer**

Once your connections are in place, you can start using them to create a project. Opening a Designer is as simple as clicking a link in the Gateway. No installation is necessary, and you can even launch the Designer from any other computer on the network. Add Tags, query databases, show status, history trends, alarms, and more in your project.

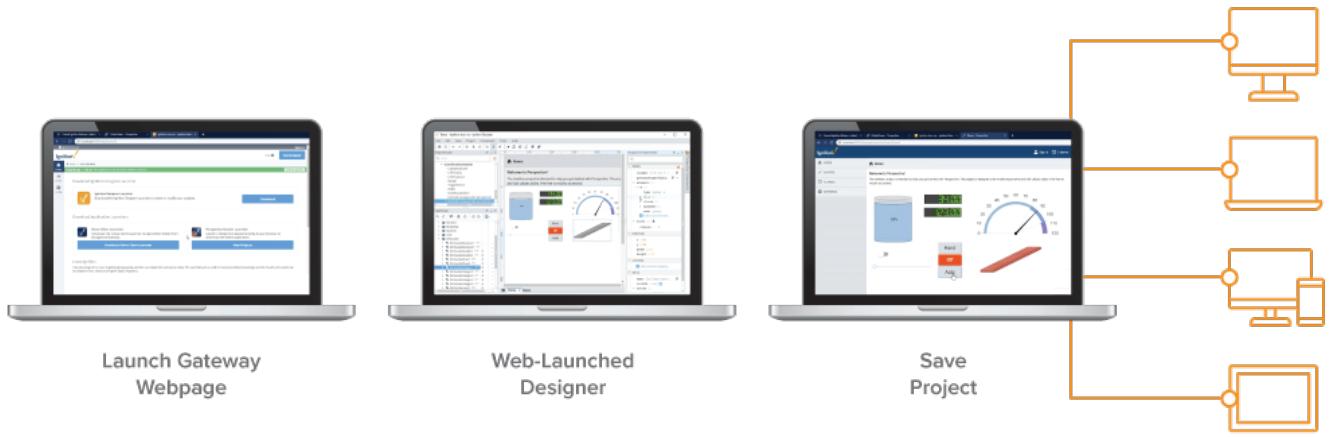
- **Save your project**

When you save, Ignition is sending all your data and all your changes back to the Gateway. Everything is centrally stored in the Gateway so you never have to worry about managing several versions or trying to combine many changes into one. You can even launch many Designers concurrently to allow multiple people to work on the same project at the same time. Projects aren't limited in scope, and you can create anything you can think of. HMI, SCADA, Reporting, Historian, Alarming, and much more.

- **Launch Clients**

Launch one Client or 20! Clients launch the same way as the Designer: simply click a link in the Gateway, no install is necessary. All of your clients talk directly to the Gateway to get new information on the fly, so there's no need to worry about them ever getting out of sync or loading changes individually to Clients.

Ignition's modular platform can support requirements of any type by selecting the modules for your needs and building processes suited for your business. Just create your project, it's all managed in one central platform and can be launched anywhere you want.



Related Topics ...

- System Architectures
- Installing and Upgrading
- Core Functions and Features
- Startup Guide

Startup Guide

Note: The Startup Guide here is was built using **Ignition 8.0**. We're currently in the process of upgraded the guide for **Ignition 8.1**. We appreciate your patience while we update the content in this section.

Are you new to Ignition? Here are a few simple steps to get going right away. In very little time, you can get a sample project up and running.

When getting started with Ignition, it helps to have a general idea about the major steps involved in designing your SCADA/HMI projects. Although there are many creative and innovative ways of using Ignition to do almost anything you want, we outlined some simple steps that can guide you in setting up a project.

1. Download and Install Ignition

Installation is simple and can be done in under 3 minutes! Let's get started and install now.

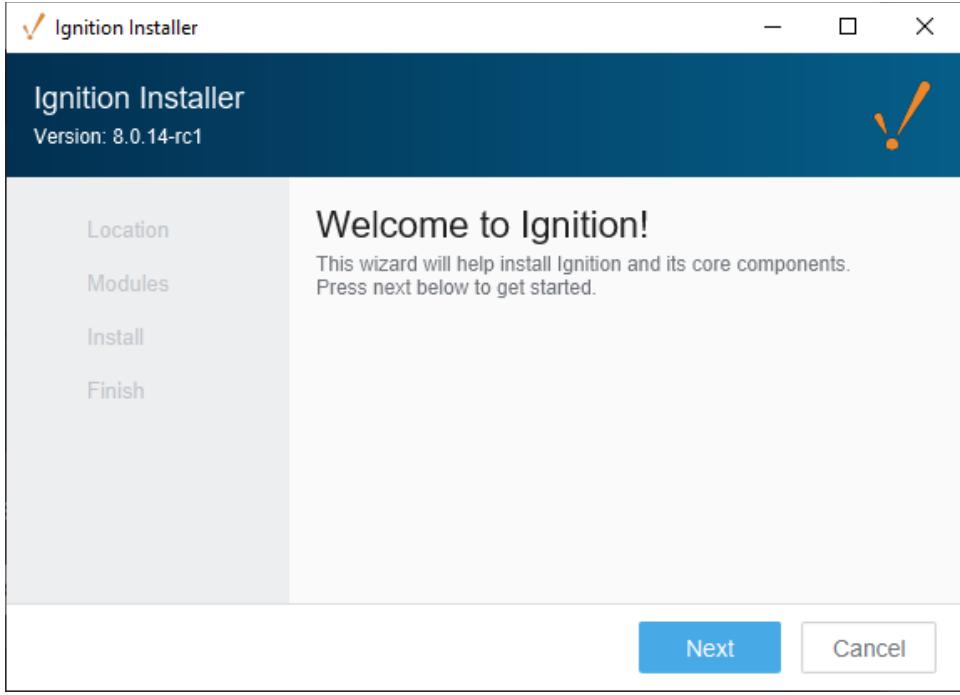
Before you install Ignition, make sure you meet the minimum system requirements:

- Dual-Core Processor (64 bit)
- 4GB RAM
- 10GB free HD space

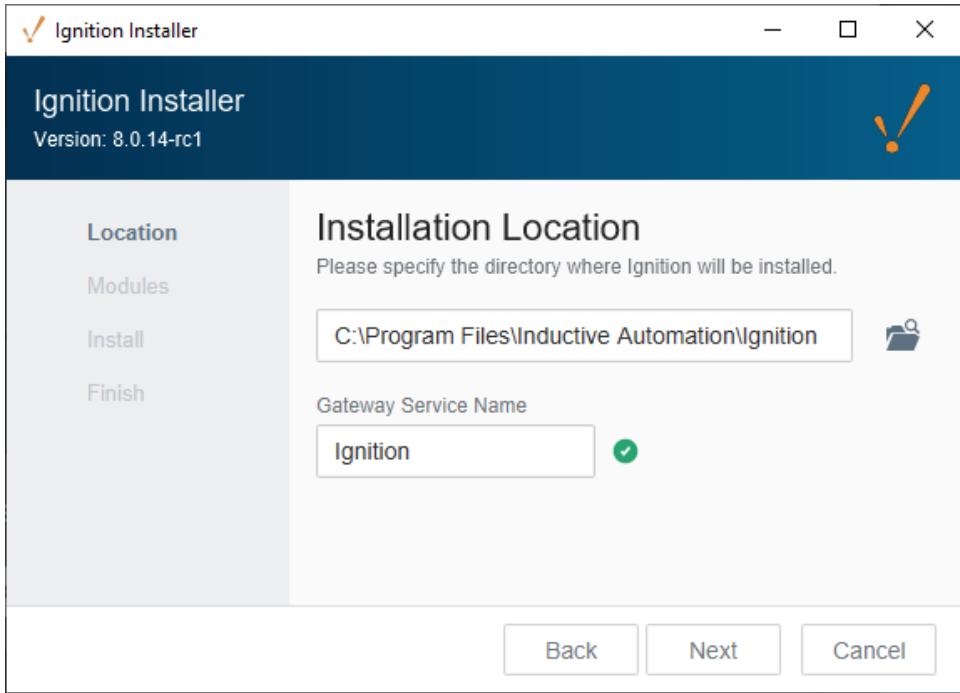
Also, ensure you have Administrator privileges to run the Ignition installer.

To Download and Install Ignition on Windows

1. Go to the Inductive Automation website at www.inductiveautomation.com
2. Select **Download Ignition** in the upper right corner of the page. The browser will determine what operating system you're running and will bring up **Download for Windows**.
3. Click **Download for Windows**. If you want to see what other operating systems are supported, click on the [Other operating systems and versions](#) link.
4. Run the downloaded file **Ignition-X.X.X-windows-x64-installer.exe** (found on the lower-left of the window if using Chrome), or go to your downloads folder and double-click the file to start installation.
5. The **Ignition Installer** window welcomes you to the Installer Wizard. To begin the installation, click **Next**.

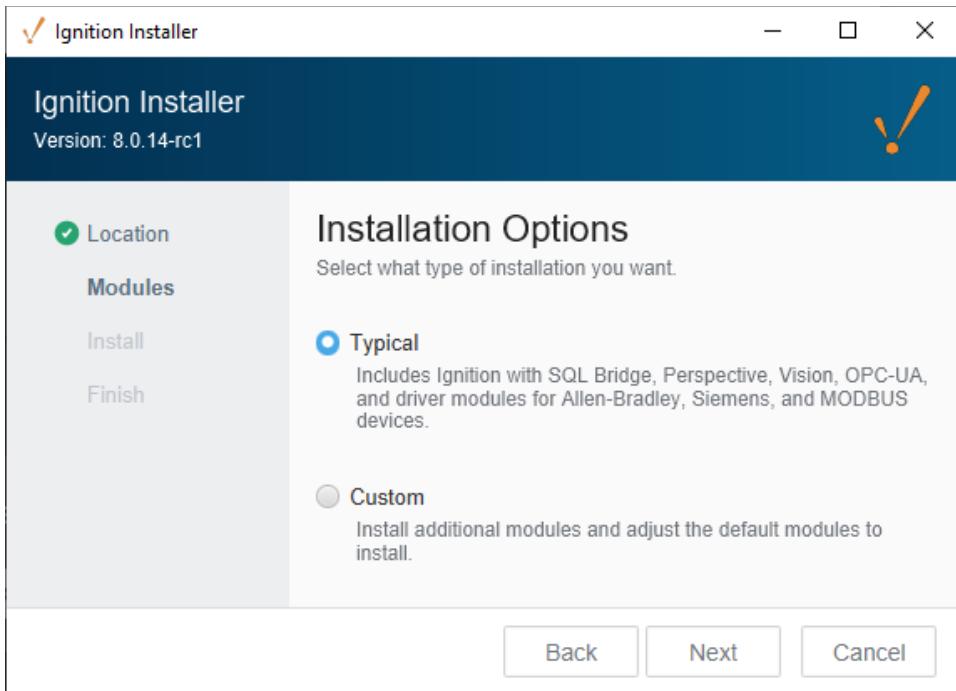


6. The **Installation Location** window will be displayed. By default, Ignition is installed under your Program Files folder: **Program Files\Inductive Automation\Ignition**. This is usually a good choice. You also have the option to choose another location. Simply click on the folder on the right side of this field to browser for another folder location. You can accept the default Gateway Service Name or enter another name. Click **Next**.

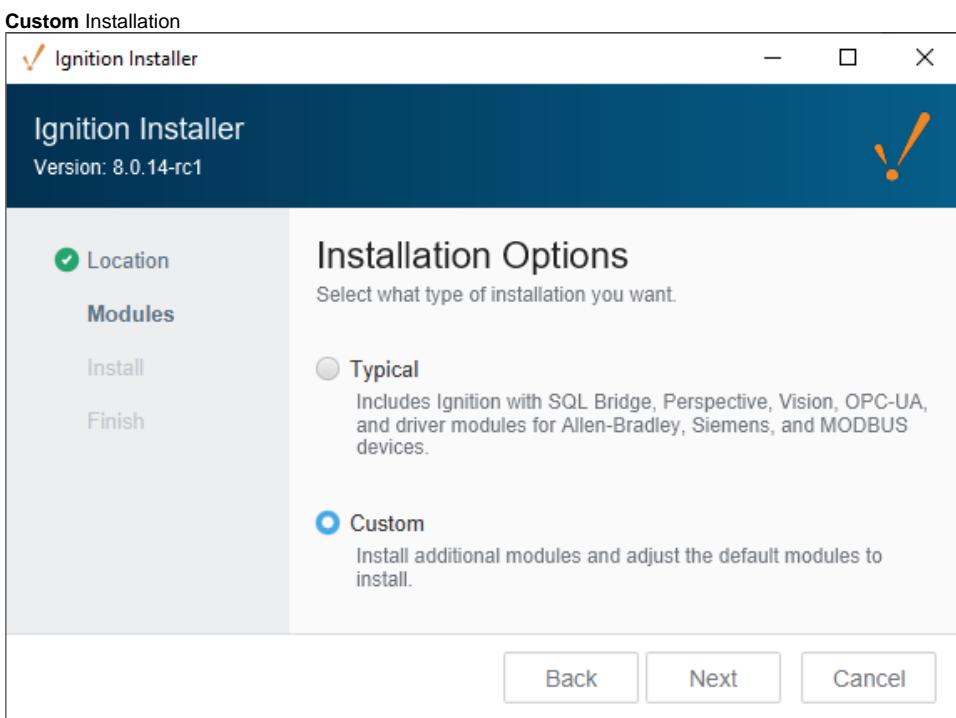


7. The **Installation Options** window will be displayed. Select either the **Typical** or **Custom** mode. The **Typical** installation includes Ignition with most of the most commonly used modules. The **Custom** installation lets you control and individually select the modules that you want installed. (Both options are shown in this step). If you selected a **Typical** installation, click **Next**.

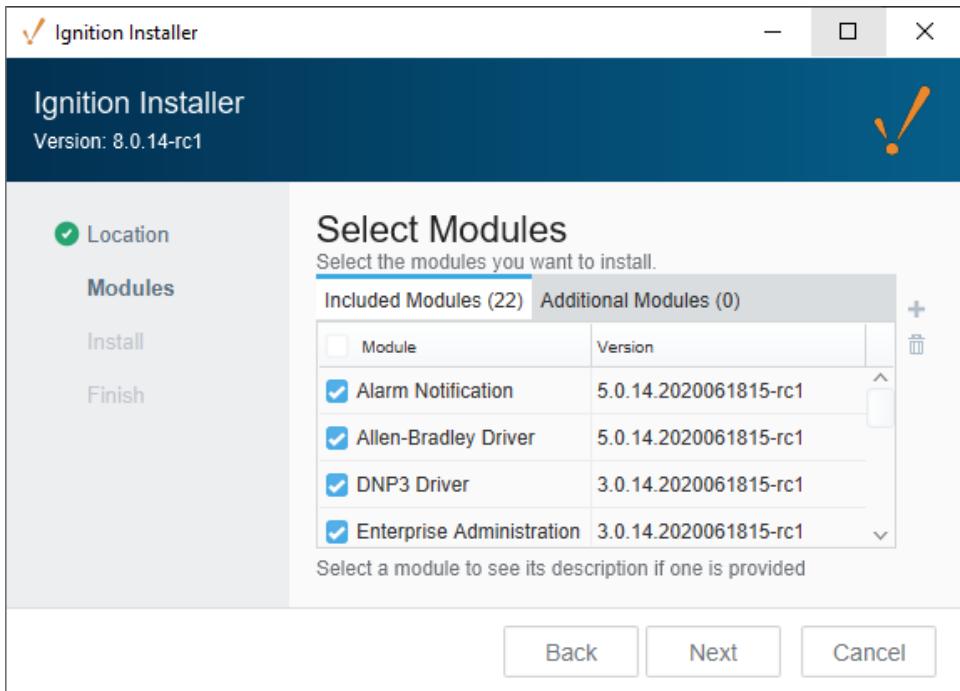
Typical Installation



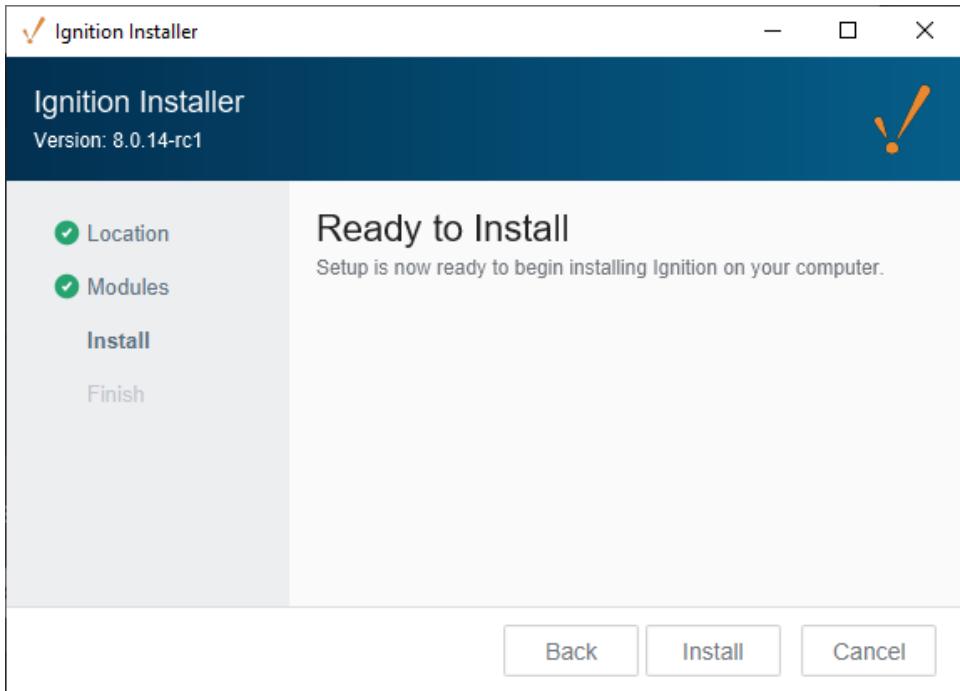
If you select **Custom** mode, check the box for the modules you want to install and uncheck those modules you don't want installed as part of your Custom Ignition installation. To see a brief description of the module, click on the module name.



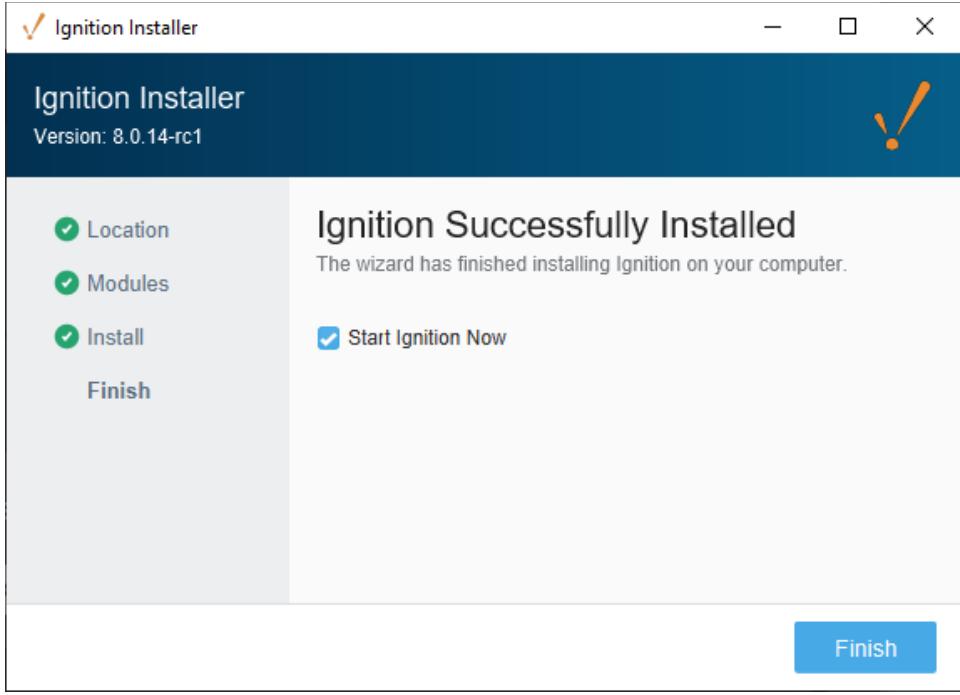
Select the modules to install, and then click **Next**.



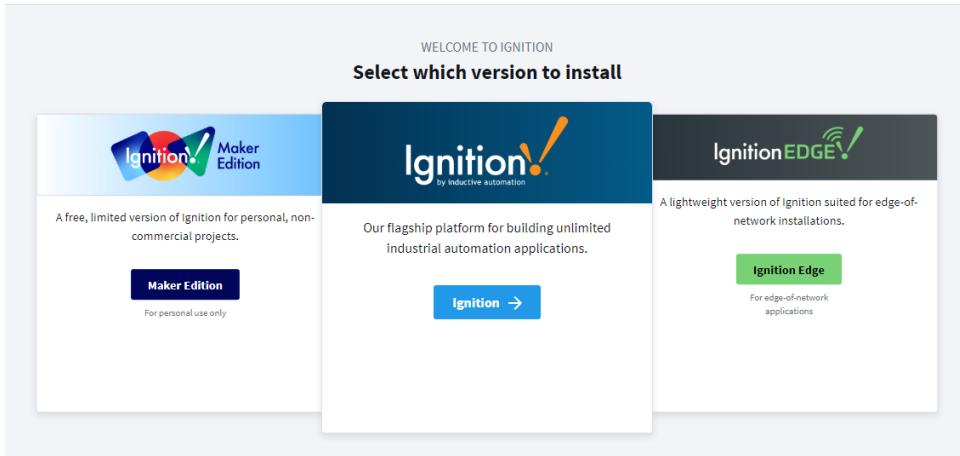
8. The wizard shows Ignition is now **Ready to Install**. If you decide to change your selections, click **Back** to go back and make your changes. Click **Install** to continue. Afterwards, a splash screen will appear indicating the installation progress.



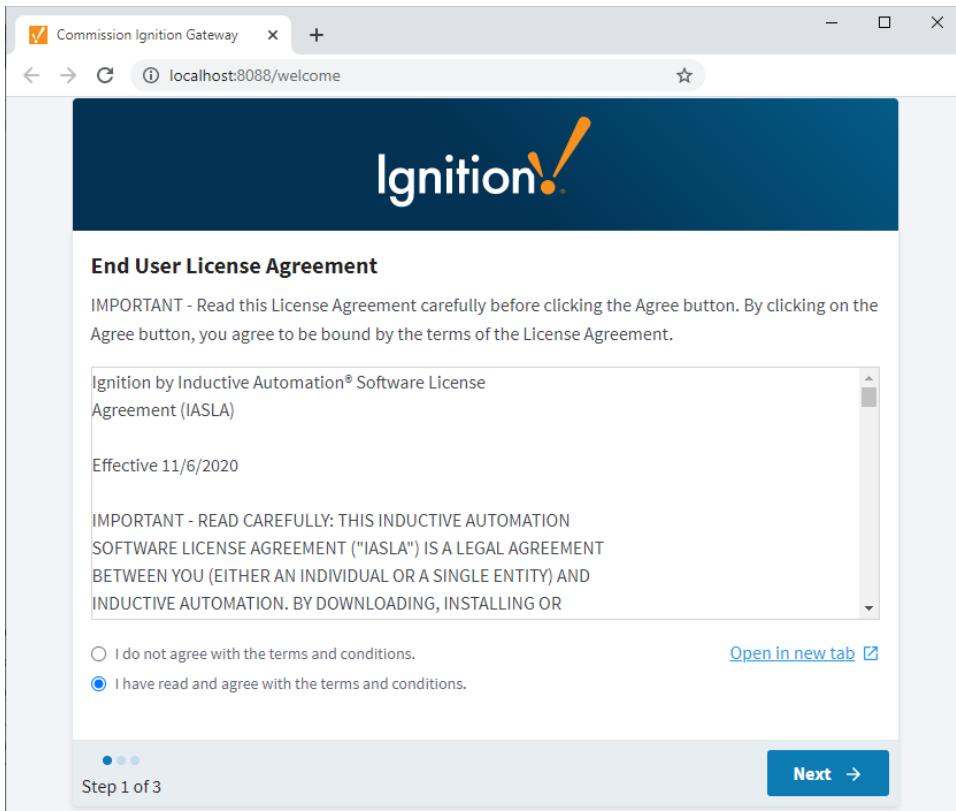
9. Once the Installer window shows that the Ignition installation is successfully installed, make sure that the checkbox for **Start Ignition Now** is selected, then click **Finish**.



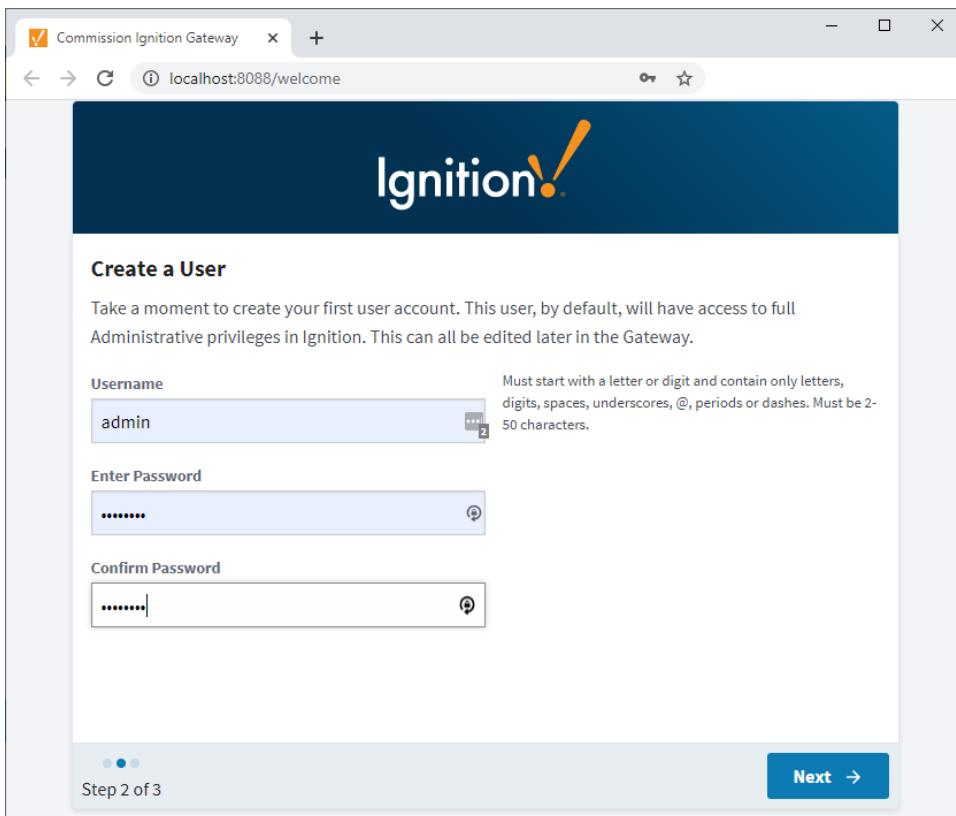
- Once Ignition is installed, your default web browser will open and you will be greeted with a **Welcome to Ignition**. You will be presented with a message to select a version to install. Select **Ignition**.



- The **End User License Agreement (EULA)** window will be displayed. Click the button acknowledging you read and agree with the terms and conditions, then press **Next**.



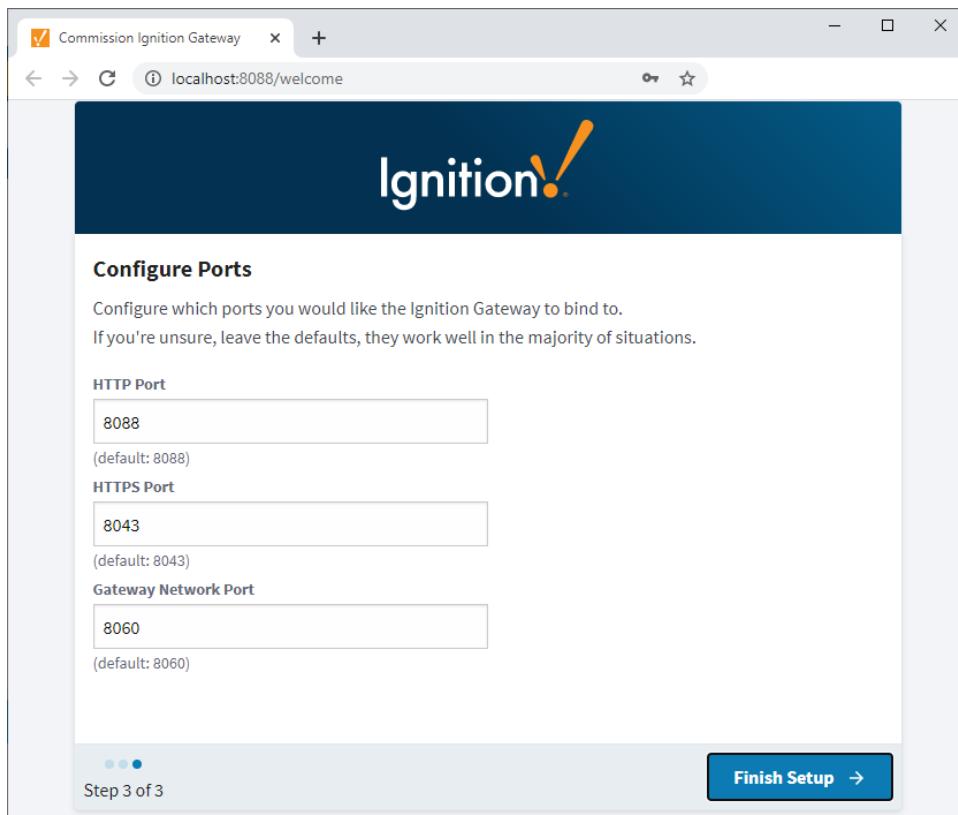
12. An **Account Setup** window will open for you to enter your first user. This first user will be the main / administrator account with full privileges in Ignition. Enter a **username** and **password**, and click **Next**.



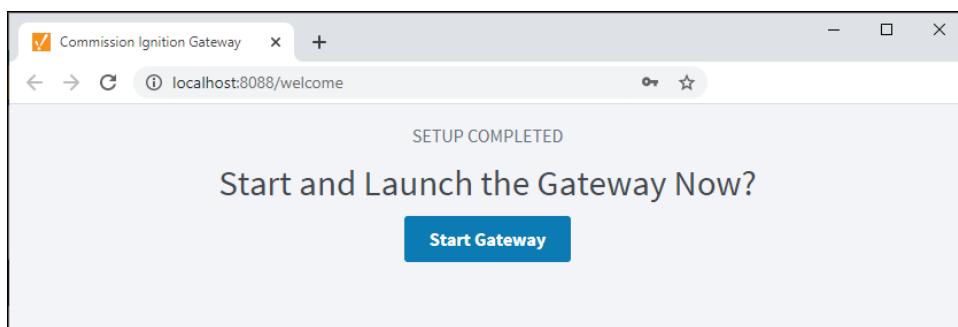
13. The **HTTP Configuration** window will open for you to configure the HTTP, HTTPS Ports, and Gateway Network Port. The default ports are: HTTP Port is 8088, HTTPS Port is 8043, and the Gateway Network Port is 8060. Commissioning will check the default ports for their availability. If no ports are available, the Gateway will iterate that port number until one is found that is available.

Existing or upgraded Gateways will not search for available ports, and will result in a faulted Gateway.

Click **Finish Setup**.



14. The **Setup Completed** window will be displayed, and now you can **Start the Gateway**. This may take a few minutes to finish.



15. That's it. Your Ignition installation is complete and the Gateway is launched. Your web browser opens the Gateway Homepage at **http://localhost:8088**. Sign in using the credentials for the administrator or user with full privileges for the Gateway as shown in Step 12 above.

Ignition

Home > Get Started

Trial Mode 1:50:06 We're glad you're test driving our software. Have fun.

WELCOME

Meet Ignition

Build It

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

Run It

Vision Client Launcher

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

 Download Vision Client Launcher

Perspective Workstation

Download Perspective Workstation to open sessions from any Ignition Gateway.

 Download Perspective Workstation

Perspective Session Launcher

Launch a Perspective session directly in your browser or download the native application.

 View Projects



Ignition installs itself as a Window Service (**Start > Control Panel > Administrative Tools > Services > Ignition Gateway**), so it starts automatically when your computer starts up.

Find more information on our [Installing and Upgrading Ignition](#) page.

THIS MAY TAKE A MOMENT

Ignition Gateway is starting



2. Launch the Gateway Webpage

The Ignition Gateway is a web server. When it is running, you access it through a web browser in just two steps.

If you're not already logged into Ignition, enter the credentials for the administrator or user with full privileges for the Gateway.

1. Figure out the IP address of the computer you installed Ignition on. If this computer is the one Ignition is installed on, then you can use **localhost** instead of the IP.
2. Open up a web browser and go to the Gateway's webpage: <http://localhost:8088>. If the gateway is installed on a different server /computer, then you'll need to know the IP address or host name: example: <http://ipaddress:8088>

The screenshot shows the Ignition Gateway's "Get Started" page. At the top, there are navigation links for "Home" and "Get Started". A green banner at the top indicates "Trial Mode 1:55:17" and says "We're glad you're test driving our software. Have fun." There is a "Help" link and a "Get Designer" button. On the left, a vertical sidebar has "Home" selected, along with "Status" and "Config" buttons. The main content area features a large orange exclamation mark icon with the text "Meet Ignition". Below it is a section titled "Build It" with a "Download Designer Launcher" button. To the right is a small icon of a laptop with a bar chart. The next section is titled "Run It" and contains three items: "Vision Client Launcher" (with a computer monitor icon), "Perspective Workstation" (with a computer monitor icon), and "Perspective Session Launcher" (with a computer monitor and keyboard icon). Each item has a "Download" button below its description.

The first time you go to the Gateway webpage, it shows you the links to download the Designer, Vision Client Launcher, Perspective Workstation, and Perspective Session Launcher.

3. To navigate back to the Gateway Homepage, click the **Home** button.



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](#)

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

[Download Vision Client Launcher](#)

Perspective Workstation
Download Perspective Workstation to open sessions from any Ignition Gateway.

[Download Perspective Workstation](#)

Perspective Session Launcher
Launch a Perspective session directly in your browser or download the native application.

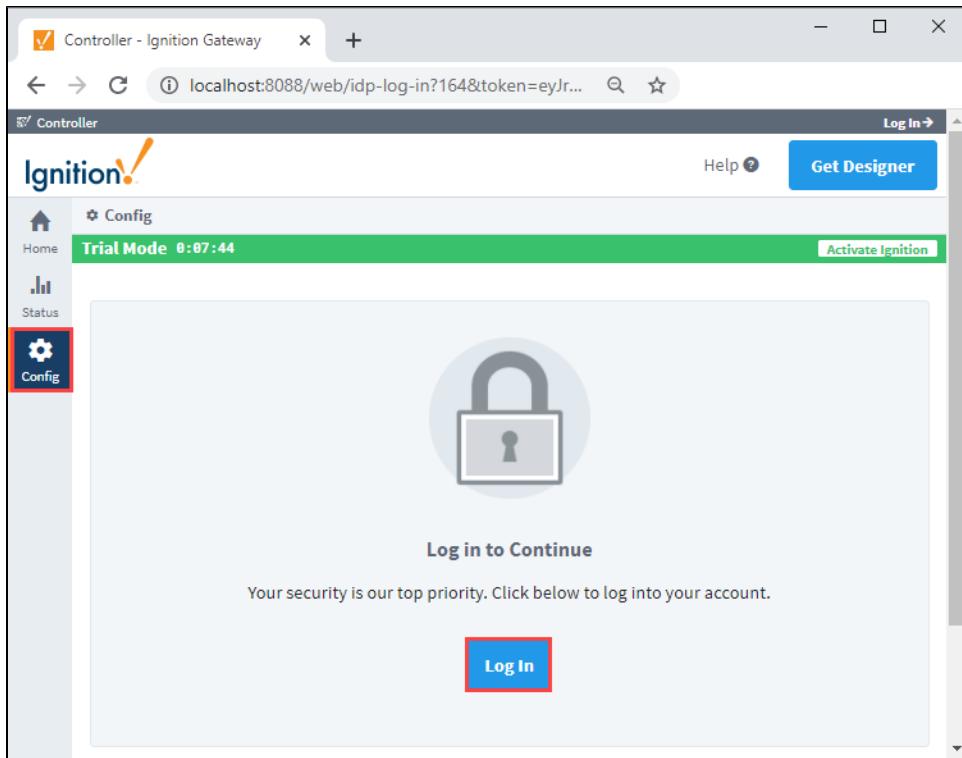
[View Projects](#)

3. Connect to a Device

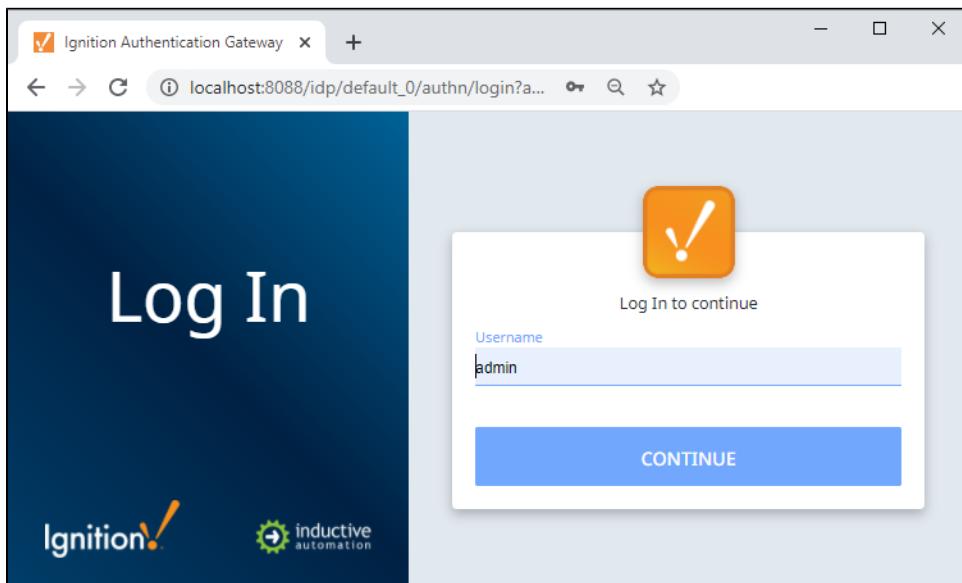
With the Ignition OPC UA module and device drivers installed, connecting to a device is simple.

With the Ignition OPC UA module and device drivers installed, connecting to a device is simple. The Programmable Device Simulator, as described below, includes Generic, Dairy and SLC simulator programs. Furthermore you can modify the existing programs to provide custom data points. It provides functionality that allows you to create your own simulator program with outputs you define. For more in depth information, refer to [Programmable Device Simulator](#).

1. If you are not already logged in to the Gateway webpage, click on the **Config** tab and press the **Log In** button.

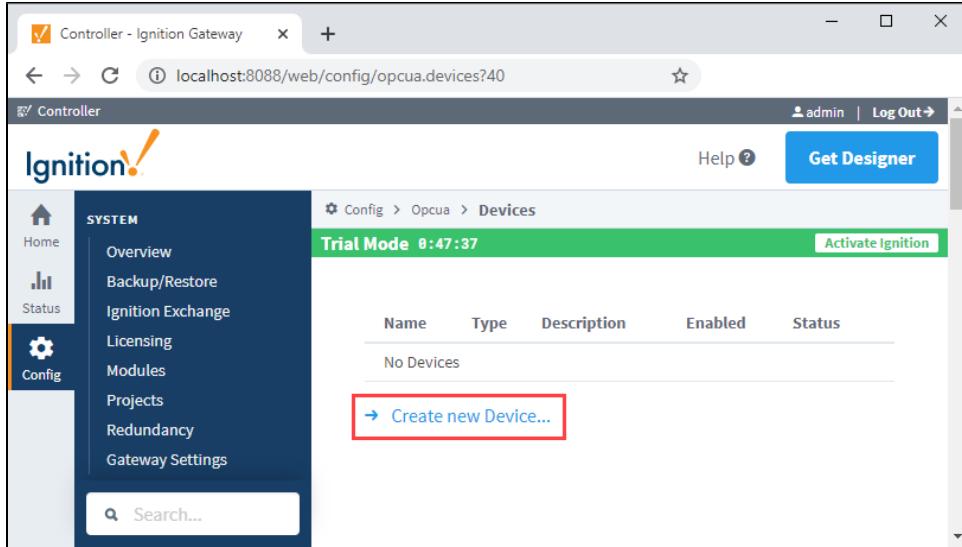


2. Enter the credentials and password for the administrator or user with full privileges for the Gateway.

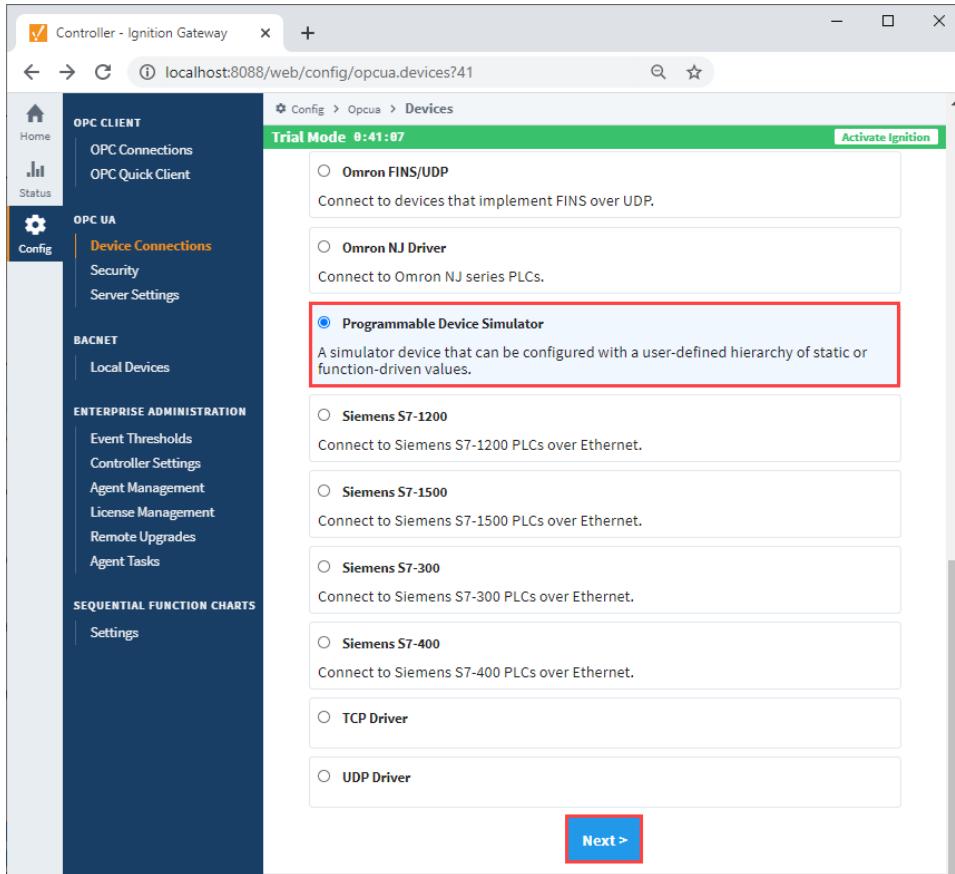


3. Scroll down to the **OPC UA** section on the left to **Device Connections**.

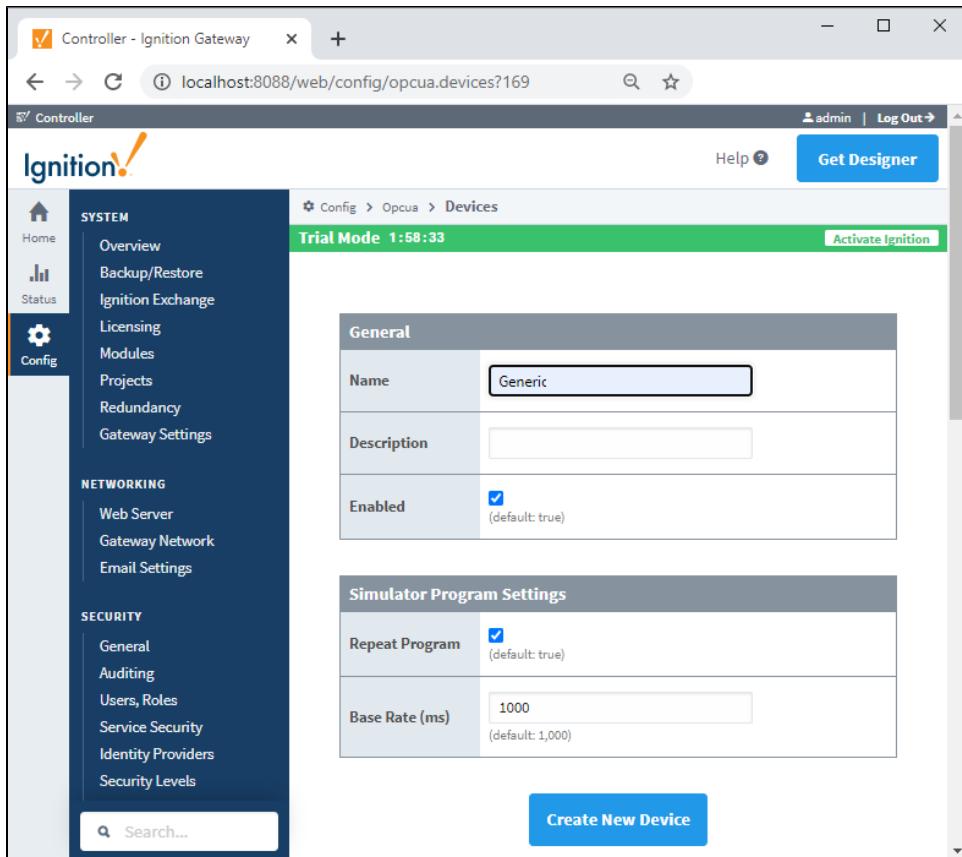
4. To add a new Device Connection, click the **Create new Device...** link.



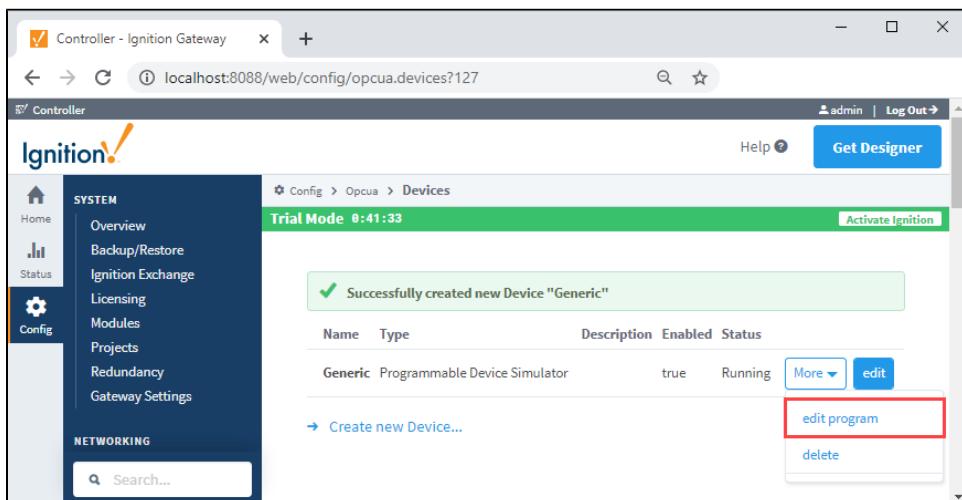
5. Select the **Programmable Device Simulator** and click **Next**.



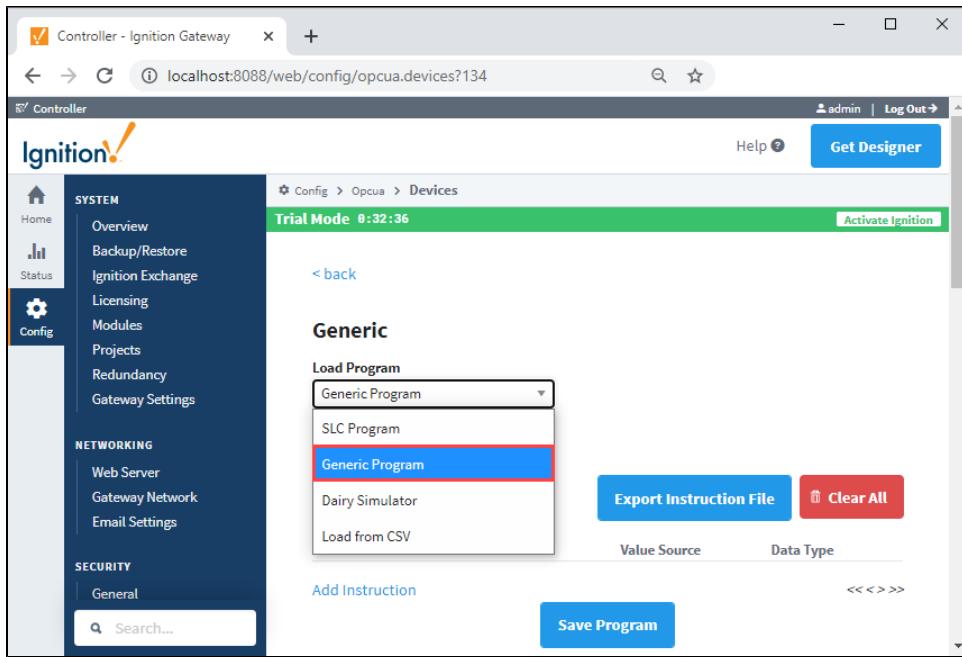
6. Give the new connection a name and click **Create New Device**.



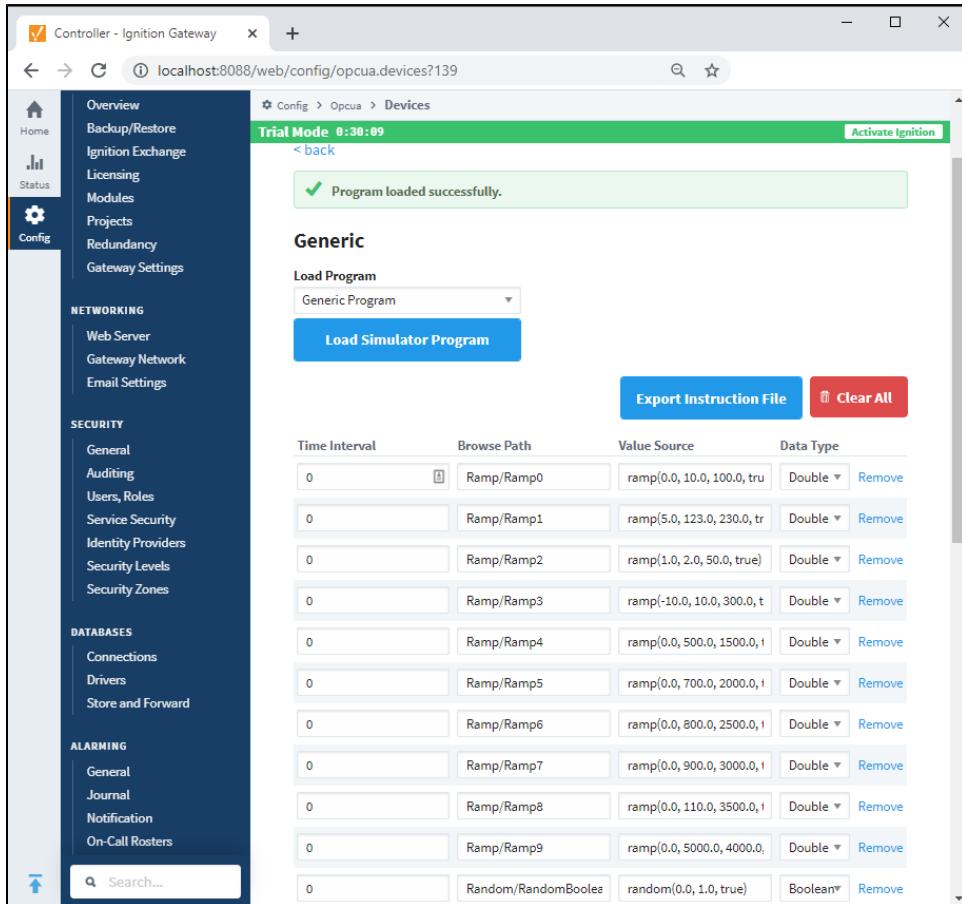
7. The window will refresh and you'll see your device was successfully created with a status of "Running". Now you can select the program for the simulator by clicking **More > edit program**.



8. From the **Load Program** dropdown, select a program (i.e., Generic Program) and click the **Load Simulator Program** button.



9. You will see a number of instructions which expand multiple pages. This is how the simulator creates Tags and sets their values. Click the **Save Program** button at the bottom of the page.



10. You have the option to modify the program before or after saving if you want to change some of the functionality. If you want to add an instruction, click the **Add Instruction** link under the last instruction on the screen. You can also **Remove** the instruction if necessary.

Ignition can connect to many different devices natively with the built-in OPC UA Server.

- Allen Bradley Ethernet
- Modbus
- Siemens
- UDP and TCP Driver
- DNP3
- Omron NJ Driver
- Programmable Device Simulator
- Third Party OPC Servers

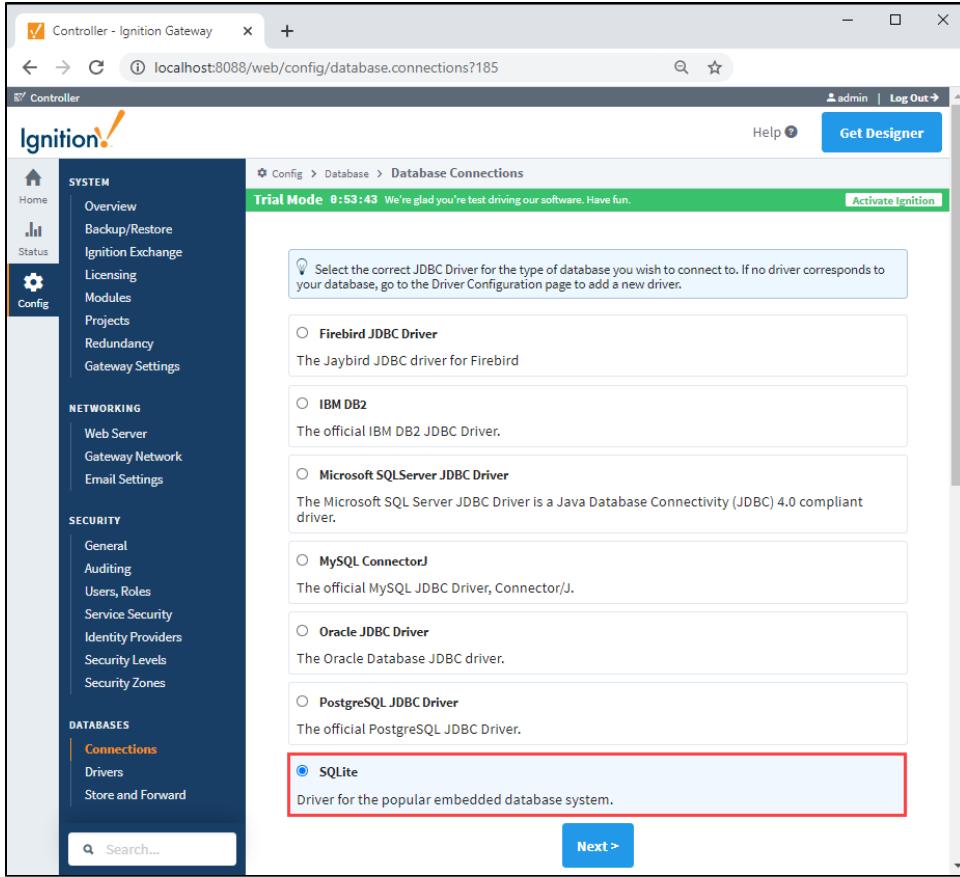
To learn more, go to the [OPC UA](#) and [Devices](#) pages!

| Name | Type | Description | Enabled | Status |
|---------|-------------------------------|-------------|---------|---------|
| Generic | Programmable Device Simulator | | true | Running |

4. Connect to a Database

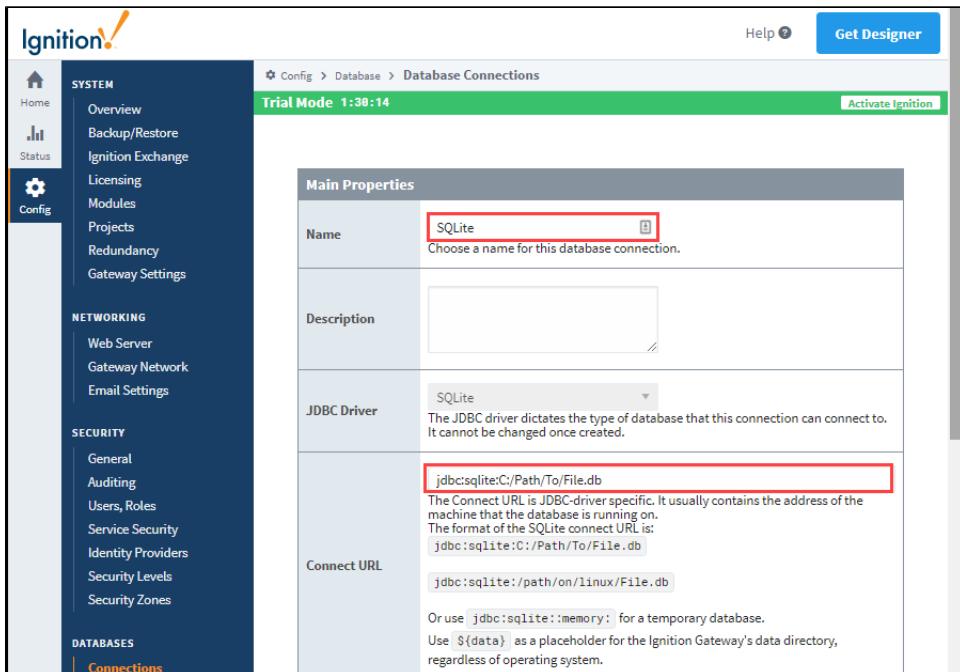
After you [install your database](#), you can connect it to Ignition. Alternatively, you can have Ignition create an internal SQLite database, which is ideal for testing, demonstrations, and just getting started. This next section will show how to create and connect Ignition to a SQLite database.

1. If you are already logged into Ignition, then go to the **Config** section of the Gateway Webpage, otherwise login as the user with full / administrative privileges.
2. From the **Config** section, navigate to the **Databases > Connections** page.
3. Click the **Create new Database Connection...** link.
4. Select the **SQLite** option and click the **Next >** button.



5. On the Database Connections page, enter a **Name** and **Connect URL**. You can leave the other fields blank. The Connect URL will default to : **`jdbc:sqlite:C:/Path/To/File.db`** (On Windows systems). Part of the Connect URL is a file path. The gateway will attempt to create a SQLite database at the specified path (assuming there isn't already one there, in which case it simply connects to that database).

The SQLite Database Connection does not require a user name or password to connect. In a production system you'll likely want to connect to a server based database instead, but when you're just getting started the SQLite database is easy to use. For now, scroll to the bottom and click the **Create New Database Connection** button.



A "success" message will appear when the database connection is created.

The screenshot shows the Ignition Gateway Webpage. The left sidebar has a dark blue background with icons for Home, Status, and Config. Under SYSTEM, it lists Overview, Backup/Restore, Ignition Exchange, Licensing, Modules, Projects, Redundancy, and Gateway Settings. A search bar at the bottom says "Search...". The main content area has a green header bar with the text "Trial Mode 0:35:58" and a "Get Designer" button. Below the header, the URL is "Config > Database > Database Connections". A green success message box says "Successfully created new Database Connection "SQLite"" with a checkmark icon. A table below shows one connection: "Name: SQLite, Description: SQLite, JDBC Driver: SQLITE, Translator: Valid". There are "delete" and "edit" buttons next to the row. At the bottom, there's a link to "Create new Database Connection..." and a note: "Note: For details about a connection's status, see the [Database Connection Status](#) page."

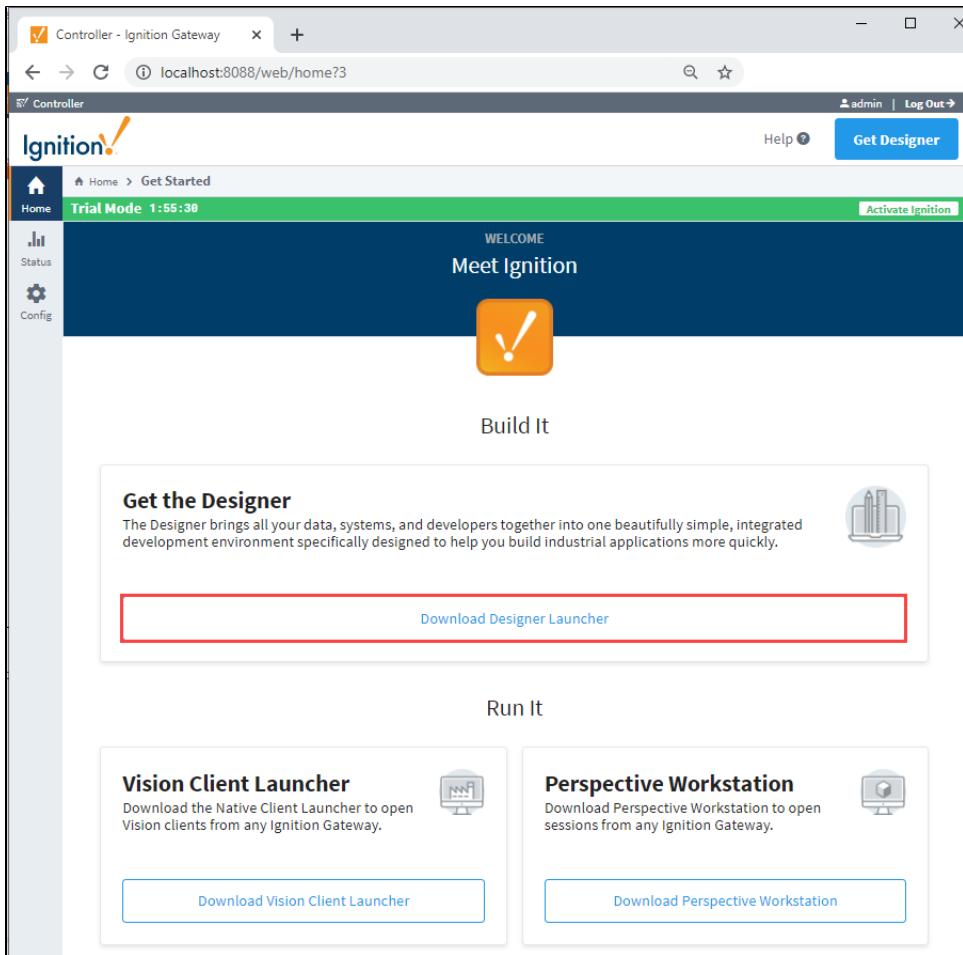
To get detailed descriptions for a number of different database connections, see [Connecting to Databases](#). You can also go to [Database Connections](#) to learn how databases are used in Ignition and how to get started.

This screenshot is identical to the one above, showing the Ignition Gateway Webpage with a successful database connection creation message and a table listing the "SQLite" connection.

5. Open the Designer

Now that you set up a device ([Step 3](#)) and database ([Step 4](#)) connections, let's open the Designer and create a project.

1. If you are already logged into Ignition, then go to the **Home** tab of the Gateway Webpage, otherwise login as the user with full / administrative privileges.
2. Under Get the Designer, click the **Download Designer Launcher** button.



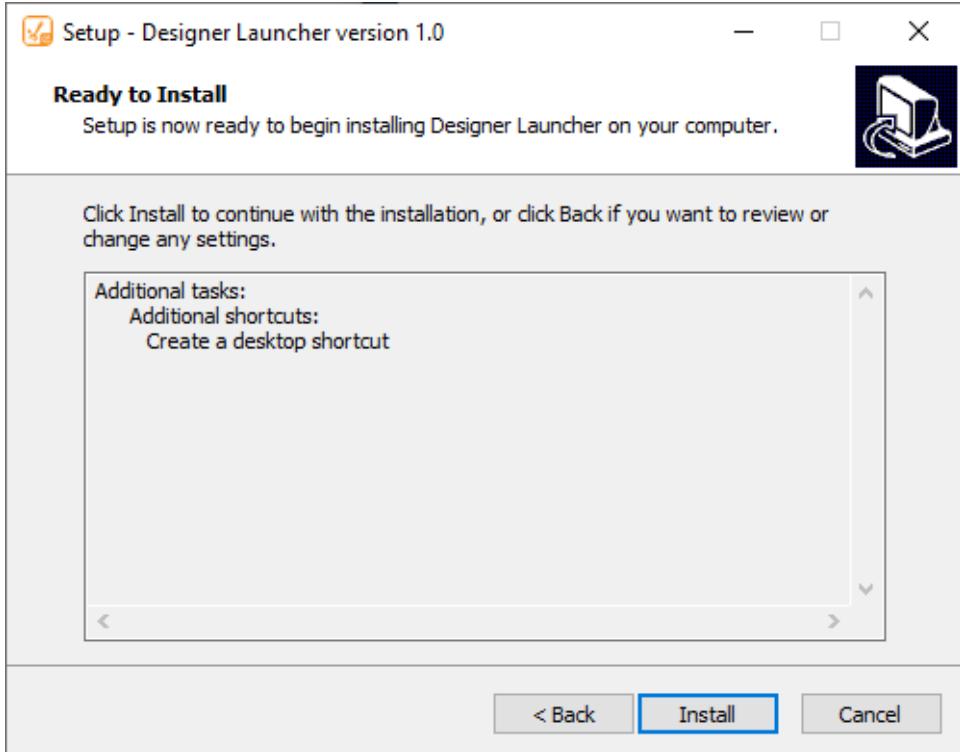
3. Select the **Download** button for the the platform you are running on: Windows: Mac OS, or Linux. In this example, we are running on Windows.

The screenshot shows the Ignition Gateway Designer Launcher Download page. At the top, it says "Controller - Ignition Gateway" and "localhost:8088/web/designer-launchers?9". The header includes "admin | Log Out" and "Get Designer". Below the header, it says "Trial Mode 1:33:43 We're glad you're test driving our software. Have fun." and "Activate Ignition". On the left, there's a sidebar with "Home", "Status", and "Config". The main content area has a heading "Ignition! Designer Launcher Download" and a sub-heading "Designer Launcher". It says "Download the Designer Launcher" and provides a "Download for Windows" button. To the right, there's a section titled "Alternative Designer Launcher" with links for "Windows" (62.3MB), "Mac" (50.7MB), and "Linux" (72.6MB). A modal window is open, asking "What do you want to do with DesignerLauncherSetup.exe (45.9 MB)? From: 127.0.0.1" with options "Run", "Save", and "Cancel". Below the modal, there are three numbered steps: 1. Run the Designer Launcher Installer, 2. Follow Setup Instructions, and 3. Provide Firewall Access.

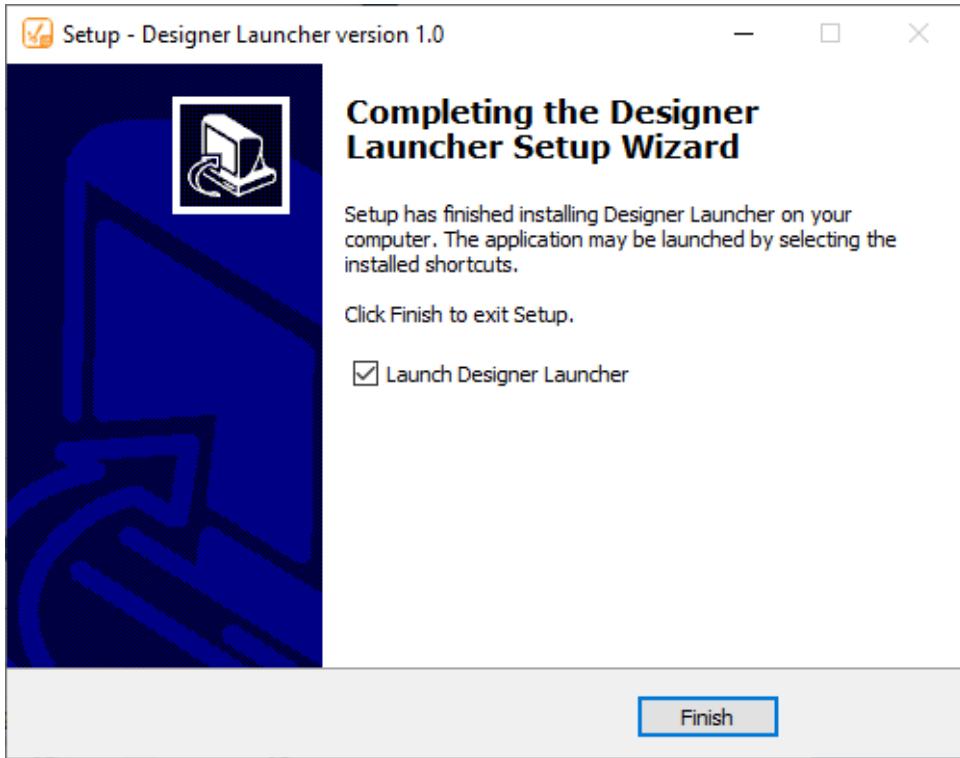
4. Run the downloaded file (**DesignerLauncher.exe**) found on the lower-left of the window, or go to your downloads folder and double-click the file to run it.
5. The Select Additional Tasks window will open. Here, you can select additional tasks that you would like to setup to perform while installing the Designer Launcher. It's a good idea to **Create a desktop shortcut** so it's easily accessible when you want to open the Designer Launcher. Click **Next**.

The screenshot shows the "Select Additional Tasks" window for the Designer Launcher setup. The title bar says "Setup - Designer Launcher version 1.0". The main content area has a heading "Select Additional Tasks" and a sub-heading "Which additional tasks should be performed?". It says "Select the additional tasks you would like Setup to perform while installing Designer Launcher, then click Next." Below this, there's a section for "Additional shortcuts:" with a checkbox labeled "Create a desktop shortcut" which is checked. At the bottom, there are "Next >" and "Cancel" buttons.

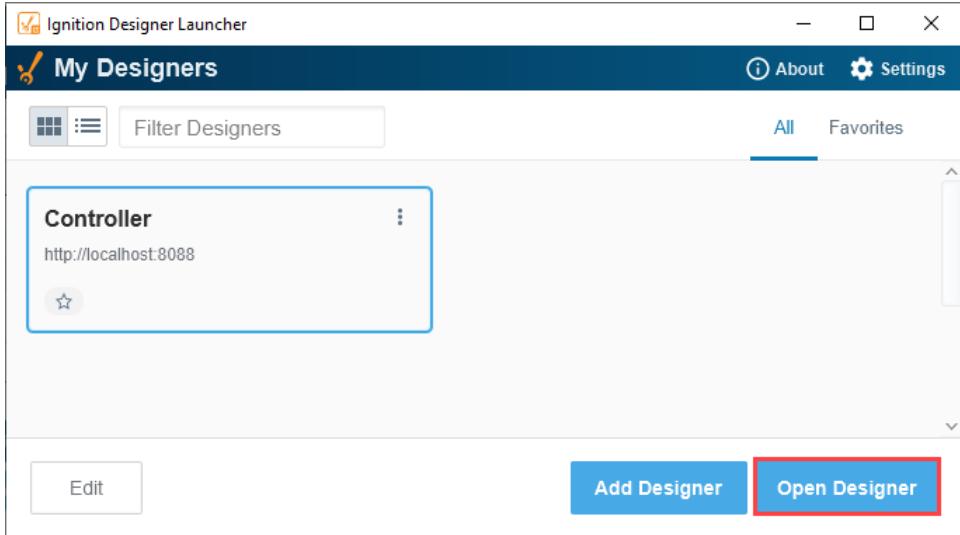
6. Now, you're Ready to Install the Designer Launcher on your computer. Click **Install** to continue.



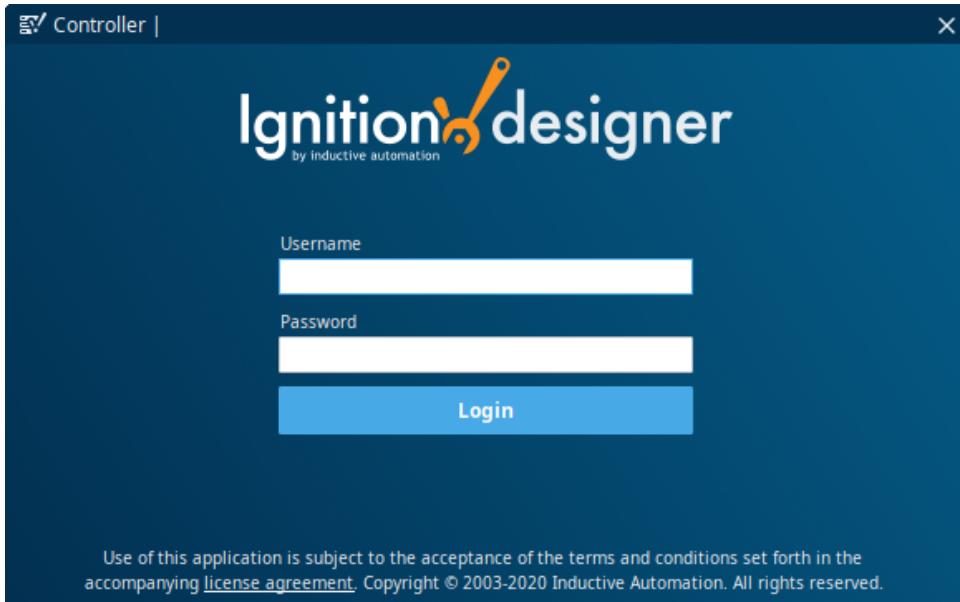
- Once the Designer Launcher is installed, complete the setup by checking the box to **Launch Designer Launcher** and click **Finish**.



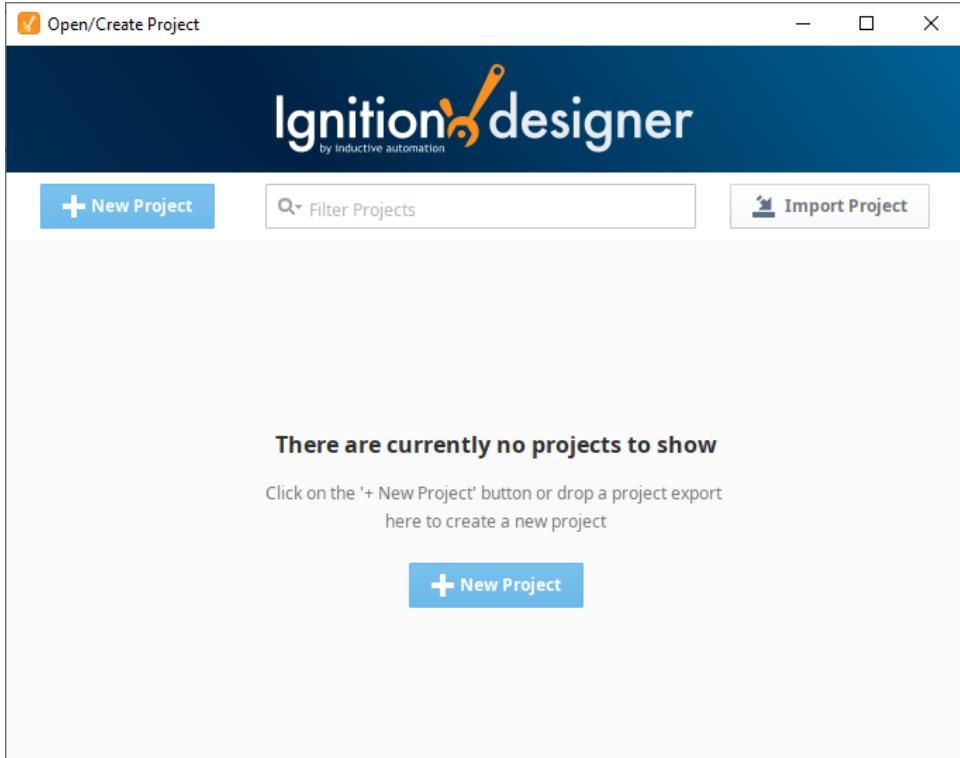
- The Ignition Designer Launcher window will open. Here you can select any Designers you have permission to launch. If you are running this for the first time, by default, you will need to add a Designer. Click **Open Designer** to launch an existing Designer. Refer to the [Launchers and Workstation](#) page for detailed information on adding a Designer, including creating a Designer Shortcut, adding to Favorites, and more.



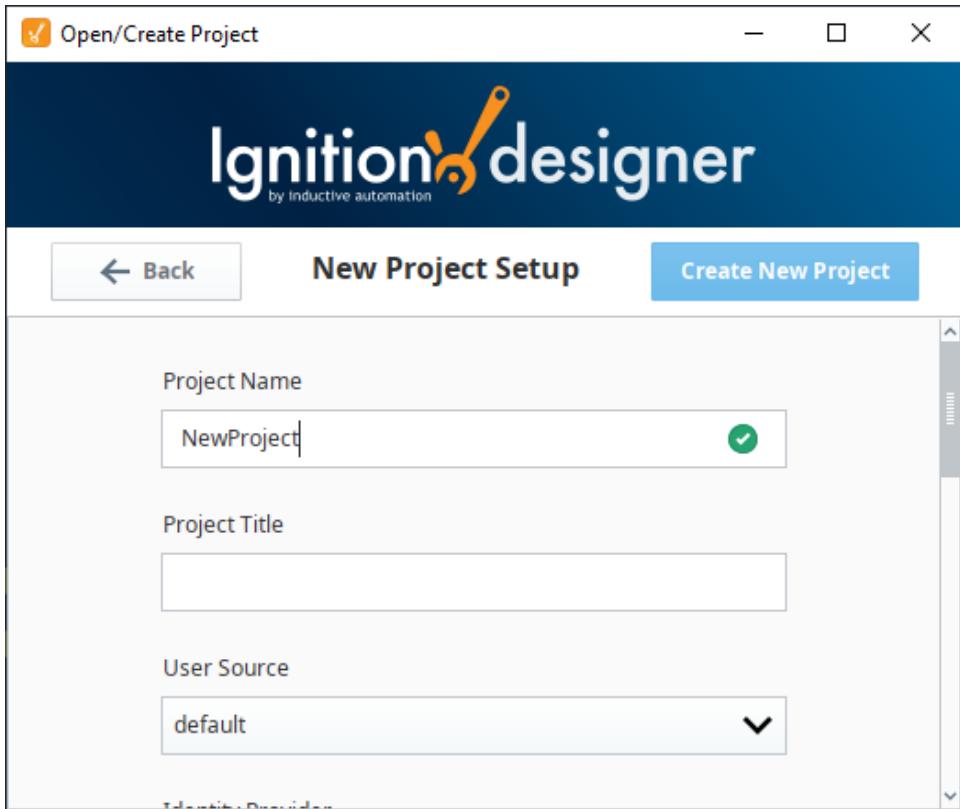
9. The Designer starts up and the login window is displayed. Enter the credentials for the administrator or user with full privileges for the Designer, and click **Login**. It should be the same as the login for the Gateway **Config** section.



10. The **Open/Create Project** window is displayed giving you two options: create new project or open an existing one. Let's create a new project by clicking **+ New Project**.

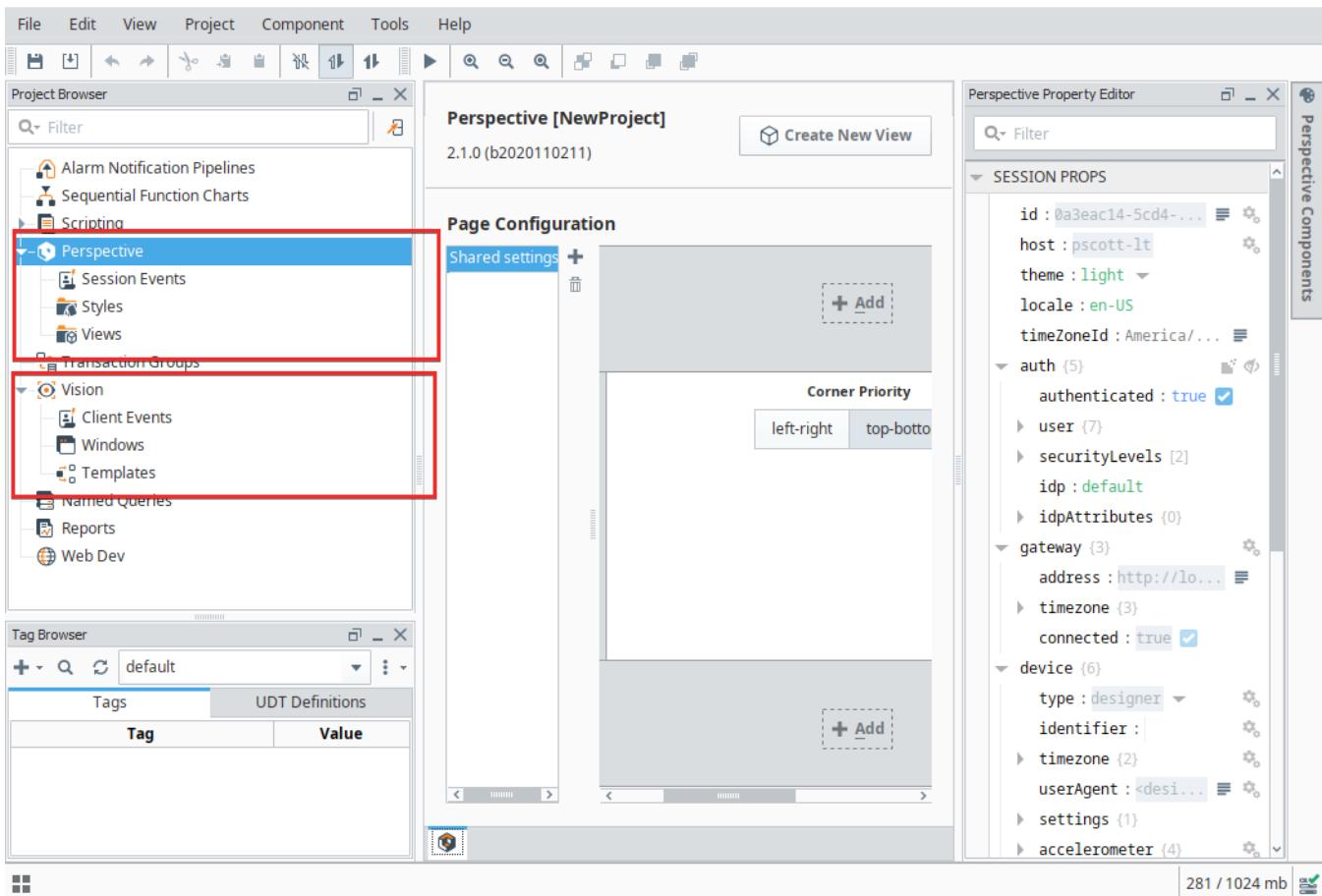


11. This opens the New Project Setup window. Enter the **Project Name** (required) and any other configuration settings you need for your project. Most settings are optional.
Refer to the [Project Creation Settings table](#) for a description of each property. When finished, click **Create New Project**.



12. When you open the Designer interface for the first time, the [Perspective Page Configuration](#) opens if you have the Perspective Module installed. If you don't have the Perspective Module installed, the Vision Designer window will open. You can begin designing in either Perspective or Vision from the Designer interface, as shown in the image below. Expand the Perspective and Vision folders and you'll notice that the folder contents are different. In Perspective, you add components to Views. In Vision, you add components to Windows.

Learn more about the [Designer](#), [Perspective Designer Interface](#), and [Vision Designer Interface](#).

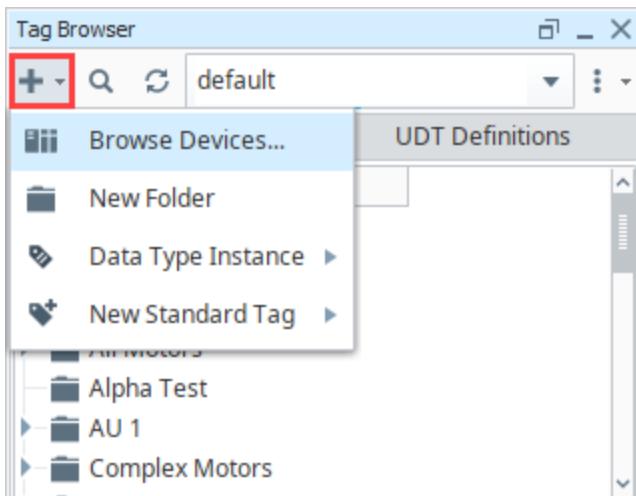


6. Create Tags

With the Designer open, a great first step is to create some Tags. Once there, Tags are all configured in the **Tag Browser** panel. You'll use these Tags for realtime status and control and to store history with the Tag Historian.

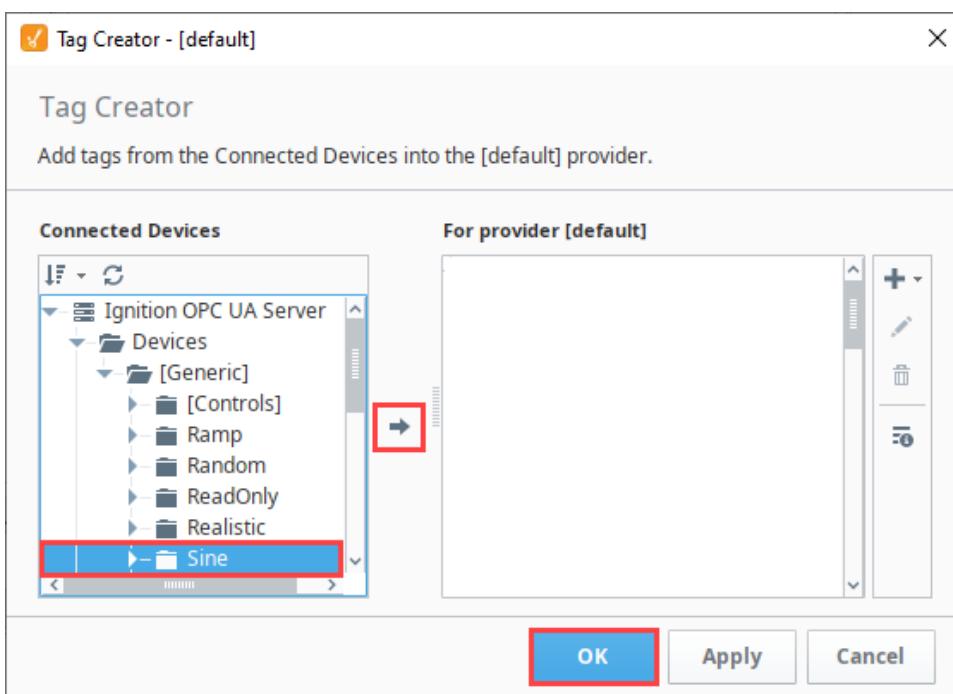
If you are connected to a device in [Step 3](#), the easiest way to create some Tags is by dragging the Tags from the Ignition OPC UA Server.

1. On the left side of the Designer, you will see the Project Browser and Tag Browser. In the Tag Browser, click on the **Add** icon to open the context menu and select **Browse Devices**. You can view all your connected devices and add Tags.

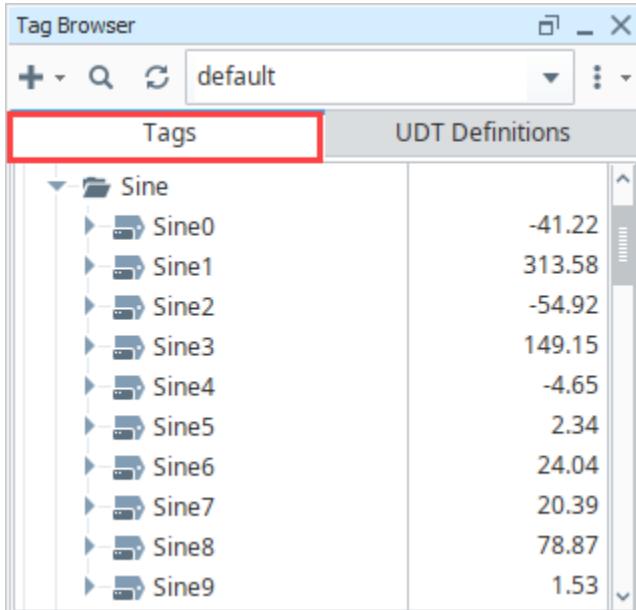


2. By default, you have a connection to the internal **Ignition OPC UA Server**. Under Connected Devices, expand the Ignition OPC UA Server to find your list of devices, and then expand the device folder to see some Tags inside. Highlight the Tags you want (for example, as in the image below, the entire **Sine** folder is selected). Click the right arrow icon to move your selected folder or Tags to the **For Provider** area. Click **OK** to copy the Tags to your Tag Browser. Ignition keeps the same hierarchy in the Tag Browser as in the PLC.

example, as in the image below, the entire **Sine** folder is selected). Click the right arrow icon to move your selected folder or Tags to the **For Provider** area. Click **OK** to copy the Tags to your Tag Browser. Ignition keeps the same hierarchy in the Tag Browser as in the PLC.



3. That's it. You now have some Tags. Click the **Tags** tab and open new **Sine** folder to see what was added. You can see their values come in and start updating automatically. By default, they update at a 1 second rate.



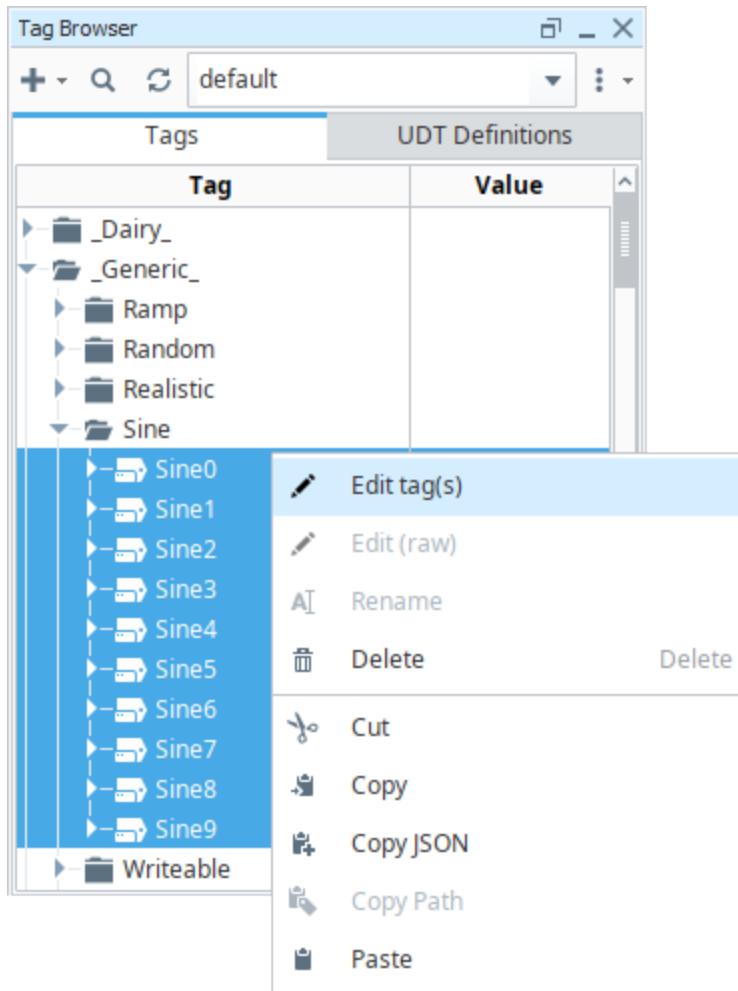
To learn more about about Tags, see the [Tags](#) page.

| Tag Browser | | |
|-------------|-------|-----------------|
| default | | |
| | Tags | UDT Definitions |
| ↳ Sine | Sine0 | -41.22 |
| | Sine1 | 313.58 |
| | Sine2 | -54.92 |
| | Sine3 | 149.15 |
| | Sine4 | -4.65 |
| | Sine5 | 2.34 |
| | Sine6 | 24.04 |
| | Sine7 | 20.39 |
| | Sine8 | 78.87 |
| | Sine9 | 1.53 |

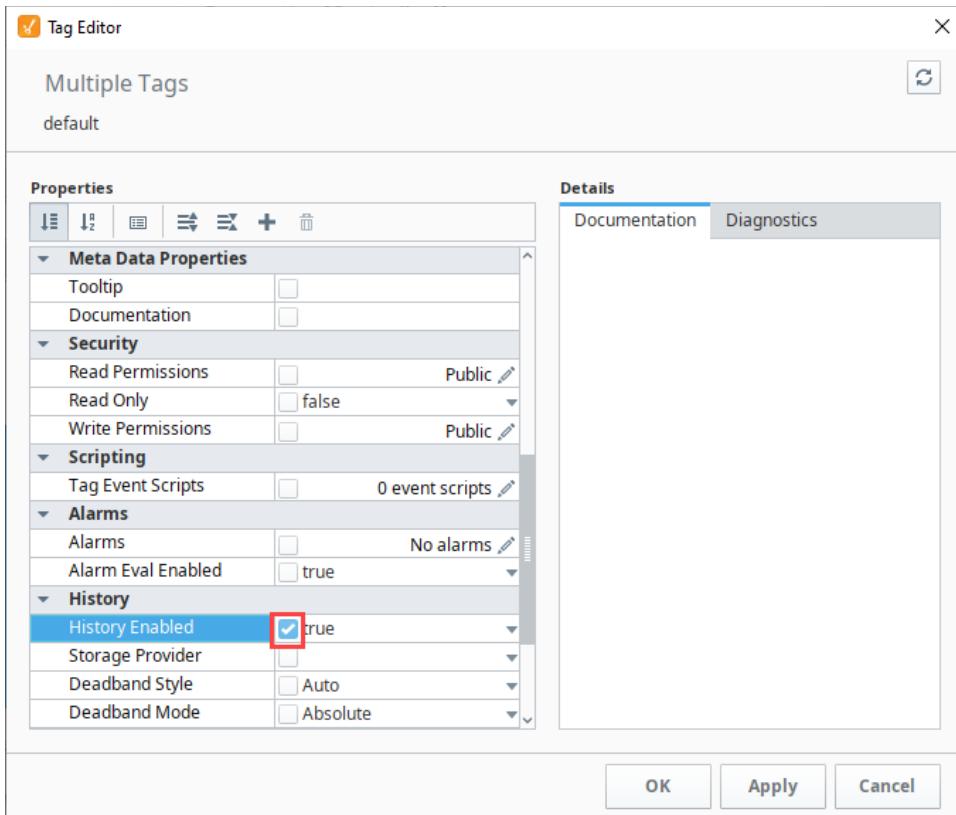
7. Add History to Tags

Storing history for your Tags is simple, and once that's done we can display it on screen in a variety of ways. If you are connected to a database ([Step 4](#)) , you are ready to add history to some of our Sine Tags.

1. In the Designer, go to the **Tag Browser** and select one or more Tags.
For example, select all the **Sine** Tags in the Sine folder by holding the **Ctrl** key and clicking on several Tags.
2. Right-click on the selected Tags, and then select **Edit tag(s)**.



3. The Tag Editor window is displayed. Here, you can edit multiple Tags and change the data type, scaling options, metadata, permissions, history, and alarming. Scroll down to **History** on the left side of **Tag Editor**, and set **History Enabled** to 'true'.

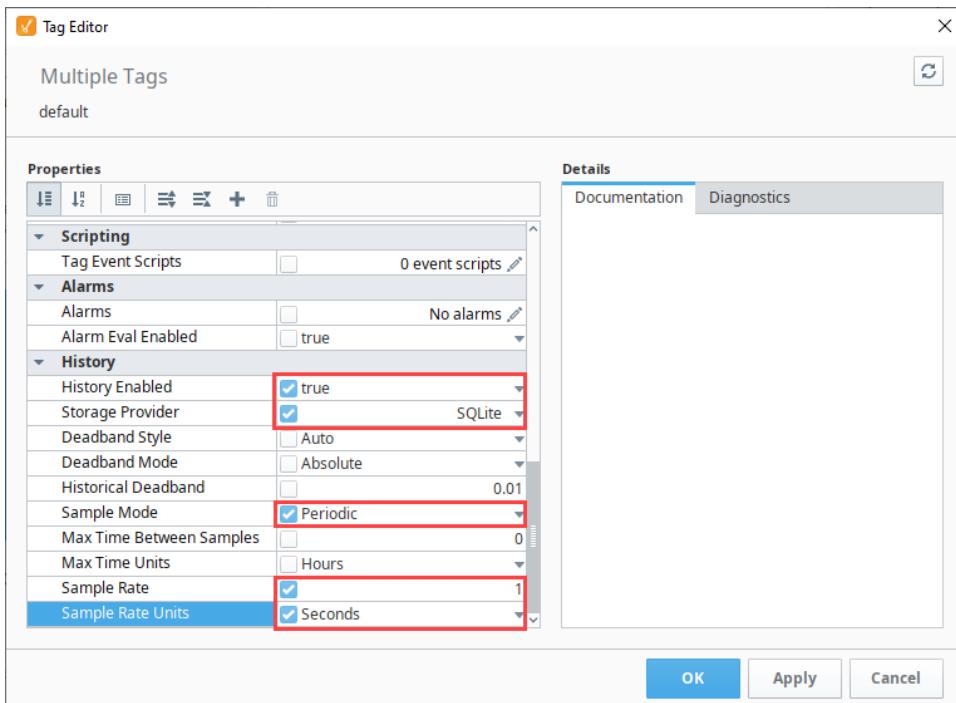


4. Choose a database from the **Storage Provider** dropdown, like the one we set up in [Step 4](#) above.

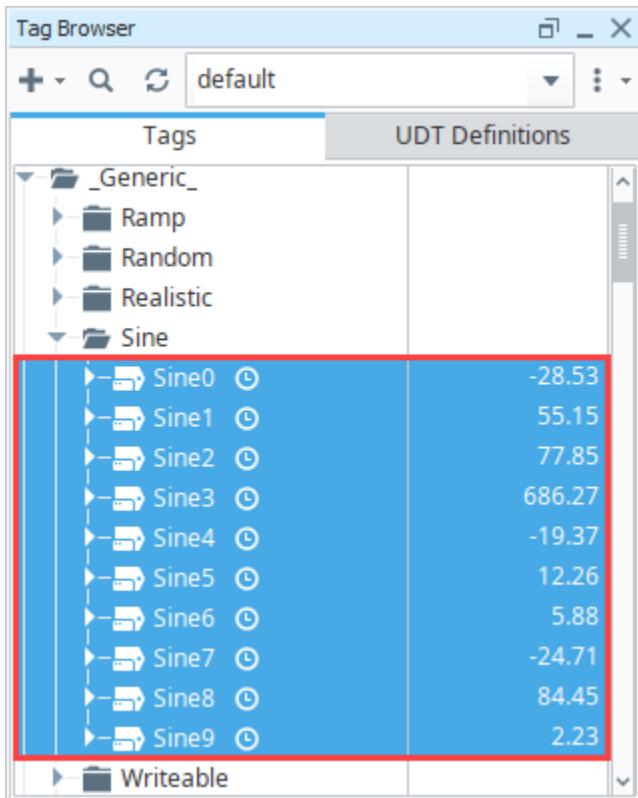
5. To set the rate that data is logged:

- Change the **Sample Mode** to **Periodic**.
- Set the **Sample Rate** to **1**. (Data will be logged every **1** second).
- Verify that the **Sample Rate Units** is set to **Seconds** (which is the default).

6. Click **OK**.

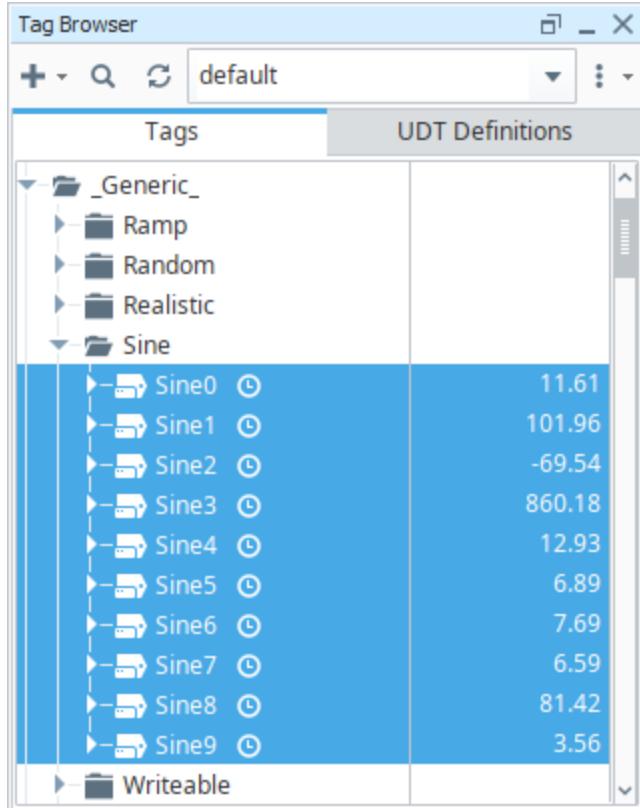


7. Now look in the **Tag Browser**, to the right of each **Sine** Tag that is storing history, an icon (⌚) appears letting you know it is setup.



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

To learn more about the Tag Historian, see the [Tag Historian](#) page.

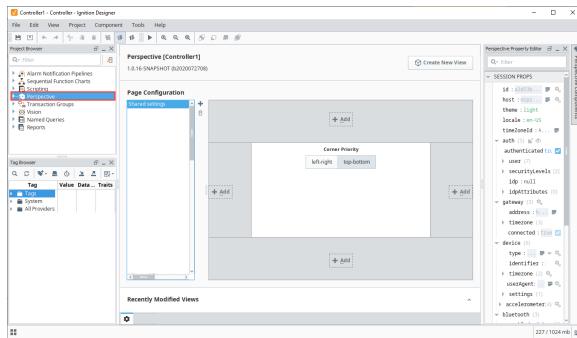


8. Adding Components

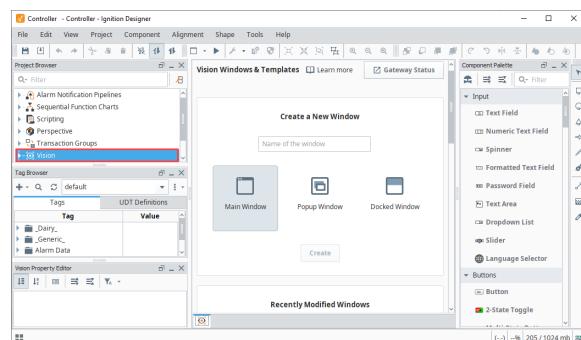
Building your industrial application starts by adding components. Whether you're using the Perspective Module or the Vision Module, adding components is easy, but they each have their own visualization system. It's for this reason, the next step in the Startup Guide is separated into two separate pages. Click the link below to navigate to the respective module's page.

When you open the Designer interface for the first time, the Perspective Page Configuration opens assuming you have the Perspective Module installed, as shown in the Perspective Session image below. If you don't have the Perspective Module installed, the Vision workspace will open.

Creating a Perspective Session



Creating a Vision Client



Related Topics ...

- [Gateway Backup and Restore](#)
- [Licensing and Activation](#)
- [Alarming](#)
- [Security](#)
- [Localization and Languages](#)

[In This Section ...](#)

Startup - Perspective Session

Before we talk about how to add components to [Perspective Views](#), it's important to first understand all the objects that comprise a [Perspective Session](#). It will help you determine your design strategy within a browser-based design environment and how components inside a View will behave before you ever drag a component into a View.

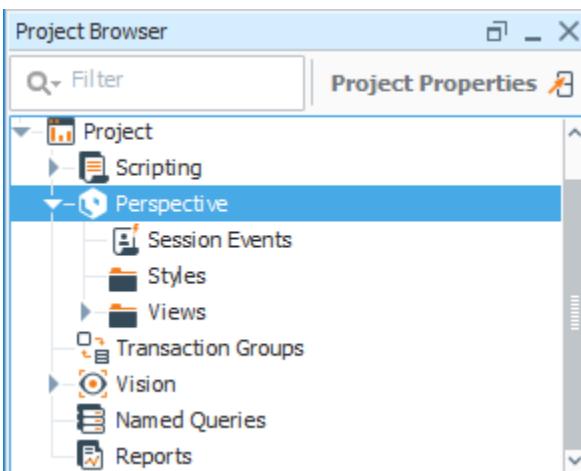
With that in mind, we recommend reading the [Perspective Sessions](#) section of the manual to fully understand how all the objects that make up a Perspective Session work together to create dynamic pages. You may also want to peruse the page on new [navigation strategies](#) that are offered in Perspective.

On this page, we demonstrate how to add components to a View and bring them to life by showing live values, displaying Tag History data on a chart, configuring pages for navigating within a session, and launching a Perspective session.

1. Creating Views and Adding Components

Views are the [primary unit of design](#) and can be displayed across the top, bottom or sides of a page, or even a whole page in a Perspective Session. You can drag and drop components into a View, and make them display text, show Tag Values, and display Tag History data.

1. With the Perspective Designer open, go to the [Project Browser](#), and expand **Perspective**.



2. In Perspective, views are the primary unit of design and a container provides a way of organizing and laying out components within a view. There are a variety of container types that support different layout strategies. Refer to [Views and Containers in Perspective](#) to learn more about container types and configuring them.

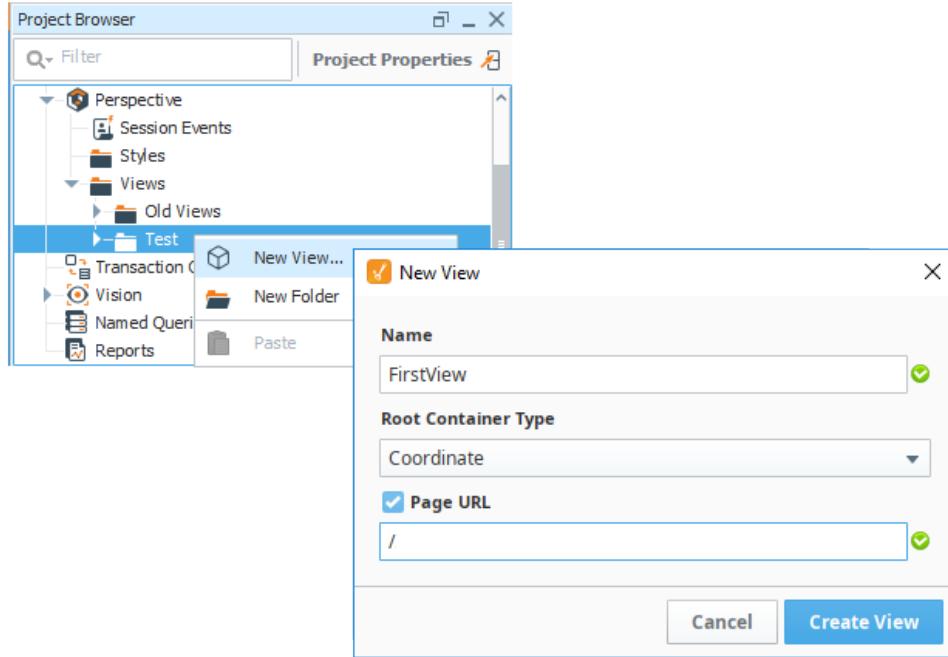
To keep this example simple and get you started creating your first view, we used a [Coordinate](#) container type.

- a. In the Project Browser, click on the **Views** folder and right click on **New View...**
- b. Select a **Root Container Type** from the dropdown. Select **Coordinate** container type.
- c. Check the **Page URL** checkbox if you want your view attached to the Page URL.
- d. Click **Create View**.



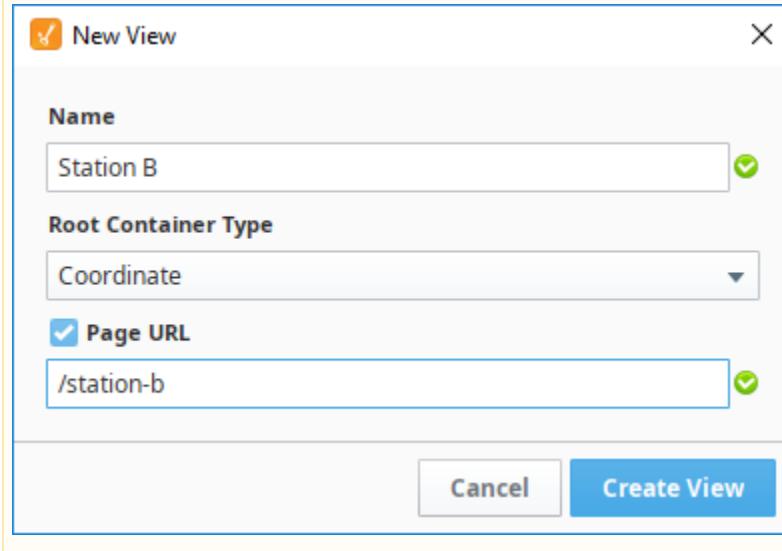
Match the URL with the Folder Structure

Perspective will match the **Page URL** with the View's folder structure upon creation. In the example below, the FirstView was created in the Test folder. Perspective knows to match the URL to the View.

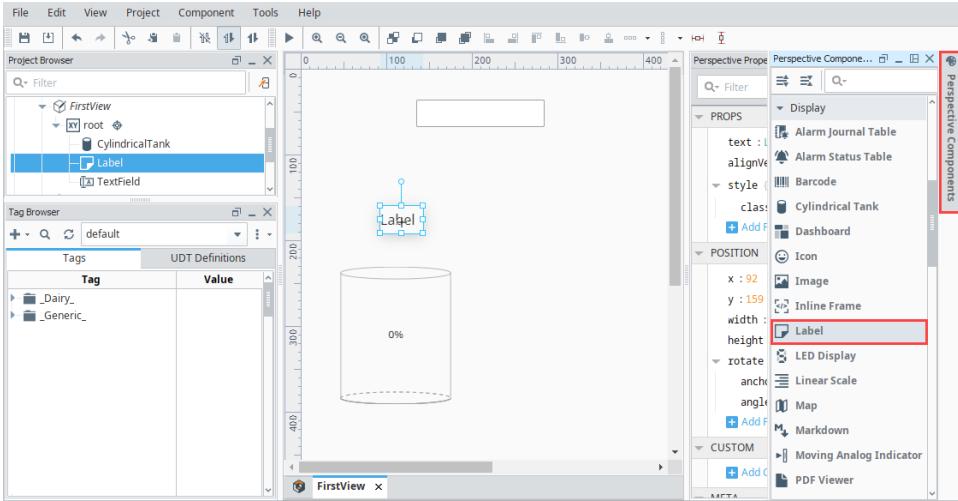


Note:

If your View Name contains a space, such as "Station B", the Page URL will replace the space with a hyphen. Notice how the Page URL automatically gets filled in when the View Name is typed in.



3. The Designer will refresh and now you're ready to begin adding components into your view. On the right side of the Designer, click on the grayed out **Perspective Components** banner to open the **Components Palette**. Drag the **Text Field**, **Label**, and **Cylindrical Tank** components into the View.

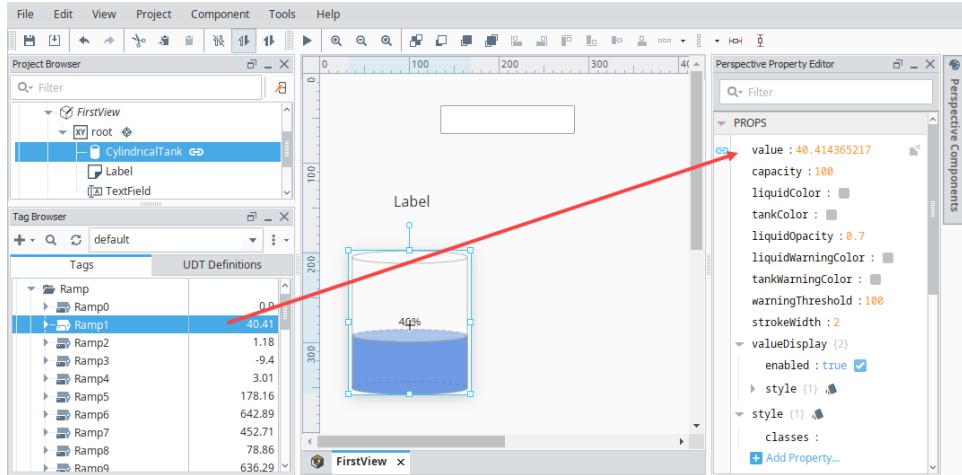


4. To add Tag values to each of the components, click on the component and enter the value in the **Perspective Property Editor**, or drag a Tag to the **value** property.

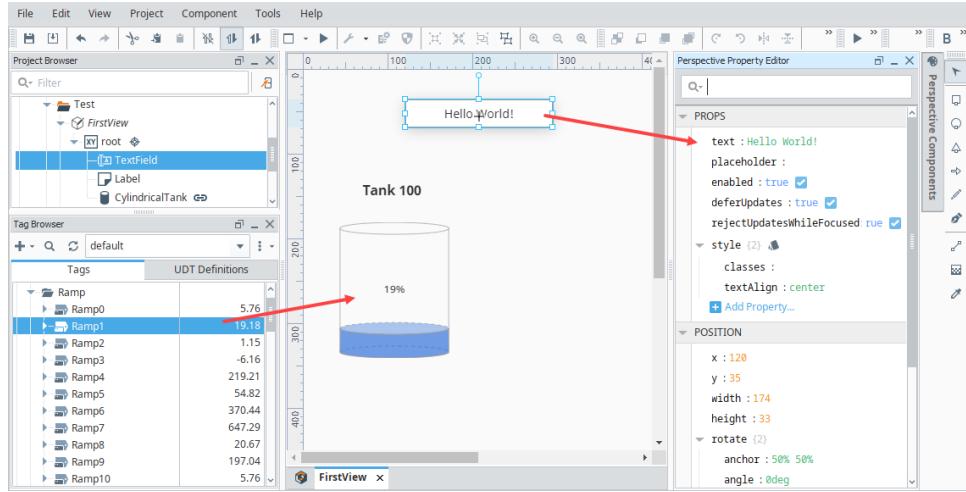
a. **Cylindrical Tank** - Drag a Tag from your Tag Browser on to the **value** property. This example uses a **Ramp1** Tag.

Note: This example requires that you are connected to the [Programmable Device Simulator](#) and set the program to Generic.

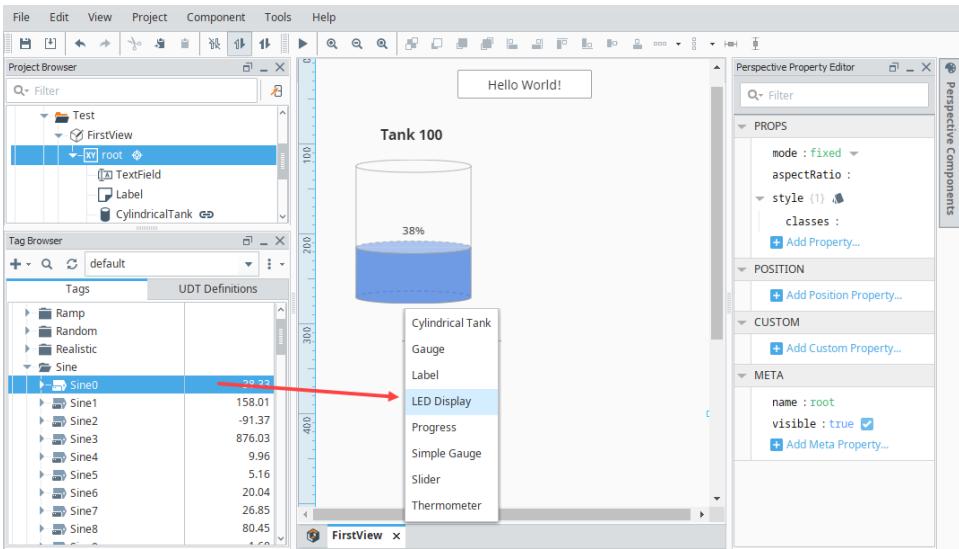
- b. **Label** - Enter 'Tank 100' next in the **Text** property field.
 c. **Text Field** - Enter 'Hello World!' in the **Text** property field. While the component is selected, you can also click and drag the arrow handles to resize them and move them in the Designer workspace.



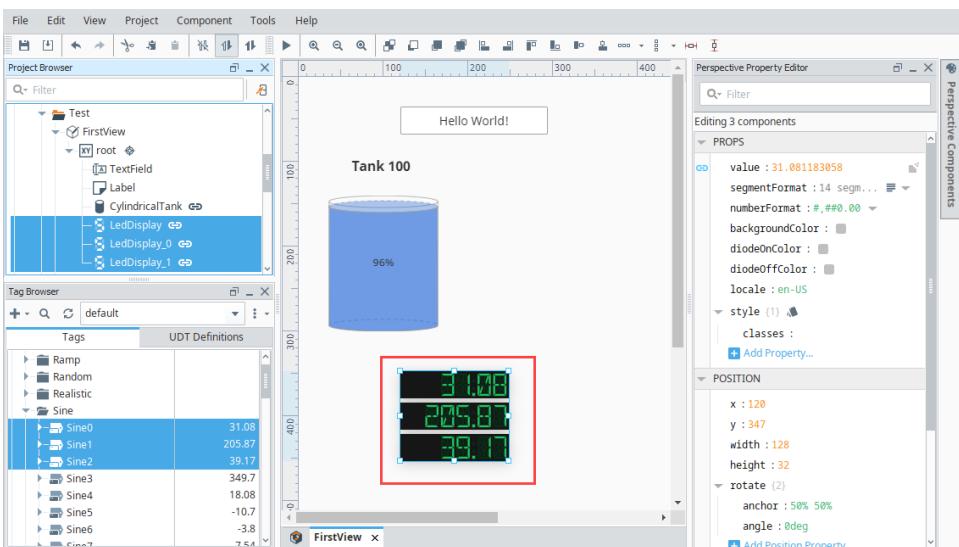
Once all the values are added to the components, they will look like the following image in your View.



- Another way and the simplest way to add a Tag to component is to go into the **Tag Browser**, select the Tag (i.e., Sine0) you want and drag it into your view. A popup will appear giving you several selections for components to use. We selected an **LED Display** to show our Tag Value.

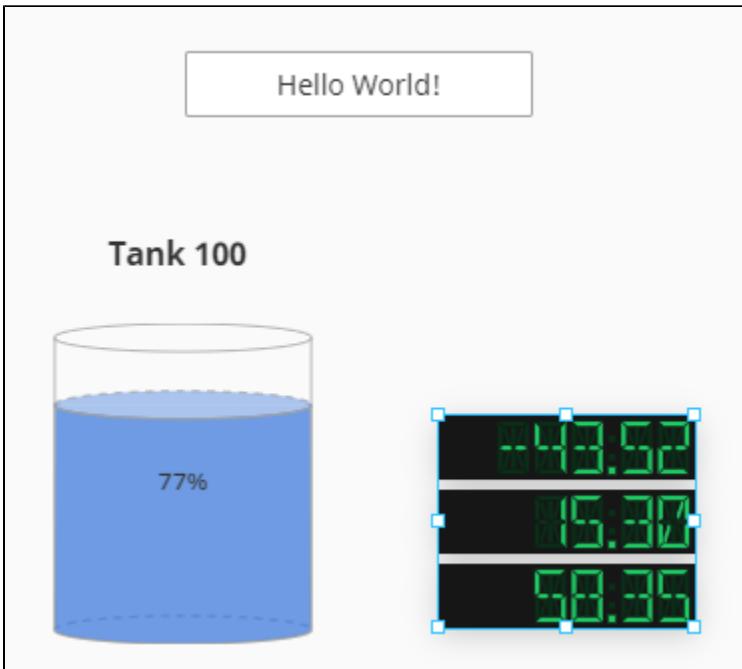


We created two more LED Displays using Sine1 and Sine3.



- Save your project by selecting **File > Save** in the menubar.

To learn more about bindings, go to the [Bindings in Perspective](#) page.

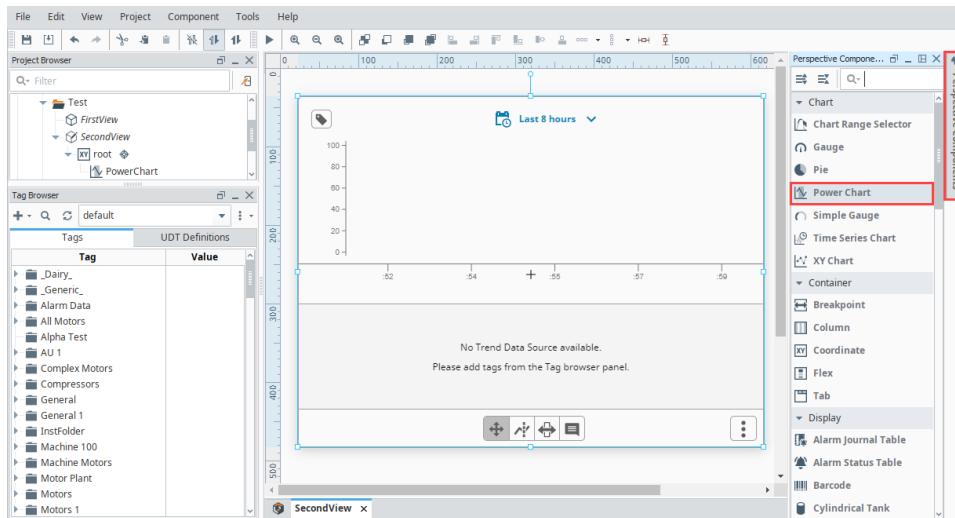


2. Showing Tag History in a Power Chart

Displaying Tag History data using a [Power Chart](#). This requires that you have Tag History already set up (See step 7 on the previous Startup Guide page). Refer to the [Configuring Tag History](#) page for more details.

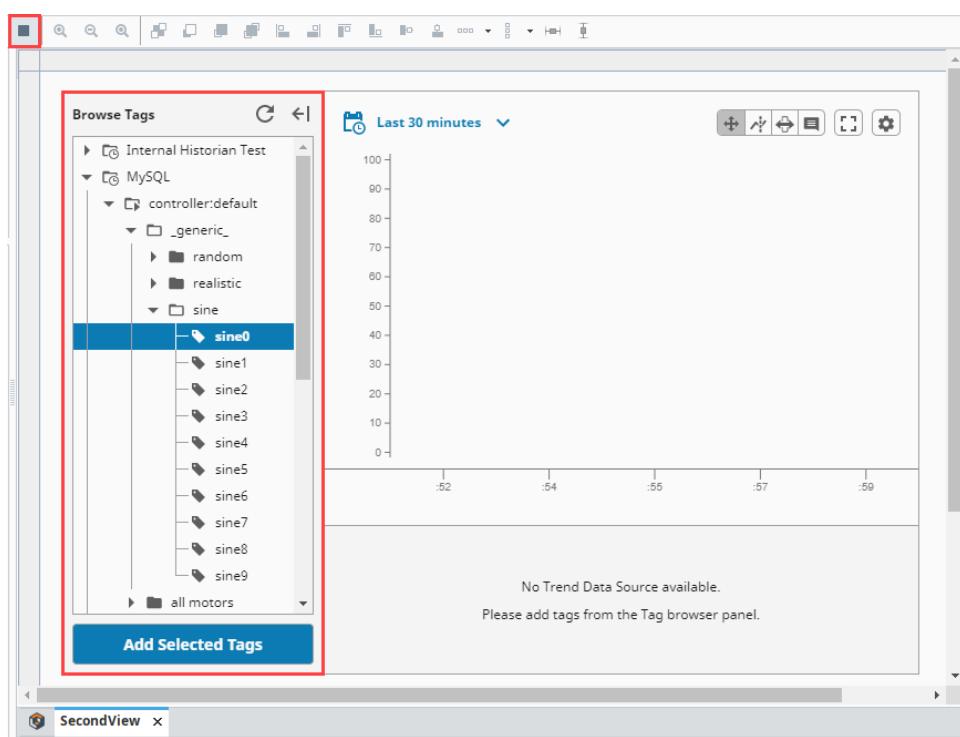
Let's create a second view (i.e., **SecondView**), add a Power Chart component to show history on the values of several Sine Tags.

1. Right click on the Views Test folder and create a new view (i.e. **SecondView**).
2. Once you have your second view created, drag a **Power Chart** component on to your view.

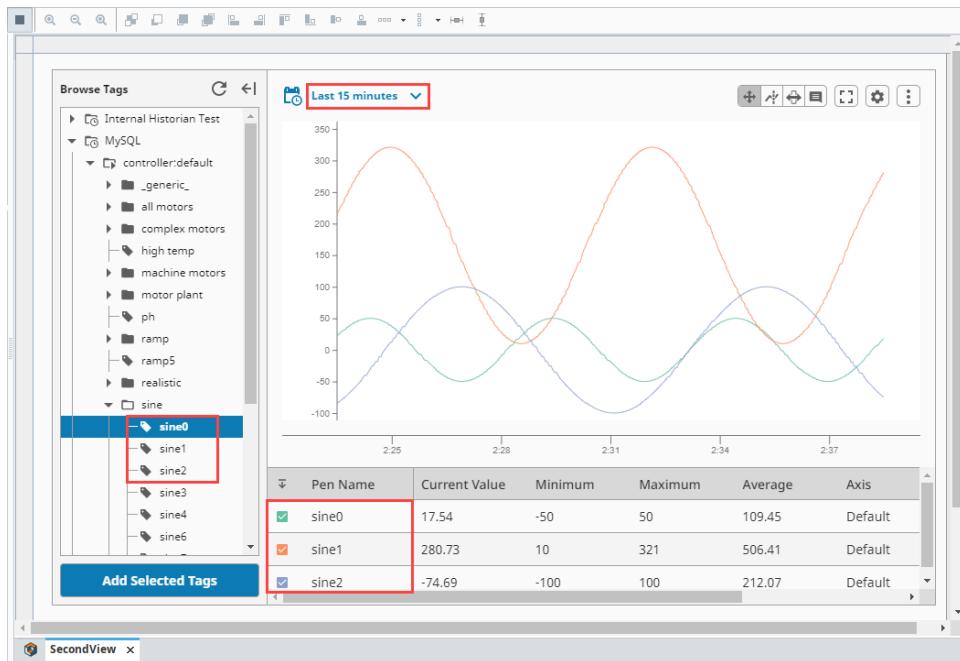


3. With the Power Chart selected, put your Designer in **Preview Mode** and click the Tags icon in the top left of the chart. This will open the built-in Tag Browser in your Power Chart where you can add or drag Tags into your chart.

Note: You will need to have Tag History configured and stored in a database on the Tags you want to add to the Power Chart. If you don't see the Tag(s) you want to add in the built-in Tag Browser, you will need to enable **History** on the Tag in the Tag Editor.

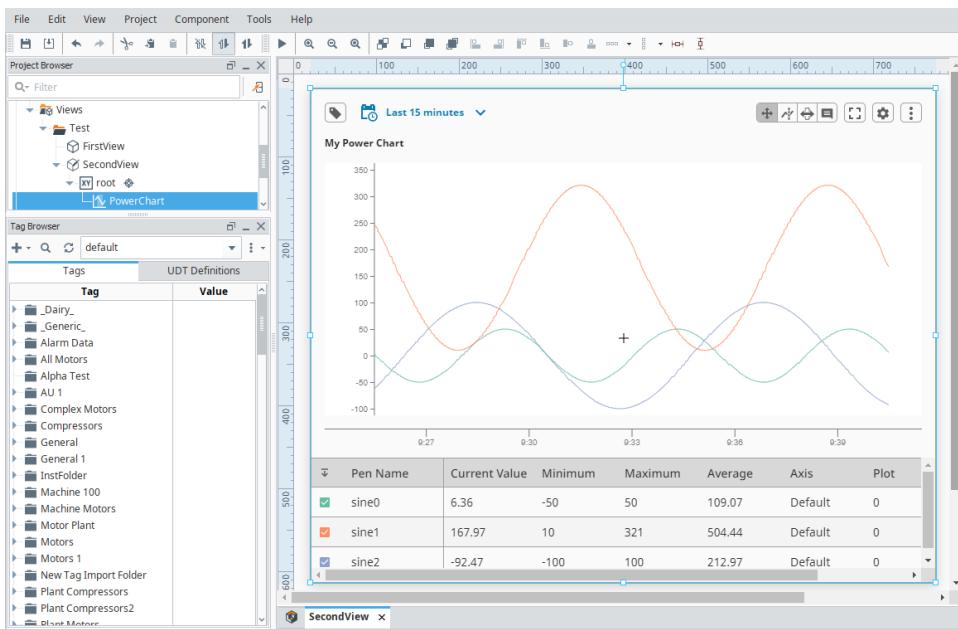


4. Select some Tags and click the **Add Selected Tags**. You can also change the **Time Range Selector** if you want to show your Tags in Realtime or Historical. In the following image, we added the **Sine0**, **Sine1** and **Sine2** Tags. We also selected **Realtime**, and **15 minutes** as our timeframe to display the Sine values on the chart.

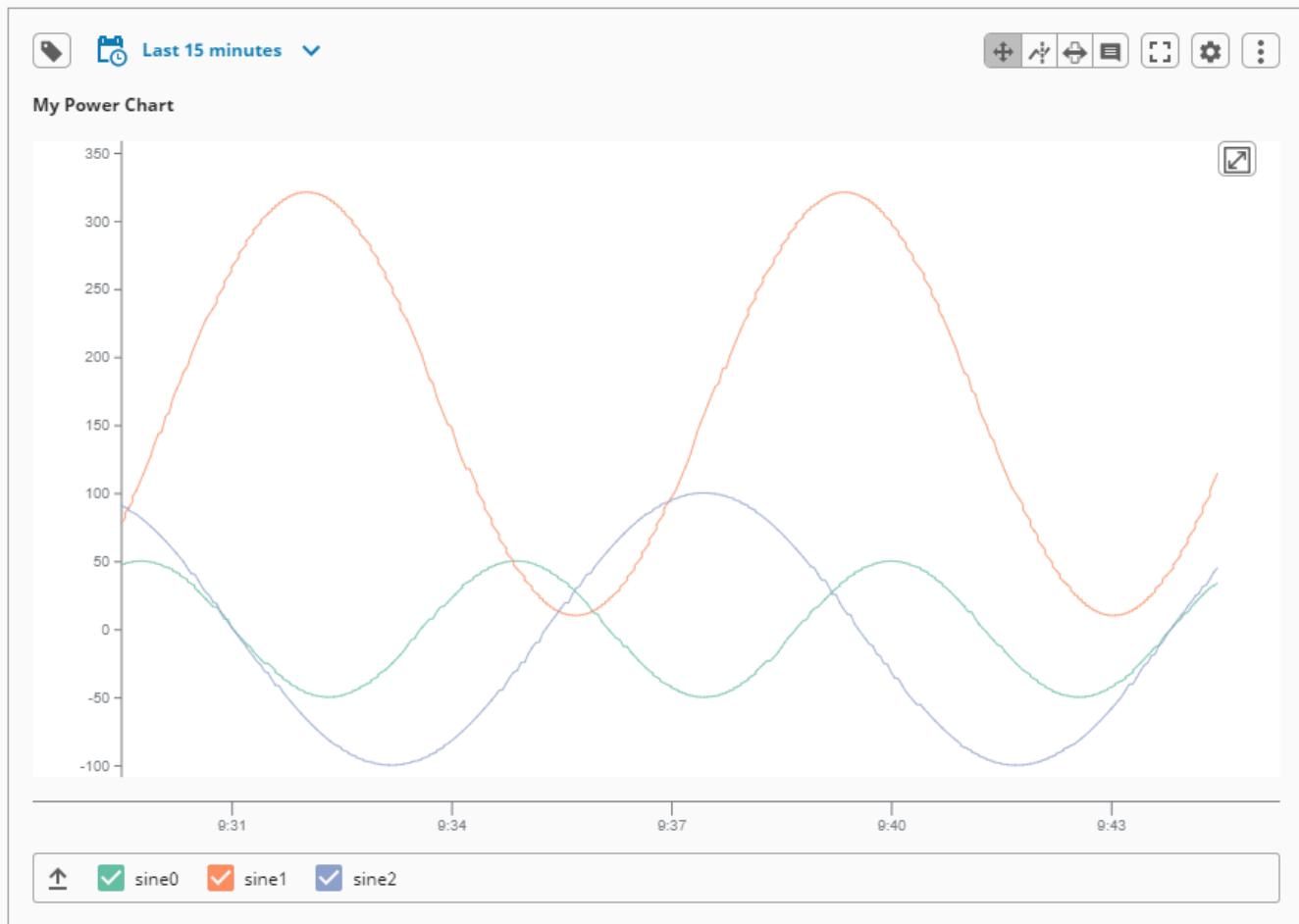


5. You'll notice that the Power Chart will immediately start plotting data and the Tags you added are shown at the bottom of the chart. That's all it takes to create a Power Chart.
The Power Chart has a host of settings to customize how you can display your data: axes, plots, time range selector, pen styles, and much more. To learn more, refer to [Perspective Power Chart](#) page.

Note: The Power Chart component has a responsive breakpoint so it fits better on mobile devices. The default breakpoint is 750px and is configurable. For more information, go to the [Perspective Power Chart](#) page.



6. Save your project.



3. Navigating in a Perspective Session

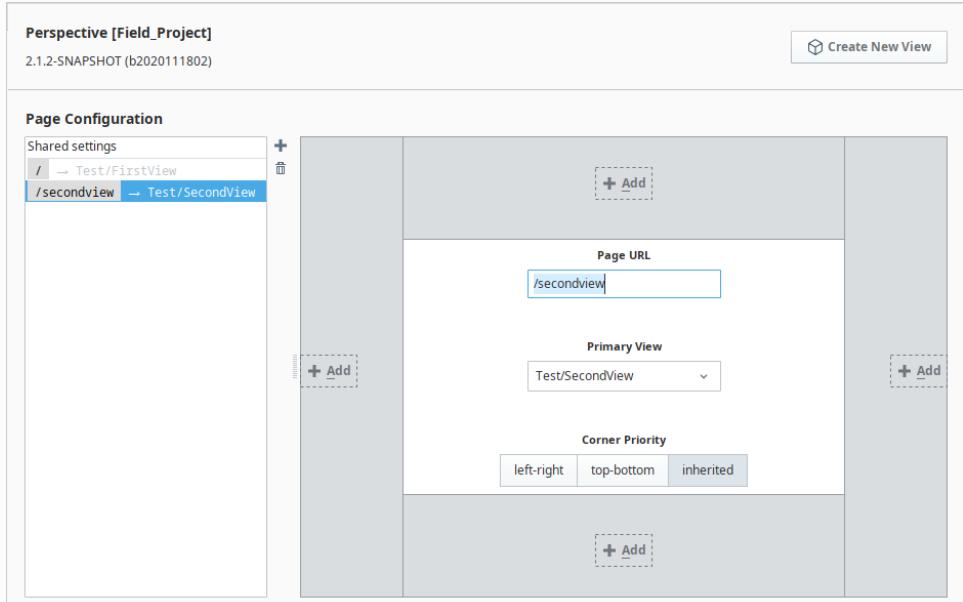
One of the most important aspects when launching a Perspective Session is having a solid navigation design so the user knows where they are, where they've been, and where they are going. How you configure your page settings dictate how the browser navigates and displays each page of your Perspective Session.

A **Page** is the main navigational element in a Perspective Session and represents a collection of views that are displayed in a browser window. When we created the two views in the [Creating Views and Adding Component](#) section above, we also configured the Page URL associated with each new view. You have the option of configuring the Page URL in a separate step if you don't want to do it at the time you create your new view. This is addressed in [Pages in Perspective](#).

1. Open the **Page Configuration** by clicking the Tool icon () tab at the bottom left corner of your Designer workspace. You'll notice that the two views (i.e., FirstView and SecondView) also have their Page URLs created. This was done at the same time we created each view.

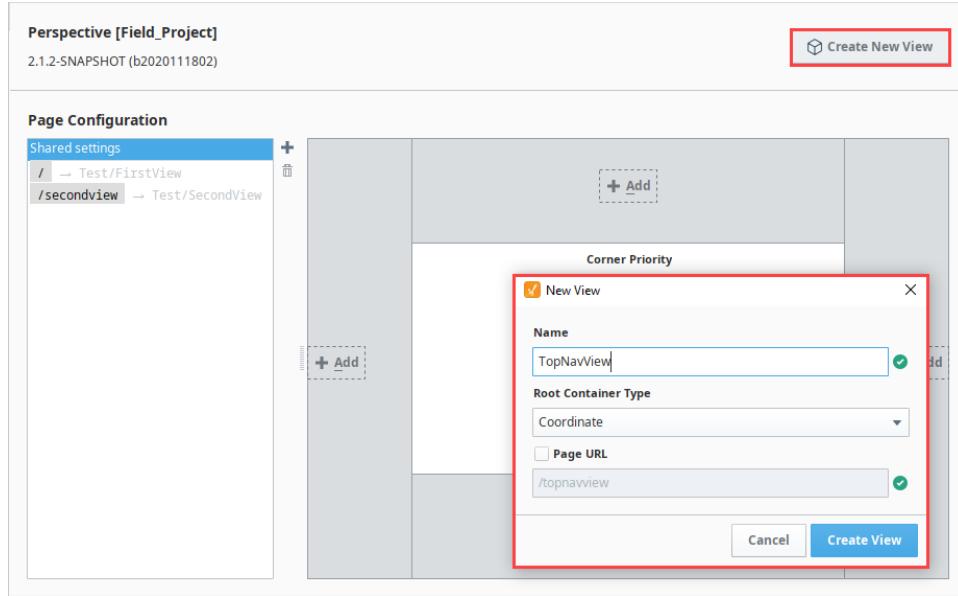
 **First View mounted to the root**

You'll notice the first View created in a project is automatically mounted to the root. The FirstView defaulted to "/" which denotes a "root" page.

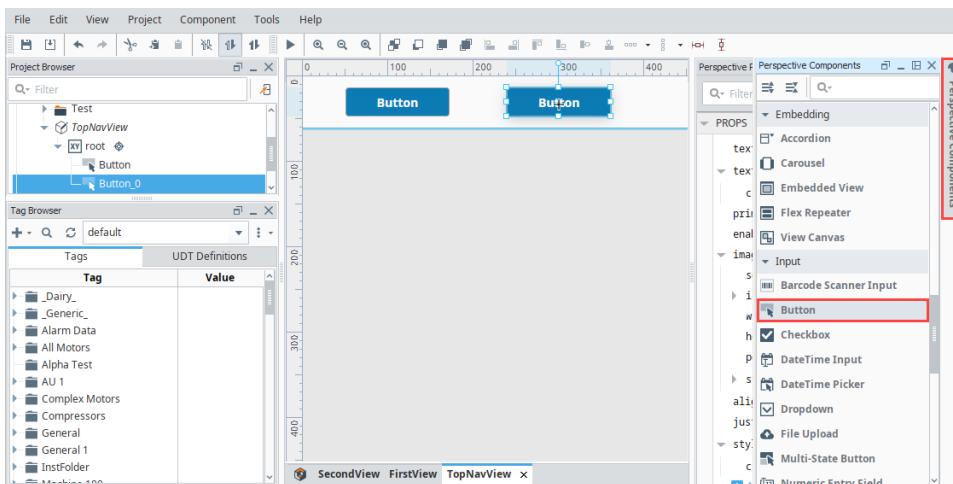


2. Now, let's create a navigation view (i.e., **TopNavigationView**) to be shared by all pages in our project. This page will contain components that will link to different pages allowing users to navigate from one page to another in a Perspective Session.

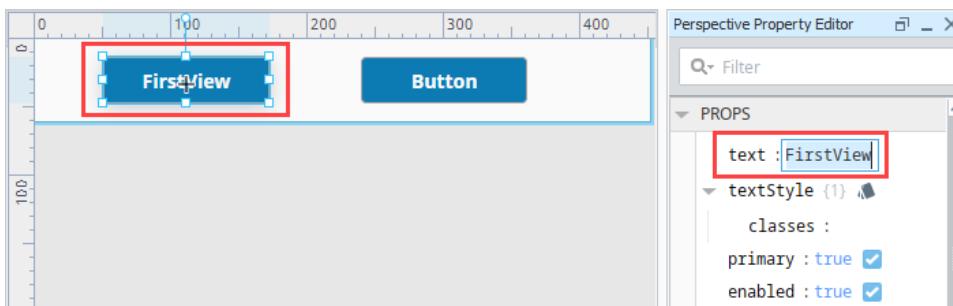
- a. At the top of the Page Configuration, click on **Create New View** (i.e., **TopNavigationView**).
- b. Use the default **Root Container Type** (i.e., **Coordinate**).
- c. *Do not check the **Page URL** checkbox to create the URL page. This isn't necessary because it is a shared setting.*
- d. Click **Create View**.



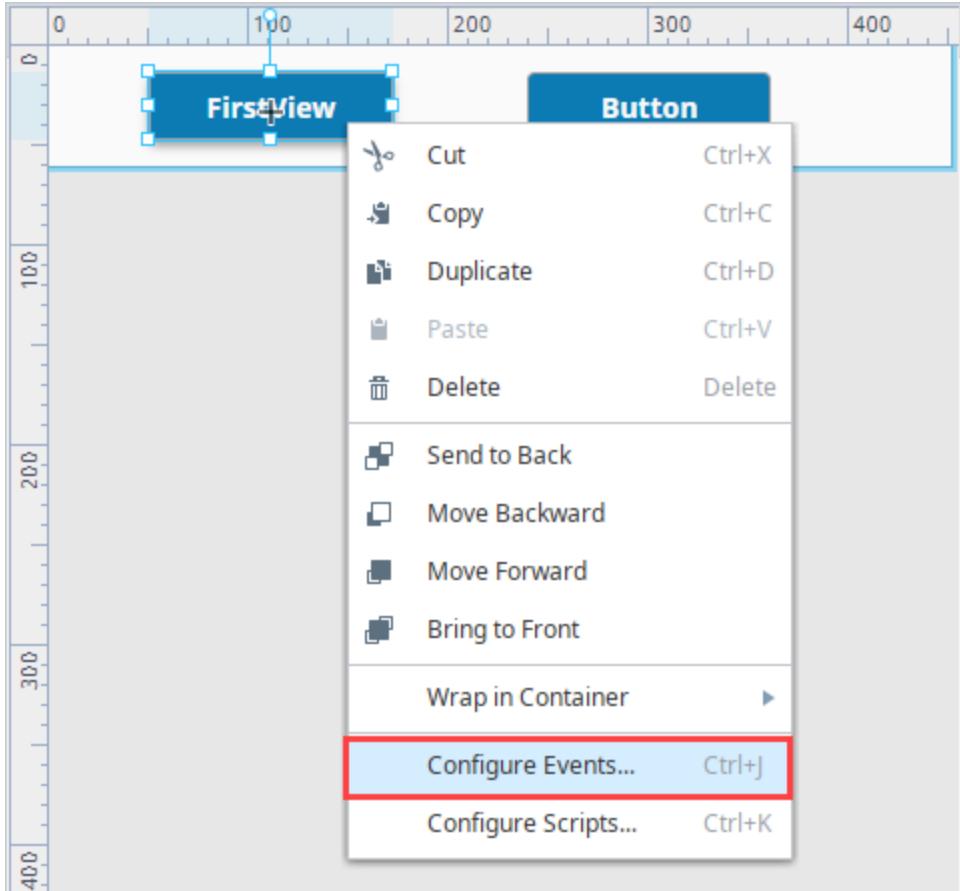
2. Go to the **Project Browser** and select your new view (i.e., **TopNavigationView**). This will show the handles for resizing the view. Resize the root container so the view looks like a banner or bar at the top of the workspace.
3. Add two Button components to navigate between pages. You can drag two **Button** components from the Perspective Component Palette or drag one in and duplicate (Ctrl D) the first Button.



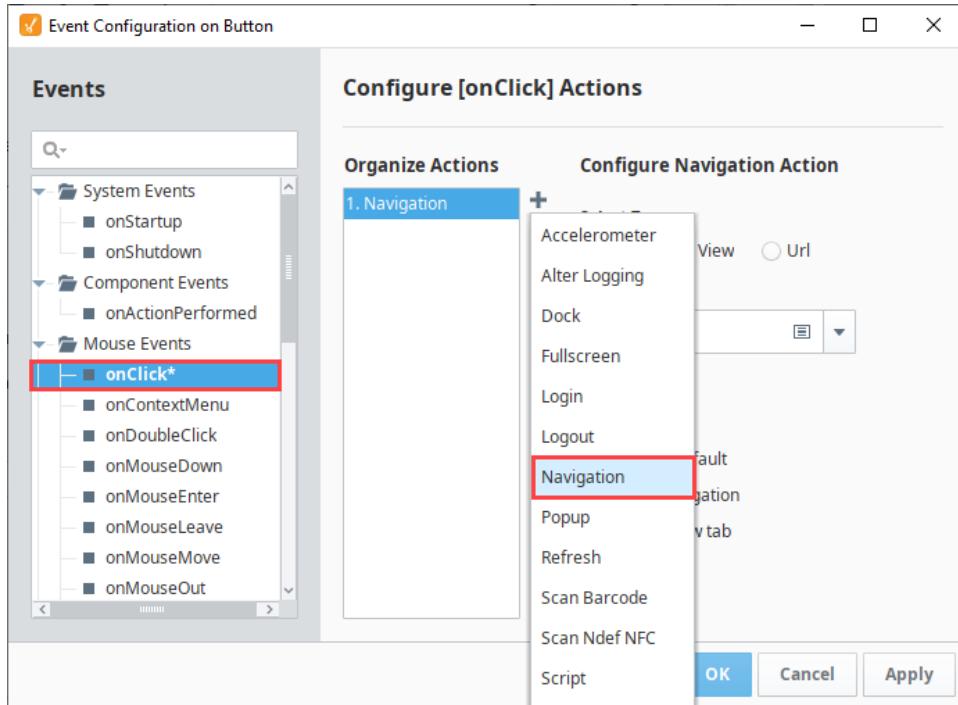
4. Assign each Button component unique text. Select the first Button component, and change the 'text' property to 'FirstView' in the **Perspective Property Editor**.



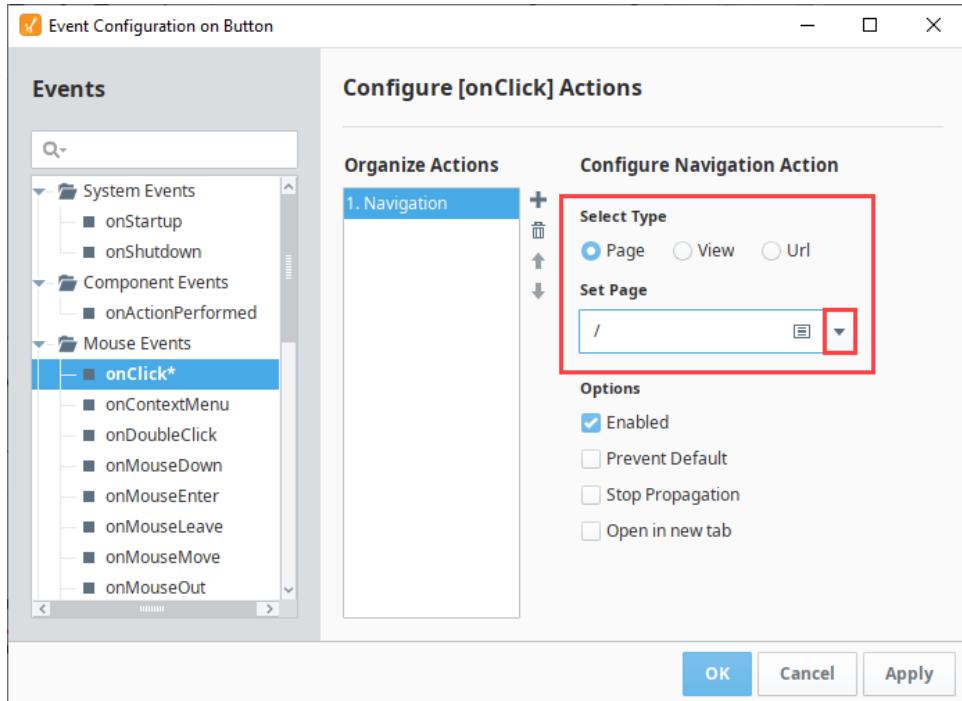
With the **FirstView** button still selected, right click and choose **Configure Events...** (You can also click **Components** on the top menubar and choose **Configure Events...**).



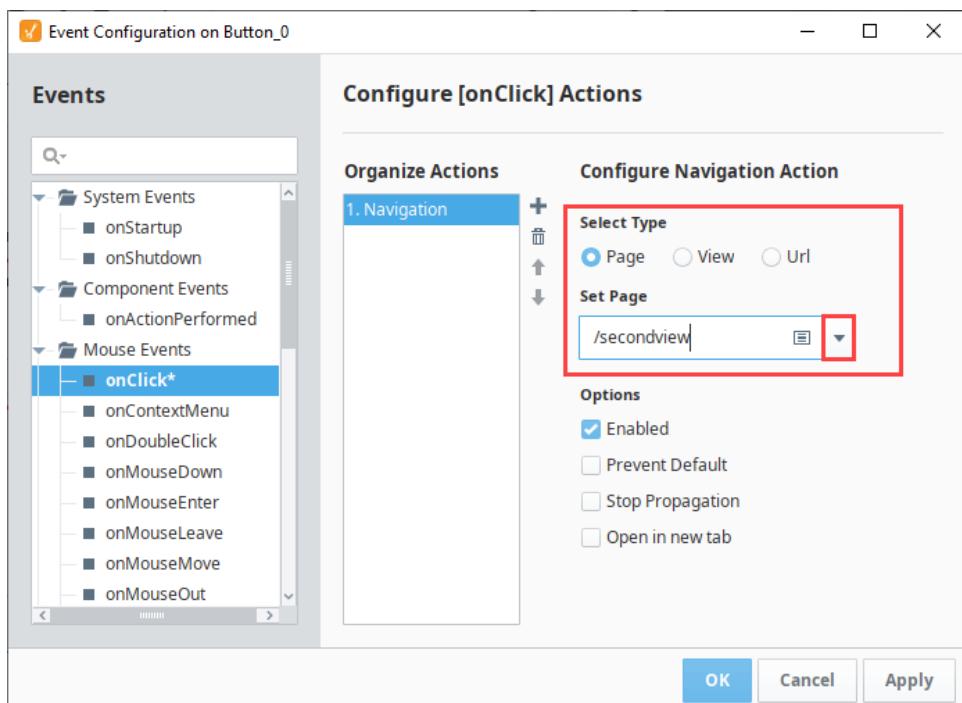
5. This opens the **Event Configuration on Button** window.
 - a. **Expand Mouse Events**, and select **onClick**.
 - b. Under **Organize Actions**, click the (+) icon and select **Navigation** from the dropdown.



- c. Under **Navigation Mode**, click **Page**, and select the "/" from the dropdown which is the **Page Url** for the **FirstView**.
- d. Click **OK**.



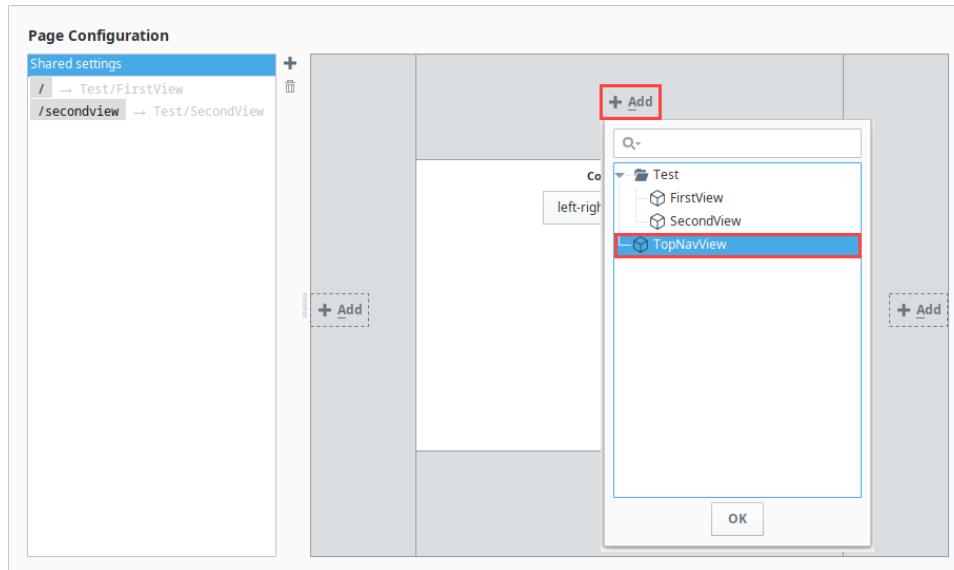
6. Now let's repeat Steps 3 and 4 for the second Button component:
 - a. Select your **second** Button component, and change the 'text' property to '**SecondView**' in the **Perspective Property Editor**.
 - b. With the **SecondView** button still selected, right click and choose **Configure Events...**
7. The **Event Configuration on Button_0** window will open.
 - a. **Expand Mouse Events**, and select **onClick**.
 - b. **Under Organize Actions**, click the (+) icon and select **Navigation** from the dropdown.
 - c. Under **Navigation Mode**, click **Page**, and select the "**/secondview**" from the dropdown which is the **Page Url** for the **SecondView**
 - d. Click **OK**.



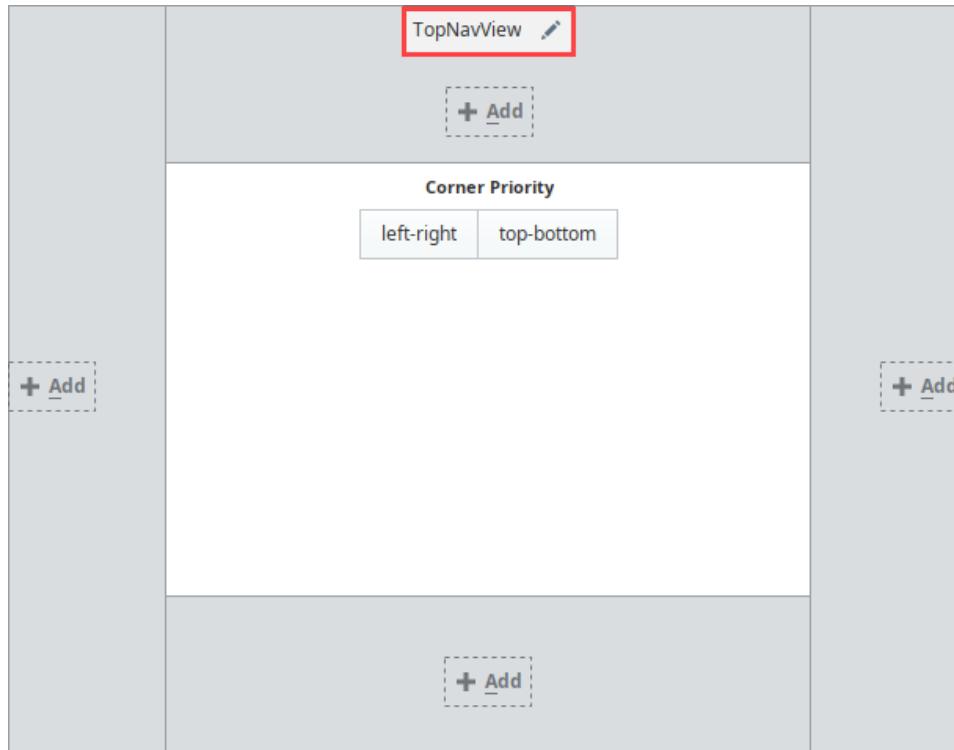
8. Lastly, set the navigation view (i.e., **TopNavigationView**) as a docked view so it appears on the top of every page in your project.
 - a. Click on the Tool icon (gear icon) at the bottom of your Designer workspace to open the **Page Configuration**.

- b. Select **Shared Settings** under Page Configuration, click on the **+ Add** button at the **top** of the Page Layout. Select your navigation view (i.e., **TopNavView**) from the dropdown, and press **OK**.

i If the dropdown is empty or you are missing views, the **pages** (not views) were never created. Go back to Step 2 in Navigating in a Perspective Session and configure your pages. Once your pages are configured, you will see them in the dropdown list.



Your navigation view (i.e., **TopNavView**) will appear at the top of your page layout.



9. You can edit any of the docked view properties by clicking on the name of view in the page layout. Edit any of the properties and click **OK**. Regardless of your **TopNavView's** height, the **Size** property here will determine the height of the view in your session.

Page Configuration

Shared settings
/ → Test/FirstView
/secondview → Test/SecondView

Configure Docked View

View
TopNavigationView

Display Resizable? Modal?
 visible false false

Content Anchor
 push fixed

Size Auto Breakpoint
 150 480

Dock ID Handle
 hide

Handle Icon

View Parameters
 Add Object Member...

Remove **OK** **Cancel**

10. When you're finished setting up your Page Configuration, **Save** your project. Now the **TopNavigationView** will be shared on all the pages of your Perspective Session.

The screenshot shows the SIMATIC Manager interface with the following components:

- Project Browser:** Shows a tree structure with **Views**, **Test**, and **TopNavigationView**. **TopNavigationView** is selected and highlighted in blue.
- Tag Browser:** Shows a list of tags under **default** category, including **_Dairy_**, **_Generic_**, **Alarm Data**, **All Motors**, **Alpha Test**, **AU 1**, **Complex Motors**, **Compressors**, **General**, and **General_1**.
- Perspective Property Editor:** Shows properties for the selected **TopNavigationView**.
 - PROPS** section:
 - defaultSize {2}**: width: 451, height: 60
 - dropConfig {2}**: **udts [0]**, **dataTypes [0]**
 - loading {1}**: mode: non-bl...
 - CUSTOM**: **Add Custom Property...**
 - PARAMS**: **Add View Parameter...**

4. Launching a Perspective Session

Once your project is created and pages are configured, you're ready to launch a Perspective Session.

There are two ways to launch a Perspective Session:

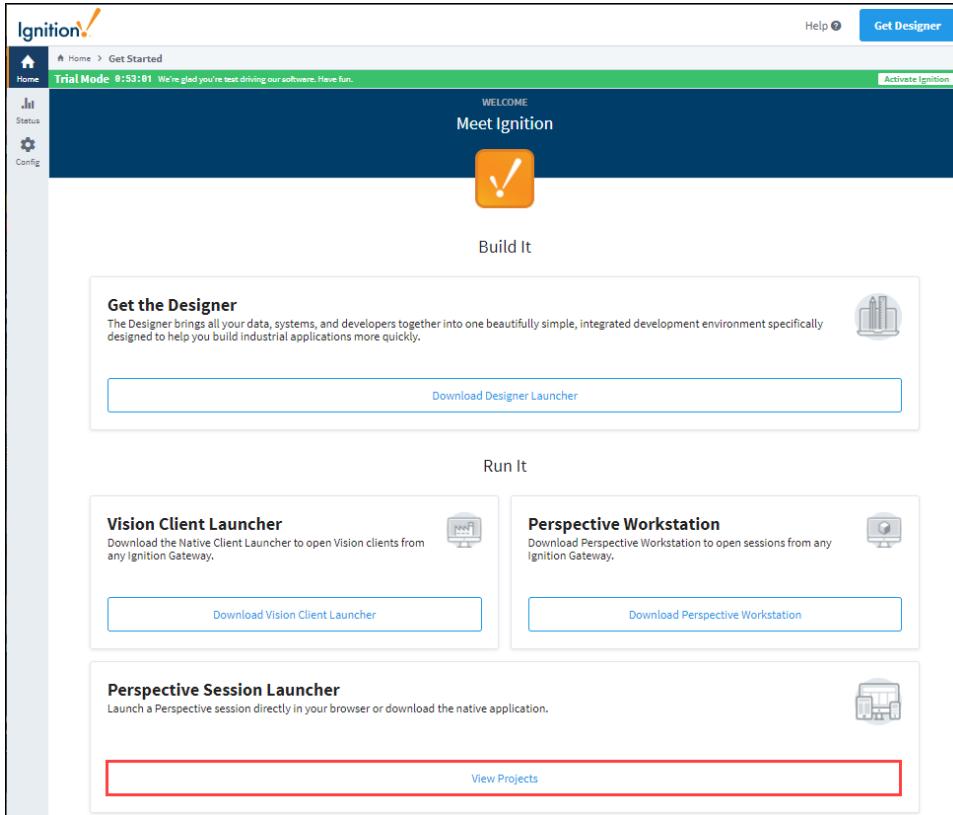
1. From the **Home** tab on the **Gateway Webpage**, click the **View Projects** button below the **Perspective Session Launcher**. All the projects in your Gateway will be displayed. Click the **Launch Project** button for a project.

- Manually enter the web address of the project in your browser.
- From the **Designer** on the menubar, click **Tools > Launch Perspective > Launch Session**. This method is great if you are both a designer and a user.

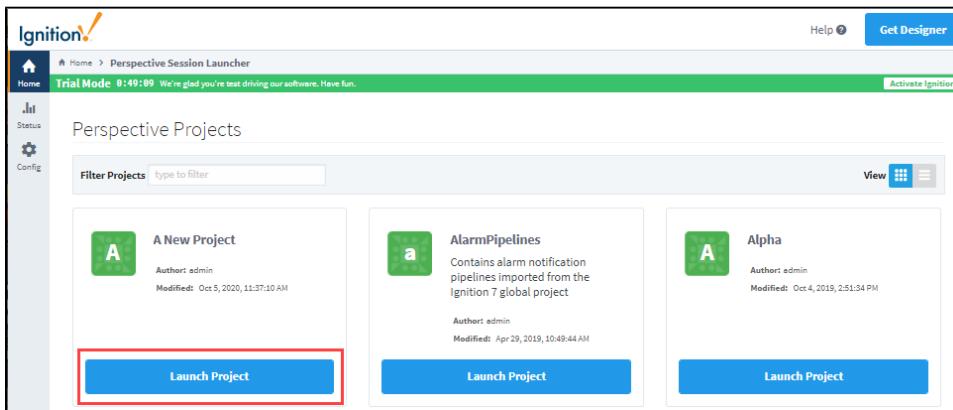
See [Launching a Perspective Session](#) for more information.

Gateway Webpage

- Open a new browser window and enter the **IP address** of the Ignition project Gateway. You can also use '**localhost:8088**'.
- On the Gateway Webpage you will see the **Home** page. Under the Perspective Session Launcher, click **View Projects**.

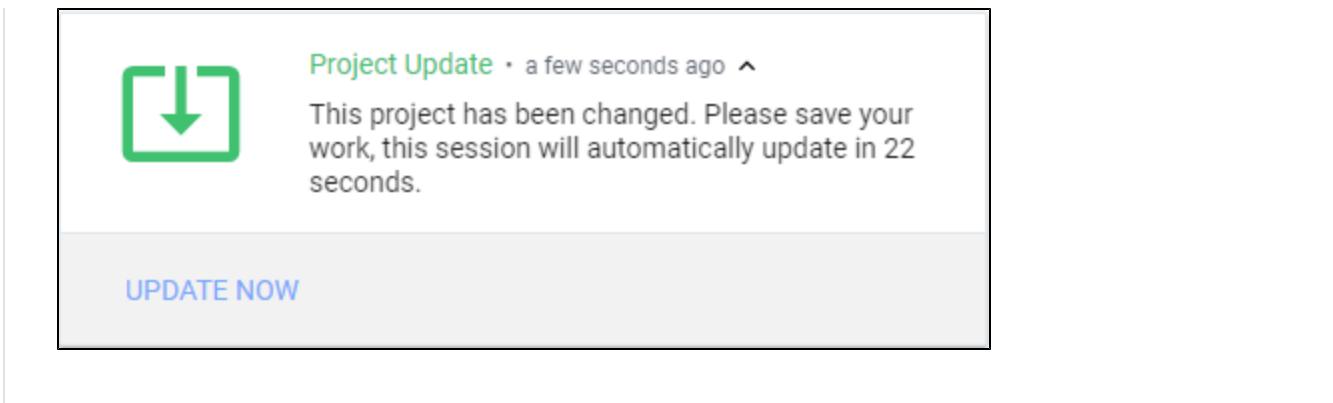


- The window will refresh with a list of Perspective projects. Click the **Launch Project** button to launch a project in a Perspective Session.

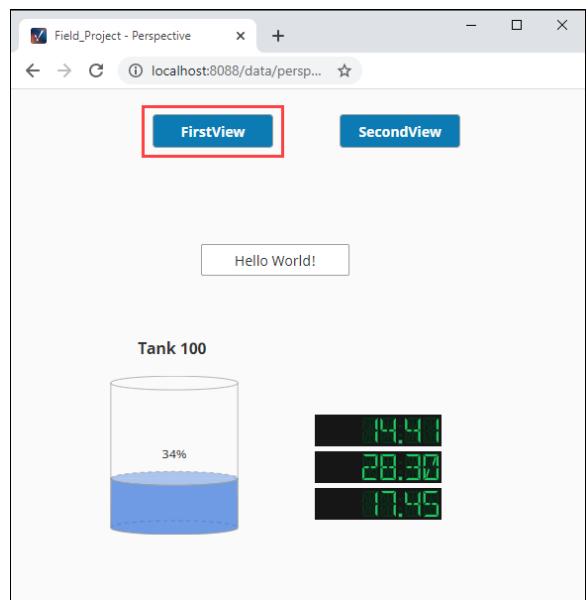
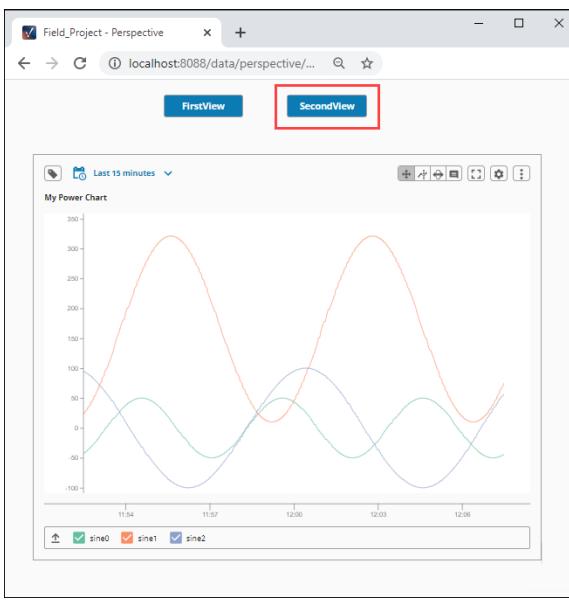


Project Updates

If you have a Perspective Session open and a change was made in the Designer that was saved, an Update Notification window will appear in your session if this option is enabled in Project Properties of the Designer. Your session will automatically update in 30 seconds or you can click **Update Now**. If the Update Notification is not enabled, your session will update immediately.



Here is what your Perspective session will look like in your browser window. Toggle between the **FirstView** and **SecondView** to switch between pages in your session. The FirstView displays the components and the SecondView displays the Power Chart.

| FirstView | SecondView |
|---|---|
|  A screenshot of a browser window titled "Field_Project - Perspective". The address bar shows "localhost:8088/data/perspective...". Below the title bar are two buttons: "FirstView" (highlighted with a red box) and "SecondView". The main content area contains: <ul style="list-style-type: none">A "Hello World!" button.A "Tank 100" visualization showing a cylinder with a blue liquid level at 34%.Three digital displays showing values: 14.41, 28.30, and 17.45. |  A screenshot of a browser window titled "Field_Project - Perspective". The address bar shows "localhost:8088/data/perspective...". Below the title bar are two buttons: "FirstView" and "SecondView" (highlighted with a red box). The main content area is a "My Power Chart" showing three sine wave graphs (sine0, sine1, sine2) over time from 11:54 to 12:08. The chart has a y-axis ranging from -100 to 300. |

Related Topics ...

- [Perspective Designer Interface](#)
- [Perspective Sessions](#)
- [Navigation Strategies in Perspective](#)
- [Containers in Perspective](#)
- [Pages in Perspective](#)

Startup - Vision Client

Overview

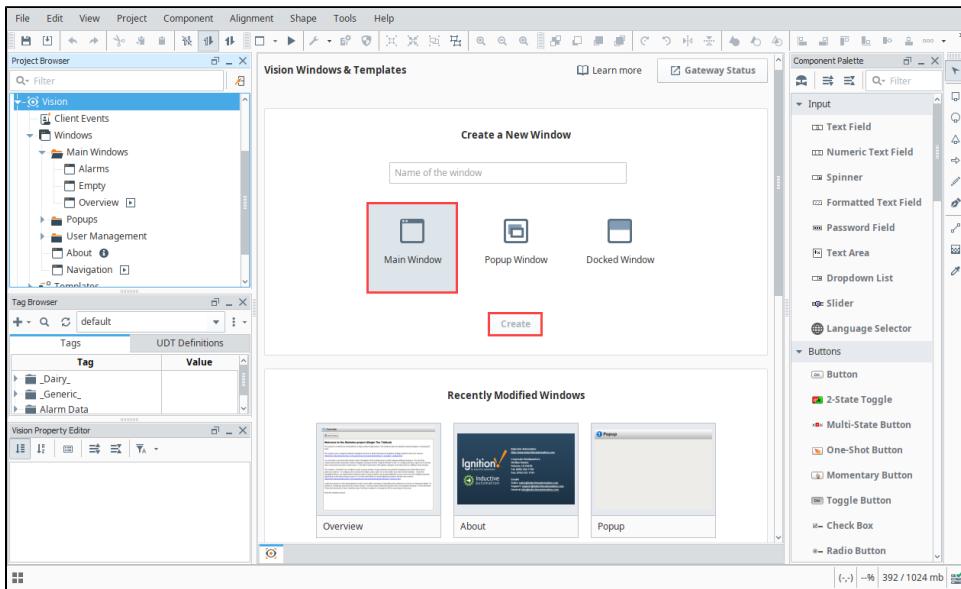
Vision Windows are the basic building blocks for all your Vision projects. Each window can contain any number of components that can display Tag values, run scripts, write values to the database, display Tag History data on an Easy Chart, and more. When you publish your project, these windows are loaded into the Vision Client where any number of windows can be opened at one time.

On this page, we'll walk you through how to add components to a Vision Window and bring them to life by showing live values, displaying Tag History data on an Easy Chart, and launching a project in a Vision Client.

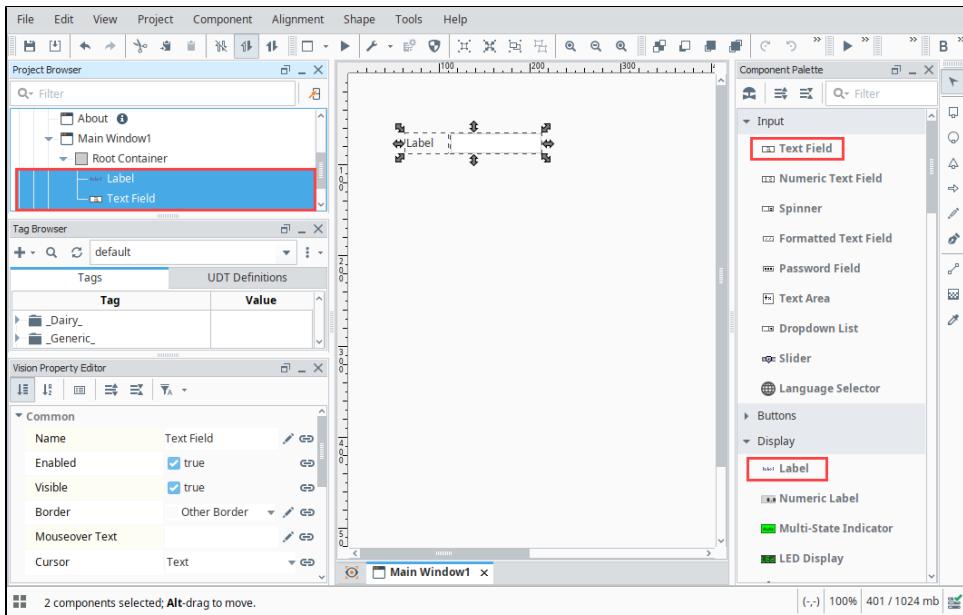
1. Adding Components to Vision Windows

Windows are the [primary unit of design](#) for Vision Clients and can be displayed as a whole page, as a popup, or docked across the top, bottom or sides of a Client. Components can do everything from displaying text, to showing Tag values, to displaying reports, and more. You can bind components to each other, to Tags, to values in the database, or any combination. Let's drag a few components into your project windows and show some live values.

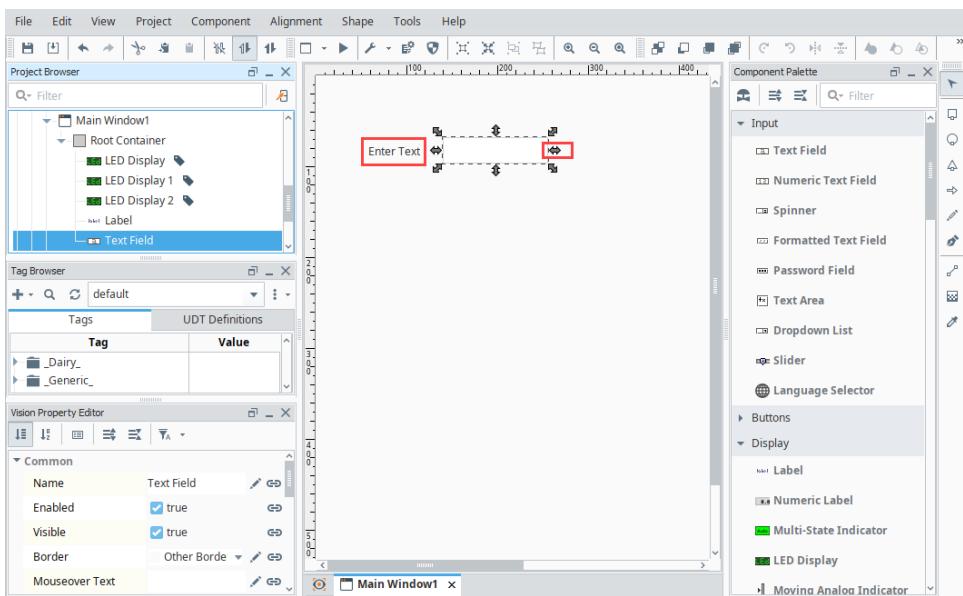
1. From the Vision Designer, you can either go to the **Project Browser** and right-click on **Windows** to create a **Main Window**, or click on the **Main Window** image in the Designer workspace, enter a window **Name** (i.e., Main Window1) and click **Create**. You have the option to create folders for organizing your windows. This first window you create will automatically open when a client starts, but that can be changed. We will talk more about that later.



2. The window will refresh and you can begin clicking and dragging components from the **Component Palette** on the right onto the Designer workspace. Add a **Label** and a **Text Field** component.



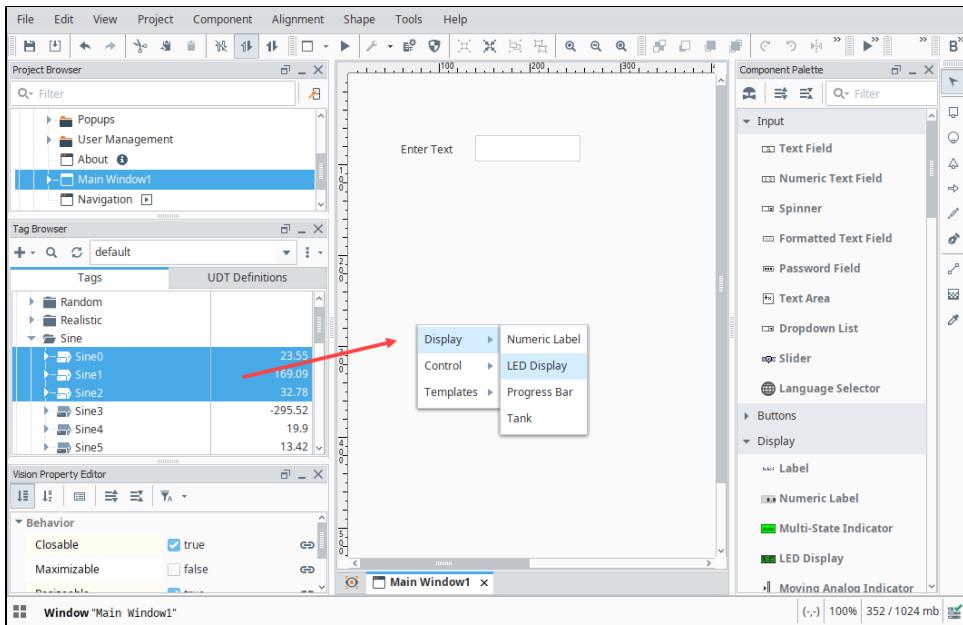
3. Double-click on the **Label** component to type directly into it, or go to the **Text** property of the Label in the **Property Editor**, and rename it to say "Enter Text."
4. Select the **Text Field** component and click-and-drag the arrow handles to resize the component.



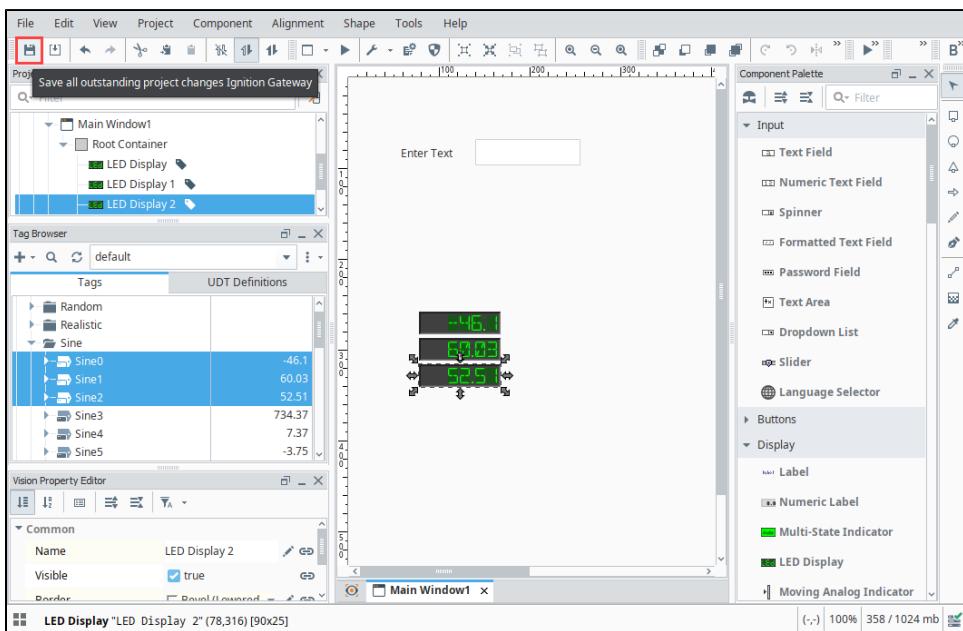
5. Now let's add some Tag values to the window.

Note: This step assumes you are already connected to Generic simulator. Refer to [Connecting to a Device](#).

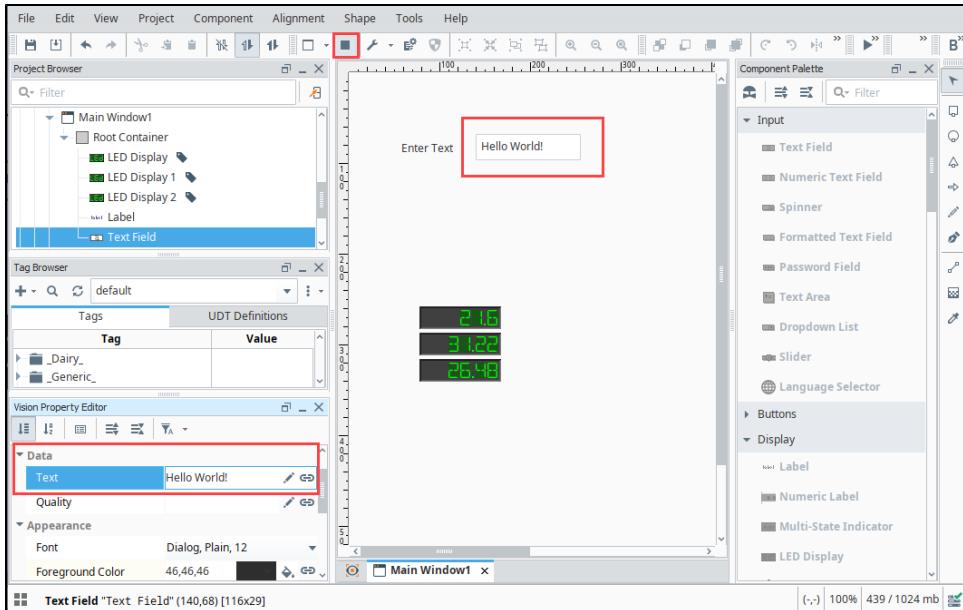
The simplest way to add Tag values is to go into the **Tag Browser** and drill down to the Tag(s) you want and drag them onto the window. Select a few Tags (i.e., **Sine Tags**) and drag them into the window. You will see a popup asking you what type of components you want to use to show these Tag values. Select **LED Display**.



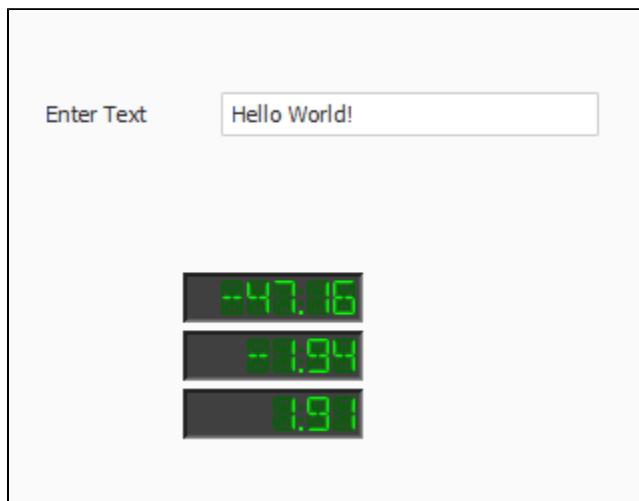
- Now you can see some live values on your window. Don't forget to **Save** your project.



- You can Preview your project in the Designer by clicking the **Preview Mode** button in the top menubar. This allows you to interact with your windows as if you were using a Vision Client. The Preview Mode button changes between a gray triangle and a gray square depending on which mode you are currently in. Look for the gray triangle and click on it to go into Preview Mode. In this example, we entered "Hello World!" in the Text Field. You can see Property Editor, "Hello World!" also appears in the Text property of the Text Field.



Learn more about [Working with Vision Components](#) and [Vision Property Bindings](#).

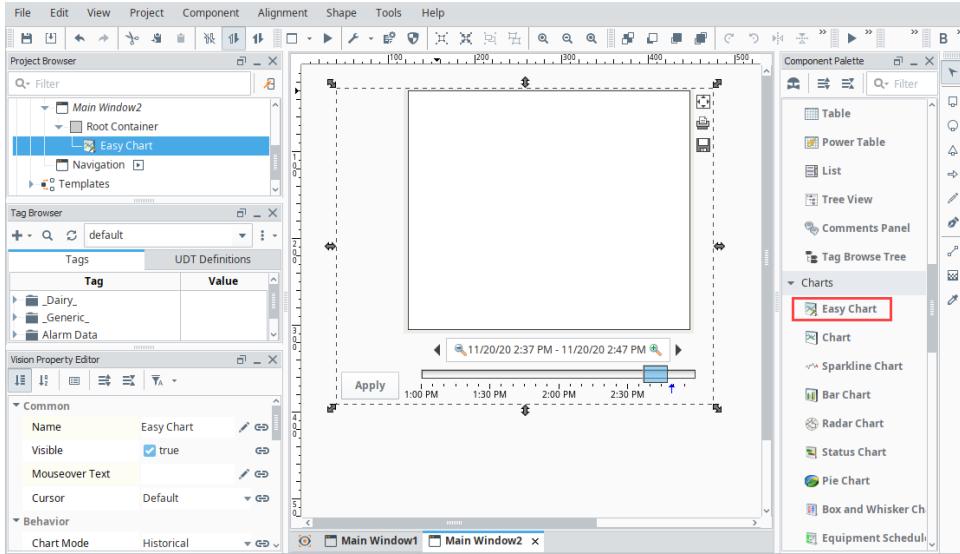


2. Showing Tag History in an Easy Chart

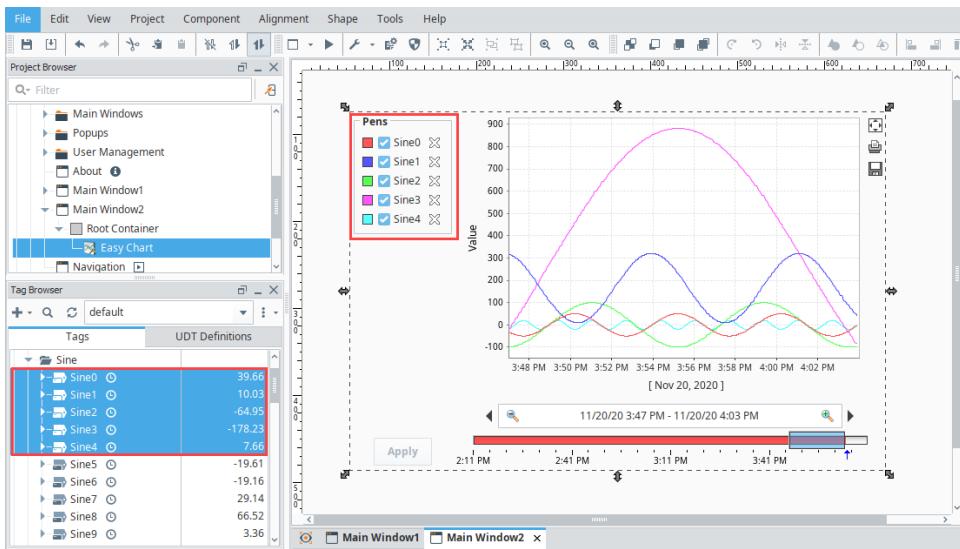
Next we'll display Tag History from our Tags using an [Easy Chart](#). Showing Tag History requires that you have Tag History already setup (See [step 7](#) on the previous [Startup Guide](#) page). Refer to the [Configuring Tag History](#) page.

In this example, create a second Main Window (i.e., **Main Window2**), add an Easy Chart component to show history on the values of the LED Displays.

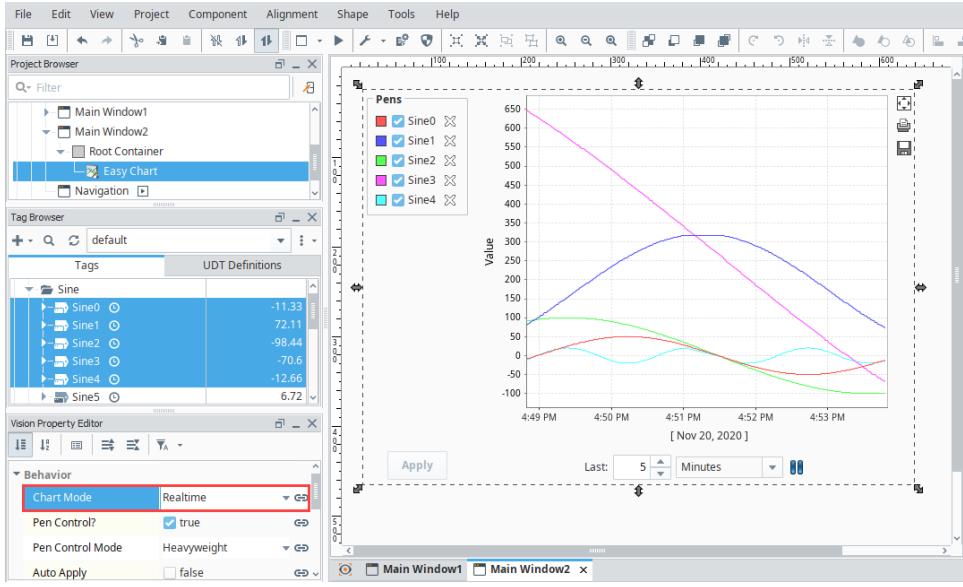
1. Once you have your second Main Window created, drag an **Easy Chart** component on to your window (i.e., **Main Window2**).



2. In the **Tag Browser**, the first five Sine Tags are logging data. Drag these Sine Tags to the Easy Chart. The Easy Chart will immediately fetch the results and trend the history. When you first drag Tags to the Easy Chart, not much data will be displayed because you just started logging data.

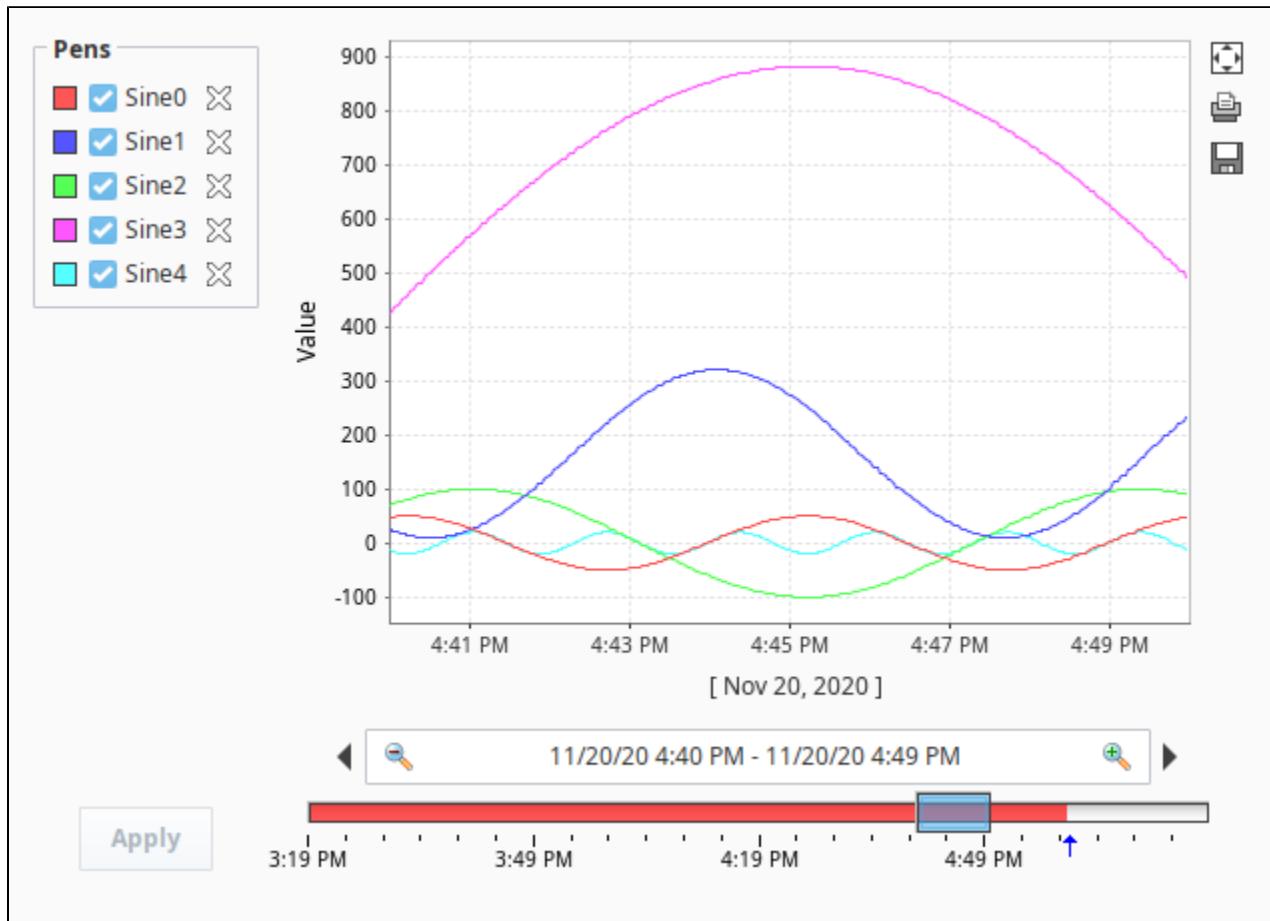


3. To show more data, set the **Chart Mode** property from Historical to **Realtime** in the **Property Editor**.



- From the top menubar, save your project by clicking **File > Save**.

Learn more about the [Vision Easy Chart](#).

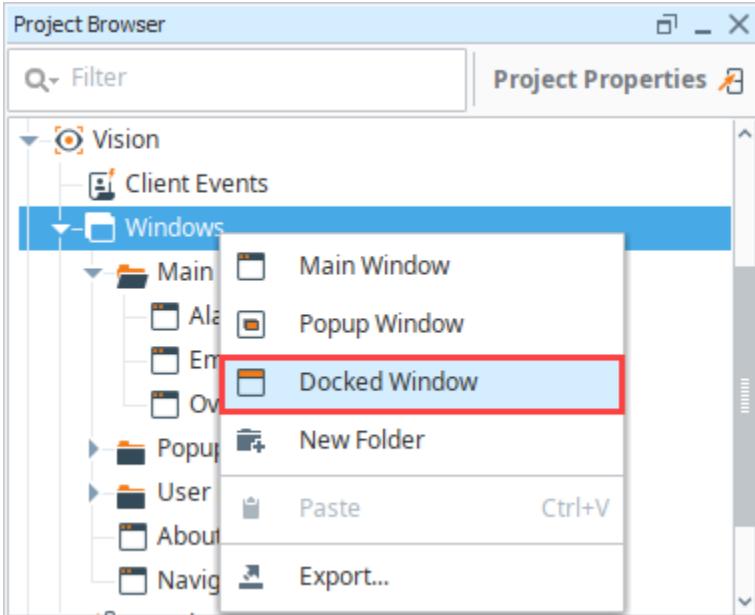


- Navigating in a Vision Client

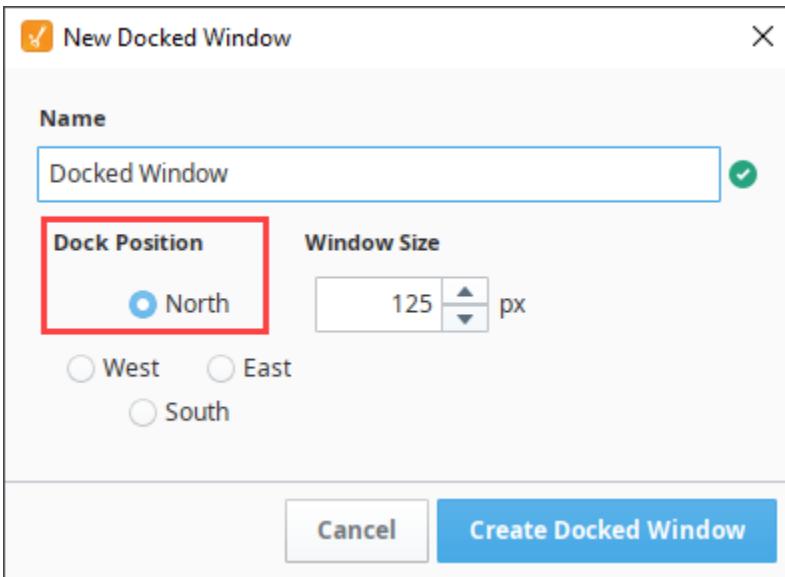
It's important to have a navigation strategy that allows you to navigate seamlessly between different windows in the Vision Client. You can choose from several types of navigation strategies, such as a [Tab Strip](#), or [Tree View](#). The [Navigation Strategies in Vision](#) page provides some things to consider to help you determine the best navigation strategy for your project and your users.

In this example, we are going to use a simple Docked window and two Button components to switch between the two windows we created in the previous sections.

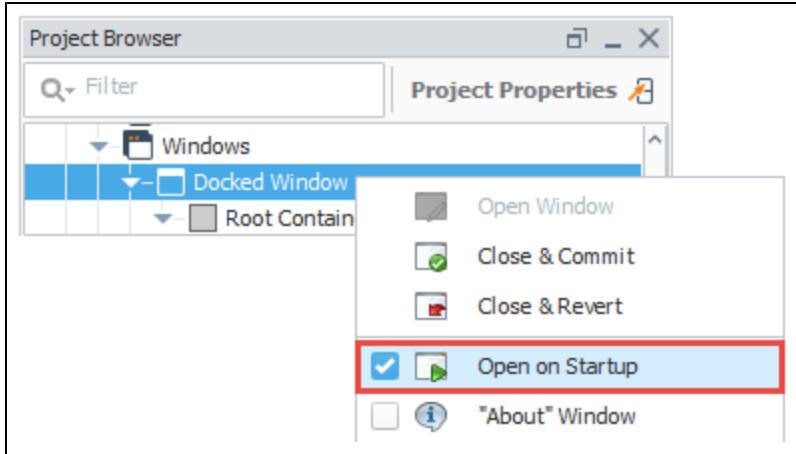
1. First, let's create our docked window. In the **Project Browser**, right click on **Windows** and select **Docked Window**.



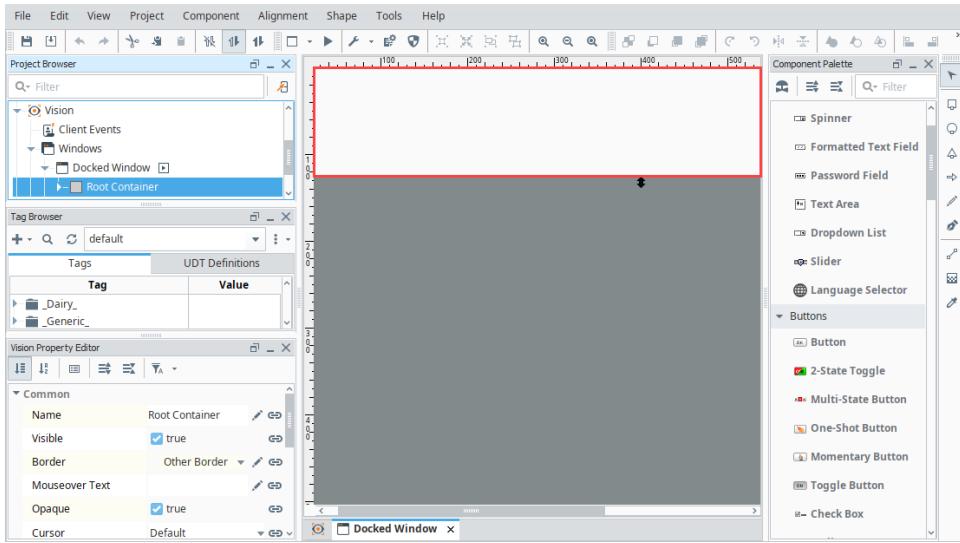
2. A New Dock Window will appear for you to select the **Docked Position**. You can choose to display a Docked Window across the top, sides or bottom of your Vision window. In this example, we named our window **Docked Window** and selected **North**. Click **Create Docked Window**.



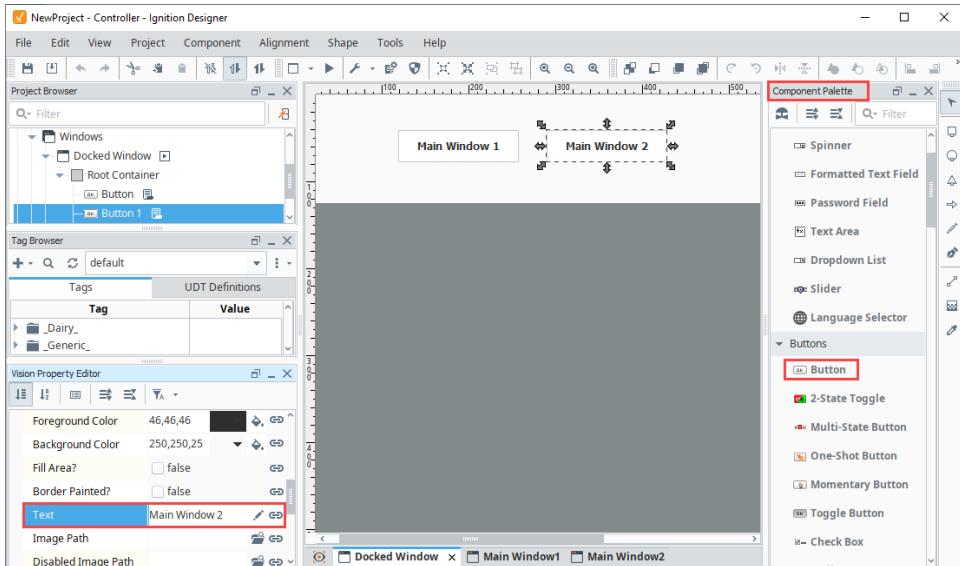
3. Right click on your new Docked Window and check **Open on Startup**. Your Docked window will now open on startup in the client and will be shown above all main windows in the Vision Client. You can do the same to any window to toggle this option on any window. For now, we only want two windows with this set, the first window from above (Main Window 1) and this Docked Window.



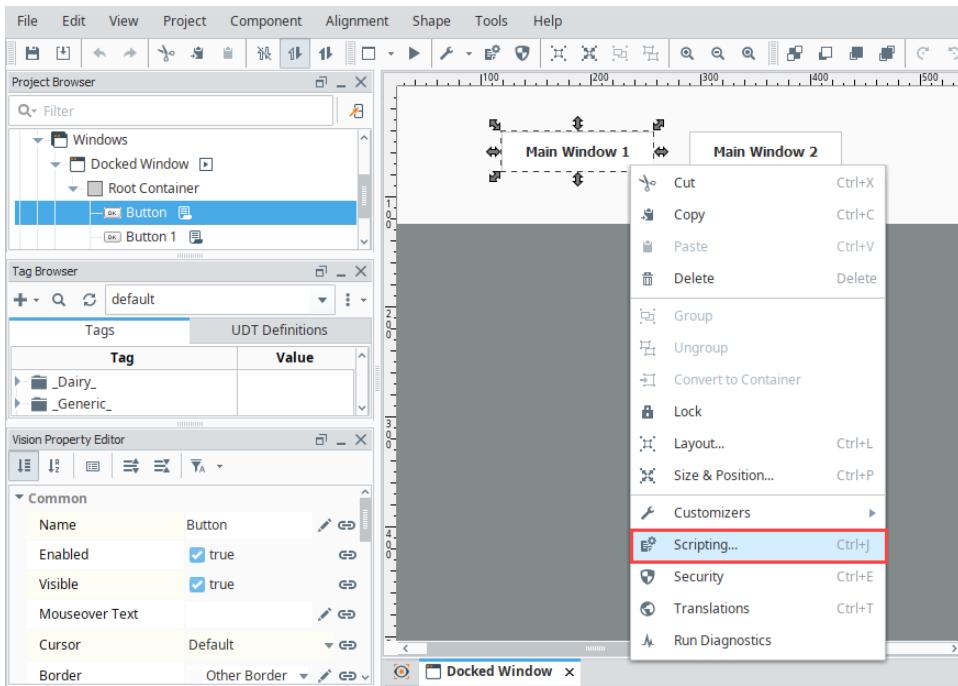
Your Docked Window will look like the following in the Designer.



4. Now, let's drag a couple of **Button** components from the Component Palette to the Docked Window, or you can duplicate (Ctrl D) the first button to create the second button. We'll use these Button components to navigate from one window to another in the Vision Client.
5. Change the **Text** property of the Buttons to something meaningful (i.e., **Main Window 1** and **Main Window 2**).

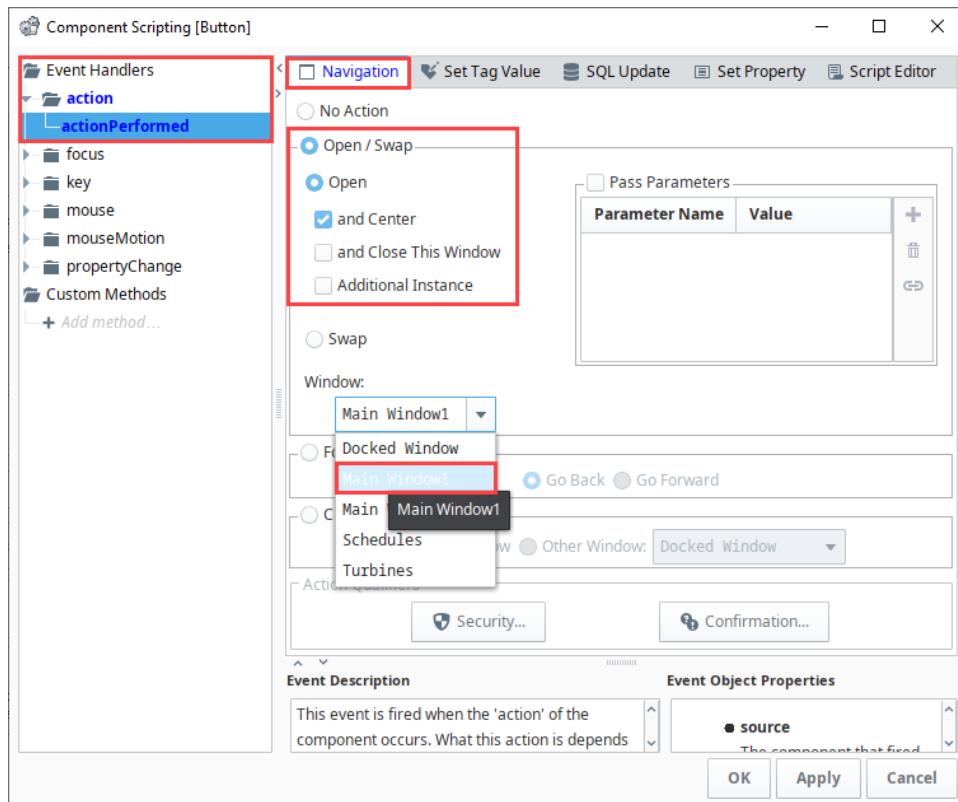


6. Now you need to add scripting to each of your Buttons so they know what window to open when pressed. Right click on your first Button component (i.e., **Main Window 1**) and select **Scripting**.



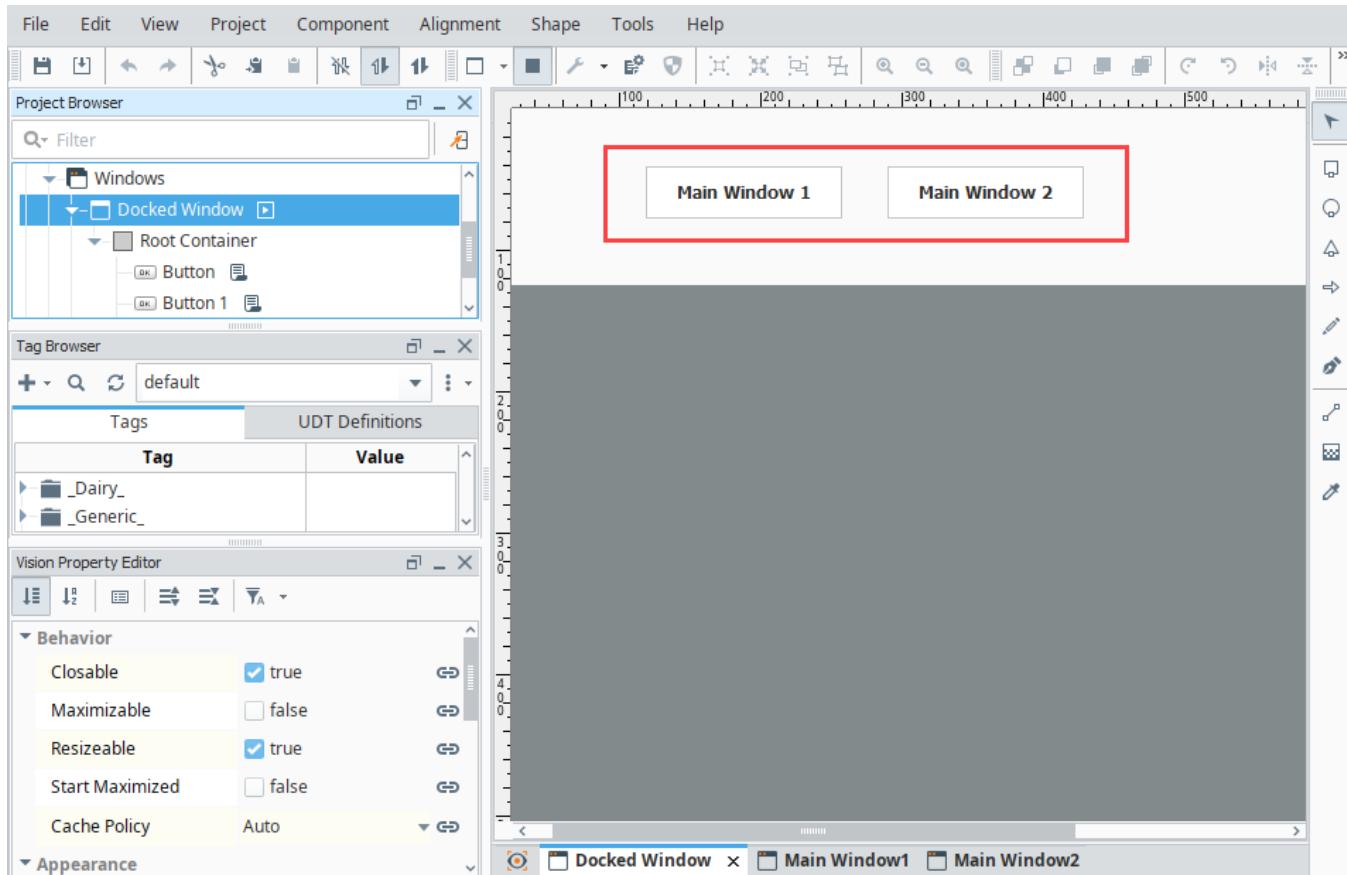
7. This opens the **Component Scripting [Button]** window. Here you have to tell the buttons to swap one window for the other when the Button is pressed. In other words, it closes the current window you have open, and then opens the other window in its place. Use the following steps to apply your script to your first button (i.e., **Main Window 1**):

- Select the **Main Window 1** button.
- Under **Event Handlers**, expand the **action** folder, and select **actionPerformed**.
- Select the **Navigation** tab, select **Open / Swap**, then **Open**, then **and Center**.
- From the **Window** dropdown, select the **Main Window1**.
- Click **OK**.



8. Repeat Step 6 for your second button (i.e., **Main Window 2**):

- a. Select the **Main Window 2** button.
 - b. Under **Event Handlers**, expand the **action** folder, and select **actionPerformed**.
 - c. Select the **Navigation** tab, select **Open / Swap**, then **Open**, then **and Center**.
 - d. From the **Window** dropdown, select the **Main Window2**.
 - e. Click **OK**.
9. The image below shows how your Docked Window will look in the Designer. Remember to **Save** your project.



4. Launching a Vision Client

The Vision Client Launcher option can be installed as an "all user" application on a Windows operating system providing launcher access to all users on the same machine from a single installation. If the Vision Client Launcher was already installed, subsequent install attempts are treated as an upgrade and you will not be prompted to select the install mode. This option will only appear on new launcher installations or after uninstalling a prior version of the installer. You will see this in the example below.

Vision Client Launcher Destination Folder

When installing for "all users," the installer will be placed in the following folder: "%Ignition Installation Directory%\Program Files\Inductive Automation"

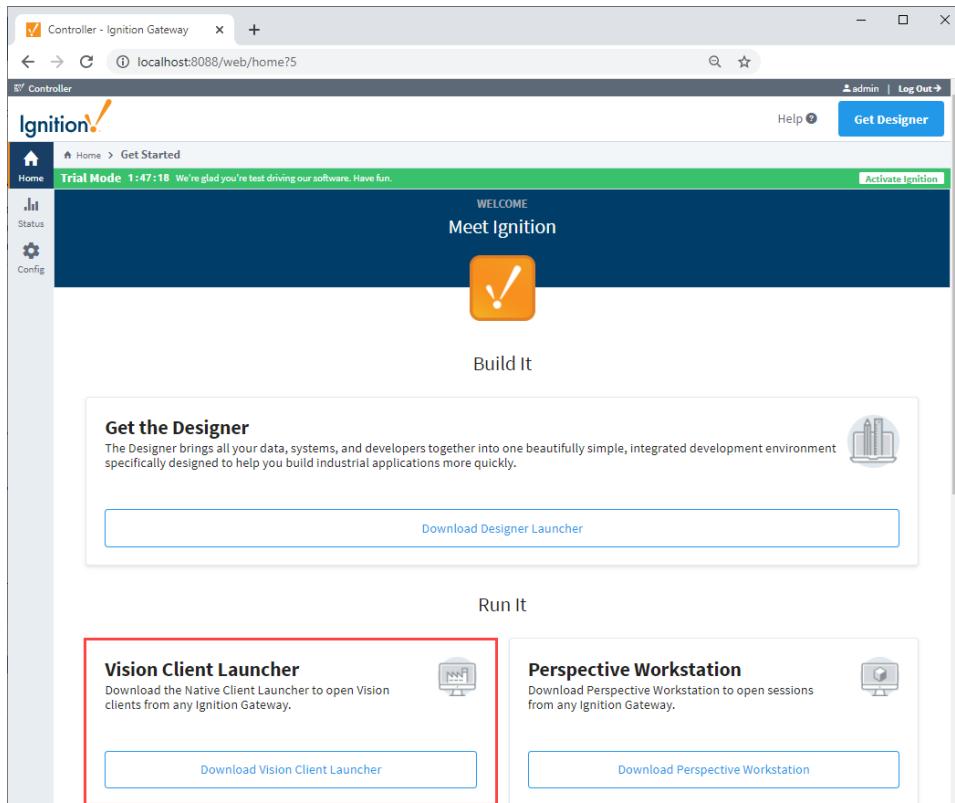
When installing for "me only," the installer will be found under the "\AppData\Roaming\Inductive Automation" folder.

Now that we [created a project](#) with a few windows and some basic navigation, let's launch a Vision Client. There are three ways to launch a Vision Client:

1. From the Gateway webpage under the **Vision Client Launcher** logo, click the **Downloads** button. This method must be used after you create your first project, typically on a new installation of Ignition and a new release of the Vision Client Launcher application.
2. From the **Vision Client Launcher** executable or a desktop shortcut, if you have used them previously.
3. From the **Designer** on the menubar, click **Tools > Launch Project > Launch (windowed or full-screen)**. This method is great if you are both a designer and a user.

Launching a Vision Client from the Gateway Webpage

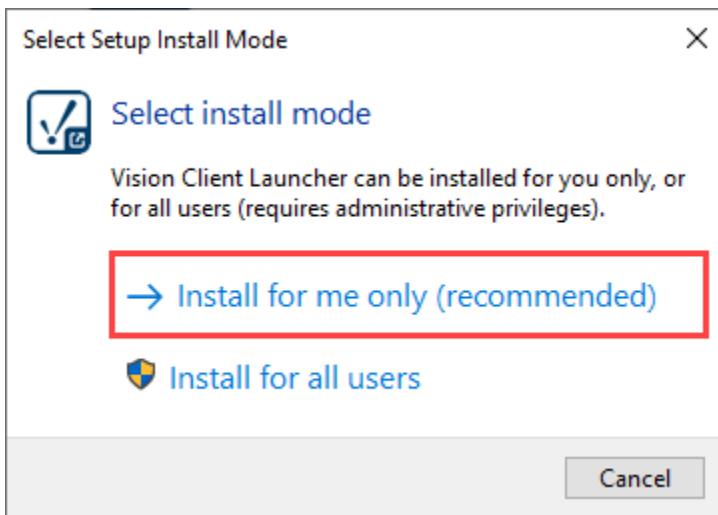
1. Open a new browser window and enter the **IP address** of the project Gateway . You can use '**localhost:8088**'.
2. If you're not already logged into Ignition, enter the credentials for the administrator or user with full privileges for the Gateway. You must have a project created. (The project in this example is called Controller). If this is your first time launching a Vision Client after installing and designing in Ignition, use the Vision Client Launcher on the Gateway webpage.
3. From the **Home** tab on the Gateway webpage, click **Download Vision Client Launcher**.



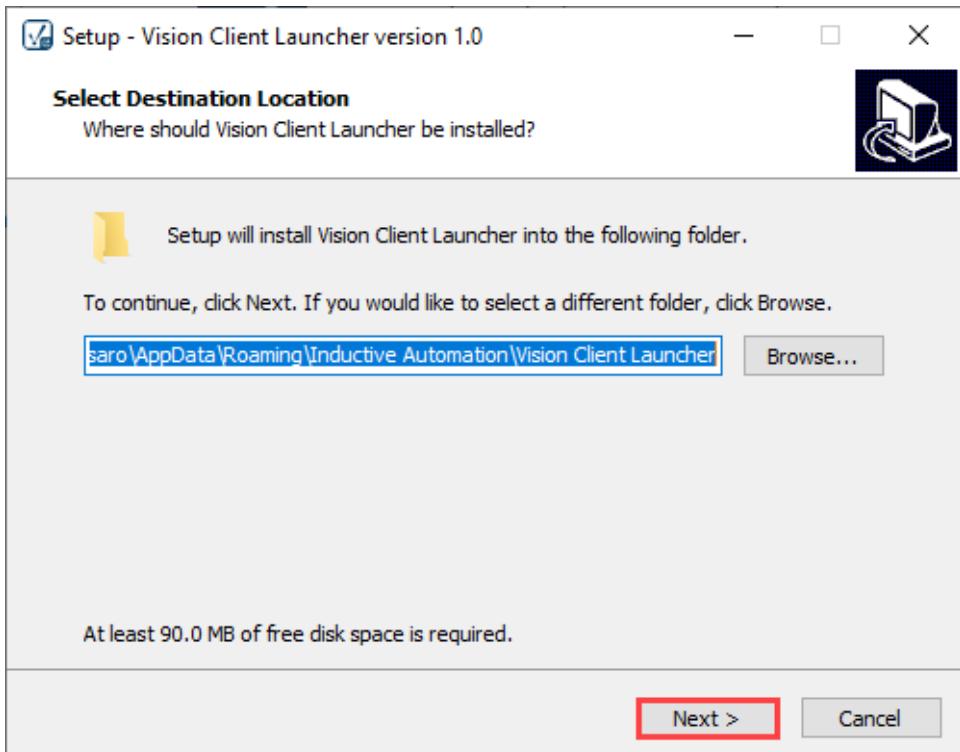
4. Select the **Download** button for the platform you are running on: Windows, Mac OS, or Linux. This example shows the installation steps for a Windows operating system.

The screenshot shows the Ignition Gateway web interface with the URL localhost:8088/web/client-launchers?8. The page title is "Controller - Ignition Gateway". The main content area is titled "Vision Client Launcher Download" and displays a green banner stating "Trial Mode 1:04:12. We're glad you're test driving our software. Have fun." Below this, there's a message: "We've detected you're on Windows. Download the Vision Client Launcher for Windows and follow these steps below to install." A large blue button labeled "Download for Windows" is highlighted with a red box. To the left, there's a sidebar with icons for Home, Status, Config, and Help. To the right, there's a section titled "Alternative Vision Client Launcher" listing download links for Windows (61.3MB), Mac (50.1MB), and Linux (72.0MB).

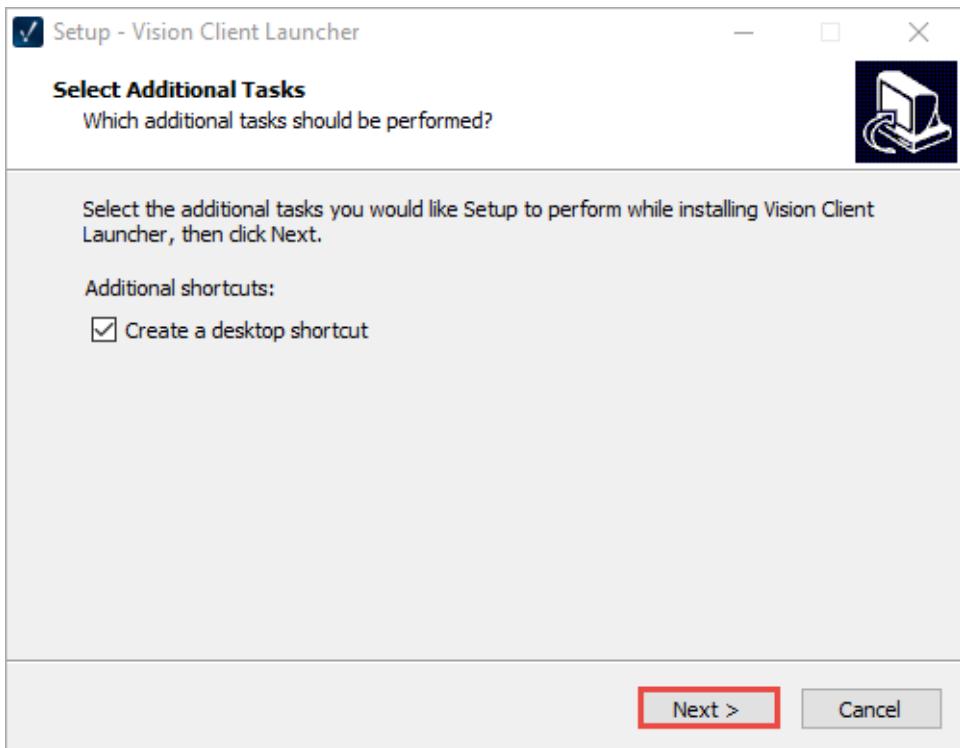
5. Run the downloaded file **VisionClientLauncher.exe** (found on the lower-left of the window if using Chrome), or go to your Downloads folder and double-click the file to run it.
6. If this is the first time installing the Vision Client Launcher, you will be prompted to select an install mode. Select either "**Install for me only**," or "**Install for all users**."



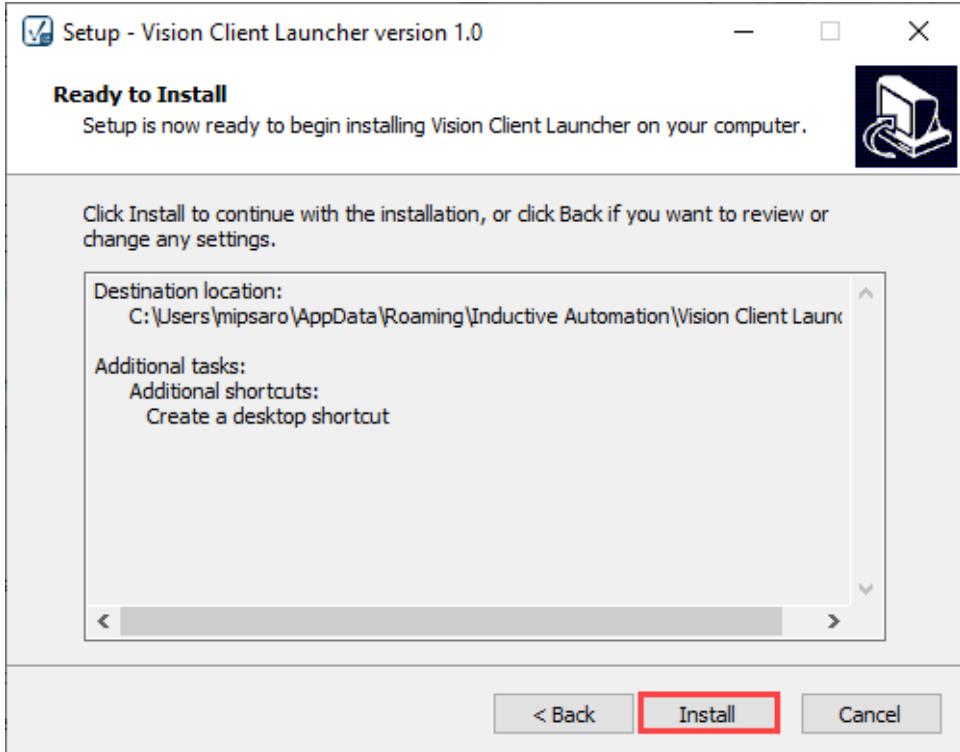
7. The Install Wizard will select a default destination location folder for the Vision Client application files. If you are "installing for me only", the destination folder will be different from the "all users" destination folder. You also have the option to select a different folder. Click **Next**.



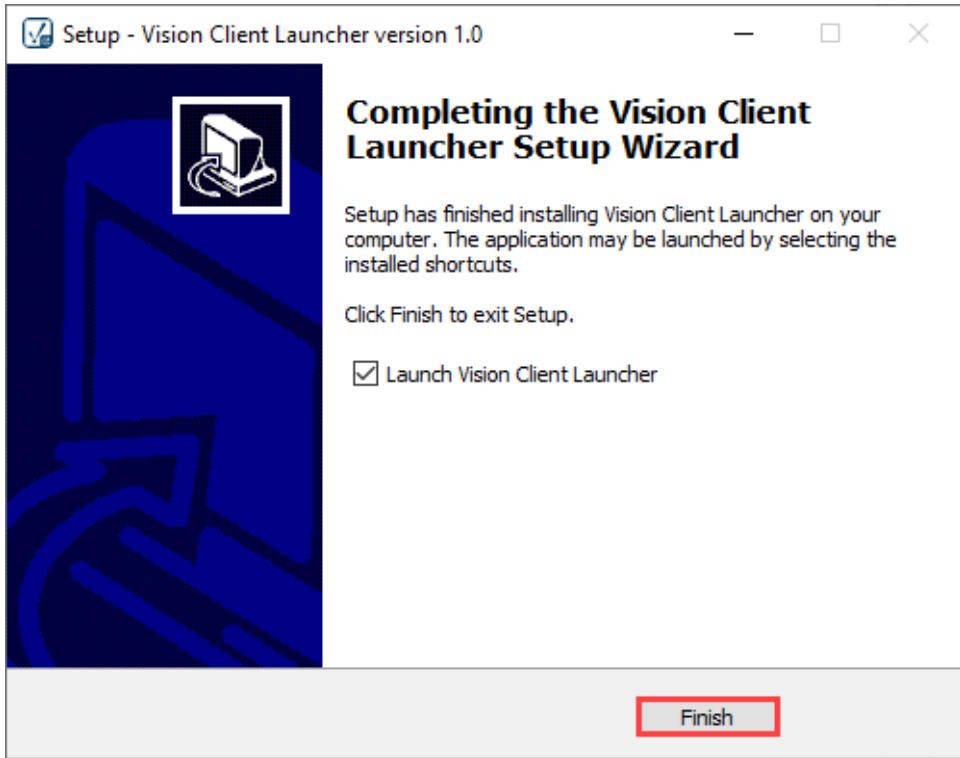
8. The Setup - Vision Client Launcher wizard will open. You'll notice that a desktop shortcut is checked and will be created so you don't have to run the Vision Client Launcher each time you want to open a client. Click **Next**.



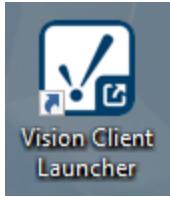
9. The Ready to Install window will open. Click **Install**.



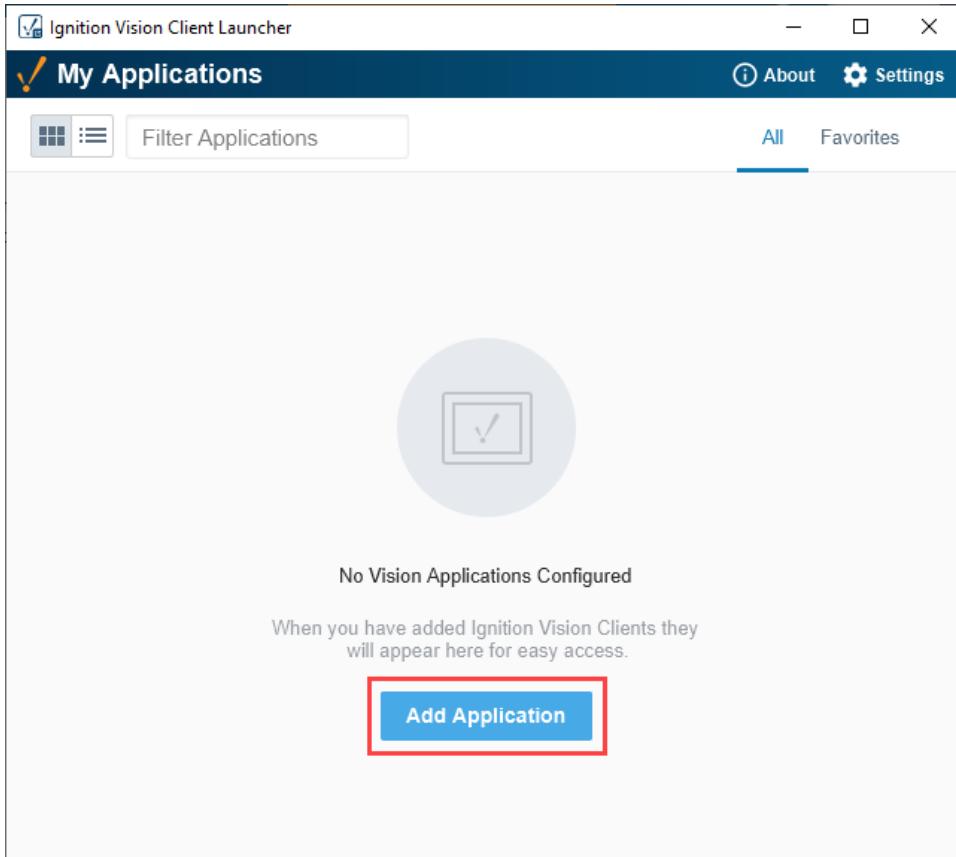
- When the Vision Client Launcher completes the install, click **Finish**.



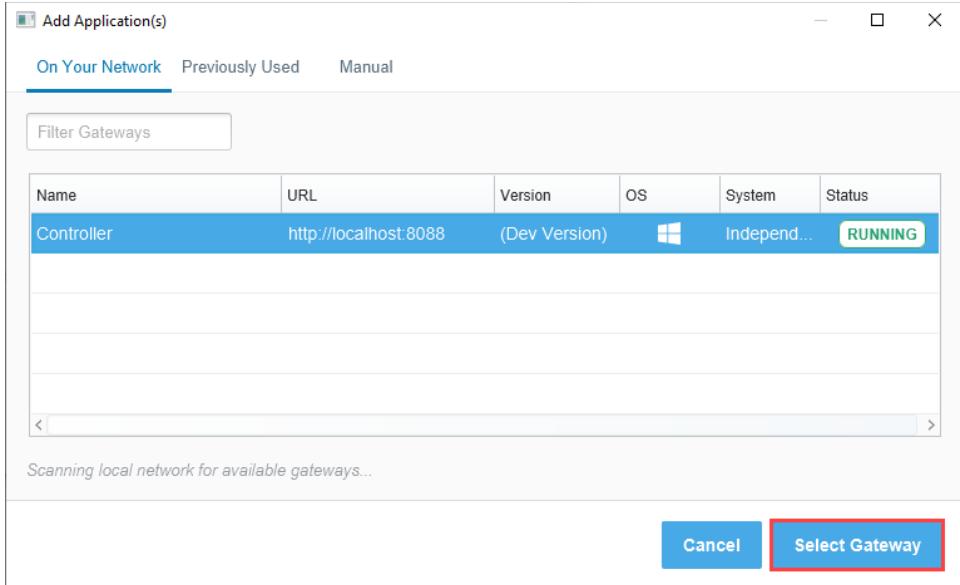
- The Vision Client Launcher shortcut icon will be placed on your desktop and looks like the following.



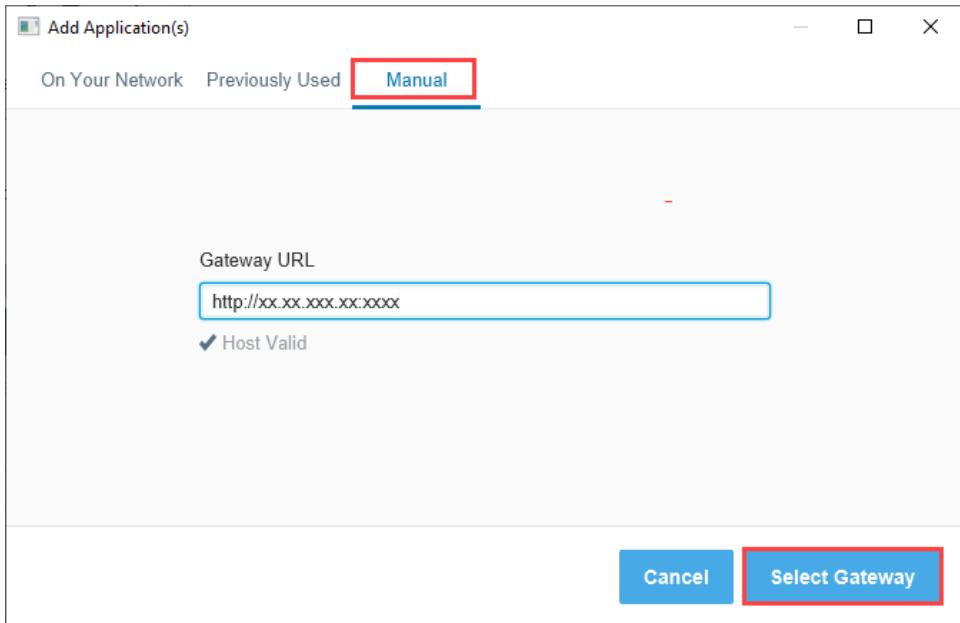
12. The Ignition Vision Client Launcher will open, but you won't have any Vision Clients (applications) available if this is your first time using the Vision Client Launcher. Click **Add Application**. This will take you to the same window to browse for local and remote Gateways.



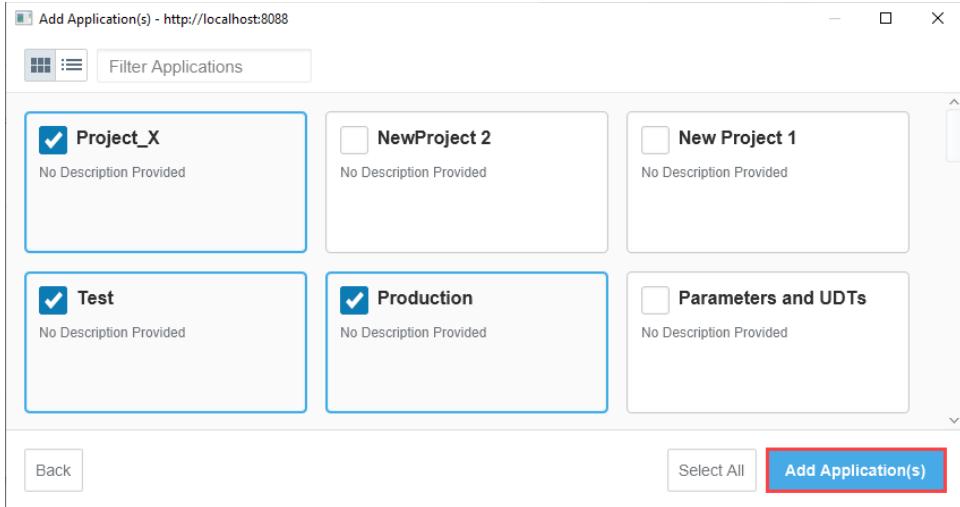
13. The window will refresh and show your local Gateway. It will also check for other Gateways on your network. Select a Gateway and click the **Select Gateway** button.



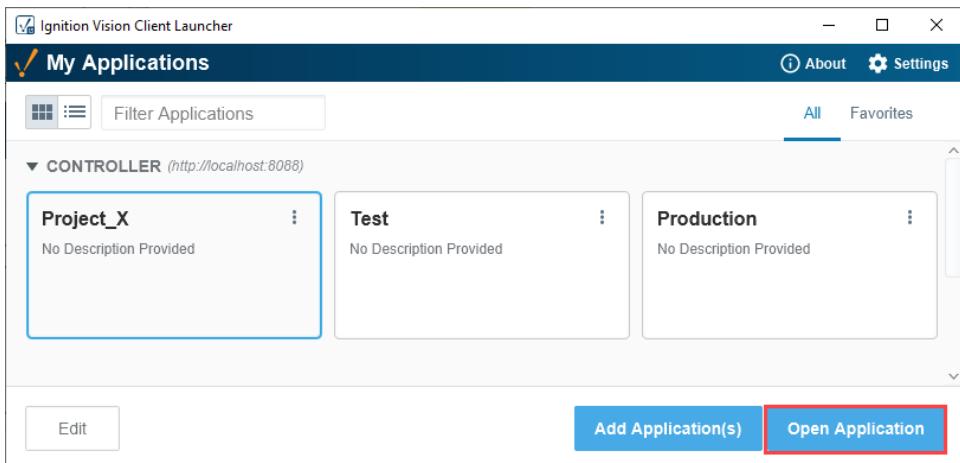
14. If there is an issue automatically detecting your Ignition Gateway, you can manually add your Gateway. Select **Manual** tab at the top of the window. Enter your **Gateway URL or IP address** (i.e., **localhost:8088**), and click **Select Gateway**.



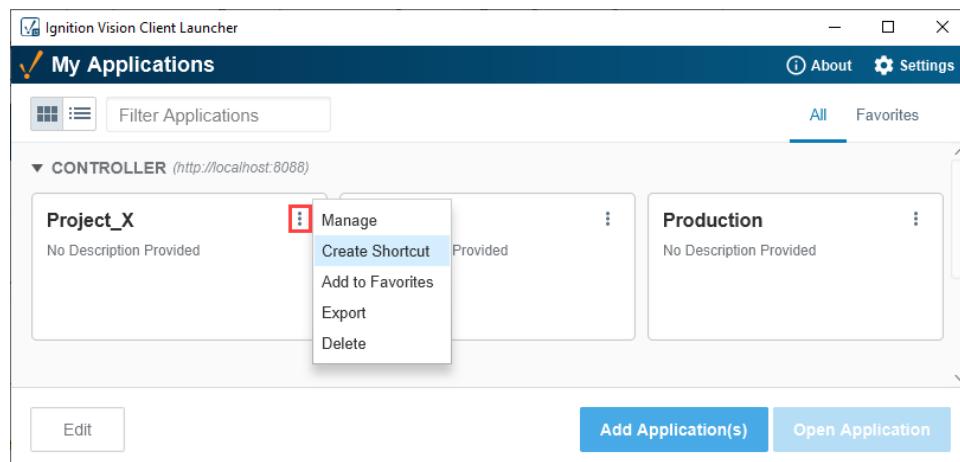
15. Once you add your Gateway, the Vision Client Launcher will automatically find all your Vision projects, but as we mentioned earlier, if this is your first time using the launcher, you won't have any applications available. Select your Vision project(s) and click **Add Application(s)**.



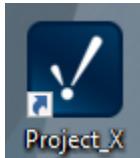
16. The following image is what you will see the next time the Client Launcher is opened. The Vision Client Launcher is installed and opens the same way you open any program. Simply click the application and click **Open Application**.



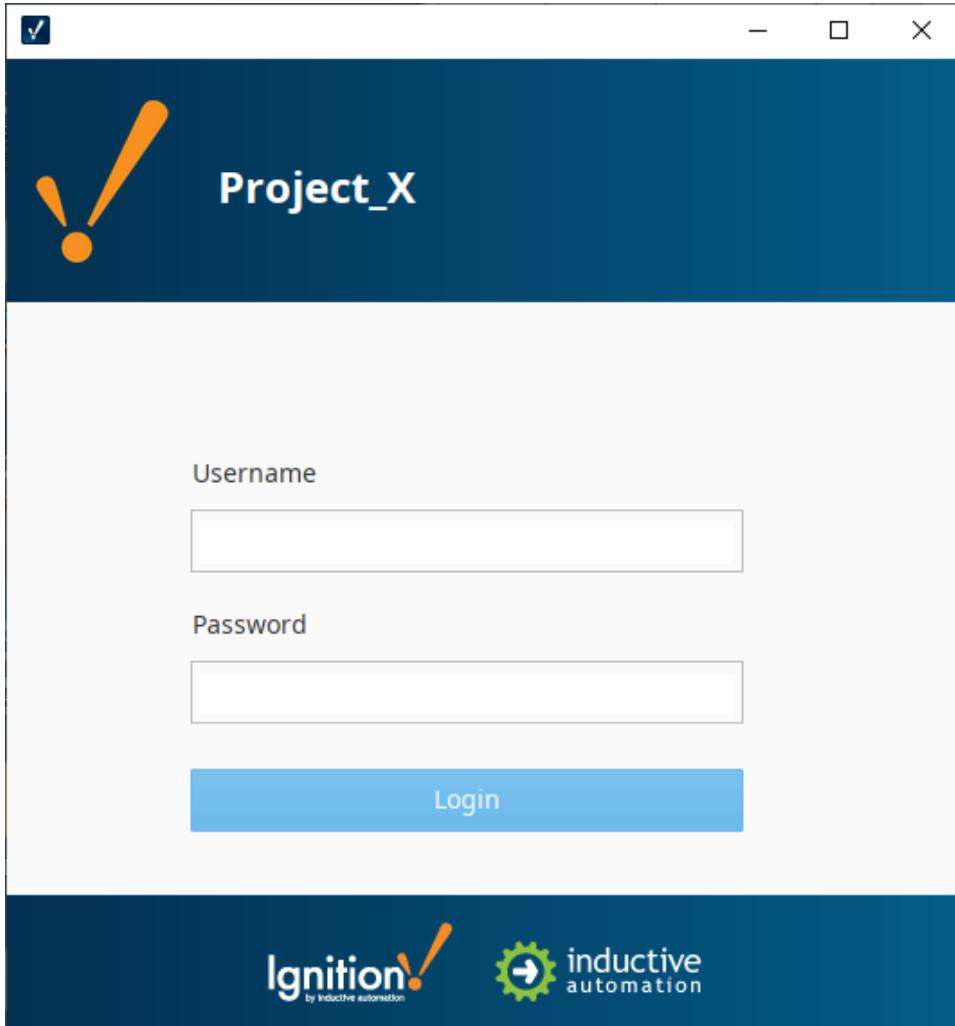
You can also launch the client by clicking on the three dot menu and selecting **Create Shortcut**. The project shortcut will be placed on your desktop so you don't have to run the Vision Client Launcher each time you want to launch a project.



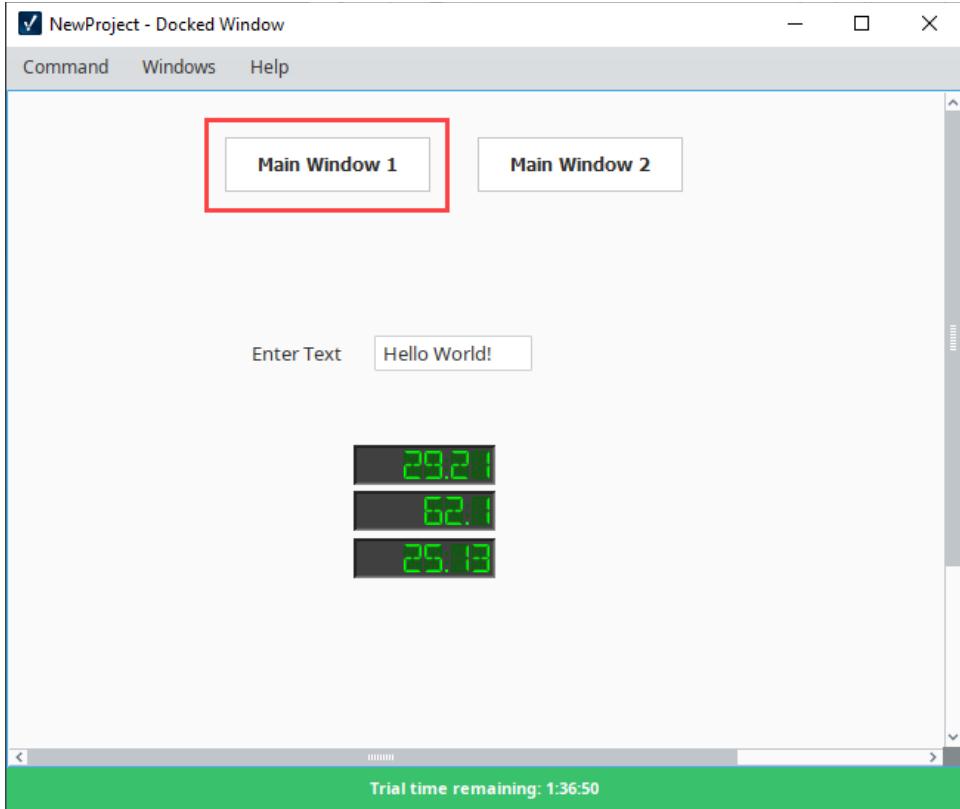
Next time, you can double click the shortcut on your desktop to open your project in the Vision Client.



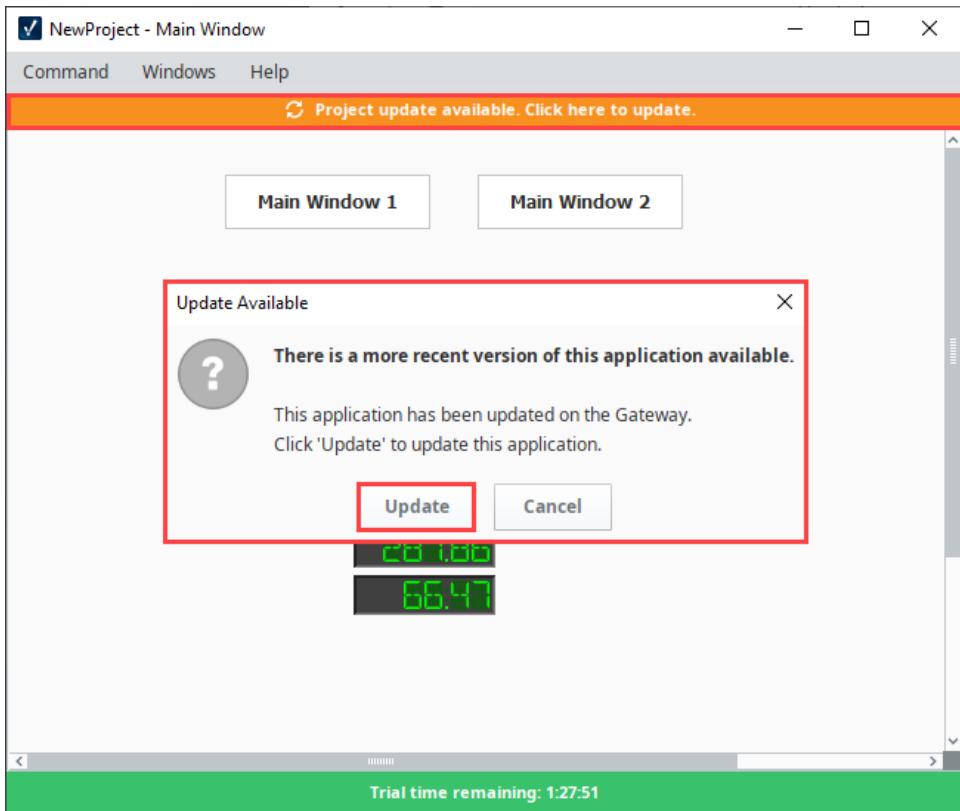
17. Login to the Vision Client using the user credentials you set up when Ignition was installed.



18. After you log in, you can see your Vision Client window displaying your project. Using the example we created in the [Adding Components to Vision Windows](#), we are showing **Main Window 1** which opens up on startup. Then click on **Main Window 2** to see the Easy Chart from [Showing Tag History in an Easy Chart](#) section.



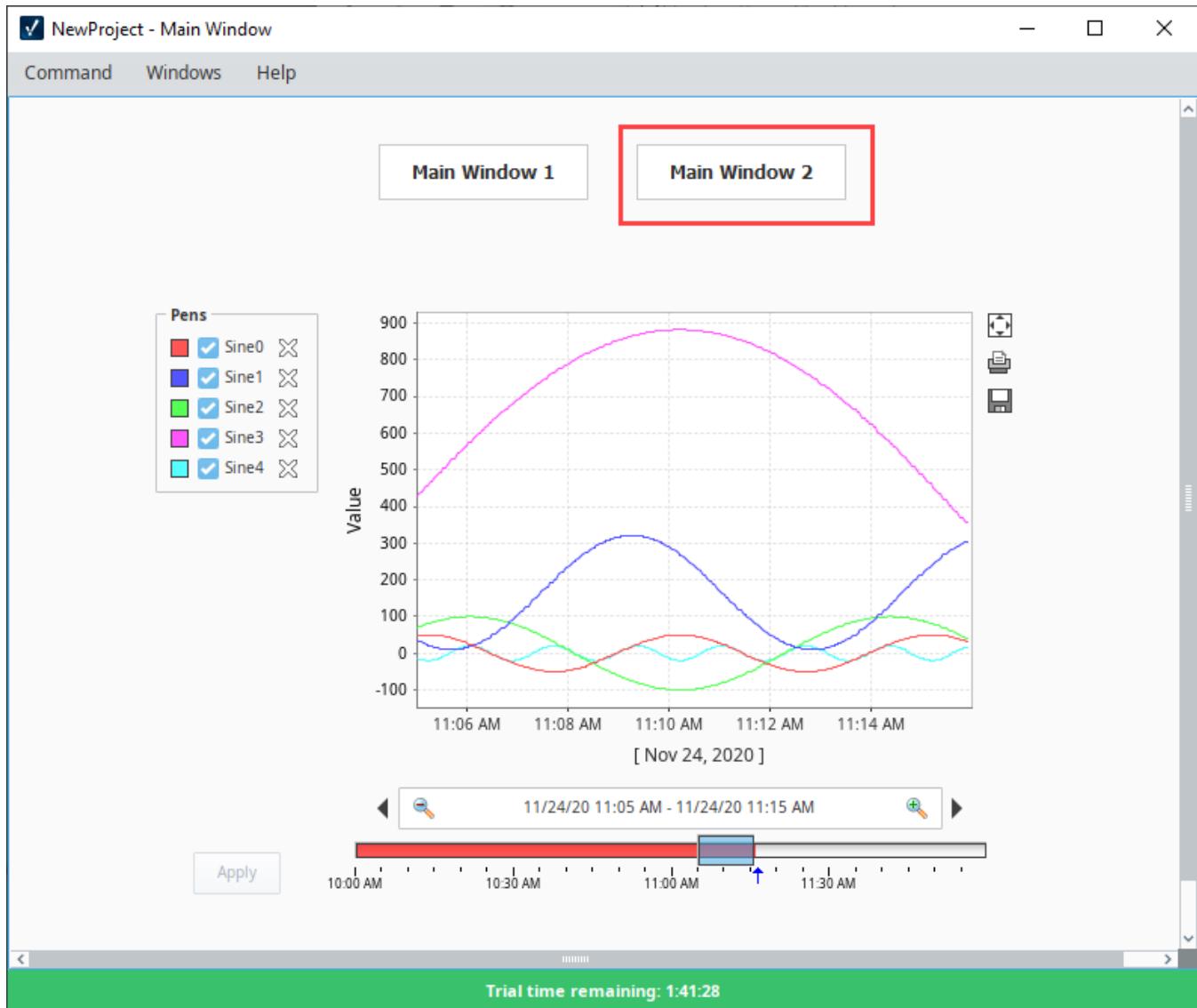
19. Without closing the Vision Client window, go back into the Designer and make a change to your project, and click **File > Save**. Your Vision Client will show an orange notification banner stating that there is an update to the project. Click on the banner and a dialogue box will open stating there is a more recent version of your application available. If you want to run it, click **Update**.



20. The Vision Client will then update itself.

You can launch as many Clients as you want! If you have other computers on the same network as the Gateway computer try launching on them too. Make sure that your Gateway computer doesn't have a Firewall enabled, or it is allowing traffic on port 8088 - the default port for the Ignition web server. The address to use on other computers is <http://ipaddress:8088> where ipaddress is the IP address of the computer Ignition is installed on.

Learn more on the [Vision Client Launcher](#) page.



Related Topics ...

- Component Events
- Using the Vision Easy Chart
- Configuring Tag History
- Binding Types in Vision

Installing and Upgrading Ignition

The installation process for Ignition is designed to be as simple as possible. You can find installers on our Downloads page: <https://inductiveautomation.com/downloads/ignition/>.

You'll find the dedicated Installers (just called "installer" for all major operation system. You'll also find ZIP files. Both options install Ignition, but the installers are generally easier to use. Download an installer, and head to the appropriate section below. For information on ZIP file installation instead, go [here](#).

Make sure your system meets our minimum system requirements (also found on our downloads page).

Install Ignition

1. Download an installer from our downloads page: <https://inductiveautomation.com/downloads/ignition/>.

Installing on a Headless Server?

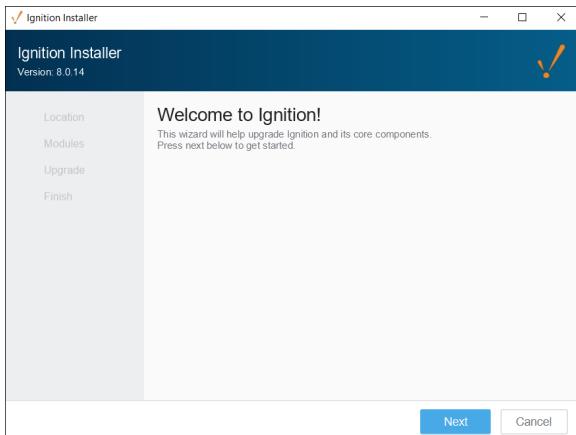
The steps below detail installing Ignition in a graphical environment. If you're attempting to install Ignition on a headless server, you can use the same installer, but you'll likely want to refer to the [Command Line Installations](#) page for details on command line arguments, which allow you to do things like run the installer in unattended mode.

Installing on Headless Linux Distros

When installing Ignition on a headless Linux distro, you'll want to install the fontconfig package, as some Ignition features use the package.

```
sudo apt-get install fontconfig
```

2. Run the installer and follow the steps in the installation wizard.



Linux Installations

Before you can run the installer on Linux, you must first make the installer executable. From terminal:

```
chmod +x ignition-x.x.x-linux-xxxinstaller.run
```

Alternatively, your desktop environment may have some graphical interface to make the installer executable. Check your desktop's documentation for more details.

On this page ...

- [Install Ignition](#)
 - [Ignition Default Installation Directories](#)
- [Quick Start Configuration](#)
- [Start or Stop Ignition](#)
- [Upgrade Ignition](#)
 - [Updating License for an Upgrade](#)
 - [Upgrading a Redundant Pair](#)
- [Reset the Password](#)
- [Uninstall Ignition](#)



INDUCTIVE
UNIVERSITY

Installing Ignition on Windows

[Watch the Video](#)



INDUCTIVE
UNIVERSITY

Installing Ignition on Linux

[Watch the Video](#)



INDUCTIVE
UNIVERSITY

Installing Ignition on Mac

[Watch the Video](#)



macOS Installations

There is a known issue with running the installer on macOS. If you're receiving warnings about being unable to open the installer, see our [Knowledge Base Article](#).

In most cases you can use the default settings in the installer. This section describes what each setting does.

Default Installation Directories

This determines the installation directory. You're free to pick the directory you wish. See the [Ignition Default Installation Directories](#) section below for the default paths.

Gateway Service Name (Windows Only)

This field allows you to choose the service name of Ignition, as it appears in the operating system's service listing.

Installation Options

Allows which modules are included in your base installation. Your choices are **Typical** and **Custom**. The Typical option will use a select list of commonly used modules. The Custom option allows you to install specific modules. In most installations the Standard option will suffice, but do note you can always add modules and remove modules after the installation process.

Start Ignition Now

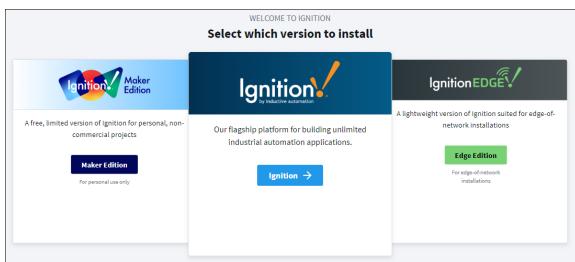
Once the installer finishes, you'll be provided with an option to start the Gateway immediately. In most cases this is the desired option. You would only opt out of this choice if you didn't want the Gateway to start once the installer finishes.

Once finished, a web browser will open, and start off the System Commissioning process.

Install As a Service (Linux and MacOS only)

Allows you to install Ignition as a service. For Windows systems, Ignition is installed as a service by default as it is a requirement for Ignition to work on a Windows server.

3. After the installation wizard finishes, a web browser will open and start the commissioning process. At this point you'll need to select your edition.



Standard Ignition - Unlimited edition of our flagship platform. Designed for commercial use.

Ignition Edge - A lightweight version of Ignition suited for edge-of-network installations.

Maker Edition - A free, limited version of Ignition for personal, non-commercial projects.

4. Follow the steps in the commissioning process to complete installation.

The System Commissioning process is a series of steps that further configure your Gateway. This process begins after installation or upgrading. In addition, it can be re-triggered via a password reset with the [Gateway Command-Line Utility](#).

This feature is new in Ignition version **8.1.1**.
[Click here](#) to check out the other new features

As of 8.1.1, commissioning can now set the Gateway Network port in addition to the HTTP and HTTPS port.

Clean installs of Ignition will check the default ports for their availability. If no ports are available, the Gateway will iterate that port number until one is found that is available. These ports can be modified during the commissioning process.

Existing or upgraded Gateways will not search for available ports, and will result in a faulted Gateway.

Activate License (Maker Edition Only)

When installing Maker Edition, you'll be asked to provide a License Key (an 8 digit string) and an Activation Token (a considerably longer string). Both of these can be acquired for free from your Inductive Automation account: <https://account.inductiveautomation.com/>.

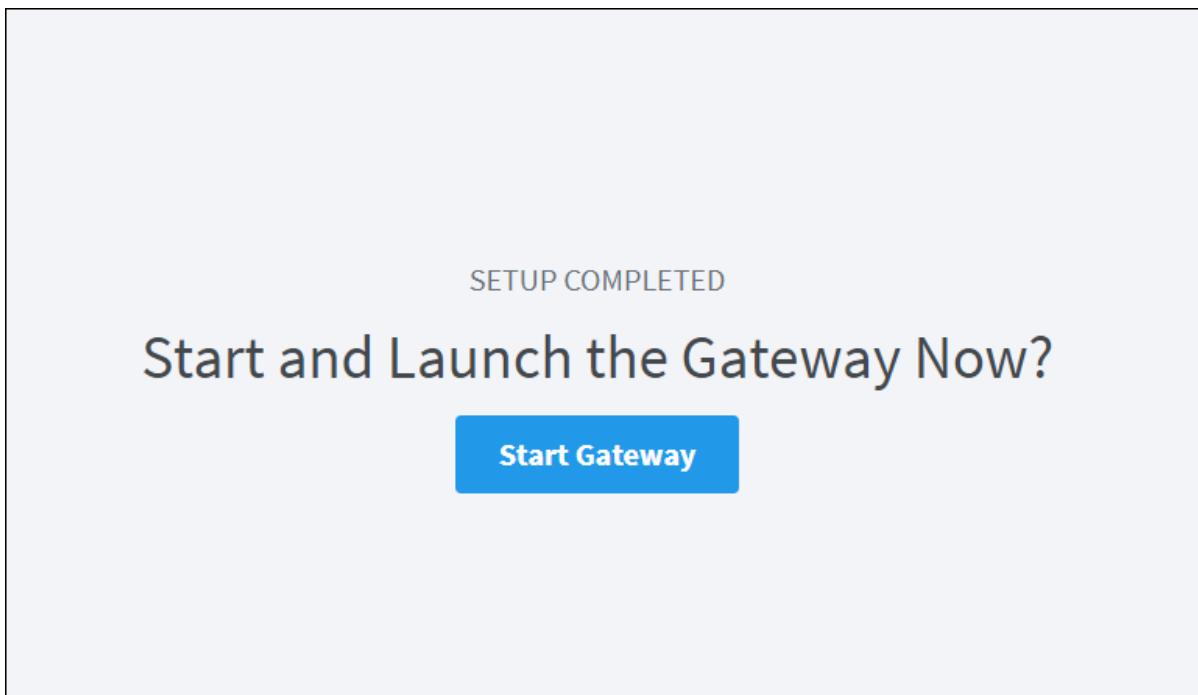
Create a User

This step asks you to create an initial user. This user will have full access to Ignition, although you're free to modify this user later. You'll need this user account to make changes to Ignition at all, so make sure you remember these credentials. Otherwise you'll need to perform a password reset: see the [Gateway Command-Line Utility](#) page.

HTTP Configuration

Allows you to specify the ports Ignition uses. By default, HTTP is **8088**, and HTTPS is **8043**.

Once you finish the commissioning process, your service will start.



Ignition Default Installation Directories

The default directories for each operation system are listed below.

Windows

```
C:\Program Files\Inductive Automation\Ignition
```

MacOS

```
/usr/local/ignition
```

Linux

```
/usr/local/bin/ignition
```

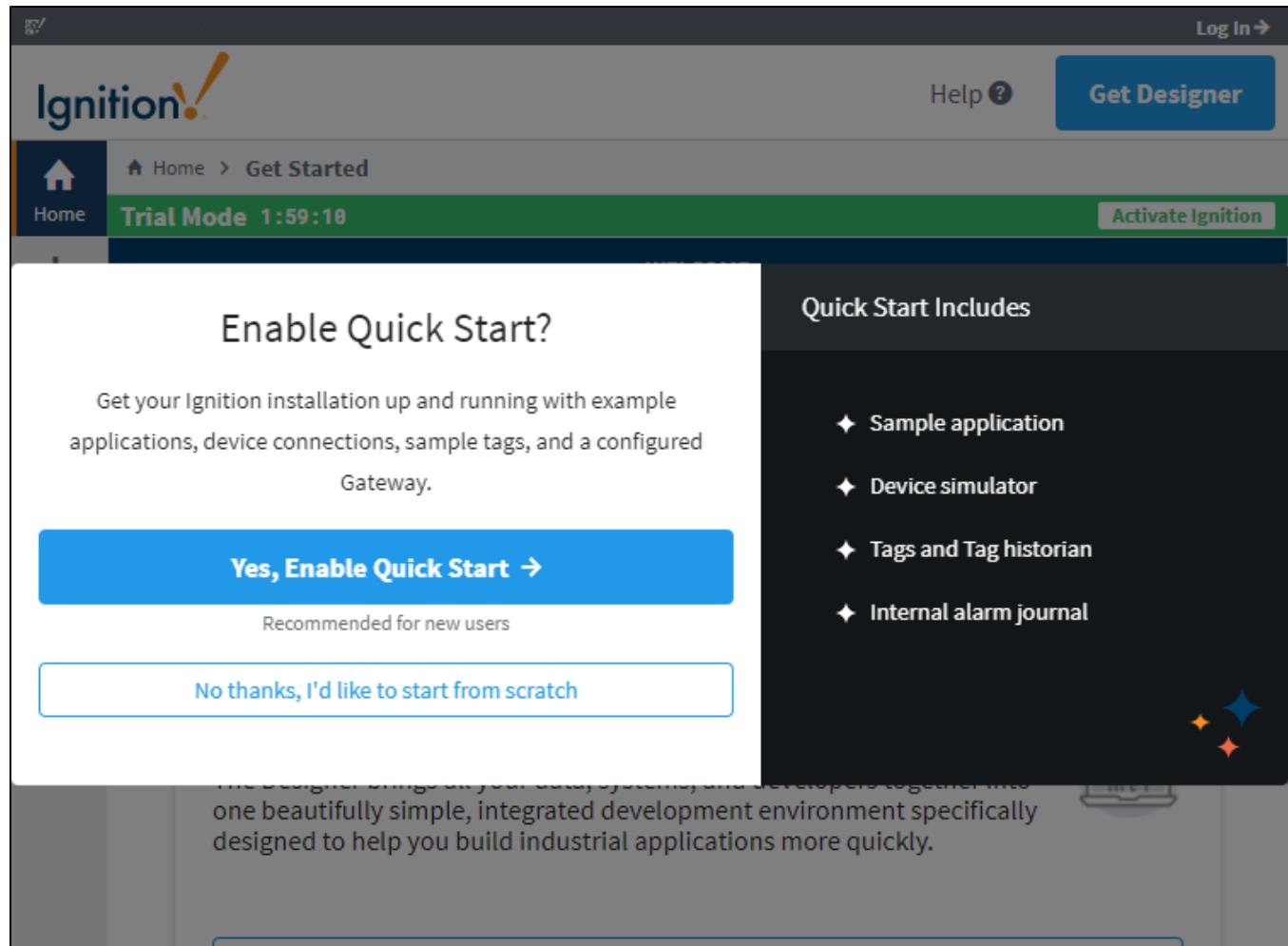
Quick Start Configuration

This feature is new in Ignition version **8.1.0**
[Click here](#) to check out the other new features

Once the installation process has finished, you'll be redirected to the Gateway Webpage. From there you'll be asked to opt into the Quick Start configuration, which makes several configuration changes to the system. Quick Start can only be enabled upon a new installation.

The following changes are made once Quick Start is enabled:

- Creates an internal SQLite database connection.
- Configures a device connection to a [Programmable Device Simulator](#) instance, and adds Tags from the simulator into a Tag provider.
- Configures an internal alarm journal.
- Creates a sample project.



Start or Stop Ignition

After installing with the Installer, Ignition will have a service representing it on the operating system. By default Ignition will start when the system starts, but you can manually start and stop the system with the following terminal commands:

Windows

The examples below assume "Ignition" is the service name used during installation (which also happens to be the default service name). You may need to modify the examples below using your system's service name.

```
net start Ignition  
net stop Ignition
```

Mac

```
/usr/local/ignition/ignition.sh start  
/usr/local/ignition/ignition.sh stop  
/usr/local/ignition/ignition.sh restart
```

Linux

The examples below use the default installation directory

```
/usr/local/bin/ignition/ignition.sh start  
/usr/local/bin/ignition/ignition.sh stop  
/usr/local/bin/ignition/ignition.sh restart
```

Upgrade Ignition

Upgrading Ignition is as easy as running the installer. When you run the installer from the same directory as a previous install, our installers are able to detect that Ignition is already installed. If Ignition is already installed, the installer will ask if you would like to proceed with the upgrade. A wizard takes you through the upgrade in a similar workflow as installation workflow.

Ignition installers can be used to upgrade any edition of Ignition (i.e., Edge, Maker, or Standard) with the exception of ARM as that will require a zip installer for upgrading.

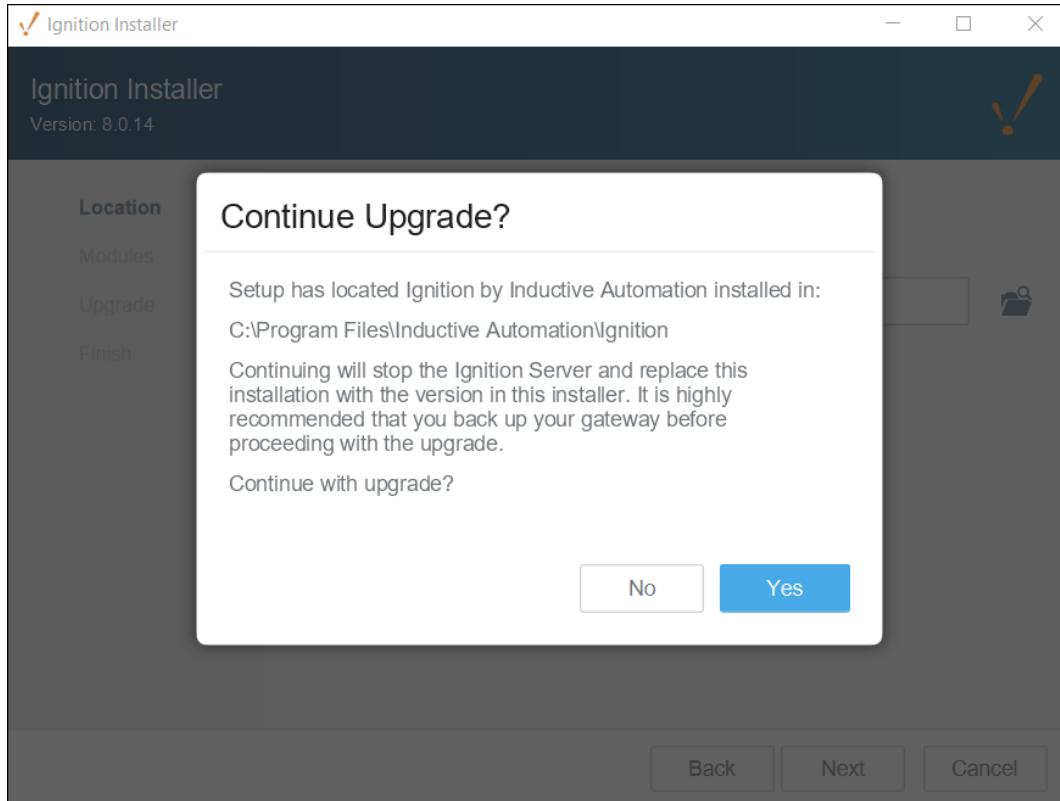
Linux Installer

Older versions of the Linux installer would place files across multiple different directories. Running an 8.0.14+ installer will move all those files to the specified installation directory.

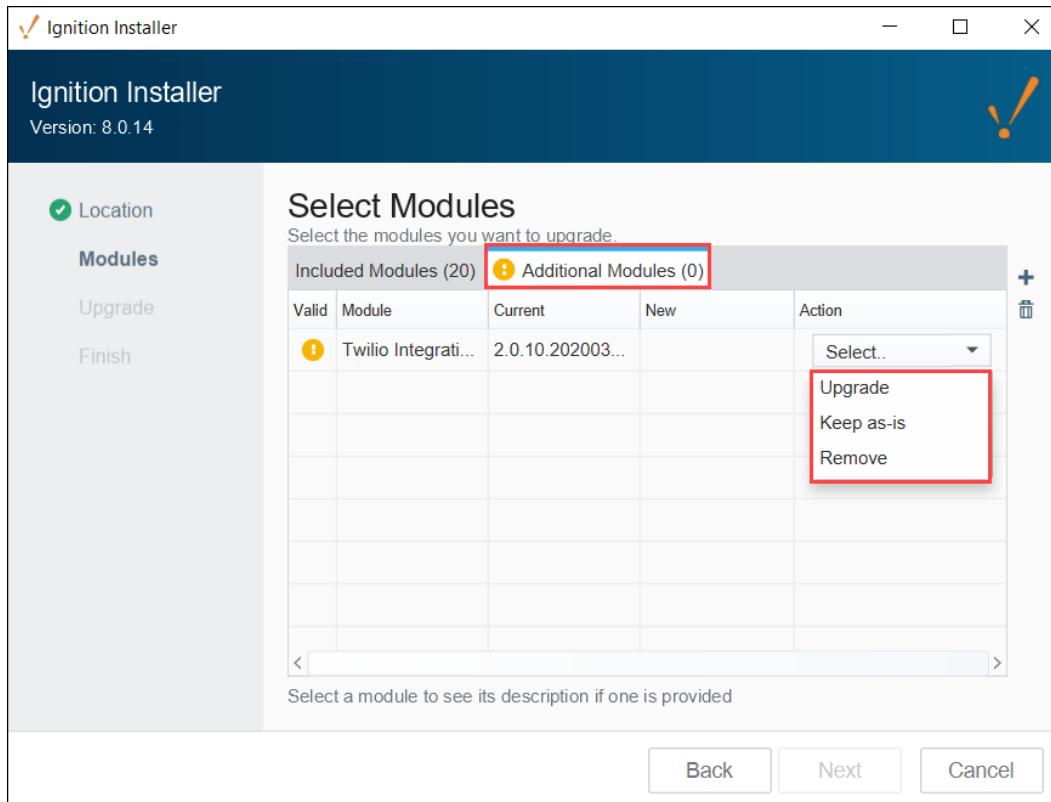
JDBC Drivers and Upgrading

Sometimes new versions of Ignition will incorporate new JDBC drivers, which are used to connect to SQL databases. When upgrading Ignition, the JDBC drivers will **not** be upgraded. The reason being, is that you should only need to change the JDBC driver if you upgrade your database. See the [JDBC Drivers and Translators](#) page for more details.

During the upgrade process, the Gateway will have to restart, so make sure to account for downtime when upgrading.



When you upgrade Ignition, you'll also be given the option to update any additional modules you currently have installed, including Third-Party modules. On the Select Modules screen, there is a tab for the modules included in your version of Ignition, and a tab for Additional Modules you may be using. Under the Action column, you can choose to upgrade or remove the module or leave it unchanged.



Updating License for an Upgrade

If you are updating from one major version to the next (i.e, upgrading from version 8.0.# to version 8.1.#) , you will need an updated license. You can do this before upgrading so there is no unlicensed time after the upgrade.

1. Let us know.

You can reach Inductive Automation by phone at **(800) 266-7798**.

Some Major upgrades include new modules or have modified the functionality of pre-existing modules. If your license needs to be updated, call us and we can guide you through it. If you are updating from one version to the next, or from 7.0.0 all the way to the latest version, we can talk you though what is different and what new functionality is available. See the [New in this Version](#) page for more specifics about what was updated in the various versions.

2. Reload your license.

Once your license is updated in our system, go to your Config section of the Gateway webpage, and click System > Licensing. Click on the Update License icon .

For a detailed guide on reloading a license, see updating a License on this page. If you are coming from an older version, the Gateway may look different. Don't worry, all this functionality is still there, but you may have to un-activate then re-activate instead if the Update License button is missing.

Upgrading a Redundant Pair

There are a few extra things to consider when upgrading a redundant pair of Gateways. When performing the upgrade locally, the order of the upgrade technically doesn't matter. However, we recommend upgrading the Master first. This allows the Backup to take over when the Master is down for an upgrade, and the Master can resume operations when it comes back online. At this point the Backup can be upgraded without interruption. The same considerations mentioned above should also be applied to the redundant pair, but you only need a Gateway Backup file from the Master Ignition Gateway.

If the upgrade will be performed remotely using the Enterprise Administration Module, then the Backup Gateway should be upgraded first. This allows the EAM controller to manually tell clients/sessions to communicate with the newly upgraded Backup Gateway, even though the Master will be a different version, allowing time to upgrade the Master.

Reset the Password

To reset the main password for the user with full privileges, use the Gateway Command-line Utility ([gwcmd](#)) utility. The '[gwcmd](#)' utility triggers system commissioning, and prompts the user to reset the username and password for the main user (with full privileges) after the [Gateway](#) restarts. Linux and Mac [OS](#) platforms also use the '[gwcmd](#)' utility for password reset. For more information, see the [Gateway Command-line Utility - gwcmd](#) page.

Uninstall Ignition

Note: Uninstalling Ignition will remove any projects or Gateway level configurations you made. If you wish to preserve anything, make sure to take a [Gateway backup](#) first.

To uninstall Ignition, run the uninstaller file. This file can be found in the main Ignition installation directory, for example:

Windows

```
C:\Program Files\Inductive Automation\Ignition\Uninstall.exe
```

Mac

```
/usr/local/ignition/Uninstall.app
```

Linux

```
/usr/local/bin/ignition/Uninstall.run
```

Command Line Installations

This page contains command line argument keys that can be used to modify the installation process. Each operating system supports launching with arguments, but with some small modifications to ensure they are passed through to the installer:

- Windows: a `--%` arg should go before installer args
- Mac: requires `open -a` command with `--args` before the args to pass to the installer
- Linux: a `--` arg should go before installer args

Arguments should be quoted.

Arguments Table

The table below lists the supported arguments.

| Command Line Argument Key | Description | Possible Values | Example |
|---------------------------|---|---|--------------------------------|
| unattended | <p>The unattended install mode to run the installer in. It runs with all default values unless otherwise specified.</p> <p>The unattended upgrades remove known IA modules that are deprecated, leaves third party as they are, and upgrades currently installed bundled modules. Additionally, unattended upgrades do not trigger the Ignition service to start automatically. The Ignition service <i>must</i> be started by a user.</p> <p>The argument also works with the uninstaller, allowing for unattended uninstalls.</p> | <ul style="list-style-type: none">• graphical - run with full size installer window but no interaction• minimal - shows a progress bar only unless errors occur• text - feedback on the command line only, no GUI• none - completely silent, no feedback except for the log file when complete regardless of success or fail | "unattended=minimal" |
| textMode | Whether to run this installer in <i>interactive</i> text mode regardless of graphical support. Note that <code>unattended</code> supersedes this value. | true - run in text mode | "textMode=true" |
| location | Overrides the default installation directory, allowing you to specify where Ignition should be installed. | A fully qualified file path | "location=/usr/local/ignition" |
| logLevel | The logging level for the installer. The default is INFO, however more detail can be provided, revealing more information about the install process, as well as the installer's internal operations. | Options are (from most granular to least granular): <ul style="list-style-type: none">• trace• debug• info• warn• error | "logLevel=trace" |
| user | Sets the ownership of the install to this user. | The username (as recognized by the operating system) that should be the owner of the installation. | "user=myUser" |
| serviceName | <p>Sets the service name for the Gateway. Defaults to "Ignition". When running the graphical installer, the service is installed immediately on Windows and set for optional future installs on Linux/Mac .</p> <p>When using this key with the unattended key during an upgrade, the existing service on windows will be removed, and replaced by a new service.</p> | my-gateway | "serviceName=my-gateway" |

Example Commands

The following examples demonstrate how arguments can be used when running the installer from. For the purposes of these examples, any mention of a user directory will use "`yourUser`", and the Ignition version number in the installer name will be "`8.0.14`". These should be replaced by appropriate values when attempting to run the installer on your system.

Windows

```
# Runs the installer on command line only and requires user interaction.  
C:\Users\yourUser\Downloads\ignition-8.0.14-windows-x64-installer.exe --% "textMode=true"  
  
# Runs the installer on command line, changes the default install directory, and does not require user interaction.  
C:\Users\yourUser\Downloads\ignition-8.0.14-windows-x64-installer.exe --% "unattended=text" "location=C:\Ignition"
```

Mac

```
# Runs the installer on command line only and requires user interaction.  
open -a /Volumes/IgnitionInstaller/ignition-8.0.14-osx-installer.app --args "textMode=true"  
  
# Runs the installer on command line, changes the default install directory, and does not require user  
interaction.  
open -a /Volumes/IgnitionInstaller/ignition-8.0.14-osx-installer.app --args "unattended=text" "location=  
/Users/yourUser/Desktop/test"
```

Linux

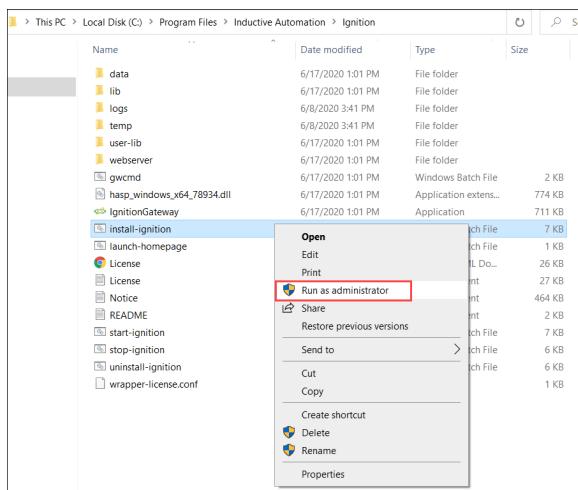
```
# Runs the installer on command line only and requires user interaction.  
"homes/yourUser/Downloads/ignition-8.0.14-linux-x64-installer.run" -- "textMode=true"  
  
# Runs the installer on command line, changes the default install directory, and does not require user  
interaction.  
"homes/yourUser/Downloads/ignition-8.0.14-linux-x64-installer.run" -- "unattended=text" "location=/home  
/yourUser/Desktop/test"
```

ZIP File Installations

Windows

For a new installation from a ZIP file, do the following:

1. Unzip the contents of the ZIP file. Wherever this file is unzipped is where Ignition will be installed so be sure to unzip this file in a suitable location. For this example, I placed my zip file contents in "C:\Program Files\Inductive Automation\Ignition\" which is the default Windows installation directory for Ignition.
2. Once your file contents are unzipped in a suitable location, run the executable file named **install-ignition.bat** to install the Windows service like below:



3. Similarly, run the executable file named **start-ignition.bat** to start the Ignition service.
4. Navigate to the Gateway's web interface (<http://localhost:8088>) to start the commissioning process. If any errors occur during startup, they will be logged to logs/wrapper.log

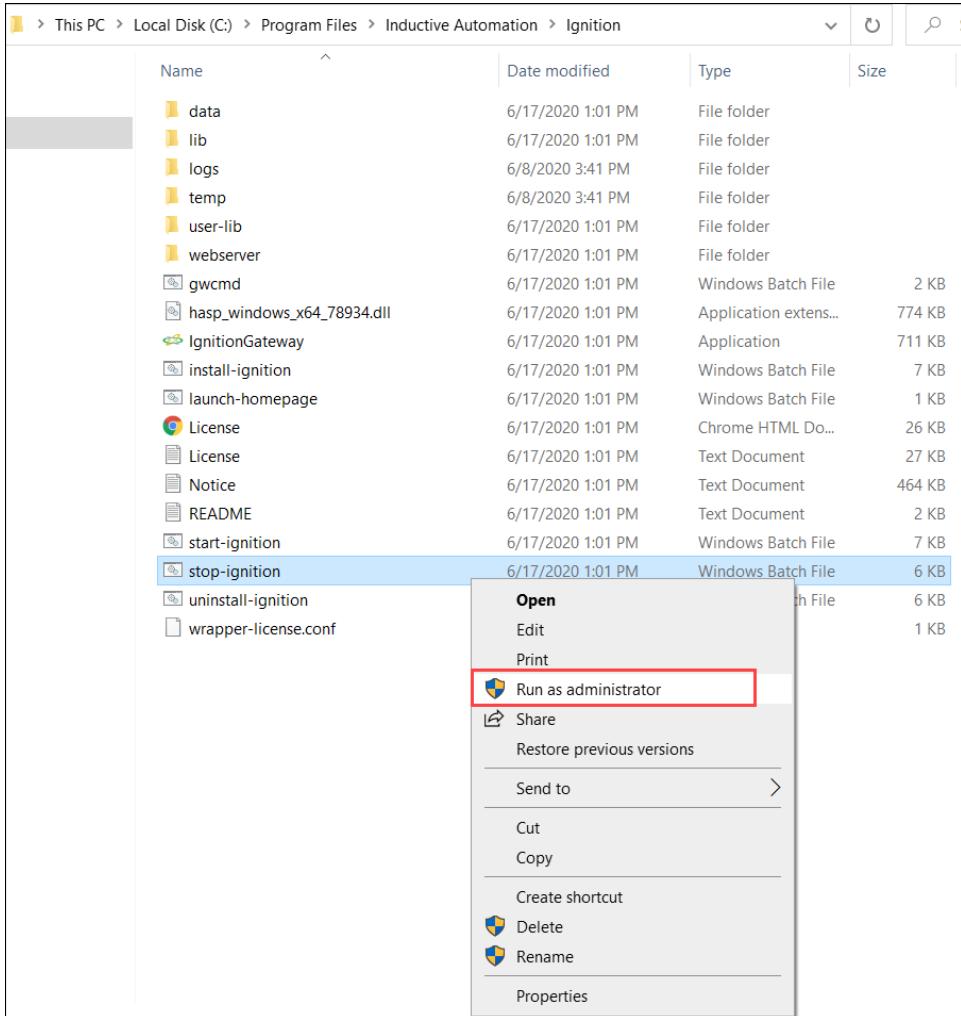
Upgrading an Existing Windows Installation

If you are upgrading from a preexisting installation, follow these steps:

1. Take a [Gateway backup](#) from the running system and store in a safe location.
2. Stop Ignition by running the **stop-ignition.bat** file located in the Ignition installation directory.

On this page ...

- [Windows](#)
 - [Upgrading an Existing Windows Installation](#)
- [Linux \(Including ARM\)](#)
 - [Upgrading an Existing Linux Installation](#)
- [Mac](#)
 - [Upgrading an Existing MacOS Installation](#)
- [Deploying Ignition to Another System](#)
 - [Requirements](#)
 - [Example](#)



3. Delete both the **lib** and **webserver\webapps** folders.
4. Unzip the ZIP installer for the version you're upgrade to into the current folder and allow files to be overwritten.
5. Unzip the new **jre-win** runtime in **lib\runtime** folder.
6. If this is a major version upgrade (such as 8.0 to 8.1) then the **data\ignition.conf** file may have changed. If this is a minor version upgrade (8.0.1 to 8.0.2) then skip to step 8.
7. To upgrade the ignition.conf file for a major version upgrade, run the following from command line in the ignition folder:

```
lib\runtime\jre-win\bin\java.exe -classpath "lib/core/common/common.jar" com.inductiveautomation.ignition.common.upgrader.Upgrader . data logs file=ignition.conf
```

8. Start Ignition. Run **start-ignition.bat**

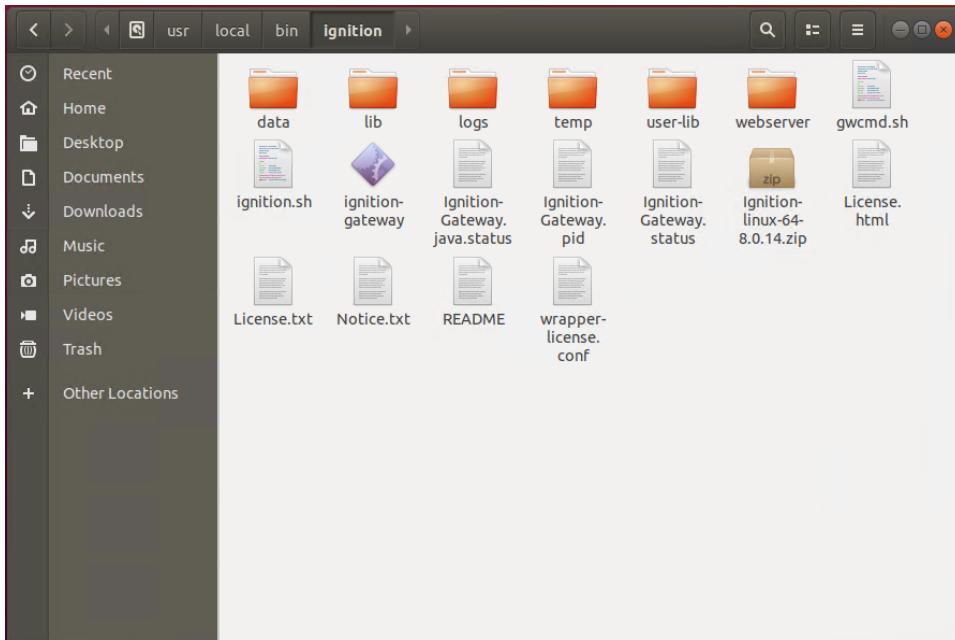
At this point, the Gateway will start, completing the upgrade.

Linux (Including ARM)

Note: 64-bit zip installers are available for ARM machines.

For a new installation from a ZIP file, do the following:

1. Unzip the contents of the ZIP file. Wherever this file in unzipped is where Ignition will be installed, so be sure to unzip this file in a suitable location. Below, you can see the contents of the zip file in /user/local/bin/ignition which is the default Linux installation directory for Ignition.



2. Open up a terminal command, navigate to the directory where you unzipped the installer and make the **ignition-gateway** file and all **.sh** files executable. You can run the following commands:

```
chmod +x ignition-gateway  
chmod +x *.sh
```

3. Execute **ignition.sh**:

```
./ignition.sh start
```

A screenshot of a terminal window titled 'root@ubuntu: /usr/local/bin/ignition'. The window shows the command 'root@ubuntu:/usr/local/bin/ignition# ./ignition.sh start' being run. The output indicates the gateway is starting, waiting for Ignition-Gateway, and then shows it is running with PID 5895.

```
root@ubuntu: /usr/local/bin/ignition# ./ignition.sh start  
Starting Ignition-Gateway...  
Waiting for Ignition-Gateway.....  
running: PID:5895  
root@ubuntu: /usr/local/bin/ignition#
```

That's it. You can navigate to the Gateway's Webpage (<http://localhost:8088>) to start the commissioning process. If any errors occurred during startup, they'll be recorded in **logs/wrapper.log** file.

You can also add the unzipped folder to your system PATH by adding the folder path to a file such as .profile or .bashrc. This allows you to start programs like the **Gateway Command-line Utility - gwcmd** from the command line without specifying a complete path to the unzipped folder.

Upgrading an Existing Linux Installation

If you are upgrading from a preexisting installation, follow these steps:

1. Take a [Gateway backup](#) from the running system and store it in a safe location.
2. Stop your Ignition service. Open a terminal window and navigate to your Ignition installation directory (/user/local/bin/ignition is the default install directory location and it is also the location we use in the installation steps above) and run the command below:

```
./ignition.sh stop
```

```
root@ubuntu:/usr/local/bin/ignition
File Edit View Search Terminal Help
root@ubuntu:/usr/local/bin/ignition# ./ignition.sh stop
Stopping Ignition-Gateway...
Stopped Ignition-Gateway.
root@ubuntu:/usr/local/bin/ignition#
```

3. Delete both the "lib" and "webserver/webapps" folders.
4. Unzip the contents of the new zip installer into the current directory and allow files to be overwritten
5. Set the executable permission on all .sh files in the unzipped folder by running this command:

```
chmod +x *.sh
```

6. Unzip the new runtime:

```
tar -C lib/runtime -xf lib/runtime/jre-nix.tar.gz
```

7. If this is a major version upgrade (8.0 to 8.1) then the data/ignition.conf file may have changed. If this is a minor version upgrade (8.0.1 to 8.0.2) then skip to step 8.
To upgrade the ignition.conf file for a major version upgrade, run the following from command line in the ignition folder, run the following:

```
lib/runtime/jre-nix/bin/java -classpath "lib/core/common/common.jar" com.inductiveautomation.ignition.common.upgrader.Upgrader . data logs file=ignition.conf
```

8. Start Ignition. Run:

```
./ignition.sh start
```

Mac

For a new installation from a ZIP file, do the following:

1. Open up Terminal.app and change directories to the same directory the ZIP install is located. If you downloaded to your desktop then it would be:

```
cd ~/Desktop
```

2. Become root:

```
sudo su
```

3. Unzip Ignition (Replace X.X.X with the actual build number). This unzips the contents of Ignition-osx-X.X.X.zip in /usr/local/ignition which is the default MAC OS installation directory for Ignition.

```
unzip -d /usr/local/ignition Ignition-osx-X.X.X.zip
```

4. Change directory to the directory where you unzipped the contents of the zip installer or /usr/local/ignition in this example.

```
cd /usr/local/ignition
```

5. Make the make the following files executable:

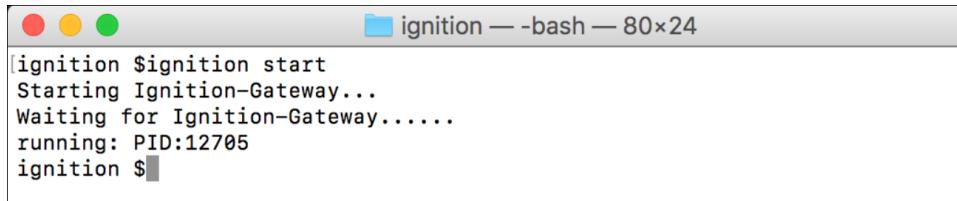
```
chmod +x *.sh  
chmod +x ignition-gateway
```

6. Create a symlink in /usr/local/bin

```
cd /usr/local/bin  
  
ln -s /usr/local/ignition/ignition.sh ignition
```

7. You can now start Ignition from Terminal.app using the [starting and stopping commands](#) listed above. Start Ignition with:

```
ignition start
```



```
[ignition $]ignition start  
Starting Ignition-Gateway...  
Waiting for Ignition-Gateway.....  
running: PID:12705  
ignition $]
```

A screenshot of a Mac OS X terminal window titled "ignition — bash — 80x24". The window shows the command "ignition start" being run and its output: "Starting Ignition-Gateway...", "Waiting for Ignition-Gateway.....", and "running: PID:12705". The prompt "ignition \$" is visible at the bottom.

That's it. You can navigate to the Gateway's web interface (<http://localhost:8088>) to start the commissioning process. If any errors occurred during startup, they'll be recorded in **logs/wrapper.log** file.

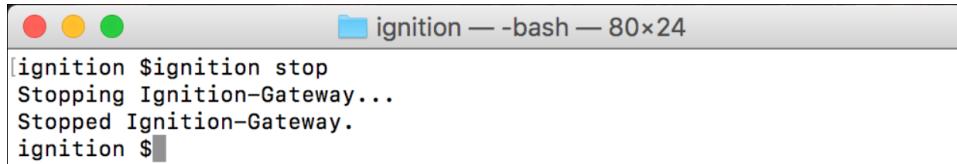
You can also add the unzipped folder to your system PATH by adding the folder path to a file such as .profile or .bashrc. This allows you to start programs like the [Gateway Command-line Utility - gwcmd](#) from the command line without specifying a complete path to the unzipped folder.

Upgrading an Existing MacOS Installation

If you are upgrading from a preexisting installation, follow these steps:

1. Take a [Gateway backup](#) from the running system and store in a safe location.
2. Stop Ignition. Run:

```
ignition stop
```



```
[ignition $]ignition stop  
Stopping Ignition-Gateway...  
Stopped Ignition-Gateway.  
ignition $]
```

A screenshot of a Mac OS X terminal window titled "ignition — bash — 80x24". The window shows the command "ignition stop" being run and its output: "Stopping Ignition-Gateway..." and "Stopped Ignition-Gateway.". The prompt "ignition \$" is visible at the bottom.

3. Delete both the **lib** and **webserver/webapps** folders.
4. Unzip the new ZIP into the old Ignition directory, allowing files to be overwritten.
5. Set the executable permission on all .sh files and the ignition-gateway file:

```
chmod +x *.sh
chmod +x ignition-gateway
```

6. Unzip the new runtime:

```
tar -C lib/runtime -xf lib/runtime/jre-mac.tar.gz
```

7. If this is a major version upgrade (8.0 to 8.1) then the **data/ignition.conf** file may have changed. If this is a minor version upgrade (8.0.1 to 8.0.2) then skip to step 8.

To upgrade the ignition.conf file for a major version upgrade, run the following from command line in the ignition folder, run the following:

```
lib/runtime/jre-mac/bin/java -classpath "lib/core/common/common.jar" com.inductiveautomation.ignition.common.upgrader.Upgrader . data logs file=ignition.conf
```

8. Start Ignition. Run:

```
ignition start
```

This feature is new in Ignition version **8.1.2**
[Click here](#) to check out the other new features

Note: Updated the user manual to include preconfiguring a [Gateway](#) in the following section, Deploying Ignition to Another System. The instructions apply to Ignition version 8.1.2 and future versions of Ignition. It does not apply to prior Ignition versions preceding 8.1.2.

Deploying Ignition to Another System

Some users are interested in "preconfiguring" an Ignition Gateway. That is, configuring an instance of an installation, and then moving the installation files to a new server. This can be useful in cases where you're attempting to push out new Ignition installations via command-line interface to multiple separate systems.

Normally, you need to install Ignition manually on each of these systems and walkthrough the commissioning process to create an initial user and select an Ignition edition, which defeats the purpose of this whole deployment strategy. Thus, the following example will demonstrate how to deploy a preconfigured Ignition Gateway that has already been commissioned and is ready to start up.

Requirements

You'll need the following:

- A ZIP installer for the OS and Ignition Version you're planning to deploy on. ZIP installers can be found on our [Downloads](#) page.
- An existing Ignition Gateway where you already accepted the End User License Agreement (EULA). This existing Gateway must match the Ignition version of the system you're going to deploy to. If you don't have an install available, simply run through the installation process once, agree to the terms in the EULA, and finish the rest of the commissioning process. Accepting the EULA is important because you'll need to manually do this once.
- An Ignition Gateway backup from an existing system. We'll use this to define the initial user, and carry over any other resources we want on the new system. This can be a backup from the Gateway you used in the previous requirement. We'll use this backup to inject a user account that only you have credentials for.

Example

1. Locate the EULA hash from your existing Ignition Gateway. You will find the hash in the `commissioning.json` file in Gateway's installation directory: `%installDirectory%/data/commissioning.json`.

Note:

You'll only find this file on systems that have completed the commissioning process.

Part of the commissioning process involves accepting the EULA. The steps in this guide involve skipping the commissioning process, including the EULA. Thus, to be able to follow along with this guide you'll need to review the terms in the EULA and manually accept them on a separate system.

2. Open the file in a text editor, and copy the value for the `eulaSetup.eula` key, including the quotation marks. For example, assuming a file looked like the entry below, you would copy "XXXXX" (the actual hash will be much longer). Save this value for later, as we'll need to reference it in a future step.

```
{  
  "isCommissioned": "COMMISSIONED",  
  "connections.useSsl": "false",  
  "eulaSetup.eula": "XXXXX"  
}
```

3. Next, take the Ignition installer and unzip it into a directory that's easily accessible. This directory will be the **base directory** that we'll ultimately end up deploying to on one or more different servers.
4. In the base directory, edit the [Gateway's configuration file](#) (`ignition.conf`) using the text editor of your choice. The file is located at `%baseDirectory%/data/ignition.conf`
5. Part of the installation process normally includes an edition selection, but we can bypass that by modifying the configuration file to explicitly state which edition the Gateway should be set to. Located the "Java Additional Parameters" line in the configuration file. The file should have some entries like the following:

```
# Java Additional Parameters  
wrapper.java.additional.1=-Ddata.dir=data  
#wrapper.java.additional.2=-Xdebug  
#wrapper.java.additional.3=-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=*:8000
```

at the end of that section, add the following:

```
wrapper.java.additional.2=-Dedition=%value%
```

Where `%value%` is the edition, you want the deployed system to use: "edge" for Ignition Edge, and "maker" for Ignition Maker. For "standard" edition, simply omit the "`=%value%`" portion. For example, the file should look something like the following when configuring for Standard edition:

```
# Java Additional Parameters  
wrapper.java.additional.1=-Ddata.dir=data  
#wrapper.java.additional.2=-Xdebug  
#wrapper.java.additional.3=-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=*:8000  
wrapper.java.additional.2=-Dedition
```

And the file would look like the following if deploying Edge edition:

```
# Java Additional Parameters  
wrapper.java.additional.1=-Ddata.dir=data  
#wrapper.java.additional.2=-Xdebug  
#wrapper.java.additional.3=-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=*:8000  
wrapper.java.additional.2=-Dedition=edge
```

6. Save the changes you made to the configuration file.
7. Also in `%baseDirectory%/data/`, create a text file and call it `commissioning.json`.
8. Edit the newly created commissioning file, and set contents to the following:

```
{  
  "isCommissioned": "COMMISSIONING",  
  "connections.useSsl": "false",  
  "eulaSetup.accepted": "true",
```

```
"eulaSetup.eula": "xxxxxxxxxxxxxxxxxxxxxx",
"authSetup.username": "admin",
"authSetup.password": "[e811cfc7]fc3b28ad5f73331c94790129f0931437ac31d59376b0e4b15df580a8841ba165"
}
```

The JSON above does a couple of things, but notably defines an initial user on the new system. The user is "**admin**", with a user of "**password**". Later, we'll restore from our Gateway backup which will replace this initial user, but for now, we simply need to have a user to bypass commissioning.

9. In the same commissioning file, replace the value of the `eulaSetup.eula` key with the value you took from step 2.
10. Save the commissioning file.
11. At this point, you're ready to deploy. Take the contents of your base directory, move them to a new server, and perform a normal ZIP installation. When the Gateway starts, it will bypass the commissioning process and the Gateway will immediately startup.
12. Once the Gateway is running, you can use the **admin** user we configured earlier to access the newly deployed instance of Ignition. From here, you can restore from the Gateway Backup file mentioned in the requirements section of this guide. You can do this [from command-line](#) which would look something like the following:

```
gwcmd --restore /path/to/your/GWBK
```

Installing or Upgrading a Module

Module Configuration

All module configuration is done from Gateway Webpage in the **Config > Modules** section. From here you can add, remove, and restart modules. Modules integrate their settings into the Gateway configuration tree, and therefore do not offer settings in this section.

| Name | Version | Description | License | State | More | restart |
|---------------------------|------------------------|--|-----------|---------|--------|-----------|
| Alarm Notification | 5.0.0 [02101051112] | Provides alarm notifications via email | Activated | Running | [More] | [restart] |
| Allen Bradley Driver | 5.0.0 [02101051112] | Allen Bradley driver suite for the OPC UA module. | Activated | Running | [More] | [restart] |
| BACnet Driver | 3.0.0 [02101051112] | A driver supporting BACnet (Distributed Network Protocol) device. | Activated | Running | [More] | [restart] |
| Enterprise Administration | 3.0.0 [02101051112] | A remote Gateway administration system, allowing you to monitor multiple Gateways and automate tasks from a single controller. | Activated | Running | [More] | [restart] |
| Logix Driver | 4.0.0 [02101051112] | A driver for communicating with Allen Bradley Logix controllers (CFC), and includes Intrinsic version 21 support. | Activated | Running | [More] | [restart] |
| Modbus Driver | 6.0.0 [02101051112] | A driver for communicating with devices via Modbus TCP. | Activated | Running | [More] | [restart] |
| Omron Driver | 3.0.0 [02101051112] | A driver for Omron NJ series PLCs. | Activated | Running | [More] | [restart] |
| OPC-UA | 8.0.0 [02101051112] | Provides Ignition's OPC UA client and server functionality. | Activated | Running | [More] | [restart] |
| Perspective | 1.0.0 [02101051112] | A module that provides modern, responsive HTML based graphical interfaces for Ignition projects. | Activated | Running | [More] | [restart] |

On this page ...

- [Module Configuration](#)
- [Install or Update a Module](#)
- [Uninstall a Module](#)
- [Restart a Module](#)
- [Module Status](#)



Installing or Upgrading a Module

[Watch the Video](#)

Install or Update a Module

1. On the Gateway Webpage, select **Config > Modules** to open the **Module Configuration** page.
2. Scroll to the bottom on the list, find the blue arrow, and click the **Install or Upgrade a Module** link.

To **install** a module, choose its `*.mod1` file and press "Install".
To **upgrade** a module, install the new version on top of the existing version.
Modules can be [downloaded](#) from our website.

Choose File No file chosen

Install

3. Click **Choose File**, select a .modl file that you have previously downloaded.

Go to the Inductive Automation [downloads page](#). Search for the module you want and save it to your Downloads folder on your computer.

Third Party Modules

All third party Ignition modules require the Ignition platform to be installed.

To install third party modules:

- Install Ignition: See [Installing and Upgrading Ignition](#)
- Once Ignition is installed, download the module and install it in the Ignition Gateway: See [Installing or Upgrading a Module](#)

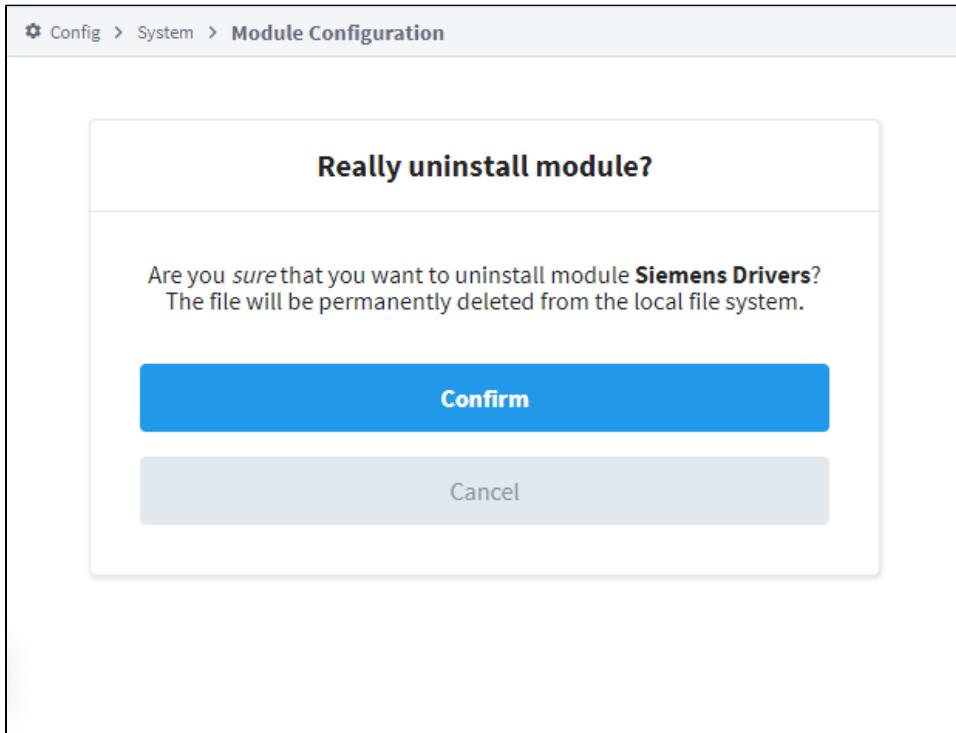
| SepaSoft, Inc. MES Modules for Ignition | | |
|---|---|---------------------------|
| OEEDT Installer Module | OEEDT-Installer-module.modl (8MB) | Version: 1.9.0.1610111600 |
| SPC-Installer-module | SPC-Installer-module.modl (10MB) | Version: 1.9.0.1610111606 |
| Instrument Interface-module | Instrument-Interface-module.modl (1MB) | Version: 1.9.0.1610111601 |
| Recipe-Installer | Recipe-Installer-module.modl (8MB) | Version: 1.9.0.1610111609 |
| Productionsimulator-Installer | Production-Simulator-module.modl (58KB) | Version: 1.9.0.1610111559 |
| Trace-Installer | Trace-Installer-module.modl (7MB) | Version: 1.9.0.1610111555 |
| Web Service-module | Web-Service-module.modl (2MB) | Version: 1.9.0.1610111611 |
| Barcode Scanner-module | Barcode-Scanner-module.modl (61KB) | Version: 1.9.0.1610111610 |

Go back to the **Install or Upgrade Module** screen and browse your Downloads folder for your module, select it, and click **Open**. Ignition will copy the path to the module to the Browse field and all that's left for you to do is click **Install**.

4. Click **Install**. When the page reloads you can now see the module you installed in the list of modules.

Uninstall a Module

1. On the Gateway Webpage, select **Config > Modules** to open the **Module Configuration** page.
2. Scroll to the the module you want to uninstall.
3. To the right of module name, click the **More button** and select uninstall. A confirmation message appears.



- Click **Confirm**. The module is no longer installed and is removed from the list.

Modules are hot-swappable, this means these actions can be performed while the system is running. The isolated nature of modules ensures that performing one of these actions only affects that particular module, and any modules which depend on it. For example, uninstalling the SQL Bridge module will not affect any running Vision module clients.

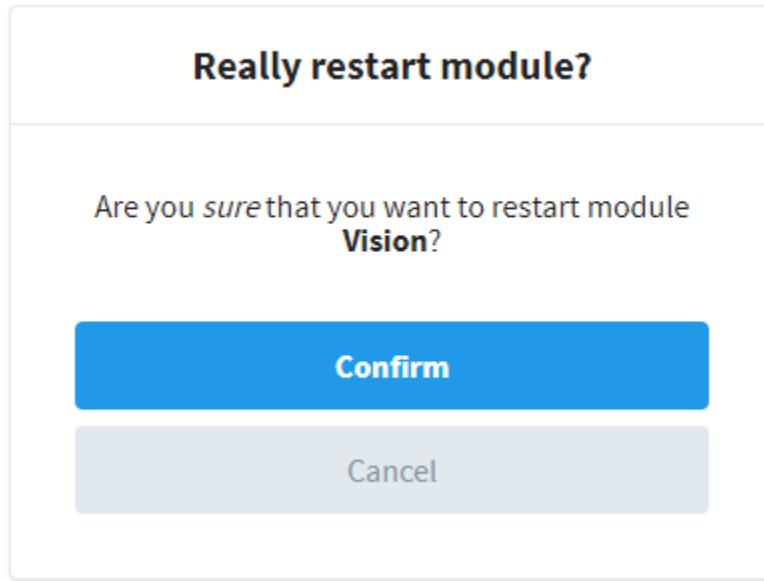
Restart a Module

Because of the isolated nature of modules, the other modules are not affected by the restart (unless they depend on that particular module).

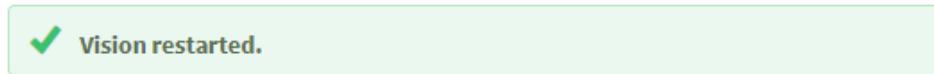
- On the Gateway Webpage, select **Config > Modules** to open the **Module Configuration** page.
- Scroll to the the module you want to restart.
- To the right of module name, click the **restart** button.

| | | | | | | |
|---------------------|----------------------|---|-------|---------|--------|----------------|
| Tag Historian | 3.0.3 (b2019080717) | Turns any database into a powerful historian that can store and drive data in Ignition. | Trial | Running | More ▾ | restart |
| UDP and TCP Drivers | 5.0.3 (b2019080717) | Drivers for receiving and parsing UDP or TCP packets. | Trial | Running | More ▾ | restart |
| Vision | 10.0.3 (b2019080717) | A module that provides web-launched HMI/SCADA clients. | Trial | Running | More ▾ | restart |

- Click Confirm at the confirmation message.



Ignition will give you a confirmation message when the module has restarted.



Module Status

The installed module list on the **Module Configuration** page also provides some basic information about the **State** of the module. The **Version**, **Licenses**, and **State** columns are all displayed in the list. Module licensing is performed centrally in **System > Licensing**, so the values here are only for information purposes.

The module State can be either Running or Loaded:

- Running indicates the module programs are actively performing tasks, for example, the Alarm Notification modules shows as Running because it is continually checking for condition changes that need to set the alarm on or off.

- Loaded indicates the module program or resources are available but nothing is running, for example, the Symbol Factory module makes all the symbol images available for your use.

| Name | Version | Description | License | State | More | restart |
|---------------------------|------------------------|--|---------|---------|-----------------------|--------------------------|
| Alarm Notification | 5.0.3 (b2019080717) | Provides alarm notifications via email | Trial | Running | <button>More</button> | <button>restart</button> |
| Allen-Bradley Driver | 5.0.3 (b2019080717) | Allen-Bradley driver suite for the OPC UA module. | Trial | Running | <button>More</button> | <button>restart</button> |
| DNP3 Driver | 3.0.3 (b2019080717) | A driver supporting DNP3 (Distributed Network Protocol) device. | Trial | Running | <button>More</button> | <button>restart</button> |
| Enterprise Administration | 3.0.3 (b2019080717) | A remote Gateway administration system, allowing you to manage Gateways and automate tasks from a single controller. | Trial | Running | <button>More</button> | <button>restart</button> |

Related Topics ...

- [Licensing and Activation](#)

Ignition 8 Upgrade Guide

Should I Upgrade?

It may not be advisable for all users to immediately upgrade production systems to 8. It is highly recommended that all users who wish to upgrade should first install Ignition 8 on a separate server and validate all of the functionality required by their system before deploying it into production. Please read this guide in its entirety to understand the changes introduced in Ignition 8 and evaluate if they will impact your installation(s).

Expected Downtime

If you are upgrading from Ignition 7.9 or before to Ignition 8, there are some changes that may require a downtime window, even when using Ignition's Redundancy. These changes are covered in the rest of the upgrade guide. For example, for anyone using Vision, all Vision Clients will need to be updated to run using the new Vision Client Launcher, which will need to be installed on each client PC.

Upgrading from 8.0 to 8.1

This section details notes and advisories for users planning on upgrading from 8.0 to 8.1.

Changes to Be Aware Of

Tag Historian Licensing

In 8.1, the Tag Historian module licensing has been adjusted.

1. Licensing for the full Tag History works identically to before. If you have a normal Tag Historian license, and are using the standard Tag Historian functionality, there are no changes.
2. For customers with a Scale-Out architecture, the Tag Historian could be used on the front-end Gateways unlicensed. This would provide a message of "Partially licensed" on those Gateways, even though the configuration was an acceptable architecture with no need for additional licensing. In 8.1, we added the ability for the Tag Historian to be licensed for read-only operations, to eliminate that message. This is available for free for users upgrading from existing licenses 7.9 and 8.0 licenses that have Upgrade Protection by contacting your sales representative at Inductive Automation.
3. A small set of customers use third-party modules that act as a storage engine for Ignition's Tag Historian. Previously, Inductive Automation did not require a Tag Historian license for a module to act as an engine for our Tag Historian (i.e. to use our API for storage and retrieval). Starting in 8.1, a Tag Historian license is needed in order for these modules to function. If you are using one of these modules in 7.9 or 8.0 and you have Upgrade Protection on your license, Inductive Automation will provide a free addition to your license, so no additional cost is required upon upgrade. If you have Upgrade Protection, Inductive Automation will attempt to auto detect if you're running one of these modules to update your license in our system automatically, so you would only need to click the refresh license button in your Ignition Gateway to pull in the change. Some third party modules cannot be auto detected, so call your sales representative if a third party historian module does not work on upgrade and you have an existing 7.9 or 8.0 license with Upgrade Protection, and we will add the necessary parameters to your license.

If your system is one of the few that uses third-party historian engine modules, it is highly advised that you review the system after upgrading to 8.1 and ensure the historian system is working as expected. If there are any issues with historian system after upgrade, and clicking refresh on your license hasn't resolved the issue, please contact your sales representative for assistance.

Gateway Scan of Project Directory

As of 8.1, the Gateway scans the project directory every **five minutes** for changes. This is an important change for users that have a Version Control System with their Gateway. Previously in 8.0, the scan time was every **10 seconds**, which proved to be resource intensive.

Users who want to change this scan rate back to the 10 second frequency can do so by placing the following system flag in the ignition.conf file:

```
-Dignition.projects.scanFrequency=10
```

JxBrowser Library Update

On this page ...

- [Should I Upgrade?](#)
- [Expected Downtime](#)

Upgrading from 8.0 to 8.1

- [Changes to Be Aware Of](#)
 - [Tag Historian Licensing](#)
 - [Gateway Scan of Project Directory](#)
 - [JxBrowser Library Update](#)
 - [Gateway Authentication with Identity Providers](#)
 - [Perspective Script Transforms](#)

Upgrading from Ignition 7 to 8.1

- [Upgrade Process](#)
 - [Using the Installer to Upgrade \(Preferred Approach\)](#)
 - [Importing Projects into a Fresh Install](#)
- [Changes That Require User Action](#)
 - [Look and Feel changes](#)
 - [Redundancy Connection Configuration](#)
 - [Replacement for the system.tag.browseConfiguration Function](#)
 - [OPC UA Certificate Management](#)
- [Discontinued Features](#)
 - [Vision Mobile Module Discontinued](#)
 - [External Database "SQLTags" Providers No Longer Supported](#)
 - [32 bit Support](#)
 - [Project Rollback](#)
 - [OPC UA Server Stale Threshold](#)
 - [Staging and Publishing](#)
 - [OEM Lock](#)
 - [OPC UA Server Exposed Tags Auditing](#)
 - [OPC UA Server "ReadOnly" Role](#)
- [Changes to Be Aware Of](#)
 - [Third-Party Modules](#)
 - [Tag Historian Licensing](#)
 - [Default Security Certificate](#)
 - [Native Client Launchers Replaced](#)
 - [Introducing Ignition Perspective](#)
 - [Introducing Identity Providers](#)
 - [System Commissioning](#)
 - [Security Levels](#)
 - [Expanded Auditing Scope](#)
 - [Java Web Start Discontinued](#)
 - [Embedded Java](#)
 - [Project Inheritance & "Global" Upgrade Logic](#)
 - [Project Resource Storage](#)
 - [Designer Concurrent Editing and Conflict Resolution](#)
 - [Tag System Changes](#)
 - [UDT Performance](#)
 - [UDT Parameter Usage](#)

We updated our bundled JxBrowser library from version 6 to version 7. For most users, this has no ramifications at all.

For users that have intentionally written scripts against the JxBrowser library, those scripts may have broken as a result of the update. Take a look at the [JxBrowser migration guide](#) for more details.

Gateway Authentication with Identity Providers

In Ignition 8.1, Identity Providers are used to authenticate users on the gateway, whereas version 8.0 and prior used User Sources. To make the upgrade process seamless, your newly upgraded 8.1 gateway will create a new Ignition Identity Provider, and point it to the System User Source (under Gateway Settings prior to upgrade). For this to work, the Ignition Identity Provider needs to be able to view the list of users from the backing User Source.

In cases where Active Directory or Hybrid User Sources were set as the System User Source, you'll want to make sure that the list of users from the underlying user source are available. Otherwise, some users may not be able to log into the gateway after upgrading. You can validate this before upgrading by viewing the [Manage Users](#) page. If one or more users are missing from the resulting list, then you'll want to refine how the user source builds that list. Based on the system's user source, you may need to check some settings:

- For Active Directory User Sources, it's a good idea to verify the **User Search** and **User List filter** settings on the User Source, making sure they're filtering on the criteria you expect.
- For Hybrid User Sources, the **List Users from Active Directory** will determine where the list of users is generated from. If the resulting user list is missing users, you may need to toggle this setting.
- For Database User Sources using the "Manual" Mode, make sure the **List Users Query** is able to return the users you'd expect.

If, after upgrading, you're unable to log into the Gateway you can always perform a [gateway password reset](#) to troubleshoot any authentication issues.

Perspective Script Transforms

Arrays in script transforms will now automatically be returned as valid JSON. This change technically modifies the value returned by the transform (i.e., potentially adding quotation marks around string elements), which may impact some existing bindings. If you're making use of Script Transforms that interact with arrays, then you may want to reexamine their configuration after upgrade.

Upgrading from Ignition 7 to 8.1

Ignition 8.1 is a major update across virtually all subsystems and modules. Inductive Automation strives to maintain its commitment to backwards compatibility and a straight-forward upgrade process. However, given the vast scope of changes in 8.1, there are inevitably special considerations that need to be evaluated and actions taken in order to successfully upgrade an existing system. This document seeks to identify those issues and provide guidance for users who are migrating from an Ignition 7 system. To discover the new features of Ignition 8.1, please see the [What's New](#) section.

Note: In some cases, it is not advisable or possible to upgrade from Ignition 7 to 8.1 at this time. Please read this guide carefully before upgrading a production Gateway!

Upgrade Process

When upgrading from Ignition 7 to Ignition 8.1, there are two general approaches.

Using the Installer to Upgrade (Preferred Approach)

Using the Installer to upgrade is the easiest approach. Simply run an Ignition 8 installer on your existing server (similar to how upgrades normally work). As always, we highly advise taking a Gateway backup before upgrading.

Importing Projects into a Fresh Install

We have found that some users prefer to completely uninstall their Ignition 7 Gateways, install a fresh copy of Ignition 8.1, and then manually import projects and Tags from the old Ignition 7 Gateway, one-by-one. We generally **discourage** this practice, as it results in unintended results. We recommend you take the "Using the Installer to Upgrade" approach instead. However, if you must import Ignition 7 resources into an Ignition 8.1 Gateway, then we recommend you use Gateway Backups instead (when possible).

The reason for this, is that a Gateway Backup triggers an upgrade check that looks over the Gateway Backup and makes modifications to the resources so they run as expected on an Ignition 8.1 system. The same check is triggered when running the installer. Project and Tag imports do **not** trigger this check. As a result, you may need to make manual modifications to the imported resources.

- Script Based Tag Editing
- OPC UA Default Configuration Changes
- OPC UA Discovery with Third-Party Clients
- Changes to system.alarm.listPipelines
- SQLite Storage
- More Information

For example, the project system in Ignition 8.1 is drastically different, so running the installer directly on an Ignition 7 Gateway or importing an Ignition 7 Gateway Backup into an Ignition 8.1 Gateway will trigger all of the updates listed under the [Project Inheritance & "Global" Upgrade Logic](#) heading below. Individually importing projects one-by-one will not trigger the upgrade logic, so any legacy "shared" scripts may not work in Ignition 8 without user intervention.

The same could be said for Tag imports. Ignition's built-in OPC UA server connection on Ignition 7 used the name "Ignition OPC-UA Server", but Ignition 8.1 changed the name to "Ignition OPC UA Server", removing the hyphen "-" character. As a result, your Ignition 7 Tags are likely using the old name, while a freshly installed Gateway is using the new name. Thus, OPC Tags imported from an Ignition 7 file will likely break upon import as the OPC Server name won't match, so you'll need to either A) rename the OPC UA server connection or B) find-and-replace the OPC UA server name on all of the Tags.

Changes That Require User Action

These changes will require manual action on the part of the system administrator/project author. In other words, these are features or situations that have a direct corresponding solution, but for various reasons cannot be automatically upgraded.

Look and Feel changes

The general look and feel of the Ignition Designer and Vision Client has been updated. This may result in subtle changes to the appearance of Vision Clients depending on the components used by a given project. It is not possible to revert to the old "look and feel" as it's not compatible with Java 11.

This means screens that have been developed in version 7 may look different after upgrade. Notably, the default font on many Vision components is larger, so you may need to resize the font to match your desired look. Manual configurations to Vision resources, such as manually selecting a font or background color for a component, will be preserved upon upgrade. However, visual properties left with default values may appear differently after upgrade.

Note: It is highly advised to upgrade version 7 projects to version 8.1 on a test server and verify the look of the project, before upgrading the production servers.

Redundancy Connection Configuration

Redundancy communication is now accomplished through a Gateway Network connection between the master Gateway and the backup. This mechanism provides greater security and performance than the previous communication channel, but will require the system administrator to re-establish the connection.

The settings for the communication channel are still configured in the Gateway under Configure>Redundancy, and are still established from the Backup to the Master, but must be modified before the backup will successfully connect to the master after upgrade.

Replacement for the `system.tag/browseConfiguration` Function

The Tags system in Ignition changed dramatically in Ignition 8, which makes the Ignition 7 function `system.tag/browseConfiguration` obsolete. The function has been removed from the code base, so scripts using the old `browseConfiguration` function will break upon upgrade. The functionality has been replaced by [system.tag/getConfiguration](#).

OPC UA Certificate Management

Prior to Ignition 8, Ignition's built-in OPC UA client implicitly trusted the certificate of any server it connected to, and Ignition's built-in OPC UA server implicitly trusted the certificate of any client connecting to it. This is no longer true. New certificate management pages for the client and server have been added to the configuration section of the Gateway under "OPC UA > Security". From this UI, trusted certificates can be imported and quarantined certificates can be marked as trusted. Ensuring the remote certificate is trusted, is required for all secured inbound and outbound connections.

On upgrade, all secured inbound and outbound connections other than the default "loopback" connection will be faulted until the remote certificate is explicitly marked as trusted.

Discontinued Features

These are areas that have been removed or discontinued, as new methodologies have been adopted to replace the functionality.

Vision Mobile Module Discontinued

The Vision Mobile Module is no longer supported in Ignition 8. The new Ignition Perspective Module provides a robust, modern solution for mobile visualization. Users should migrate their mobile visualization needs to be based on Perspective applications going forward.

A few important points:

- The Mobile Module will continue to be supported on Ignition 7.9 for the remainder of its respective Long Term Support (LTS) terms.
- All customers who own the Mobile Module and have current TotalCare Upgrade Protection contracts are entitled to a license of Perspective Mobile upon upgrade.

External Database “SQLTags” Providers No Longer Supported

The “Database Provider” and “Database Driving Providers” are no longer supported in Ignition 8. These systems were very early ways to bridge outside data into Ignition’s Tag system and have been replaced over the years with much easier and faster methods of data integration. If you are still using these features, please contact us to explore migration strategies.

32 bit Support

Support for 32-bit platforms other than ARM has been discontinued.

Project Rollback

The Project Rollback feature has been discontinued in Ignition 8.

OPC UA Server Stale Threshold

The Stale Threshold setting on Ignition’s OPC UA server has been retired. The setting was originally used in earlier versions of Ignition as a means to help some of our drivers determine when values should be marked as stale. Our drivers have since gained their own methods to determine staleness. The setting has little use in Ignition 8, so it was removed.

Staging and Publishing

Ignition 8 features a completely revamped project system. This new project system does not have different “staging” and “published” versions of each project. When you upgrade from Ignition 7, any projects that were split into separate staging and publish versions will have their staging versions imported into Ignition 8. This is to prevent any loss: the staging version is always at least as and usually more up-to-date than the published project.

Staging and publishing was not included in Ignition 8’s project subsystem for two reasons:

1. Staging and Publishing was not consistently implemented across various modules, leading to confusion about which types of resources participated in it.
2. Projects are not self-contained; they rely on tags and external connections to devices, databases, etc. For this reason, the staging and publishing feature was only effective in limited circumstances.

You can emulate the old staging and publishing system by simply having a “staging” copy of your project and either move resources between projects or copy & overwrite them in order to “publish” your changes.

OEM Lock

The “OEM Lock” feature that was used to encrypt project resources by authors is not currently supported in Ignition 8. Project authors that used this feature may migrate their projects to 8.0, as the owner license will decrypt Ignition 7 resources, but they should be aware that the resources will currently remain unencrypted.

End users who upgrade OEM-Locked systems to 8.0+ will not be able to view the project resources.

OPC UA Server Exposed Tags Auditing

When a Tag made available via the “Exposed Tags” feature of Ignition’s OPC UA server is written to by an OPC UA client, the write is not currently entered into the audit log.

OPC UA Server “ReadOnly” Role

The undocumented “ReadOnly” role no longer has any effect.

Changes to Be Aware Of

Ignition 8 has many additional changes and improvements that system administrators should be aware of. Although none of them present a significant issue, some may directly affect end users (for example, launching clients), and some can be leveraged to improve the security and performance of your system.

Third-Party Modules

Please consult individual module authors concerning upgraded module availability. Modules built for Ignition 7 **will not work** in Ignition 8 without modification.

Tag Historian Licensing

See the Tag History Licensing changes under the [Upgrading from 8.0 to 8.1](#) section.

Default Security Certificate

Ignition installations no longer come with initial security certificates.

Native Client Launchers Replaced

The Native Client Launcher has been replaced by the [Designer Launcher](#) and [Vision Client Launcher](#), which offer similar functionality. Due to the aforementioned "Default Security Certificate" change, secure connections between the launchers and Gateway will require valid certificates are in place.

Introducing Ignition Perspective

Ignition Perspective is an entirely new visualization system, built from the ground up to allow you to create modern, high performance, and highly functional pure-web applications that can be viewed anywhere. Learn more about Perspective by visiting the [Perspective pages](#) in the User Manual, or the [Inductive University](#).

Using Perspective without Upgrading

The Ignition Perspective module is one of the most exciting new features of Ignition 8, offering an entirely new and innovative way to visualize data and deploy applications to mobile devices. Many users will want to begin leveraging this module right away, and will feel compelled to upgrade for that purpose.

It is important to understand how various Ignition features, in particular the Gateway network and remote services, can be leveraged to allow immediate usage of Ignition Perspective without the potential risk or disruption necessary to upgrade a running production Gateway.

As part of the base platform, the Gateway Network allows you to create connections between Gateways, including between 8 and 7. The Remote Tag Provider allows you to use Tags from one Gateway on another, including both realtime and historical values. Remote Tag Providers created in Ignition 8 targeting Ignition 7 Gateways will support reading, writing, real-time subscriptions, and Tag history queries. You cannot edit or modify the Tags.

Therefore, to develop and deploy Perspective applications with minimum impact on your production system you can:

1. Install Ignition 8 onto a different computer or VM instance.
2. Limit the installation to the Perspective module (though, of course, other modules can be used).
3. Create a Gateway network connection between the Ignition 8 Gateway and the Ignition 7 Gateway.
4. Create a Remote Tag Provider on the Ignition 8 system for each provider you'd like to access.

Unless previously changed, remember that the Ignition 7 Gateway will employ the default "service security" settings for the remote Tag Gateway network service. This will allow read only data, but if you want to support writes, you'll need to change the policy by going to [Configure>Security /Service Security](#) in the Gateway.

Introducing Identity Providers

Ignition 8 adds support for connecting to SAML and OpenID Connect Identity Providers. In addition, Ignition can act as an Identity Provider, using a User Source as a backing source of users. On upgrade, a new Ignition Identity Provider will be created that points to a single user source, allowing users in that user source to authenticate against the Identity Provider. You can learn more about this on the [Configuring Identity Providers](#) page.

For more information, see the [Gateway Authentication with Identity Providers](#) under the Upgrading from 8.0 to 8.1 section.

System Commissioning

Upon installing Ignition, the user is now presented with a "system commissioning" web based walkthrough. This process must be completed for new installations, and requires that an initial user is created for the newly commissioned Gateway.

As of Ignition 8.0, there is no longer a pre-defined default "admin" user. System administrators should choose a secure root username and password for optimal system security.

Security Levels

Ignition 8 introduces the concept of [Security Levels](#), which are abstracted attributes that can be given to users, and used as a means to restrict access to different parts of the system. Security Levels can be assigned to users based on User Roles, Security Zones, a combination of both, or neither, as the Expression Language can be used to determine which level any given user falls under.

Many subsystems in Ignition will automatically migrate over to security levels upon upgrade, such as Tag read and write permissions.

Expanded Auditing Scope

Ignition's built-in auditing system was very limited in version 7, recording only a handful of events. Most events that registered in the system had to originate from a project, so changes made at the Gateway level were largely ignored, save for Tag writes.

In Ignition 8, the auditing system's reach has broadened greatly. The system now accounts for configuration changes made at the Gateway level. Check out the [Auditing Actions Reference](#) page for a full listing of actions that are audited.

Java Web Start Discontinued

Support for Java Web Start has been discontinued by Oracle and the OpenJDK community as of Java 11. This technology has been used in Ignition since its original release to launch the Designer and Vision clients, though we have also offered alternative launching tools for years.

In anticipation of this change, the designer and client launching tools have been greatly improved in Ignition 8. In addition to a fresh and more functional UI, the launch tools now manage the Java runtime environment used by the client (see next point), and can help you track many projects and Gateways. There are separate launchers for Vision Clients and the Designer, representing the natural distinction between the target users of these products, and the launchers can be downloaded for Windows, Mac, and Linux directly from the Gateway homepage.

Java Web Start is no longer supported, please use the Designer and Client launchers to launch those applications.

Embedded Java

Ignition no longer requires Java to be installed on the Gateway or client machines. Native versions of the Ignition installer and launchers have been made available for 64-bit Windows, Mac, and Linux, as well as ARM, for use in edge devices. These distributions include embedded JREs (Java Runtime Environment) that are provided by Inductive Automation and managed by Ignition.

The native client launchers manage retrieving the current version of the JRE as necessary for the client and designer runtimes. Updates to the embedded JREs are managed by Inductive Automation and are made available through updates to Ignition. The client launchers will automatically update their local versions as necessary, ensuring that Vision Clients and the Designer are always running the latest version available on the Gateway.

Ignition no longer requires Java to be installed on server or client machines. System administrators should monitor Inductive Automation for important system and security updates, and can sign up for important notifications at <http://support.inductiveautomation.com>

Project Inheritance & “Global” Upgrade Logic

The project system in Ignition 8 has been improved and now features a more flexible inheritance model. Each project may have a “parent” project, and will inherit all of the resources of that parent project. That project may in turn have its own parent project, and so in this way complex hierarchies of reusable resources may be designed. Within each project, inherited resources may be used by other, “local” resources. For example, an inherited Vision template could be embedded in a window, or an inherited script could be executed by a button. Inherited resources may also be overridden, allowing them to be re-defined when needed. Conversely, resources in parent projects may be marked as “not overridable” to ensure that children cannot alter them, when appropriate.

This new inheritance system makes Ignition 7’s “global” project concept obsolete. In order to maintain backwards compatibility, the following will happen when an Ignition 7 Gateway is upgraded to Ignition 8:

- The resources from the Ignition 7 special “[global]” project will be copied into an Ignition 8 standard, inheritable project called “global”.
- All other projects will have their parent project set to this new “global” project.
- All “shared” scripts are now simply project-level scripts in a top-level script package called “shared”, stored in the new “global” project. This allows all existing scripts to function without edits.
- All “project” scripts are now simply project-level scripts in a top-level script package called “project”.
- All Vision templates stored in the old global area are moved into a folder called “SharedTemplates/” in the new global project. The Vision module will automatically convert a template path like “[shared]MyTemplate” as “SharedTemplates/MyTemplate” so that no further action is needed.
- If any run-always sequential function charts are present in the global area, a new project called “run-always-charts” will automatically be created upon upgrade, and these charts will be moved into this project. This is because resource types that execute automatically like run always SFCs and Transaction Groups do not run in inheritable projects.
- If any alarm pipelines exist, a project called “alarm-pipelines” will be created and all alarm pipelines will be moved into this project. Tags can target pipelines in any project, but in the absence of a specific project, the pipeline is assumed to be in this special upgrade project.
- There is a new Gateway-wide setting called “Gateway Scripting Project”. This setting lets you specify one project which all scripts that are outside the scope of a project will execute against. For example: Tag Change Scripts and Tag expressions using the runScript expression. On upgrade, this will be set to the new “global” project so that Tag scripts can continue to access shared.* scripting functions.

Project Resource Storage

The new project subsystem in Ignition 8 uses a new file-based storage strategy, instead of the internal-database strategy used in Ignition 7. This has several significant benefits over the previous methodology:

1. All resources can now be tracked with industry standard source control tools, such as Git. Resources can be tracked individually, and changes to disk caused by source control operations (checkout, pull, etc.) will be applied automatically to the running system.
2. Project resources can be transferred between systems with simple file system operations. System administrators can use a wide range of tools to synchronize all types of resources between Gateways.
3. Project backups are simply standard zip files organized in the same way as the file system.

Designer Concurrent Editing and Conflict Resolution

Ignition 8’s Designer uses a new lock-free strategy for handling concurrent editing. We believe that this strategy is significantly more robust and will allow for more designers to work together on projects.

The Ignition 7 locking strategy (besides being frustrating when users would lock resources and then go home for the day!) is incompatible with the new file system storage strategy. Since changes can occur in the file system at any time, it isn’t possible to maintain exclusive edit locks in the Designer.

In Ignition 8, you may open and modify any resource in your project. You will be notified if the system detects that you and another user are both editing the same resource, or if the resource you’re editing has already been modified remotely. If you modify the resource anyway, and someone else has modified it first, this is called a conflict. The system automatically detects conflicts when you try to save. If any are detected, you are given the opportunity to resolve these conflicts by choosing whether to use your changes, accept the changes that happened elsewhere, or cancel your save and figure out what to do in another way.

Tag System Changes

The Tag system has undergone significant changes in Ignition 8. The majority of these changes should go unnoticed upon upgrade (apart from obvious UI changes), and most systems should immediately benefit from improved performance, faster edits, and expanded functionality. To summarize:

- "Scan Classes" have been renamed to "Tag Groups". Functionally they're identical. Scan Classes on Ignition 7 systems will gracefully migrate to Tag Groups upon upgrade.
- The Tag Editor has been gone through a major UI rework, but remains as easy-to-use as ever.
- Tags are fundamentally defined via JSON structure now. A new right-click menu has been added to the Tag Browser, allowing you to edit the raw JSON of a given Tag. Additionally, Tag exports/import files now support a JSON format.

UDT Performance

The Tag system has been redesigned to optimize for UDT oriented design. In Ignition 7, certain aspects of UDT performance led some system designers to favor standard Tags over UDTs for some applications. Such systems should be re-evaluated in Ignition 8, as use of User Defined Types will lead to much more efficient memory usage and lighter weight change processing.

UDT Parameter Usage

As mentioned in the section [Changes That Require User Action](#), the way that Parameter references are applied to Tags has changed. Instead of being replaced in a "pre-compilation" stage of Tag execution, they are now treated as true property references. This means that they can be modified, and those modifications propagate dynamically to referencing properties, causing those to be updated in turn.

An important consequence of these changes is that syntax for parameter references in Expressions has changed in a subtle, but crucial way. In Ignition 7, due to the replacement strategy, the following would have been a valid expression:

```
"The value of the parameter is: {ParamRef}"
```

In Ignition 8, this value would no longer be a valid reference. It is counter-intuitive to all other uses of reference in Ignition, and has been replaced with a more familiar syntax:

```
concat("The value of the parameter is: ", {ParamRef})
```

All existing references will be upgraded automatically to the new syntax upon system upgrade. Designers should be aware of this change when creating new Tags.

Script Based Tag Editing

The various scripting functions for editing Tags (`system.tag.editTag`, `system.tag.addTag`, etc.) have been deprecated and replaced by `system.tag.configure`. The previous functions will continue to work, but designers should learn the new function, which is far more intuitive and powerful than the previous functions.

Tags are now defined as JSON objects, which consist of properties, arrays, and sub-objects. The `system.tag.configure` function can take either a String document definition, or a JSON object that defines one or more Tags. Overrides for UDTs are created by simple redefinition of properties, and complex structures like Event Scripts and Alarm configurations will be merged with inherited definitions.

OPC UA Default Configuration Changes

New installations have some new OPC UA related defaults:

- The server binds only to localhost (i.e. no remote connections allowed)
- The server uses port 62541 instead of 4096
- The server only allows secured connections (SecurityPolicy Basic256Sha256)
- The default "loopback" connection is called "Ignition OPC UA Server" (there is no longer a "-" between "OPC" and "UA")

Upgrades are not affected by these changes; they retain their existing configuration.

OPC UA Discovery with Third-Party Clients

Ignition's OPC UA server disallows anonymous access on new installations, which can make the server endpoint difficult to discover by UA clients. To help with this, Ignition's UA server runs a separate unsecured endpoint to allow discovery. The discovery endpoint is accessible by appending `/discover` to the end of the URL:

```
opc.tcp://1.2.3.4:62541/discovery
```

Changes to `system.alarm.listPipelines`

The `listPipelines` function was changed to search a single project for pipelines. Upgrading from Ignition 7 to Ignition 8 will migrate all alarm pipelines into a project named "alarm-pipelines". To maintain backwards comparability, `system.alarm.listPipelines()` will check for a project named "alarm-pipelines", but now accepts a project name as a parameter, allowing the function to check for pipelines in any project.

SQLite Storage

Gateways can now create SQLite databases, or "Internal databases" for many of the storage subsystems (Database Connections, Historian, Alarm Journal, and Auditing), allowing your gateway to store and retrieve data without needing to connect to a remote SQL database, which is useful for development systems and demonstrations.

Note: We still recommend external SQL databases are used in production systems, as they generally offer a level of performance and redundancy not available to SQLite databases.

More Information

To learn more and to stay up to date on updates to Ignition 8, check out the following resources:

- Inductive Automation's website: many product pages have been updated with information about Ignition 8 and the Perspective Module.
 - The Blog has new information concerning updates to Ignition 8.
 - The downloads page has been modified, and will offer new nightly builds of the latest Ignition 8 updates, in addition to stable and RC versions.
- support.inductiveautomation.com
- forum.inductiveautomation.com - The online forum is a great place to provide feedback and to learn about what other users are encountering with Ignition 8.
- [Inductive University](#) has many new videos covering a wide range of Ignition 8 topics. New videos will also be released regularly.
- This User Manual is being continually updated with new content as Ignition evolves. You can check out the updates for each new version on the [New in this Version](#) page.

Related Topics ...

- [Installing and Upgrading](#)

Support, Feedback, and Resources

Support Team

Can't find it here? Go to the [Support homepage](#) for one-on-one help from our support team. Submit a Ticket, and one of our Support Engineers will follow up with you quickly.

You can reach us during business hours 6am-5pm Pacific Time at 1-800-266-7798. Support charges may apply. 24-hour support is also available, at an additional fee.

Email support is available at support@inductiveautomation.com.

Long-Term Support (LTS)

At Inductive Automation, we are constantly improving our software, both in features, functionality and reliability. We provide Long-Term Support (LTS) for certain versions of the Ignition software for a period of five years from the date of its original release. The current release branch and non-expired LTS releases fall into the Active category. This means that the software is being actively developed/patched and the IA support team can provide full customer support with issues related to these releases.

For those Ignition versions that expired, there are no new releases or patches for any issues that may arise in expired versions. You can see in the table below that LTS is currently provided for versions 7.8 through 8.0, and versions 7.7 and prior, new development and patches are discontinued.

The table below shows the original release dates, LTS status, and end date of Active support status.

| Version | Date Released | Is LTS | Date Active Support Lapses |
|---------|---------------|--------|----------------------------|
| 8.1 | Nov-02-2020 | Yes | Nov-02-2025 |
| 8.0 | Apr-08-2019 | No | Nov-02-2020 |
| 7.9 | Dec-12-2016 | Yes | Dec-12-2021 |
| 7.8 | Oct-15-2015 | No | Dec-12-2016 |
| 7.7 | Jul-16-2014 | Yes | Jul-16-2019 |
| 7.6 | May-7-2013 | No | Jul-16-2014 |
| 7.5 | Jun-14-2012 | Yes | Jun-14-2017 |
| 7.4 | Mar-29-2012 | No | Jun-14-2012 |
| 7.3 | Oct-5-2011 | Yes | Oct-5-2016 |

To learn more about our product support in detail, refer to our [Support Policy](#).

Resources

Inductive Automation Blog

Our [company blog](#) features announcements, Ignition tips, and a more personal look at the folks behind the scenes of Inductive Automation.

Knowledge Base

In our [Knowledge Base](#), you can search and view articles created by our support team for troubleshooting, known problems, and workarounds.

Community User Forum

Our [User Forum](#) is *the* place to get help from our passionate community of Ignition users. The Forum is one of the most effective ways to get help from others. It is always available and is actively patrolled by Inductive Automation staff and many knowledgeable users. Chances are you will find your question already answered in an existing post, but if not you can sign up and ask away.

More Resources on our Website

- [Ignition How-Tos](#): Step by step guides
- [White Papers and Articles](#): What's happening in the industry and with Inductive Automation

On this page ...

- [Support Team](#)
 - [Long-Term Support \(LTS\)](#)
- [Resources](#)
 - [Inductive Automation Blog](#)
 - [Knowledge Base](#)
 - [Community User Forum](#)
 - [More Resources on our Website](#)
- [Comments, Feedback, or Ideas?](#)
 - [Feedback on the User Manual](#)
 - [Feedback on the Ignition Software](#)
 - [Feedback on the Inductive University](#)
 - [Other Feedback](#)

- [Webinars on Demand](#): Watch recordings of our past live webinars
- [Tip Sheets](#): Tips and references put together for you

Comments, Feedback, or Ideas?

We encourage your feedback on both the Ignition Software and Ignition User Manual. We strive to keep in step with our community and grow the software to meet your needs. If you have something you think we should add or improve, let us know!

Feedback on the User Manual

We encourage you to give us feedback on the Ignition wiki User Manual as this will help us improve the documentation over time. You can email us at training@inductiveautomation.com.

Feedback on the Ignition Software

How can we improve Ignition? Our goal is to provide you a powerful yet easy-to-use SCADA/HMI software that does everything you need. Let us hear from you on our [features and ideas](#) page. We love to hear any ideas you have so we can continue to improve Ignition.

Feedback on the Inductive University

Do you have a question about any of our videos? You can email iu_support@inductiveautomation.com for answers or suggestions on new videos and improvements.

Other Feedback

Do you have something else to tell us? Not sure where to send it? You can send an email to us at training@inductiveautomation.com, and we will get you pointed in the right direction

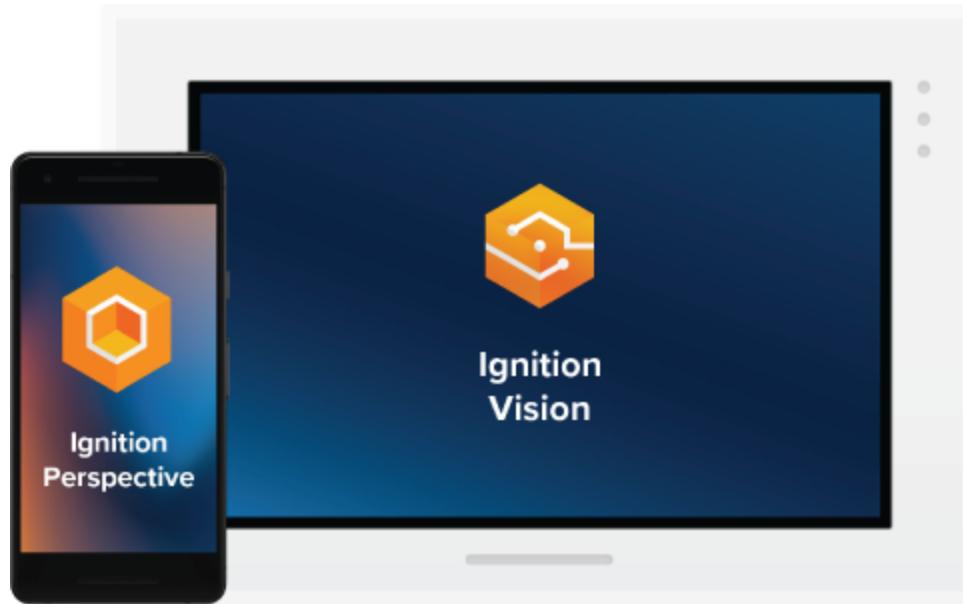
Related Topics ...

- [Installing or Upgrading a Module](#)

Perspective and Vision — Which Visualization System Is Best for Me?

Ignition's Perspective Module and the Vision Module each enable you to rapidly develop industrial applications for your site. Perspective is innovative, supporting mobile-first projects and applications optimized for web browsers. Vision is a time-tested solution for designing traditional plant-floor displays, HMs, and desktop screens. With the addition of Perspective Workstation in release 8.1, you can now run Sessions as standalone, kiosk type displays in Perspective. In addition as of 8.1, Vision and Perspective both support security with Two-Factor Authentication (2FA) or federated identities for the application you're building.

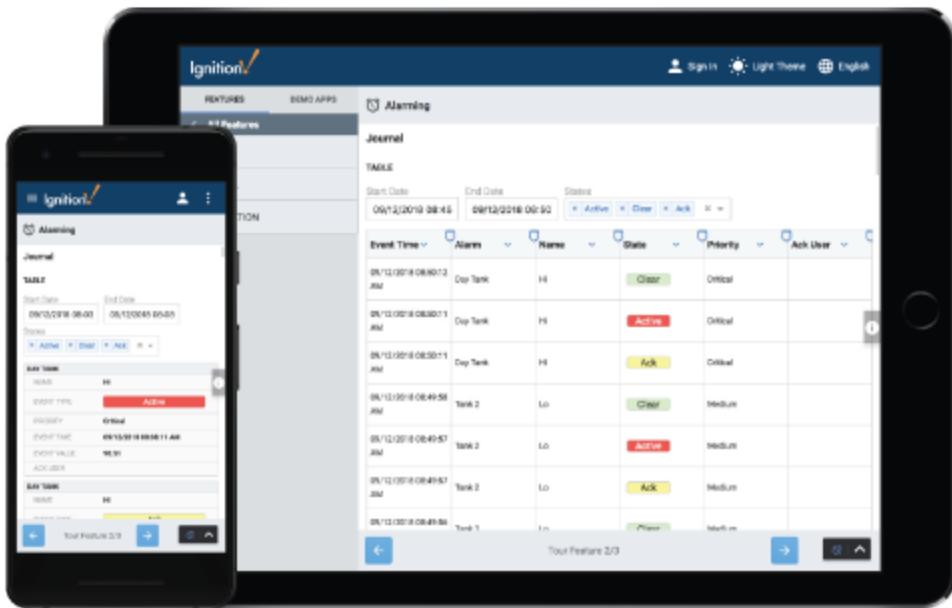
When you start a new project, there are many things to consider, such as who the users will be, how much data you need, what kind of time and resources you have to work with, visualization needs, and so forth. Deciding whether to use Perspective or Vision (or both) really comes down to which options best fit your project at the visualization level.



When to Use Perspective

If you need to build mobile-responsive applications, then Perspective is the recommended way to go. If your application needs to run on a mobile OS, namely iOS or Android, then choose Perspective. Phones, tablets, and other mobile devices can easily run Perspective Session. New in Ignition 8.1, [Perspective Workstation](#) runs in Windowed mode and also has built-in features to run in full screen Kiosk mode eliminating any distractions from the underlying operating system. Perspective also runs in web browsers on desktop PCs. If you're building an application that needs to run on phones, browsers, and desktop applications that don't have a strong need for OS resource access, Perspective is a great choice.

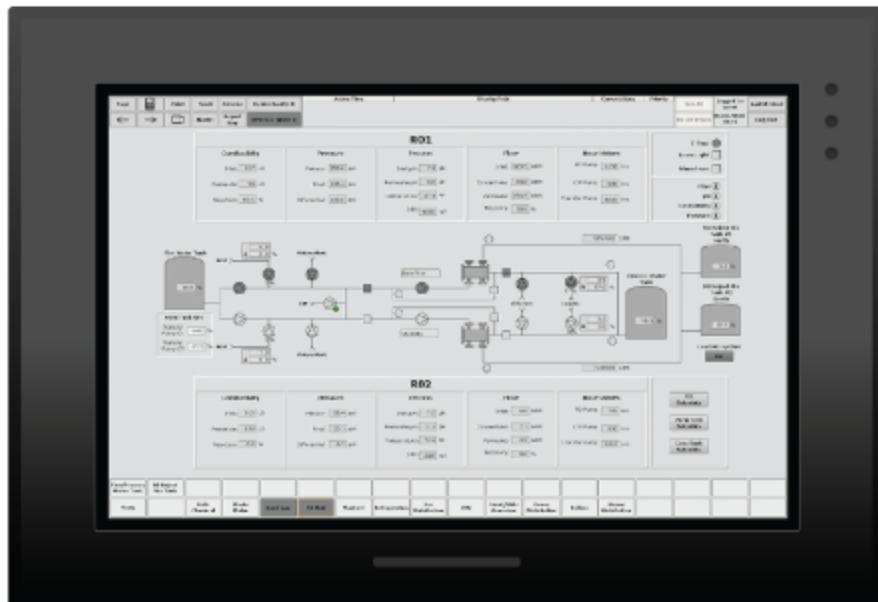
With Perspective, you can automatically adapt to fit any screen size using mobile-optimized container types. It provides the ability to use your device's sensors and intuitive touch commands, as well as message handling, flexible property bindings, CSS3 styles, and more. For example, with an application built in Perspective, you could augment alarm information by taking a picture of the faulty machine. You could scan a QR code by a machine and have the SCADA system tell you what to do, or take readings or photos in the field and tag them with GPS and other sensor data. There are so many possibilities.



When to Use Vision

Vision is and has been for many years a top-class desktop application. Vision is a great choice for traditional industrial plant-floor and desktop screens, standalone HMIs, and the like. If you need OS resources, file access, serial port access, or custom Java code running directly in the client, and you're looking for visualization through a desktop application, choose Vision.

In Ignition 8, the Vision Module got one of its biggest updates since it was first released in 2010, allowing you to make your HMIs and plant-floor dashboards faster, easier to deploy, and more beautiful than ever.



Should I Upgrade?

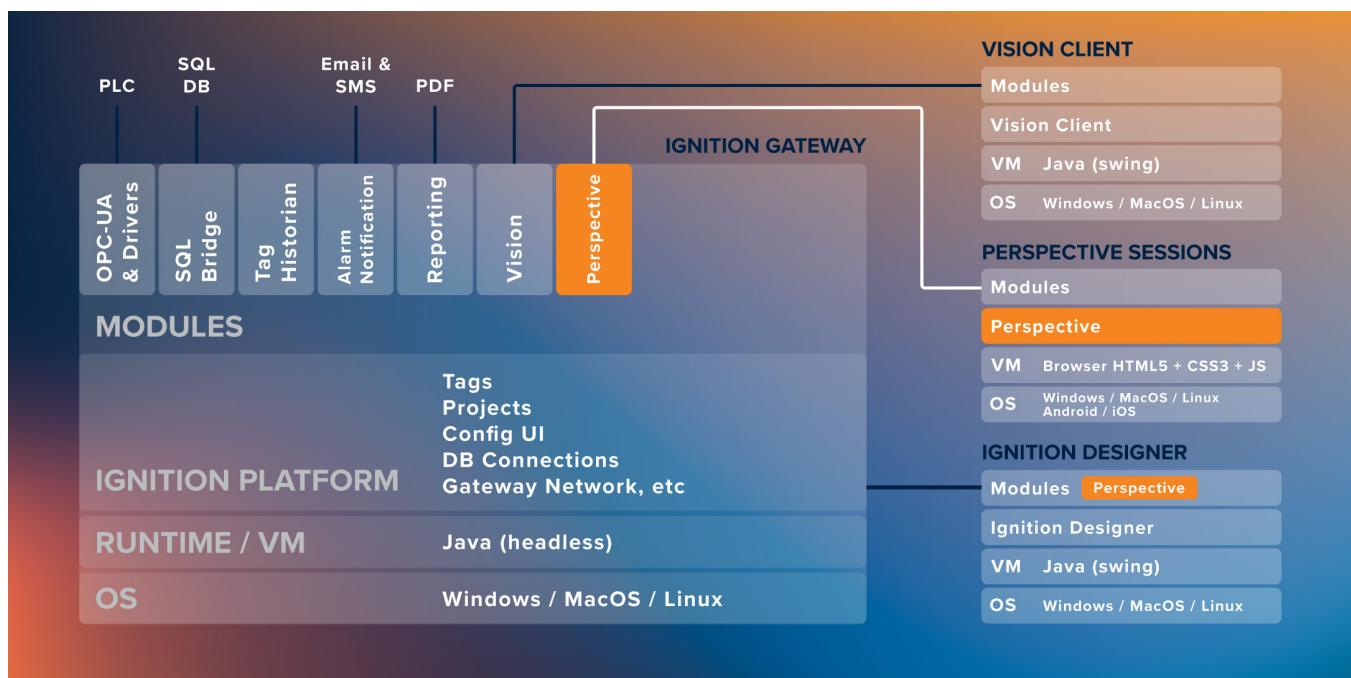
There are many considerations for using Perspective or Vision. Perspective contains familiar elements like the drag-and-drop Designer, property binding system, and scripting engine. Refer to the [Ignition 8 Upgrade Guide](#) for complete information. See also [A Vision-Oriented Guide to Perspective](#).

Behind the Scenes

A "Client" in Vision is called a "Session" in Perspective, and the two actually have very similar tech stacks. At the bottom of the stack is the OS. In addition to supporting Windows, macOS, and Linux, Perspective also supports Android and iOS. The next part of the stack is the web browser, which is very similar to the runtime/VM layer in the other stacks because it provides an environment to execute code, generate user interfaces, and communicate back to the Gateway. On top of that runs the Perspective session application itself, and then modules can augment the features of Perspective.

In the topology of a typical Ignition installation, there's the Ignition Gateway, the Ignition Designer, and Vision clients. At the base of the Gateway is the operating system, which can be Windows, Mac, or Linux. Above the OS is the Java runtime/VM layer, which is what Ignition runs on. Above the runtime/VM layer is the Ignition platform, which provides many features and services like Tags, projects, database connections, the Gateway Network, and so on. At the top of the tech stack are modules, which let you do a number of useful things like connect to devices, use databases effectively, send alarm notifications, generate reports, and, in the case of the Vision Module, create applications which run as Vision clients.

As you can see, Perspective is just another module. It installs into the Gateway and provides a design experience in the Designer just like other Ignition modules do.



Related Topics ...

- [Perspective](#)
- [Vision](#)

A Vision-Oriented Guide to Perspective

An Intro to Perspective

Perspective is a flexible and versatile approach to HMI/SCADA design. Whereas Vision clients run on any platform that supports Java, Perspective Sessions run anywhere that supports a web browser. Additionally, sessions can run in the [Ignition Perspective App](#). This opens the door for a new level of support for mobile devices and tablets. Perspective's features follow a responsive design paradigm. This added dynamic and flexibility means you'll need put a little more planning into your projects.

The purpose of this guide is to describe the core functionality of Perspective in ways that make sense for Vision users.

The largest difference between Vision and Perspective is the way they do layout and project design. Perspective is primarily designed to allow you to create mobile responsive interfaces. Perspective is an Ignition module, using the same platform as the Vision module. It has access to all of the same systems like Tags, database connections, reports, and security (though security got a few updates in Ignition 8 too).

On this page ...

- [An Intro to Perspective](#)
- [Perspective Design Considerations](#)
 - [Designing Your Main Views](#)
 - [Docked Views and Pages](#)
 - [JSON Component Properties](#)
 - [Bindings and Transforms](#)
 - [Inherited Projects](#)
 - [Components](#)
- [Comparison of Perspective and Vision Functionality](#)
- [New Features in Perspective](#)

As we developed Perspective, we took the familiar aspects of using the Ignition Designer and provided more tools and flexibility. You will find many similarities between how you design a Vision project and a Perspective project. Anyone familiar with Vision can start designing quickly. For example, you are still working with containers, components, properties, bindings, Python scripting, Tags, and databases. While it is visually different from Vision, you will find the interfaces in Perspective familiar, but updated.

In order to take full advantage of the new systems in Perspective, there are a few things that you need to think about in a different way from your past Vision projects. But first, there are a few terms that we need to define in Perspective. We will be using them a lot to talk about the differences, so please familiarize yourself with them before continuing. A more complete set of terms is provided [below](#).

- **View:** Think of this as a Vision Template, Window, and Container all rolled into one. You can put components in it, you can pass parameters into it, and you can nest them inside each other. Each view has a Layout type.
- **Layout:** There are several types of containers in Perspective. Each has its own set of position attributes for the components inside it, which is similar to the Vision Relative vs Anchored constraints. There is much more than just X, Y, width, and height now.
- **Page:** This is a new concept for Perspective. Instead of having a Vision Client with multiple windows open at the same time, you open a single Page at a time and navigate by switching pages. Each page has its own main View and any docked Views you want.
- **Session:** A [session](#) is the Perspective equivalent to a Vision Client, except it runs in the browser instead of using Java. You can have multiple browser tabs open using the same session.
- **Style Classes:** [Style Classes](#) are style configurations, such as text color and size, and margins. Styling on components in Perspective utilizes CSS, and Style Classes are user defined configurations that allow you to quickly add several styling rules to a component. Style Classes are exhibit inheritance, so making changes to a Style Class configuration will propagate those changes down to any components using the style class.

Perspective Design Considerations

With Perspective, it's more important than ever to have a plan before you start designing. Here is a short list of the things you want to think about before starting to build your visualization system. Note that this assumes you already have your Tags, database, and other Gateway items taken care of.

- Make a flowchart of your project. How will users navigate, and what is the tree structure for your pages and popups?
- Make a visual outline of each page. What docked windows do you want? What will your navigation look like?
- Get an idea of what windows you want, and how you want them to look.
- Will your project be mobile responsive? Plan the look for both a large and small version of each window.

Once you have an idea of what your session will look like, you can start designing.

Designing Your Main Views

Deciding on headers, tab strips, navigation trees or other methods of navigation is just a start in Perspective.

If you want to **keep things simple**, you can use a [Coordinate View](#). This will feel extremely similar to a Vision window since all the components have an X, Y, width, and height. Just drag your components onto the View and use the handles to stretch them to the size you want. By default, all components will be locked in using those location properties. These views will behave similar to a Vision window with all components anchored only to the top and left. You can change the Mode property in the root container to "Percent" to make the components behave similar to the Relative layout mode in Vision. Many will use this with a single docked view for navigation. This will create a project very similar to the Vision Single Tier Nav project template.

Views can be **nested** inside each other to create more complex structures. If you want a more structured layout, you can use the [Flex View](#) with other views inside it. A Flex View creates a row or column structure out of your components. For example, you could use a Flex View with a header at the top, and a coordinate container filling the rest of the view. This is another simple structure that creates a similar structure to a Vision window with an anchored header and relative layout components in the main space.

If you want to make your project **mobile responsive**, there are other types of Views that you can use. The [Column View](#) allows you to add your components, then arrange them differently for three unique sizes of screen. The [Breakpoint View](#) allows you to show two completely different views based on the size of the screen.

And of course, you can nest all of the different types of views in any combination you want. However you choose to set up your project, you will want to figure out which containers and combinations of containers will give you the look you want. Try looking at your favorite webpages for inspiration, there are a lot of great (and terrible) User Interfaces out there to give you ideas. For example, you can use a Breakpoint View that when small, contains a Flex View that is a single column of components, and when large, contains a Coordinate View that shows a diagram of your entire facility.

Docked Views and Pages

A session, much like a client, is made up of more than one view, and you will often see multiple views at the same time. In Vision, you have multiple windows open, and certain windows are designed to be docked or floating. In Perspective you have [Pages](#) instead. A Page is a view or set of views that open and close together. Most commonly, a page will have a main View and a docked view.

Navigation in Perspective means moving from one page to another, not opening and closing multiple windows. This allows you to use the forward and back buttons on your browser as a part of your navigation strategy.

There is a shared space that all pages inherit from, so you can set up docked views there, and any page that is open will automatically have those docked views open as well.

There are many settings for your docked views. You can set these views to be expandable, auto hide depending on session width, and a number of other options.



**INDUCTIVE
UNIVERSITY**

**Docked Views in
Perspective**

[Watch the Video](#)

JSON Component Properties

Perspective components have a slightly different type of [property](#). In Vision, we used a flat property structure where everything was a basic data type like integer or string, except for datasets. If you wanted a dynamic number of properties, you had to create a dataset, then use scripting to pull the values out. In Perspective, all of the properties on a component or container are JSON objects.

This means the properties have different types. They can be a Value (int, float, string, etc), an Array (a numbered set of sub-properties), or an Object (a complex set of sub-properties with key:value pairs). For arrays and objects, the sub members can be any of the three types.

Datasets can now be stored on components as an array of objects, where each object is a row, and each object has several values that make up the columns.

Bindings and Transforms

There are more options for [bindings](#) that were not possible in Vision, and this is only partly due to the new property structure. For example, there is an HTTP binding that allows you to directly connect to a web service and download or upload data.

On top of that, any binding can have [transforms](#) on it. A transform is an expression, map, or script that takes in the value of the binding and changes the output. If you chain together multiple transforms, the output of the previous transform is in input of the next. Think of number-to-color translations. Instead of creating a custom property with an expression on it and then binding a color to that custom property, you can do it all in one binding. Just create an expression binding, then add a map transform to change the output value to a color.

Inherited Projects

Another design consideration is [inheritance](#). Outside of the look and feel of a perspective session, in Ignition 8 you can have multiple projects that inherit resources. If you have views or scripts that you want every project to be able to use, you can create a master project and tell all other projects to inherit from that. You can then use the resources or allow designers to overwrite them as needed.



**INDUCTIVE
UNIVERSITY**

Project Inheritance

[Watch the Video](#)

Components

Because of all the differences between Perspective and Vision, the two modules necessarily have a different set of components. There is no plan to ever have a one-to-one parity between components, and not all of the [Vision components](#) will make their way into [Perspective components](#).

Some components are new to perspective like the Menu tree and the Link components. The Link component was not feasible in Vision, but with Perspective Sessions living in a browser it makes sense to have that component.

The library of components in Perspective is growing continually. For example, an Alarm Status Table similar to the one in Vision is currently under development. Make sure to go to the [forum](#), the [ideas portal](#), or download a [nightly build](#) if you want to see what is coming up.

Comparison of Perspective and Vision Functionality

Here is a quick comparison of summary of basic Perspective concepts, and how they are similar to / differ from their Vision analogs.

| Perspective Concept | Similar Vision Concept | Perspective Description |
|---------------------|--|--|
| Session | Client | <p>A Perspective Session is a running instance of an application, much like a Vision client. Whereas Vision clients run as independent Java programs on a user's machine, a Perspective session runs natively in a web browser (or the Logon Perspective App).</p> <p>Notably, a session can run across multiple pages in the same web browser. This is similar to how sessions function elsewhere on the Internet: log into your favorite shopping website and then open several new tabs, each one knows about your current shopping cart.</p> |
| Page | Desktop (A.K.A multi-monitor clients) | <p>A page in Perspective equates to a single page in a web browser. Pages are the main navigational unit in a session, and consist of one or more views. Each page is associated with a URL, which means the Forward and Back buttons in a web browser can be used to navigate to pages that have already been visited. Multiple pages can be open as part of the same Session, similar to how multiple desktops in a Vision Client may be open simultaneously.</p> <p>Views can be docked to specific edges of the page, or be used as a "Main View". Each page consists of at least a main view, but multiple views can be configured on a page. A page has specific regions where you can place instances of your views.</p> |
| View | Window and Template | <p>A View is the primary unit of design for Perspective. Perspective Views fill the roles of both Windows and Templates in Vision. Thus, you could think of a view as a window that can be nested inside other windows, or as a template that can be maximized or docked. Views have a root container, much like windows in Vision.</p> <p>Nesting one view inside another requires the Embedded View component. Parameters may be defined on the view, and then passed into the embedded view</p> <p>There are also components that may dynamically create view instances (like the Flex Repeater component). In this way, views act like templates just as they do in the Vision module. Parameters can be passed into a view from an external source.</p> |
| Container | Container + Layout Constraints | <p>Containers are objects that contain components. You can nest one container inside of another.</p> <p>In Perspective, the way that components reposition and resize is managed on the container, and every component contains only simple information about its size and position, which is interpreted based on the container's specifications. There are several container types, and each type is defined by the layout type it uses.</p> |
| Component | Component | Components, like in Vision, are displays, buttons, charts, labels, and other objects that display information to the user viewing the session. Components are the elements displayed in the Component Palette in the Designer. |
| Property | Property | <p>Properties serve as a place to change how a component looks or behaves, or store specific pieces of information.</p> <p>In Perspective, the property tree of a component is a JSON object, and as a result, there are only three configurable data types: value, object, and array. Rather than characterize the expected format of the property's data (e.g., integer, boolean, or string), these data types control the structure of the property tree. No further configuration of data types is required.</p> |
| Session Property | Client Tag | Every session has a configurable collection of properties that can be managed from the designer. They can be referenced from any view in the project, both from property/expression bindings and from scripting. |
| Events and Actions | Component Scripting | Event and Actions are configured similarly to the Component Scripting section in Vision. However, Perspective offers many more configurable events, and many more possible responses to these events (Actions). |
| Styles | Component Style Properties /Customizer | Styles in Perspective use CSS, and are configurable from the properties on each component. However, Perspective also offers the ability to configure style classes, which allow you to reuse configured styles across many different components. Styles can also be configured to change based on more advanced properties, like whether a mouse is hovering over the component, or how wide the viewport of the viewing device is. |

New Features in Perspective

In addition to an entirely new set of components, Perspective offers a variety of new features in the realm of Project design.

| Perspective Term | Description |
|------------------|---|
| Transform | A transform lies between a binding and the property it modifies, and provides an opportunity to change the value or format of the binding's output. For example, if a tag binding yields 0 for Normal and 1 for Faulted, we could use a transform to map 0 to Green |

| | |
|---------------------|---|
| | and 1 to Red. |
| Component Messaging | Perspective offers the ability to communicate between <i>components</i> , using a similar style of messaging that one might use to communicate between clients in Vision. You can send messages from any component using the system.perspective.sendMessage function, and you can configure message handlers on any component. This is useful for controlling the behavior of one component from another in a different view. |

Licensing and Activation

How Licensing Works

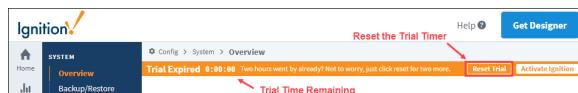
Ignition's licensing is unique and easy to use because Ignition is licensed by the server, not the client. You only need one license for your server; any clients are automatically included. In addition to that, an Ignition license is unlimited and sold based on which modules you want. There are unlimited clients, Tags, and projects. Buy only the modules you need, and don't worry about running into limits. If you want to test other modules, you don't need to do anything extra because of our built-in Trial mode. All unlicensed modules can be reset in 2 hour trial mode.

Trial Period

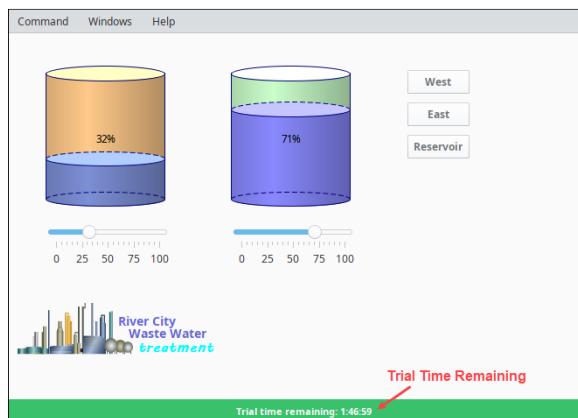
Our goal at Inductive Automation is to provide an easy way to access and learn Ignition. We want everyone to have the opportunity to try Ignition and start working with it with no restrictions. You can download Ignition from our webpage, install it, and start using it for two hours at a time. At the end of the two hour time period, all modules will stop running, but don't panic, you can reset the timer to run for another two hours. You can reset the timer as many times as you want, so go for it! The Gateway Webpage and the Designer are not affected by this trial, so you can develop for as long as you want without interruption. If you do have a license, any unlicensed modules will run in this Trial mode, but licensed modules will never timeout.

You can re-start the Trial period by logging into the Gateway, and clicking **Reset Trial** to enable another two hours of execution. The Trial Timer may be restarted any number of times. Depending on the module, you may need to take some additional actions. For example, the Vision Clients requires you to log out and back in again in order to continue the Trial.

In the Gateway, the Trial Time banner is displayed near the top of the screen. To reset the trial timer, click the Reset Trial button:



In a Vision Client (for Edge or Ignition Standard), the banner appears near the bottom of the screen.



On this page ...

- [How Licensing Works](#)
- [Trial Period](#)
- [How to Activate Ignition Edge and Ignition Standard Edition](#)
 - [Online Activation Example](#)
 - [Offline Activation Example](#)
- [How to Activate Maker Edition](#)
 - [Key and Token](#)
 - [Activation](#)
- [Adding Multiple Licenses to a Single Gateway](#)
- [Module Status](#)
 - [License Incomplete](#)
- [Updating a License](#)
- [Emergency Activation](#)
 - [How to Activate in Emergency Mode](#)
- [Unactivating a License](#)
 - [Offline Unactivation](#)



About Licensing

[Watch the Video](#)



About the Trial Period

[Watch the Video](#)

How to Activate Ignition Edge and Ignition Standard Edition

When you purchase a license for Ignition Standard edition or Edge, you receive a license key, a six-digit code that identifies your purchase. Use this license key to activate the software online through the Ignition Gateway. If you later want to add any additional modules, your account is updated and you



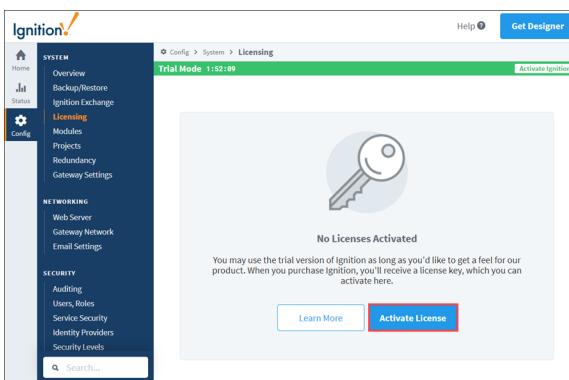
can re-use your existing license key to activate the new features. You can also deactivate your license key, and reuse it to activate Ignition on a different machine as many times as you want, allowing you to transfer a license from one Gateway to another.

You can activate your license in two ways:

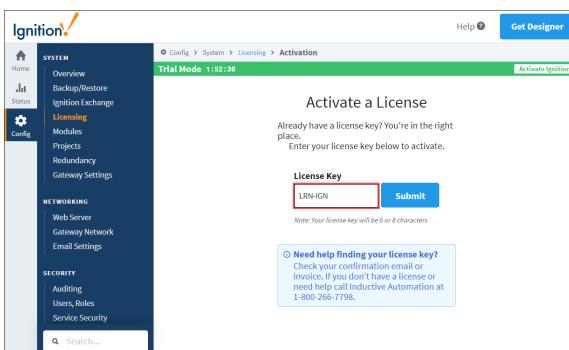
- **Online Activation** - from the **System > Licensing** section on the Gateway Webpage, your request to activate your Gateway is activated over the internet.
- **Offline Activation** - if you don't have an Internet connection, you can follow the manual activation process in the Offline Activation section.

Online Activation Example

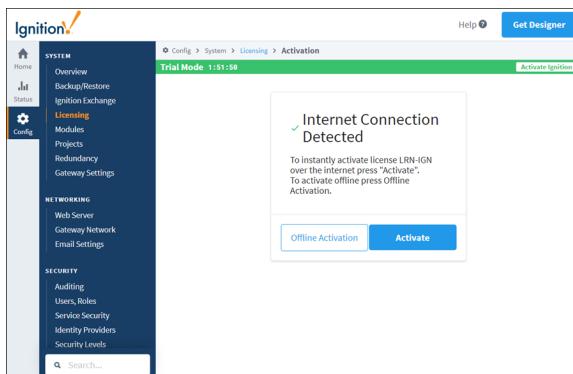
1. Go to the Gateway Webpage and select the **Config** section.
2. From the menu on the left, select **System > Licensing**. The Licensing page will appear. If you already have a license key, click the **Activate License** button.



3. The **Licensing / Activate Online** page will be displayed. Enter your **License Key** and click **Submit**.



- a. If you are connected to the Internet, click the **Activate** button.
- b. If you want to activate the license offline, click the **Offline Activation** button.



4. The Licensing page will refresh, and your Current License will be successfully activated.

Online Activation

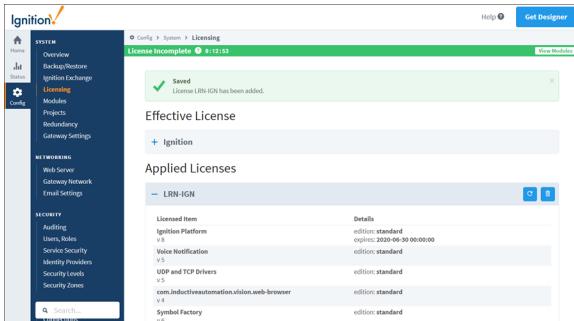
[Watch the Video](#)



INDUCTIVE
UNIVERSITY

Offline Activations

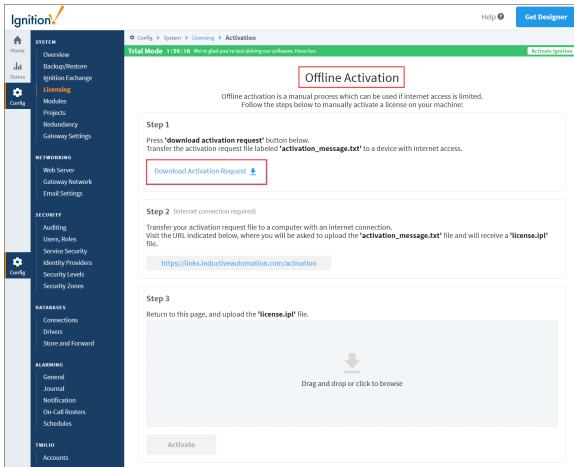
[Watch the Video](#)



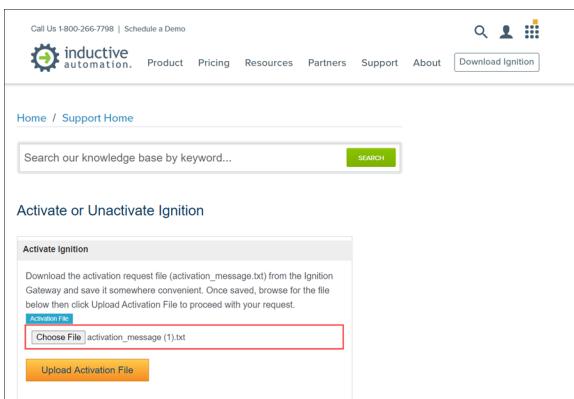
Offline Activation Example

When you do not have an internet connection, you can do the following steps to activate your license manually:

1. Go to the Config tab on the Gateway Webpage.
2. Scroll down to **System > Licensing**. The Licensing Activation page appears.
3. Click the **Offline Activation** button.
4. Click the **Download Activation Request**. An activation request file, called `activation_message.txt` is generated and downloaded.

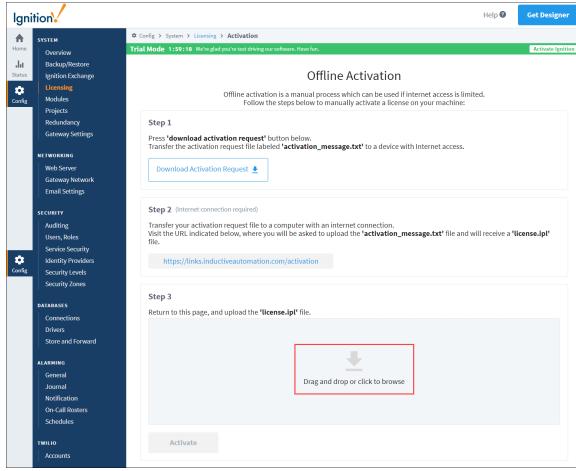


5. Take the `activation_message.txt` file to a machine with Internet access and go to [http://links.inductiveautomation.com/activation](https://links.inductiveautomation.com/activation).
6. Click **Choose File** and select the `activation_message.txt` file.



Note: Should any trouble occur with the upload process, please contact our Support department.

- Once you upload the `activation_message.txt` file, a license file called `license.ipl` is generated.
- Bring the `license.ipl` file back to the computer on which you're licensing Ignition. Back on the Offline Activation page, upload the `license.ipl` file. Then click **Activate** to finish the process.



- The Licensing page will refresh showing your license was successfully activated.
- Click **View Modules** to see a list of all your activated modules.

How to Activate Maker Edition

Key and Token

When a license is purchased you receive a Key (an eight-digit code) and a token that identify your purchase. Use this Key to activate the software online through the Ignition Gateway. You can also deactivate your license key, and reuse it to activate Ignition on a different machine as many times as you want.

Activation

When you download Ignition, you have the choice of Ignition Standard, Edge, or Maker Edition. You activate the license for Maker Edition during the installation process. When you choose Maker Edition, the installation wizard will prompt you for a license key and token. If you need help finding your key, sign in to your Inductive Automation account.



Activate License

Enter your Maker Edition license key here to continue this installation. If you don't already have a license, one can be generated online from your Inductive Account profile.

License Key

Your key will be 8 digits, AAAA-BBBB.

Activation Token

ⓘ Need help finding your key?

To generate a free Maker Edition key, sign in to your [Inductive Automation Account](#) online or create a profile. Within your account, navigate to the 'Licenses' page from the main menu.

Step 2 of 4

Next →

Adding Multiple Licenses to a Single Gateway

In order to better support our community of third-party module authors, we also allow for multiple license keys to be installed on a single Gateway. A third-party module author can issue a license key for their module directly to a customer, whereby, they can immediately do an install of the module. To learn more, check out the [Third-Party Module Showcase](#) to find and purchase modules that extend Ignition's functionality.

It is important to note that there may only be one license on it with a platform version per Gateway. The platform will look similar to the picture below. If you try and activate a second license with a platform onto a Gateway that already has a license with a platform, the new license will overwrite the previous license.

In the image below, two licenses have been applied to this Gateway, but only one is active.

The screenshot shows the Ignition Gateway's Licensing page. The left sidebar has sections for Home, Status, and Config, with Config currently selected. The main area shows the path: Config > System > Licensing. A green banner at the top says "License Incomplete" with a timer "0:47:49" and a "View Modules" button. Below it, under "Effective License", there is a box for "Ignition". Under "Applied Licenses", there are two entries: "LRN-IGN" (status: G) and "3PSW-F2P7" (status: B). A link "Activate new license..." is also present.

Module Status

It's important to know which version of a module you have installed and which version of the module you are licensed for. You could have a license for another version of a module so it's not going to work correctly until you have the correct version of the license. To verify the versions on your installed modules, go to the **Modules** page in the **Status** section of the Gateway Webpage or by clicking **View Modules** in the green banner.

Here you can see all the modules that are currently running with their version numbers and license status. The licenses are either in Trial, Activated, or Free mode. On this page, you can add or remove modules from your Ignition. If you are licensed for a module you are running, it will run in Activated mode. If you are not licensed for a module you are running, it will run in Trial mode until the Trial time is expired.

License Incomplete

Don't be alarmed by a 'License Incomplete' message on the green banner. Click on the View Modules link within the banner. It opens an informational message box showing modules that are installed. Some of the modules are in Trial Mode.

The screenshot shows the Ignition software interface. At the top, there's a green banner with the text "License Incomplete" and a timestamp "1:59:03". Below the banner, the main content area has two sections: "Running Modules" (19 / 22) and "Licensed Modules" (19 / 22). To the right, there's a "Licensing Configuration" button and a "License Details" panel showing a license key "LRN-IGN", version "8", and edition "standard". The left sidebar contains sections for SYSTEMS (Overview, Performance, Alarm Pipelines, Gateway Scripts, **Modules**, Redundancy, Reports, SFCs, Tags, Transaction Groups), CONNECTIONS (Databases, Designers, Devices, Gateway Network, Store & Forward, OPC Connections, Perspective Sessions, Vision Clients), and DIAGNOSTICS (Execution, Logs, Threads).

| Name | Version | License | Status |
|---------------------------|----------------------------|-----------|-----------|
| Alarm Notification | 5.0.0-beta0 (b2019032202) | Activated | ✓ RUNNING |
| Allen-Bradley Driver | 5.0.0-beta0 (b2019032202) | Activated | ✓ RUNNING |
| DNP3 Driver | 3.0.0-beta0 (b2019032202) | Activated | ✓ RUNNING |
| Enterprise Administration | 3.0.0-beta0 (b2019032202) | Activated | ✓ RUNNING |
| Logix Driver | 4.0.0-beta0 (b2019032202) | Activated | ✓ RUNNING |
| Modbus Driver | 6.0.0-beta0 (b2019032202) | Activated | ✓ RUNNING |
| OPC-UA | 8.0.0-beta0 (b2019032202) | Activated | ✓ RUNNING |
| Omron Driver | 3.0.0-beta0 (b2019032202) | Activated | ✓ RUNNING |
| OpcCom | 5.0.0-beta0 (b2019032202) | Activated | ✓ RUNNING |
| Perspective | 1.0.0-beta0 (b2019032202) | Activated | ✓ RUNNING |
| Reporting | 5.0.0-beta0 (b2019032202) | Activated | ✓ RUNNING |
| Vision | 10.0.0-beta0 (b2019032202) | Activated | ✓ RUNNING |
| Voice Notification | 4.9.10 (b2018112821) | Trial | ✓ RUNNING |

Updating a License

If you added one or more new module to an existing license, then you'll need to update (Reactivate) the license. You update a license from the **Config > Licensing** section on the Gateway webpage. Simply click the Reactivate button next to the license that was changed, as shown below.

The screenshot shows the Ignition Config > System > Licensing page. The "Effective License" section shows "Ignition" selected. The "Applied Licenses" section shows a license named "LRN-IGN" with a "Reactivate" button highlighted with a red box. A link to "Activate new license..." is also visible.

INDUCTIVE UNIVERSIT
Reloading a License
[Watch the Video](#)

Pressing the Reactivate button will cause the gateway to attempt to reach out to our licensing server, and will update shortly after.

If the gateway does not have internet access, it will be unable to update automatically. In this case you'll need to follow the [Offline Unactivation](#) steps mentioned on this page, followed by an [Offline Activation](#).

Emergency Activation

In cases where you may have a hardware or OS failure and you cannot deactivate a license, Ignition provides an [Emergency Activation mode](#). In this mode, you can temporarily activate your license for 7 days giving you enough time to contact [Inductive Automation Support](#).

How to Activate in Emergency Mode

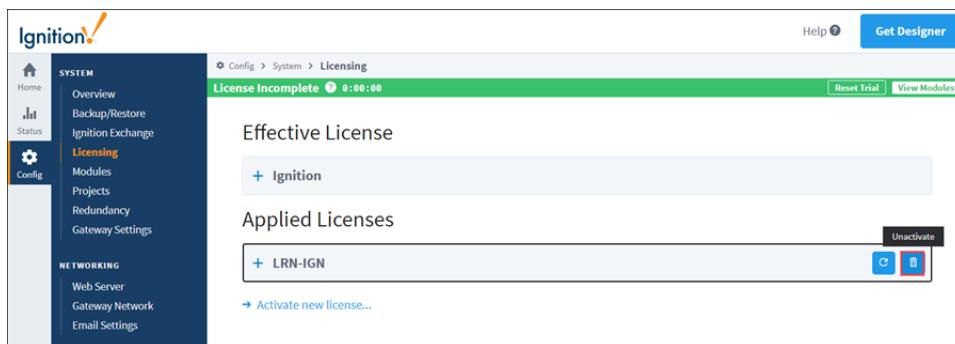
Activating your license in emergency mode is exactly the same as activating with your normal license, you don't have to do anything different because Ignition handles it all for you. See License Activation above for details on how to activate. The Gateway will know to run in emergency activation mode and it will display a timer stating how many days, hours, and minutes you have remaining in the Emergency Activation Mode banner. Any time before it expires, you can contact [Inductive Automation Support](#) to get your license fixed.

Unactivating a License

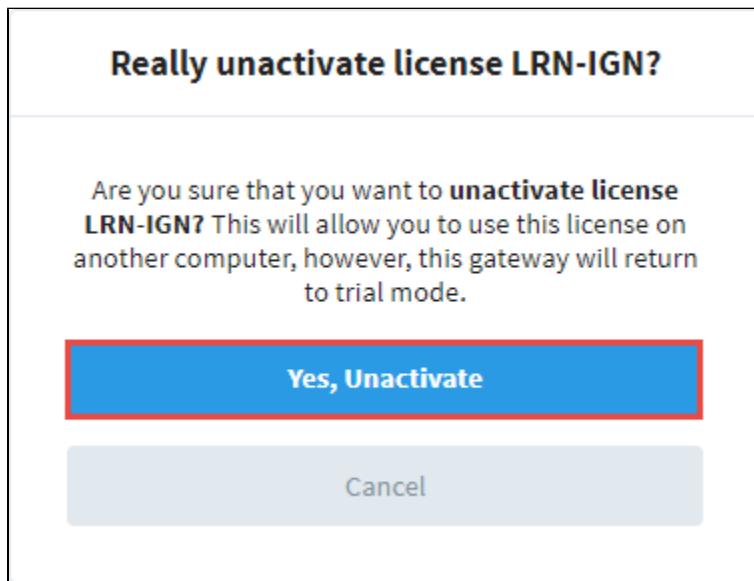
For a given license key, a limited amount of simultaneous activations are allowed at a given time. If you want to activate [Ignition](#) on a different [server](#), you must first deactivate it on the current [server](#). You can deactivate the license on one Gateway, and then activate it on a different Gateway if needed. Deactivation occurs immediately over the Internet, and makes this license available for activation on another machine.

To deactivate the [Gateway](#), do the following steps:

1. From the **Config** section of [Gateway](#), go to **System > Licensing**. The **Licensing** page is displayed and you can see the currently installed [license](#) license key.
2. Click on **Deactivate License** icon.



3. A Licensing / Confirm Unactive window will appear asking you to confirm the deactivation. Click the **Yes, Unactivate** button. It may take a minute or so for the request to finish.



4. The deactivation request will be sent to [Inductive Automation's](#) licensing servers, and the [license](#) will again be available for activation on another [Gateway](#).

Offline Unactivation

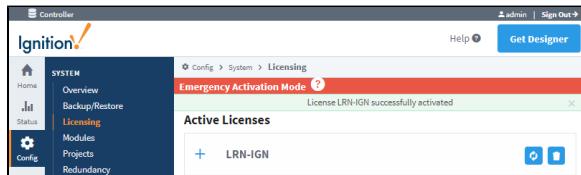
In the event the gateway is unable to reach out licensing server, it will prompt you with instructions on how to manually unactivate. Follow the instructions to complete the manual unactivation process.

[In This Section ...](#)

Emergency Activation

What Is Emergency Activation?

After activating your License Key in Ignition, you may see a banner that says Ignition is in **Emergency Activation Mode**.



What Does this Mean?

This means the License Key that you used has already been activated. Either it was used previously on another computer, or something went wrong during activation and it registered twice. Fortunately, you have plenty of time to fix this problem. Emergency Activation Mode means **you are fully licensed for 7 days**, so even at night or over the weekend, you can still run without interruption until it is fixed.

On this page ...

- [What Is Emergency Activation?](#)
 - [What Does this Mean?](#)
 - [How Do I Fix it?](#)
 - [Why Does this Exist?](#)
- [Quick Disaster Recovery Plan](#)
 - [What You Need Before an Emergency Happens?](#)
 - [Emergency Restore Steps](#)

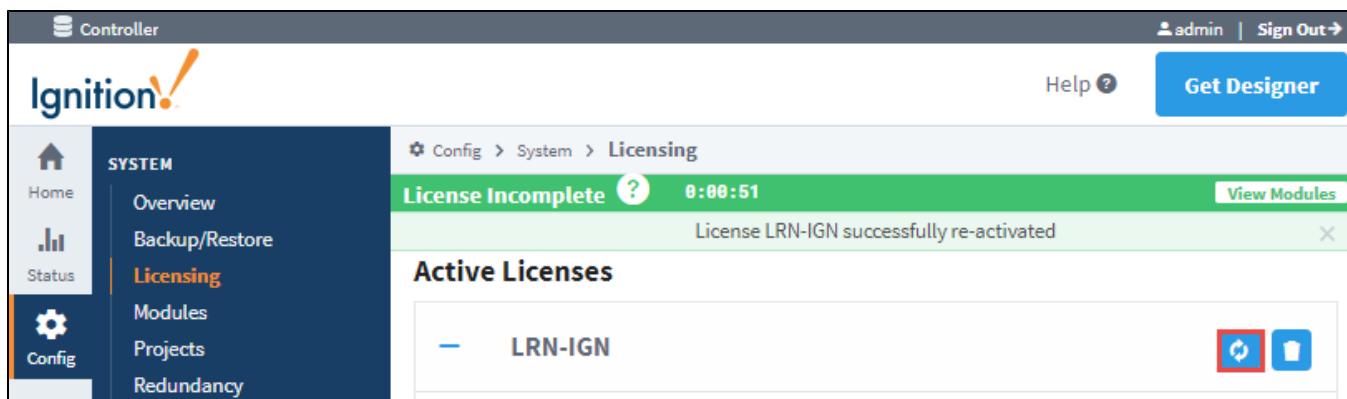


Emergency Activation

[Watch the Video](#)

How Do I Fix it?

In order to fix your License Key, you must [contact Inductive Automation](#). You can reach us by email or by phone and we will walk you through fixing your license. It is usually a very quick procedure that involves you re-activating a license after we fixed your License Key from our end. Once your license key is fixed, press the **Refresh** button and your license will be re-activated.



Why Does this Exist?

Basically, computers sometimes fail and you need to get running again, fast.

When you initially activate your copy of Ignition, it will run as long as your computer does. As we all know, sometimes there are hardware failures or your computer just stops working. There are many things you can do to minimize this, but once it happens you need to be able to get Ignition up and running again quickly. This Emergency Activation Mode allows you to do this without having to involve Inductive Automation until after your facility is back on track.

Quick Disaster Recovery Plan

What You Need Before an Emergency Happens?

If you already have Ignition installed and running in your facility, it's easy to get back to running quickly. Just make sure you have the following stored on a computer or shared drive **that is not your Ignition Gateway**:

- A [Gateway Backup](#). If you haven't already, you should set up [Scheduled Backups](#).
- Your Ignition License Key. Make sure you store the 6 digit code somewhere that you can easily retrieve it if your computer fails.
- The [Ignition software Installer](#). In an emergency, it's best to have the installer for the version you are currently using. You can always download an archived version from our website, but having one on hand is preferred. *Pro tip:* Edit the installer filename to include your License Key so it's easily available.

Emergency Restore Steps

1. Find another computer or create a new Virtual Machine. You need to get something up and running to put Ignition on first. We recommend having something on standby that is the same as the original computer. Further considerations:
 - a. Set your new computer to the same IP Address of the original Ignition Gateway. This way your clients won't need to do anything special to start running again.
 - b. Make sure your firewall is set correctly. At minimum, you must have the Ignition port open (8088 by default). In an emergency situation, you may want to just disable the firewall temporarily.
2. [Install Ignition](#) on the new computer.
3. [Load a Gateway Backup](#). Everything in Ignition is in the Gateway Backup, once it's loaded, you are running again!
4. [Activate your License](#). Your existing License Key will only work normally the first time it is used. After that, if you try to use it again, it will instead go into an **Emergency 7-Day Trial**. This gives you plenty of time to get back up and running before you have to deal with the License Key, even if your failure is on the weekend or outside Inductive Automation's business hours.
5. [Contact Inductive Automation](#). You can reach us by email or by phone (**1-800-266-7798**), and we will walk you through fixing your license.

Maker Edition

What Is Ignition Maker Edition?

Ignition Maker Edition is a community version of Inductive Automation's Ignition SCADA application. The Maker Edition enables hobbyists and those folks who want to explore and get inspired to build and automate their own projects in their home. Ignition Maker Edition is a free-license edition that can be strictly used for non-commercial and personal educational use. Users are free to make use of all the modules and features of the product while creating their own project.

Note: For educational institutions, Inductive Automation has other options for a classroom setting or senior projects. Please contact our [sales team](#) for more information on these options.



Getting Started with Maker

[Watch the Video](#)

A screenshot of the Ignition Maker Edition homepage. The top navigation bar includes the Ignition logo, "Maker Edition", "Home", "Status", and "Config" buttons, along with "Help" and "Get Designer" links. The main header reads "WELCOME TO IGNITION MAKER EDITION" and "Automate... Anything!". Below the header is a large circular icon with a yellow and red gradient and a white exclamation mark. A "Get Started" button is centered below the icon. Two main sections are displayed: "1. Build It" and "2. Run It". "Build It" shows an icon of a computer monitor with a flowchart and a gear, with a "Download the Designer" button. "Run It" shows an icon of a green sphere with various blue and purple cubes, with a "View Projects" button. At the bottom, a call-to-action reads "Join our community forum to ask questions & get answers from automation **professionals** and **hobbyists**".

What Can Maker Edition Do?

Maker Edition can do almost anything standard Ignition can do. You can seamlessly connect all your data, design any kind of application with ease, and instantly web-deploy clients to anyone, anywhere — all from one platform.

How Maker Edition Compares to Standard Edition

Make Edition differs from Standard Edition Ignition in the following ways:

- Maker licenses are free. Get one now in your [IA account](#).
- Maker Gateways can support a maximum of 10 Perspective sessions.
- Maker Gateways can run up to 10,000 Tags.
- In regards to redundancy, Maker Gateways can only be set to the [Independent](#) mode.

This feature is new in Ignition version **8.1.4**
[Click here](#) to check out the other new features

- As of 8.1.4, it is now possible to create remote Tag Providers and Tag history providers when the provider source is a Maker Gateway.

Ignition Maker Edition has limited module support .The table below represents Inductive Automation modules that are ready to use on Maker Gateways. Modules that aren't allowed to run on Maker will be removed during Maker's commissioning process.

| Module Name | Description |
|----------------------------|--|
| Alarm Notification | Provides alarm notifications via email. |
| Allen Bradley Driver Suite | Allen-Bradley driver suite for the OPC UA module. |
| Logix Driver | A driver for communicating with Allen-Bradley Logix5000 series PLCs, and includes firmware version 21 support. |
| Modbus Driver | A driver for communicating with devices via Modbus-TCP. |
| Omron Driver | Drivers for Omron PLCs. |
| OPC UA | Provides Ignition's OPC UA client and server functionality. |
| Perspective | A module that provides modern, responsive HTML based graphical interfaces for Ignition projects. |
| Reporting | Adds reporting capability. |
| Serial Support (Gateway) | Adds platform-level serial comm support and system.serial functions to Gateway-scoped scripts. |
| SFC | Sequential Function Charts are customizable logic execution environments. |
| Siemens Driver | Support for Siemens S7-300, S7-400 and S7-1200 drivers. |
| SQL Bridge | An OPC-to-SQL data logger and transaction manager. |
| Tag Historian | Turns any database into a powerful historian that can store and drive data in Maker Edition. |
| Twilio | Send SMS alarm notifications without a physical device allowing you to send SMS message via Python script. |
| UDP/TCP Driver | Drivers for receiving and parsing UDP or TCP packets. |
| WebDev | Allows you to create and serve basic web resources and respond to web requests through scripting. |

Ignition Edge

Ignition Edge is a lightweight, lean version of Ignition with software solutions designed for devices used in the field and OEM devices at the edge of the network. Ignition Edge solutions work on a wide variety of operating systems such as Linux, Windows (any version), macOS, and more. With support for ARM processors, Ignition Edge can run on devices like a Raspberry Pi or the latest generation of edge-of-network devices.

Core functionality for Edge includes the following:

- All Inductive Automation OPC UA drivers (AB, Siemens, Modbus, Omron, DNP3, UDP /TCP, serial, BACnet)
- One-week data buffer
- Two device connections
- One-way email notification
- Gateway Network connections (actual Gateway Network services are enabled with IIoT, EAM, and Sync Services)
- Limited to one project, one Tag Provider, and one Historical Tag Provider

Edge Onboard

Ignition Edge comes pre-installed on a range of edge-of-the-network devices that are IIoT ready out of the box. Edge can also be installed on virtually any industrial device.

Licensing

The new Edge version of Ignition has a separate license that works in the same way as a traditional Ignition license. The Edge license will not work if you attempt to activate it on an Ignition Gateway as it is only set up to license an Edge Gateway. However, you can activate a full Ignition license on an Edge Gateway, but this is not recommended due to the limitations of the Edge platform. Additionally, once a license has been activated on an Edge Gateway, the trial time for any unlicensed Edge plugins will be removed, and you will not be able to access them.

On this page ...

- [Edge Onboard](#)
- [Licensing](#)
- [Core Functionality](#)
 - [OPC UA Module and Drivers](#)
 - [One Week of Internal Storage](#)
 - [Device Connections](#)
 - [Projects](#)
 - [Redundancy](#)
- [Edge Plugins](#)
- [Edge Gateway](#)
- [Ignition Edge Compared with Ignition](#)
 - [Database Access](#)
 - [Gateway Scripting](#)
 - [Gateway and Project Backups](#)
 - [Third-Party Modules](#)



INDUCTIVE
UNIVERSITY

Edge Architecture

[Watch the Video](#)

Core Functionality

The following features are available on all Edge Gateways, licensed or unlicensed.

OPC UA Module and Drivers

Edge can create OPC UA client connections. The internal OPC UA server in the Edge Gateway acts only as a client and not as a server. This means Edge can connect to other outside OPC servers as a client to fetch data, but can not act as an OPC UA server for outside programs to connect to. Adding the IIoT plugin extends this functionality.

One Week of Internal Storage

An Edge installation can store certain historical data inside Ignition.

- **Tag History** - Up to one week or 10 million data points of history can be stored locally. Pruning will automatically happen, and the historian is automatically enabled and not configurable. An Edge Gateway can only have one realtime Tag Provider and one Edge historian. You can learn more about Tag history [here](#).
- **Alarm Journal** - The alarm journal will store all alarm events locally for one week of events. There are limited configuration options; the Alarm Journal can be accessed via the Edge Alarm Journal profile. The Edge alarm journal and audit log can sync under EAM.
- **Audit Logs** - Like the Alarms, the Audit log is limited to one week of local storage. It cannot be configured.

Device Connections

By default, Edge Gateways can have two device connections. Once two devices have been configured, the “add device” UI on the Gateway’s Devices config page will be hidden.

Projects

Edge installations have just one project. This project is created automatically if it did not exist, and it cannot be removed. The default project name is "Edge" and can be renamed. For more information, see [Edge Gateway Config section](#).

Redundancy

Two Edge Gateways can be set up with [redundancy](#). An Edge Gateway can only failover to another Edge Gateway (not a standard Ignition Gateway).

Edge Plugins

Plugins extend Edge's capabilities, each adding a unique set of features. There are five Ignition Edge plugins available: **Panel**, **IIoT**, **Compute**, **Sync Services**, and **EAM**. Plugins can be mixed and matched to create the system you need. Please note that a license is required to activate each of the Ignition Edge plugins. Each Edge plugin offers specific functionality that works best in certain scenarios, but because any combination of plugins can be installed, an Edge Gateway may fill multiple roles and help bridge different architectures.

| | |
|---|--|
|  Panel | <p>Ignition Edge Panel creates a standalone HMI at the edge of network. HMIs run in either a web browser or a dedicated application. Either Vision or Perspective may be selected. The Web Browser module is included with Vision.</p> |
|  IIoT | <p>Ignition Edge IIoT turns any device into an edge gateway that publishes data to an MQTT broker. (IIoT was formerly Edge MQTT.) The plugin allows access to the MQTT Transmission, Opto22 SNAP PAC driver, the Emerson ROC driver modules, and the ABB Totalflow driver (manual installation required). In addition, Ignition Edge IIoT exposes Edge's OPC UA server to external OPC UA clients.</p> |
|  Compute | <p>Ignition Edge Compute enables you to remotely run scripts and create REST APIs for interfacing with third-party applications. The plugin allows access to Python scripting, Gateway Event Scripts, and the Web Dev module.</p> |
|  Sync Services | <p>Ignition Edge Sync Services synchronizes data from the edge of the network to a central Ignition server. The plugin allows access to Sync Services, remote Tags and alarm pipelines, Tag history sync, alarm history sync, and audit log sync.</p> |
|  EAM | <p>Ignition Edge EAM brings diagnostics, automatic backup and recovery, central licensing, and project synchronization to your edge-of-network device. The EAM plugin allows this Edge Gateway to be an Agent in an EAM architecture.</p> |

Edge Gateway

All Edge installations are a limited version of the Ignition platform for a significantly reduced cost. This helps keep things simple, and these installations always work with other full versions of Ignition if you need additional functionality. In an Edge installation there are a lot of visual changes to the Gateway Webpage, and some pages have been removed to avoid confusion. You can see below that after installing Edge, the traditional orange Ignition logos are replaced with new green Edge logos so a Gateway can be easily identified. For more information, see [Edge Gateway](#).

The screenshot shows the Ignition EDGE! Get Started page. The left sidebar has tabs for Home, Status, and Config. The main content area includes:

- Download Ignition Designer Launcher:** Features an orange wrench icon, a "Ignition Designer Launcher" title, a description about creating or modifying projects, and a large green "Download" button.
- Download Application Launchers:** Features a blue exclamation mark icon, a "Vision Client Launcher" title, a description about opening Vision clients from any Ignition Gateway, and a green "Download Vision Client Launcher" button.
- Learn Ignition:** A section with two cards:
 - Quick Start Guide:** Features the Ignition logo and a description of the user manual.
 - Inductive University:** Features the Inductive University logo and a description of its video courses.
- Product Documentation:** Features icons of various Ignition modules and a description of learning about each module.
- Appendix:** Features icons of a document and arrows, and a description of a complete reference for components, expressions, and scripting functions.

Ignition Edge Compared with Ignition

Because Edge is a lean solution, it does have some functionality limitations compared to full Ignition. Ignition and Ignition Edge share the same basic platform, but Edge is a leaner version of Ignition made specifically for use in on edge-of-network devices. Ignition comes with unlimited Tags, Clients, and database connections, while Ignition Edge comes with unlimited Tags, two Clients (one local and one remote) and no database connectivity.

Database Access

Database access is disabled. This means that any database queries or bindings will not work, and the user sources that require a database connection will not be available. The Status and Configure pages of the Gateway Webpage have been updated to remove these features and reduce confusion. Despite this restriction, some "historical" storage can happen inside of Ignition.

Gateway Scripting

All Gateway level scripting is disabled including Gateway Event Scripts and Tag Event Scripts. This does not affect project or Client-scoped scripts. If you wanted to take advantage of Gateway Events Scripts and Tag Events Scripts with an Edge installation, you can do so through the use of the Compute plugin. Keep in mind that you will not see the "Gateway Events" menu item in the Designer on unlicensed Gateways. Applying a compute license will require a Designer restart before the menu item will appear.

Gateway and Project Backups

A Gateway backup made in an Edge installation will work perfectly in other Edge installations. Edge Gateway backups are not compatible with Ignition Gateways, so you may not take an Edge Gateway backup and restore it on an Ignition Gateway. Nor may you take an Ignition Gateway backup and restore it on an Edge Gateway.

However, you can still export Tags, windows, and other project resources in the Designer from full Ignition Gateways, and import them into an Edge project, or vice versa.

Third-Party Modules

Edge's core functionality and plugins largely determine which modules run on the platform. Third-party modules will not run on an Edge Gateway, with the exception any third-party module explicitly stated by a plugin, such as the [Edge IIoT Plugin](#).

In This Section ...

Edge Gateway

The Edge Gateway has many of the same options as the regular Ignition [Gateway Webpage](#) along with a few Edge -specific settings. There are three tabs on the left: Home, Status, and Config.

Home

The first time you go to the Edge Gateway Webpage, it shows you several steps to help you get started. Once you're up and running, the [Home](#) tab lets you download the Designer Launcher or Vision Client Launcher, or view a Perspective project. There are some resource links to help you get started with Ignition quickly: Inductive University, Quick Start Guide or Product Documentation where you can learn about Ignition modules, as well as the Appendix which contains a complete reference for components , expressions, and scripting functions.

When you're first introduced to Ignition, the Homepage is the landing page. It's where you'll find the Designer Launcher, Vision Client Launcher, and Perspective Sessions.

- The [Designer Launcher](#) locates all Gateways that are available on your local network. Once you open the Designer, you can access the Edge project.
- The [Vision Client Launcher](#) opens Vision Clients from any Ignition Gateway.
- The [Perspective Sessions](#) opens Perspective sessions from this Gateway.

If you're running Perspective as the visualization method for your project, you can launch a Perspective Session from the Designer, from the Gateway Webpage, or by entering the web address of the project in your web browser.

On this page ...

- [Home](#)
- [Status](#)
- [Config](#)
- [Edge Gateway Settings](#)



INDUCTIVE
UNIVERSITY

Ignition Edge Configuration

[Watch the Video](#)

IgnitionEDGE!

Help ? [Get Designer](#)

[Home](#) > [Get Started](#)

Download Ignition Designer Launcher

 **Ignition Designer Launcher**
Download the Ignition Designer Launcher to create or modify your projects.

[Download](#)

Download Application Launchers

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

[Download Vision Client Launcher](#)

 **Perspective Session Launcher**
Launch a Perspective session directly in your browser or download the native application.

[View Projects](#)

Learn Ignition

Take advantage of our tools to get designing quickly and take your ideas from concept to reality. The User Manual is a wealth of easily searchable knowledge and the Inductive University has hundreds of short videos covering the basics of Ignition.

 **Quick Start Guide**
With a wealth of easily searchable knowledge the Ignition user manual has everything you need to get started and get your questions answered.

 **Inductive University**
IU has more than 600 searchable videos and 20 courses to help you learn how to use Ignition at your own pace on any device.

 **Product Documentation**
Learn more about each Ignition module and what you can do with it.

 **Appendix**
A complete reference for components, expressions, and scripting functions in Ignition.

Status

The Status tab provides an overview of all of the systems in your Gateway while also enabling you to drill down into specific systems and get an in-depth view of what is currently happening. For complete information, see Ignition Gateway [Status](#).

SYSTEMS

- Overview
- Performance
- Gateway Scripts
- Modules
- Redundancy
- Tags

CONNECTIONS

- Designers
- Devices
- Gateway Network
- Store & Forward
- OPC Connections
- Vision Clients

DIAGNOSTICS

- Execution
- Logs
- Threads

Help ? **Get Designer** **Activate Ignition**

Trial Mode 1:12:38 We're glad you're test driving our software. Have fun.

Architecture

Gateway | Ignition-TR-89MC8R2-WS

Version: (Dev Version)
License: trial
Uptime: an hour

2% cpu
122 mb

No Redundancy
Add a redundant backup gateway to protect your system from downtime caused by failures.

No Gateway Network
Multiply the power of your Ignition Gateways by combining them into an enterprise network. Streamline the administration, monitoring, deployment, and commissioning process into one central location.

Environment

| | |
|----------------------|--------------------|
| Process Id | 10120 |
| Operating System | Windows 10 amd64 |
| Java Version | 11.0.6+10-LTS |
| Local Time | 3:53:27 PM |
| Available Disk Space | 138gb / 238gb |
| Detected NICs | 10.10.110.54 |

Systems

| | |
|-------------|----------------|
| EAM Role | Unknown |
| Modules | 14 installed |
| Performance | 3% CPU 122mb |
| Redundancy | Not configured |
| Tags | 35 tags |

Connections

| | |
|-----------------------------|----------------------|
| Designer Sessions | 0 open |
| Devices | 0 enabled |
| Gateway Network Connections | 0 / 0 connections |
| OPC Connections | 1 / 1 connected |
| Store & Forward | 0 stores quarantined |
| Vision Clients | 0 open |

Config

The Config tab provides access to configuration options for Gateway settings. This is where most of the settings that affect the whole Gateway are set up. Note that some settings such as database connections are not available for Edge. For complete information, see Ignition Gateway [Config](#).

IgnitionEDGE!

Help ? [Get Designer](#)

- Home
- Status
- Config**

- SYSTEM**
 - Overview**
 - Backup/Restore
 - Ignition Exchange
 - Licensing
 - Modules
 - Projects
 - Redundancy
 - Gateway Settings
- NETWORKING**
 - Web Server
 - Gateway Network
 - Email Settings
- SECURITY**
 - Audit Log Viewer
 - Users, Roles
 - Service Security
 - Security Zones
- ALARMING**
 - Journal
 - Notification
 - Remote Notification
 - On-Call Rosters
 - Schedules
- OPC CLIENT**
 - OPC Connections
 - OPC Quick Client
- OPC UA**
 - Device Connections
 - Security

Config > System > Overview

Plugins

Plugins extend Edge's capabilities, each adding a unique set of features. Plugins can be mixed and matched, providing you with just the right amount of functionality. Please note that licenses are required to activate each of the Ignition Edge Plugins. During the trial period, Ignition Edge Panel is available to use but is limited to a resettable two-hour trial window.

Installed



Ignition Edge Panel Licensed

Create a standalone HMI at the edge of the network. HMIs run in either a web browser or a dedicated application.



Ignition Edge IIoT Not Licensed

Turn any device into an Edge Gateway that publishes data to a MQTT broker. The plugin allows access to the MQTT Transmission, Opto22 SNAP PAC driver, and the Emerson ROC driver modules (manual installation required). In addition, Ignition Edge IIoT exposes Edge's OPC UA server to external OPC UA clients.



Ignition Edge Sync Services Licensed

Synchronize data from the edge of the network to a central Ignition server. The plugin allows access to Sync Services, remote alarm pipelines, and remote tag providers.



Ignition Edge Compute Not Licensed

Remotely run scripts and create REST APIs for interfacing with third-party applications. The plugin allows access to Gateway Event Scripts and WebDev Scripting Resources.



Ignition Edge EAM Not Licensed

Bring diagnostics, automatic backup and recovery, central licensing, and project synchronization to your edge-of-network device. The plugin, allows this Edge Gateway to be an Agent in an EAM architecture.

Edge Gateway Settings

There is an additional set of Gateway Settings for Edge that are not available in regular Ignition. These settings enable you to modify the names of the Edge project, Edge Tag provider, and Edge historian. You can also set the visualization method (Vision or Perspective) here if you are using the [Edge Panel](#) plugin.

1. To get to the settings, choose **Config > Gateway Settings** on the Edge Gateway Webpage.

The screenshot shows the Ignition Edge configuration interface. The left sidebar has a 'Config' menu item highlighted with a red box. Under the 'SYSTEM' section, there is a link 'Gateway Settings' also highlighted with a red box. The main content area is titled 'Gateway Settings' and contains three configuration fields:

- System Name:** Ignition-TR-89MC8R2-WS (with a help icon)
- System User Source:** default
- Gateway Config Role(s):** Administrator

A green banner at the top says 'Trial Mode 1:59:44 We're glad you're test driving our software. Have fun.'

2. Scroll down to the **Edge Settings** section.

| Edge Settings | |
|--------------------|--|
| Project Name | Edge The name of the default Edge project. Only one project can exist in an Edge gateway. (default: Edge) |
| Tag Provider Name | edge The name of the default Edge tag provider. Only one realtime tag provider can exist in an Edge gateway. (default: edge) |
| Historian Name | Edge Historian The name of the default Edge historian. Only one historian can exist in an Edge gateway. (default: Edge Historian) |
| Visualization Name | VISION The name of the default Edge Panel visualization method. Only one visualization can exist in an Edge gateway. Value is unused if Panel is not licensed. (default: VISION) |

3. Change the settings as desired, then click **Save Changes** at the bottom of the screen.

The following table describes the Edge settings.

| Edge Setting | Description |
|--------------------|--|
| Project Name | The name of the Edge project. Default is "Edge." Only one project can exist in an Edge Gateway. You can change the name of this project with this setting. |
| Tag Provider Name | The name of the Edge Tag provider. Only one provider can exist in an Edge Gateway. |
| Historian Name | The name of the Edge Historian. Only one historian can exist in an Edge Gateway. |
| Visualization Name | The name of the default Edge Panel visualization method (Vision or Perspective). Only one visualization system can be used in an Edge gateway. This value is unused if the Edge Panel plugin is not licensed. Default is Vision. |

Related Topics ...

- [Edge Panel](#)
- [Edge IIoT](#)
- [Edge Compute](#)
- [Edge Sync Services](#)
- [Edge EAM](#)

Edge Panel



Ignition Edge Panel enables standalone HMI functionality for one local client and one remote web-launched client at the edge of the network. Ignition Edge Panel gives you a choice between the [Vision](#) or [Perspective](#) modules. Ignition Edge Panel comes standard with the [Ignition Designer](#). Panel provides up to two launchable clients for your Edge project, one local and one remote can be open at the same time.

Ignition Edge Panel includes one week of data buffering for trending and local client fallback for mission-critical applications. The Panel plugin provides a remote terminal for operators, usually to act as a backup in case there is a network failure.

Ignition Edge Panel can send one-way alarm notifications by email through your company's SMTP server, so you can stay alert to what's happening at the edge of the network.

During the trial period, Ignition Edge Panel is available to use but is limited to a resettable two-hour trial window.

On this page ...

- [Visualization Module](#)
 - [Running Vision on Panel](#)
 - [Running Perspective on Panel](#)
- [One-Way Alarm Notification](#)

Visualization Module

An Edge Gateway with the Panel plugin can launch either Perspective sessions or Vision clients, but not both. Ignition Edge Panel comes standard with the [Ignition Designer](#),

Licensed Edge Gateways that don't have the plugin will refuse to launch either runtime. In addition, switching visualization to one module will fault the other, preventing the Designer from accessing resources to the faulted module (may require a Designer restart).

By default, the visualization module is Vision. To change the visualization module, do the following:

1. On the Edge Gateway go to **Config > Gateway Settings**.
2. Scroll down to the **Edge Settings** section.

| Edge Settings | |
|----------------------|---|
| Project Name | Edge The name of the default Edge project. Only one project can exist in an Edge gateway. (default: Edge) |
| Tag Provider Name | edge The name of the default Edge tag provider. Only one realtime tag provider can exist in an Edge gateway. (default: edge) |
| Historian Name | Edge Historian The name of the default Edge historian. Only one historian can exist in an Edge gateway. (default: Edge Historian) |
| Visualization Module | VISION The module to be used for client visualization. Only a single visualization module can be active on an Edge gateway. The setting is ignored if Panel is not part of the license. (default: VISION) |

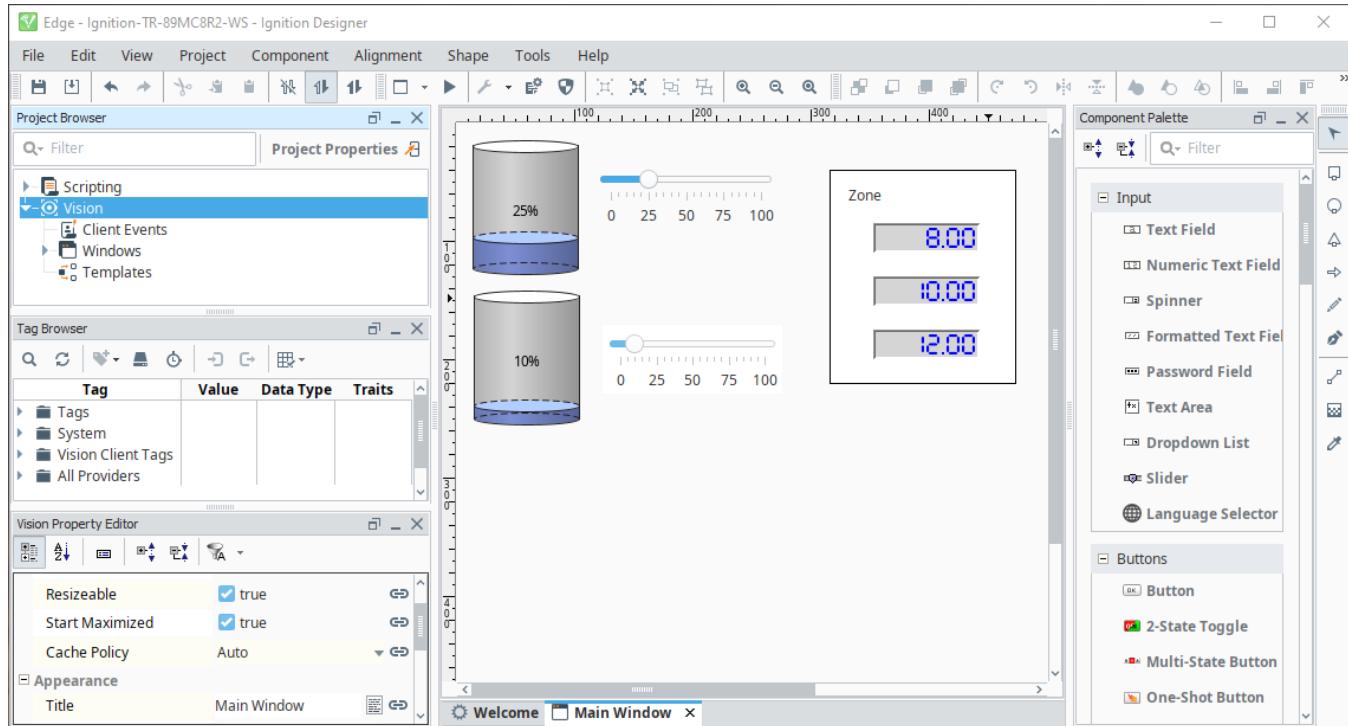
3. Under Visualization Module, click the Expand ▼ arrow then select the visualization module you want (Vision or Perspective).

| Edge Settings | |
|----------------------|---|
| Project Name | Edge The name of the default Edge project. Only one project can exist in an Edge gateway. (default: Edge) |
| Tag Provider Name | edge The name of the default Edge tag provider. Only one realtime tag provider can exist in an Edge gateway. (default: edge) |
| Historian Name | Edge Historian The name of the default Edge historian. Only one historian can exist in an Edge gateway. (default: Edge Historian) |
| Visualization Module | VISION VISION PERSPECTIVE <small>(default: VISION)</small> |

4. Click **Save Changes**.

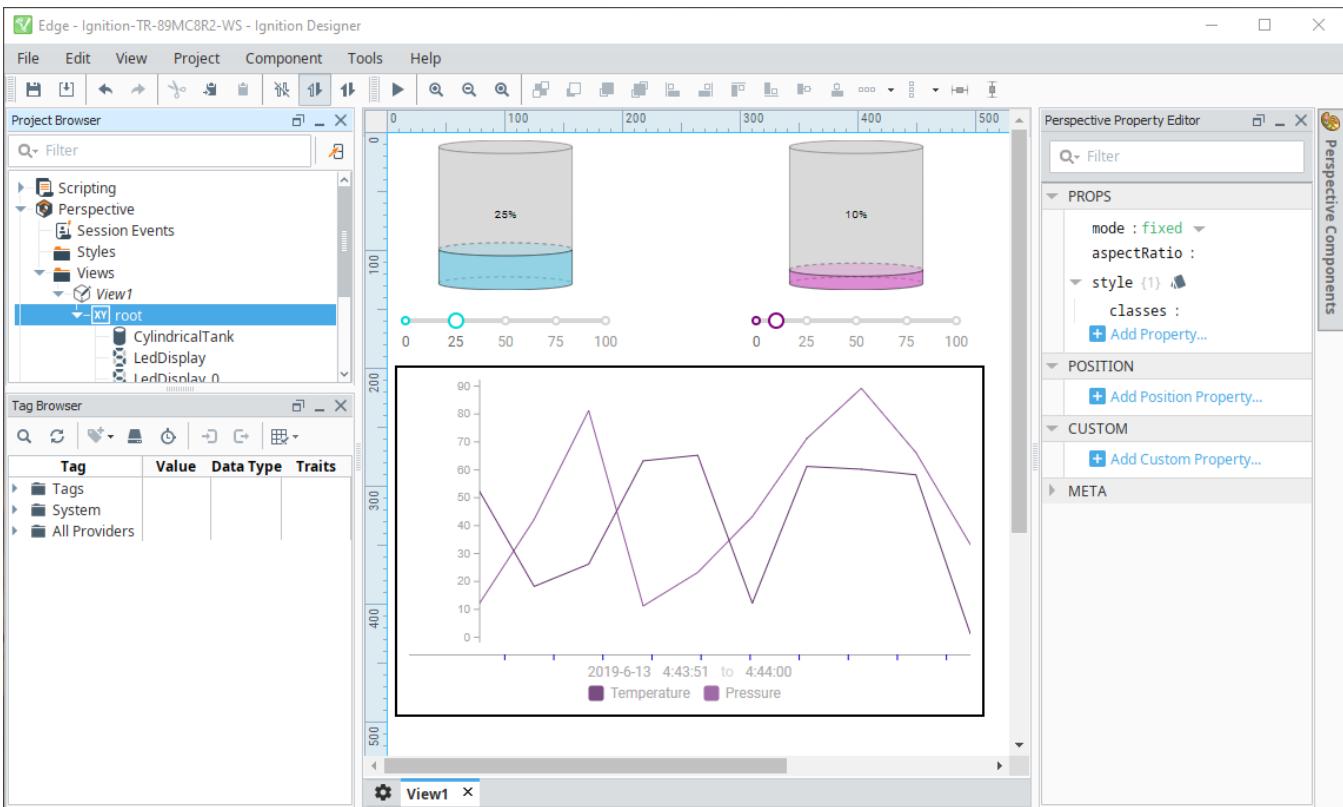
Running Vision on Panel

When the visualization module is set to Vision, only two clients are allowed: one local and one remote. The [Web Browser Module](#) is included with Vision. The following is an example of Vision running in Designer on Edge:



Running Perspective on Panel

When the visualization module is set to Perspective, only two sessions are allowed. The following is an example of Perspective running in Designer on Edge:



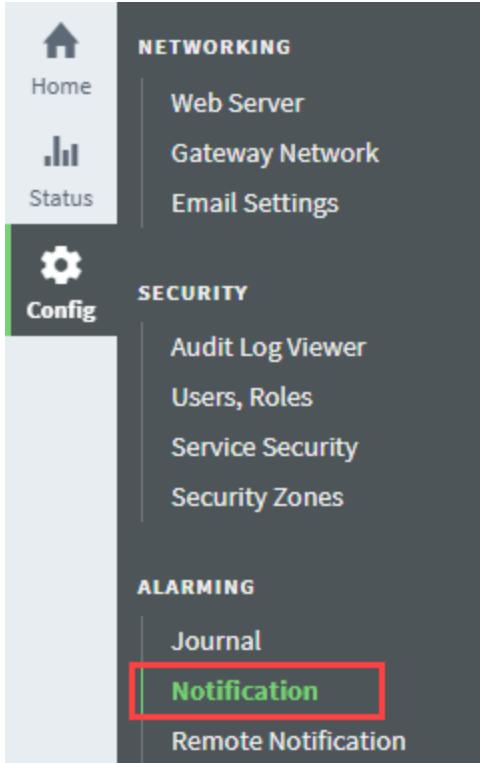
One-Way Alarm Notification

Ignition Edge Panel can send one-way alarm notifications by email through your company's SMTP server, so you can stay alert to what's happening at the edge of the network. Panel provides alarm notification in a limited form. This means the Edge Gateway has only one email notification profile and no alarm pipelines. Ignition Edge Panel includes one week of data buffering for trending and local client fallback for mission-critical applications.

Data buffer is limited to 10 millions rows, does not include database support.

To set up alarm notification, do the following:

1. On the Edge Gateway Webpage select **Config > Alarming > Notification**.



2. On the Alarming Notification screen, choose the [SMTP profile](#) and [On-Call Roster](#).
3. Set the Alarm Criteria as desired.
4. Click **Save**.

| Property Name | Description |
|------------------|--|
| Enabled | Whether notification is enabled. (Default is true.) |
| SMTP Profile | The SMTP profile that will be used to distribute the notifications. |
| On-Call Roster | The Roster of users that will be notified when an alarm meets the criteria set in this profile. |
| Minimum Priority | For a Notification to be sent from this Notification Profile, its priority must be equal to or higher than the priority specified on this property . (Diagnostic, Low, Medium, High, Critical) |
| Maximum Priority | For a Notification to be sent from this Notification Profile, its priority must be equal to or lower than the priority specified on this property . (Diagnostic, Low, Medium, High, Critical) |

For more information, see [Simple One-Way Email Notification Profile](#).

Edge IIoT



The IIoT plugin (formerly known as MQTT) allows this copy of Edge to push data directly into a Broker using the fast and lightweight MQTT data transfer protocol. In effect, this makes your computer (and whatever devices are attached to it) into a remote MQTT publisher. If you are already rolling out new smart devices that connect to your Broker, this allows you to connect all of your legacy devices into the same system.

IIoT plugin also allows UA clients to connect to the UA server on Edge.

Edge IIoT activates the following modules:

- MQTT Transmission
- Opto22 SNAP PAC driver
- Emerson ROC driver
- ABB TotalFlow

On this page ...

- [Remote Tag Provider](#)
- [Expose Edge's UA Server](#)
- [Install MQTT modules to Activte IIoT Plugin](#)
- [Install IIoT Opto22 SNAP Pac Driver or the Emerson ROC Driver](#)

Remote Tag Provider

With Edge IIoT installed, Ignition Gateways will be able to create remote Tag providers, targeting a provider on an Edge Gateway: either the system provider or the one realtime Tag provider.

For more information about remote Tag providers, see [Tag Providers](#).



The [Edge Sync Services](#) plugin also unlocks this feature.

Expose Edge's UA Server

The IIoT plugin enables UA clients to connect to the UA server on Edge. Applying this plugin will require that the [OPC UA module](#) is restarted before the UA server will be available publicly.

Install MQTT modules to Activte IIoT Plugin

After you install Edge, you need to download and install the MQTT Transmission module from Cirrus Link. You can use the free trial as long as you want so you can build and test full solutions before you buy.

1. To install the Ignition Edge IIoT plugin, first download the MQTT Transmission module from the [Inductive Automation](#) website.


Product Pricing Resources Partners Support About Download Ignition

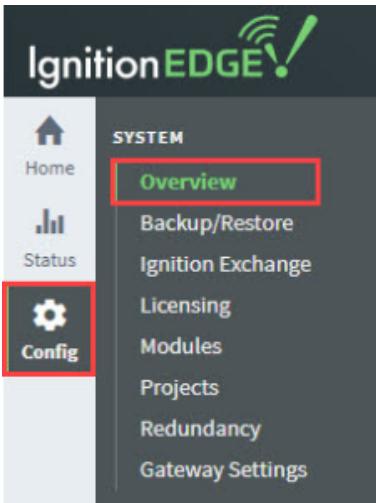
[Home](#) / [Downloads](#) / Other OS-Versions

Download Ignition

Other operating systems and versions

| Ignition | Version Archive | Strategic Partner Modules | | | | | | | | | | | | | | | |
|--|-----------------|---------------------------|---|---------|----------|--|--|--|-----------------------------------|-------|-------------------------|------------------------------|-------|-------------------------|------------------------------------|-------|-------------------------|
| Ignition Version: | | | | | | | | | | | | | | | | | |
| 8.0 | .9 | | | | | | | | | | | | | | | | |
| Ignition Version 8.0.9 STABLE <small>Released February 18, 2020 System Requirements Release Notes</small> | | | | | | | | | | | | | | | | | |
| <h3>Strategic Partner Modules</h3> <p>All third party Ignition modules require the Ignition platform to be installed.</p> <p>To install third party modules:</p> <ol style="list-style-type: none"> 1. Install Ignition: See Ignition installation guide 2. Once Ignition is installed, download the module and install it in the Ignition Gateway: See module installation guide <table border="1"> <thead> <tr> <th>Cirrus Link Solutions MQTT Modules for Ignition</th> <th>Version</th> <th>Checksum</th> </tr> </thead> <tbody> <tr> <td colspan="3">See the release notes and usage documentation for all Cirrus Link modules here.</td> </tr> <tr> <td>MQTT Distributor Module (30.5 MB)</td> <td>4.0.3</td> <td>sha-256</td> </tr> <tr> <td>MQTT Engine Module (25.7 MB)</td> <td>4.0.3</td> <td>sha-256</td> </tr> <tr style="outline: 2px solid red;"> <td>MQTT Transmission Module (21.8 MB)</td> <td>4.0.3</td> <td>sha-256</td> </tr> </tbody> </table> | | | Cirrus Link Solutions MQTT Modules for Ignition | Version | Checksum | See the release notes and usage documentation for all Cirrus Link modules here . | | | MQTT Distributor Module (30.5 MB) | 4.0.3 | sha-256 | MQTT Engine Module (25.7 MB) | 4.0.3 | sha-256 | MQTT Transmission Module (21.8 MB) | 4.0.3 | sha-256 |
| Cirrus Link Solutions MQTT Modules for Ignition | Version | Checksum | | | | | | | | | | | | | | | |
| See the release notes and usage documentation for all Cirrus Link modules here . | | | | | | | | | | | | | | | | | |
| MQTT Distributor Module (30.5 MB) | 4.0.3 | sha-256 | | | | | | | | | | | | | | | |
| MQTT Engine Module (25.7 MB) | 4.0.3 | sha-256 | | | | | | | | | | | | | | | |
| MQTT Transmission Module (21.8 MB) | 4.0.3 | sha-256 | | | | | | | | | | | | | | | |

2. Next go to the Config tab of your Edge Gateway.



3. On the Overview page, scroll down to the MQTT and click the **Install+** button.

Plugins

Plugins extend Edge's capabilities, each adding a unique set of features. Plugins can be mixed and matched, providing you with just the right amount of functionality.

Installed



Panel Trial

Ignition Edge Panel creates a standalone HMI at the edge of network that runs in either a web browser or a dedicated application.



Sync Trial

Ignition Edge Sync lets you synchronize data from the edge of network to a central Ignition server.



Compute Trial

Ignition Edge Compute allows you to remotely run scripts and create REST APIs for interfacing with third-party applications.



EAM Trial

Ignition Edge EAM brings diagnostics, automatic backup and recovery, central licensing, and project synchronization to your edge-of-network device.

Available



IIOT

[Details](#)[Install +](#)

Ignition Edge IIOT lets you turn any device into an edge gateway that publishes data to a MQTT broker.

4. Scroll down and click on **Install** or **Upgrade a Module...**

| Name | Version | Description | License | State | Action |
|---------------------------|----------------------|--|---------|---------|--|
| Alarm Notification | 5.0.6 (b2019111216) | Provides alarm notifications via email. | Trial | Running | More restart |
| Allen-Bradley Driver | 5.0.6 (b2019111216) | Allen-Bradley driver suite for the OPC UA module. | Trial | Running | More restart |
| Enterprise Administration | 3.0.6 (b2019111216) | A remote Gateway administration system, allowing you to manage Gateways and automate tasks from a single controller. | Trial | Running | More restart |
| Logix Driver | 4.0.6 (b2019111216) | A driver for communicating with Allen-Bradley Logix5000 series PLCs, and includes firmware version 21 support. | Trial | Running | More restart |
| Modbus Driver | 6.0.6 (b2019111216) | A driver for communicating with devices via Modbus-TCP. | Trial | Running | More restart |
| OPC-UA | 8.0.6 (b2019111216) | Provides Ignition's OPCUA client and server functionality. | Trial | Running | More restart |
| Siemens Drivers | 5.0.6 (b2019111216) | Siemens ST-300, ST-400 and ST-1200 drivers. | Trial | Running | More restart |
| Tag Historian | 3.0.6 (b2019111216) | Turns any database into a powerful historian that can store and drive data in Ignition. | Trial | Running | More restart |
| Vision | 10.0.6 (b2019111216) | A module that provides web-launched HMI/SCADA clients. | Trial | Running | More restart |

[Install or Upgrade a Module...](#)

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

5. Next click on **Choose file**.

To install a module, choose its *.modl file and press "Install".
To upgrade a module, install the new version on top of the existing version.
Modules can be downloaded from our website.

[Choose File](#) No file chosen

[Install](#)

6. Navigate to the folder where you downloaded MQTT-Transmission-signed.modl, select the file, and click **Open**.

7. Click the **Install** button.

8. At the License agreement, select **I accept the terms in the License Agreement**, then click the **Accept License** button.

9. At the Trusted Certificates window, click **I want to add this certificate to my trusted certificates and install the module**.

10. Click the **Add Certificate and Install Module** button.

This certificate has been signed by a trusted Certificate Authority.

Not Valid Until: 6/30/19, 5:00:00 PM
Not Valid After: 7/10/20, 4:59:59 PM
Subject Name: Cirrus Link Solutions LLC
Issuer Name: thawte SHA256 Code Signing CA
Thumbprint: [8D 8B 87 3A 51 FB F4 85 29 55 0A B3 75 1A 04 86 50 AA B0 54]

This module's certificate has been signed by a trusted Certificate Authority.

I want to add this certificate to my trusted certificates and install the module

[Add Certificate and Install Module](#)

< Back

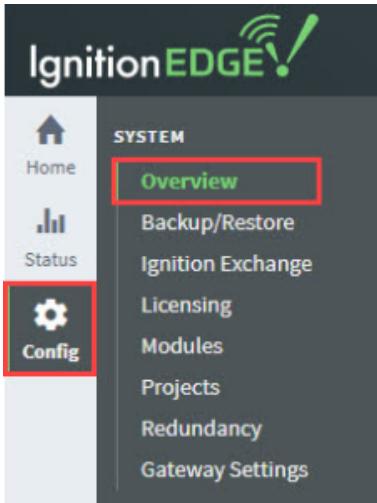
11. You'll see a message indicating the module was successfully installed. And the Module will show up on the Module Configuration page in the Edge Gateway now.

| Cirrus Link Solutions LLC | | | | | |
|----------------------------------|---------------------|--------------------------------|---------|-------------|--|
| View Certificate | | Name | Version | Description | License State |
| MQTT Transmission | 4.0.2 (b2019101100) | An Ignition Tag to MQTT Bridge | Trial | Running | More ▾ restart |
| | | | | | |

For more information, check out the documentation on the Cirrus Link website <https://docs.chariot.io/display/CLD>.

Install IIoT Opto22 SNAP Pac Driver or the Emerson ROC Driver

1. To install the Opto22 SNAP Pac driver or Emerson ROC driver, first download the driver you want from the [Inductive Automation](#) website.
2. Next go to the Config tab of your Edge Gateway.



3. Scroll down and click on **Install or Upgrade a Module...**
4. Next click on **Choose file**.
5. Navigate to the folder where you downloaded the driver(s) and select the file.
6. Click the **Install** button.
7. At the License agreement, select **I accept the terms in the License Agreement**, then click the **Accept License** button.
8. You'll see a message indicating the driver module was successfully installed. And the module will show up on the Module Configuration page in the Edge Gateway now, for example the following image shows the Opto22 SNAP Pac driver installed.

| Cirrus Link Solutions LLC | | | | | |
|--|---------------------|--|---------|-------------|--|
| View Certificate | | Name | Version | Description | |
| MQTT Transmission | 4.0.3 (b2020012800) | An Ignition Tag to MQTT Bridge | | | |
| Opto 22 groov EPIC and SNAP PAC Driver | 4.0.3 (b2020012800) | A driver for communicating with groov EPIC and SNAP PAC series controllers | | | |

Edge Compute



Ignition Edge Compute brings true edge computing to your network. Compute enables you to program directly against the web server in the Ignition Edge Gateway. You can remotely run scripts and create REST APIs for interfacing with third-party applications. It also provides Gateway Network Messaging.

Ignition Edge Compute gives your edge devices the ability to run scripts without the need to depend on the central server. Python is an easy to read, versatile programming language. Even if you don't know it at all, you will probably be able to understand a basic Python script. See [Python Scripting](#) for complete information.

On this page ...

- [Scripting in Edge Compute](#)
- [Edge Gateway Network Messaging](#)
- [WebDev Module](#)

Scripting in Edge Compute

Gateway scoped scripting, such as [Gateway Event Scripts](#) and scripts that execute on a Tag (such as [Tag Event Scripts](#) or scripts called by `runScript` on a tag/alarm property), are unlocked when the Compute plugin is applied. This allows you to use any of the Gateway scoped Event scripts for your project. You will not see the "Gateway Events" menu item in the Designer on unlicensed Gateways. Applying a compute license will require a Designer restart before the menu item will appear.

The screenshot shows the Ignition Designer's 'Gateway Event Scripts' dialog. The left sidebar lists event types: Startup, Update, Shutdown, Timer (which is selected and highlighted in blue), Tag Change, and Message. The main pane displays 'Gateway Timer Scripts' with the title 'Daily System Check @60,000ms'. Below the title, it says 'Timer scripts that are always running on the Gateway'. To the right of the title is a small clock icon. The script code is as follows:

```
1 # this script will check to see if the g
2
3
4 # check the time, fire at midnight
5 now = system.date.now()
6 if system.date.getHour24(now) == 0 and s
7
8     # run the system checks against the
9
10    # read system tags
11    tags = system.tag.readBlocking(['[Sy
12
13    # gateway uptime check
14    gatewayUptime = tags[0].value
15    if gatewayUptime < 86400:
16        project.notifyUser('manager', 'G
17
18    # gateway memory check
19    memoryUtilization = tags[1].value
```

At the bottom of the dialog are three buttons: OK, Apply, and Cancel.

Edge Gateway Network Messaging

Edge Compute provides access to call message handlers on other Gateways. This allows you to send data to other Gateways to be dealt with. You can access this functionality through the following [system.util](#) messaging functions:

- [system.util.sendMessage](#)
- [system.util.sendRequest](#)
- [system.util.sendRequestAsync](#)

WebDev Module

Edge Compute comes with the [WebDev module](#), which enables you to directly program against the [web server](#) inside the [Ignition Edge Gateway](#) and systems running Vision Clients. This gives you the ability to host web pages or files, and create full-fledged REST APIs that allow external systems to interact with the Ignition server.

Edge Sync Services



Ignition Edge Sync Services acts as a limited remote server that synchronizes data from the edge of the network to a central Ignition server.

- Remote tags
- Tag history sync
- Alarm journal sync
- Audit log sync
- Remote alarm notification profiles

Sync Services can be used with the EAM plugin or separately. For an example of architecture using Sync Services, see [Edge Architectures](#).

**On this
page ...**

- Agent Gateway
- Store and Forward
- Data Synchronization
- Sync Reset
- Remote Alarm Notification Profiles
- Remote Tag Provider

Agent Gateway

Ignition Edge Sync Services, acting as an Agent Gateway in a secure, high-performance Ignition Gateway Network, can share information with other Ignition Gateways through Distributed Services, including distributed tags, history, and remote alarming.

Store and Forward

Ignition Edge Sync Services comes with up to one-week of data buffering so it can be used to store-and-forward your data to a central server in case of network failure.

Data Synchronization

The Sync Services plugin introduces Tag History Synchronization. Tag History Synchronization allows you to sync your history data from the Edge Gateway to a full Tag History Provider on an Ignition Gateway, allowing you to save past the one week data limit. The Edge Gateway will still only see the one week of data that it can hold internally, but it will allow you to store the data in a more permanent database that can be viewed from other Ignition systems. This is setup by navigating to the Tag History Sync page link at the bottom of the Edge Gateway Configure page.



SYSTEM

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



NETWORKING

- Web Server
- Gateway Network
- Email Settings

SECURITY

- Audit Log Viewer
- Users, Roles
- Service Security
- Security Zones

ALARMING

- Journal
- Notification
- On-Call Rosters
- Schedules

OPC CLIENT

- OPC Connections
- OPC Quick Client

OPC UA

- Device Connections
- Security
- Server Settings

ENTERPRISE ADMINISTRATION

- Agent Setup

Main

Remote Gateway Name



The gateway to target for remote synchronization.

Remote History Sync Settings

Remote History Provider Name

The remote history provider to sync data to.

Remote History Sync Enabled

Whether to send local tag history data to the remote system. The remote system must have the Tag Historian module installed, and allow remote storage.
(default: false)

Max Batch Size

10000

The maximum number of data points that will be sent per batch to the remote gateway.
(default: 10,000)

Sync Frequency

10

The frequency with which data will be sent to the remote gateway. This setting will be used in conjunction with the sync schedule, if enabled.
(default: 10)

Sync Frequency Units

Seconds ▾

(default: SEC)

Enable Schedule

If enabled, the data will only be synchronized during the times specified by the pattern provided.
(default: false)

Schedule Pattern

A comma separated list of time ranges. Examples:

9:00-15:00

9pm-5am

20.30-04.30

Home

Status

Config

ENTERPRISE ADMINISTRATION

Agent Setup

SYNC SERVICES

Data Sync (selected)

Sync Reset

Config > Enterprise > Data Sync

Trial Mode 0:52:14 We're glad you're test driving our software. Have fun. [Activate Ignition](#)

| Remote Audit Settings | |
|---------------------------|---|
| Remote Audit Profile Name | <input type="text"/> |
| | The remote audit profile to sync data to. |
| Remote Audit Sync Enabled | <input type="checkbox"/> (default: false) |
| | 10 |
| Sync Frequency | The frequency with which data will be sent to the remote gateway. This setting will be used in conjunction with the sync schedule, if enabled. (default: 10) |
| Sync Frequency Units | Seconds ▾ (default: SEC) |
| Enable Schedule | <input type="checkbox"/> If enabled, the data will only be synchronized during the times specified by the pattern provided. (default: false) |
| Schedule Pattern | <input type="text"/> A comma separated list of time ranges. Examples: 9:00-15:00 9pm-5am 20.30-04.30 |

Home

Status

Config

Enterprise Administration

Agent Setup

Sync Services

Data Sync (selected)

Sync Reset

Remote Alarm Journal Settings

| Remote Alarm Journal Settings | |
|-------------------------------|---|
| Remote Alarm Journal Name | <input type="text"/> |
| | The remote alarm journal to sync data to. |
| Remote Journal Sync Enabled | <input type="checkbox"/> (default: false) |
| | 10 |
| Sync Frequency | The frequency with which data will be sent to the remote gateway. This setting will be used in conjunction with the sync schedule, if enabled. (default: 10) |
| Sync Frequency Units | Seconds ▾ (default: SEC) |
| Enable Schedule | <input type="checkbox"/> If enabled, the data will only be synchronized during the times specified by the pattern provided. (default: false) |
| Schedule Pattern | <input type="text"/> A comma separated list of time ranges. Examples: 9:00-15:00 9pm-5am 20.30-04.30 |

Save Changes

The following table describes the tag history synchronization properties.

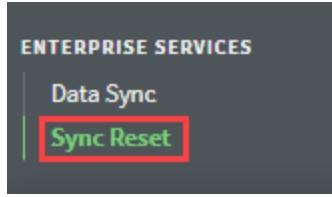
| Main | |
|--------------------------------------|--|
| Settings | |
| Remote Gateway Name | The gateway to target for remote synchronization. |
| Remote History Sync Enabled | |
| Remote History Provider Name | The remote history provider to sync data to. |
| Remote History Sync Enabled | Whether to send local tag history data to the remote system. The remote system must have the Tag Historian module installed, and allow remote storage. Default is false. |
| Max Batch Size | The maximum number of data points that will be sent per batch to the remote gateway. |
| Sync Frequency | The frequency with which data will be sent to the remote gateway. This setting will be used in conjunction with the sync schedule, if enabled. |
| Sync Frequency Units | Units are defined as milliseconds, seconds, minutes, hours, days, weeks, months, and years. |
| Enable Schedule | If enabled, the data will only be synchronized during the times specified by the pattern provided. |
| Schedule Pattern | A comma separated list of time ranges that will determine when data will be sent over the network. |
| Remote Audit Settings | |
| Remote Audit Profile Name | The remote audit profile to sync data to. |
| Remote Audit Sync Enabled | Whether Remote Audit Sync is enabled. Default is false. |
| Sync Frequency | The frequency with which data will be sent to the remote gateway. This setting will be used in conjunction with the sync schedule, if enabled. |
| Sync Frequency Units | Units are defined as milliseconds, seconds, minutes, hours, days, weeks, months, and years. |
| Enable Schedule | If enabled, the data will only be synchronized during the times specified by the pattern provided. |
| Schedule Pattern | A comma separated list of time ranges that will determine when data will be sent over the network. |
| Remote Alarm Journal Settings | |
| Remote Alarm Journal Name | The remote alarm journal to sync data to. |
| Remote Journal Sync Enabled | Whether Remote Alarm Journal settings are enabled. Default is false. |
| Sync Frequency | The frequency with which data will be sent to the remote gateway. This setting will be used in conjunction with the sync schedule, if enabled. |
| Sync Frequency Units | Units are defined as milliseconds, seconds, minutes, hours, days, weeks, months, and years. |
| Enable Schedule | If enabled, the data will only be synchronized during the times specified by the pattern provided. |
| Schedule Pattern | A comma separated list of time ranges that will determine when data will be sent over the network. |

Tag History Sync is unique in that if the connection to the remote Ignition Gateway is severed, the Edge Gateway will use its week of storage as a Store and Forward buffer, allowing you to store up to a week of data before data is lost. Once the connection is restored, the Edge Gateway will send data over based on the max batch size and data frequency until all previous data is sent.

Sync Reset

Edge has a way to reset synchronization state for internal historian providers. If you restored a Gateway backup onto a system with existing locally "syncable" configuration files, then the backup may have IDs from a range beyond the local files. Use the Reset Sync actions to reset your local sync configuration.

To do a **Sync Reset**, go to Edge Gateways under **Config > Enterprise Services > Sync Reset**.



The screenshot shows the 'Sync Records' configuration screen. At the top, there's a green header bar with the text 'Trial Mode 1:29:03' and a 'Activate Ignition' button. Below the header, there's a table with two rows:

| Key | Current Sync ID | |
|--------------------|-----------------|------------------------|
| Edge Alarm Journal | 0 | <button>Reset</button> |
| Edge Audit Profile | 0 | <button>Reset</button> |

Below the table, a note in a blue-bordered box reads: **Note:** If you have restored a gateway backup onto a system with existing locally "syncable" configuration files, then the backup may have IDs from a range beyond the local files. Use the reset actions here to reset your local sync configuration.

Remote Alarm Notification Profiles

Sync Services allows alarms in the Edge gateway to invoke remote alarm pipelines, via a remote alarm notification profile (configured on the Edge gateway).

Local alarm pipelines may not be used with Edge.

For more information, see [Alarm Notification Profiles](#).

Remote Tag Provider

With Edge Sync Services installed, Ignition Gateways will be able to create remote tag providers, targeting a provider on an Edge Gateway: either the system provider or the one realtime tag provider.

Note: The [Edge IIoT](#) plugin also unlocks this feature.

Edge EAM



The EAM plugin allows an Edge Gateway to act as an agent in EAM architecture. The Edge Gateway can only be an agent, not a controller. EAM enables you to bring diagnostics, automatic backup and recovery, central licensing, and project and Tag synchronization to your edge-of-network device.

The EAM plugin activates the [Enterprise Administration module](#). Some features of Ignition Edge EAM require the [Enterprise Administration module](#) to be installed on the central Ignition Gateway.

For an example of architecture using EAM, see [Edge Architectures](#).

**On this
page ...**

- [Gateway Backup and Recovery](#)

Gateway Backup and Recovery

The [EAM](#) enables automated backup and quick recovery for all Gateways through a [Gateway](#) backup archive.

Launchers and Workstation

Ignition features several launchable runtimes: Vision Clients, the Designer, and Perspective Sessions. To help with managing these separate systems, Ignition Gateways provide several "launcher" programs for each runtime. These launchers are the main means of launching an Ignition runtime application. For the purposes of clarity, the following are considered "Launchers":

- **Designer Launcher** - launches the [Designer](#) against the configured Gateway.
- **Vision Client Launcher** - opens Vision Clients from an Ignition Gateway.
- **Perspective Workstation** - opens Perspective sessions from an Ignition Gateway, in a standalone desktop program.

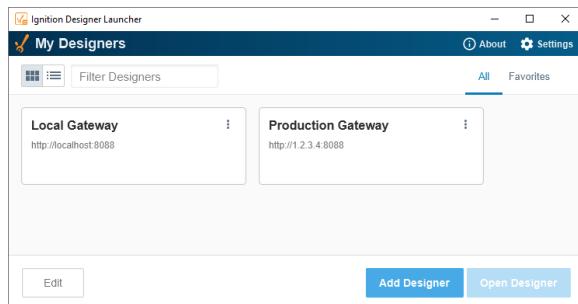
The application launchers and Perspective Workstation mentioned above, all download, install and configure their property settings in the same way. Some property settings are unique to a specific launcher application and will be detailed on that specific launcher application's page.

This page describes the download and installation process common to the launcher applications and Perspective Workstation, as well as the common property settings.

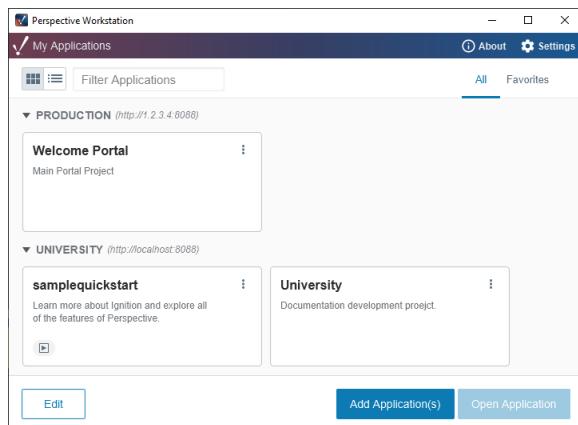
What Is an Application?

To understand the purpose of a Launcher, we need to understand what an application is. An Application is a configuration in a launcher which is associated with a project.

Below we see the Designer Launcher, which is the launcher for the Designer. The launcher has two Applications configured: one titled "Local Gateway", and the other titled "Production Gateway". Selecting either application and clicking "Open Designer" will launch the Designer for the gateway selected.



The Vision Client Launcher and Perspective Workstation work under a similar pattern, except each application represents a project in a gateway. Multiple projects from a single gateway can be represented in a launcher. In the image below, there is a "Welcome Portal" application under the **Production** gateway, while the **University** gateway features two applications each leading to a different project.



Launcher Download and Installation

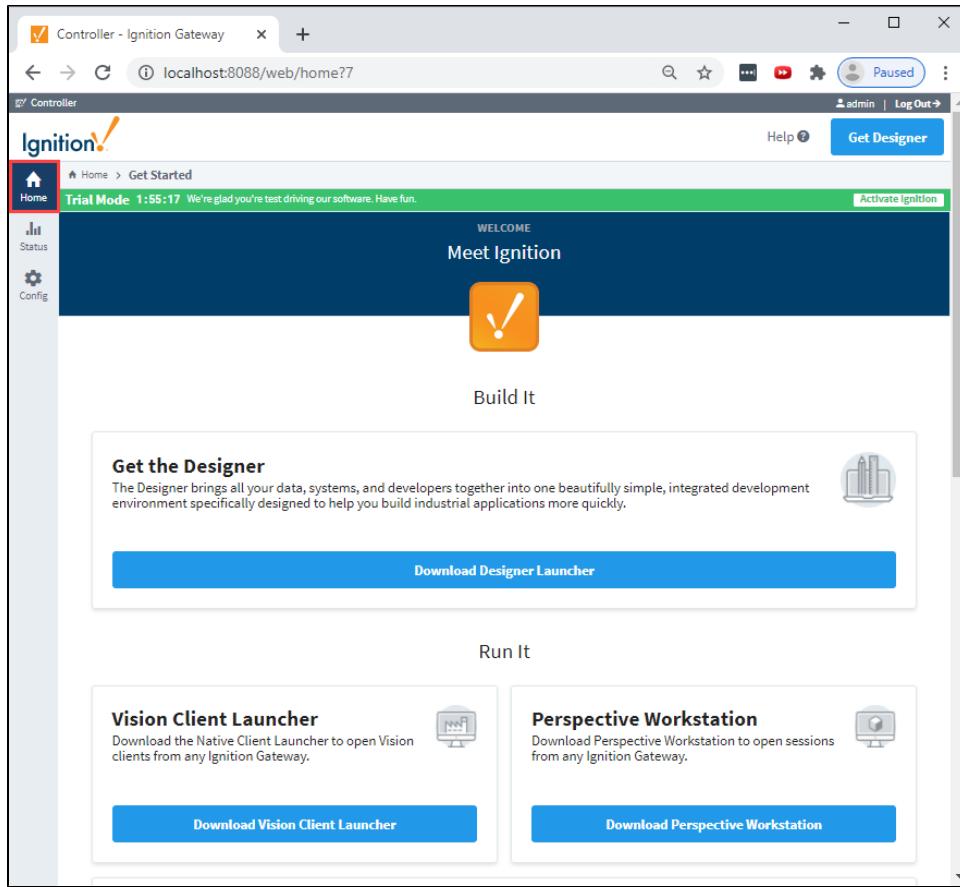
This section demonstrates the common download and installation steps for all the launcher applications. Each launcher application has an install wizard that will guide you through the installation process. The installation steps are common for all the launchers with a few exceptions depending on the operating system you're using.

On this page ...

- [What Is an Application?](#)
- [Launcher Download and Installation](#)
 - [Upgrading Launchers](#)
 - [Adding Applications](#)
 - [Launcher Settings](#)
 - [Property Settings](#)
 - [Security Certificates](#)
 - [Certificates Signed by a Certified Authority](#)
 - [Options Menu](#)
 - [Windows Silent Installations](#)
 - [Hosted Launcher Installers](#)

For the sake of brevity, this section will focus on installing the Designer Launcher, but the steps for installing the Vision Client Launcher and Perspective Workstation are all very similar.

1. Go to the **Home** tab of the Gateway webpage. You will see buttons that lead to separate download pages for each of the launchers. Click the desired button and you'll be redirected. We'll click on **Download Designer Launcher**.



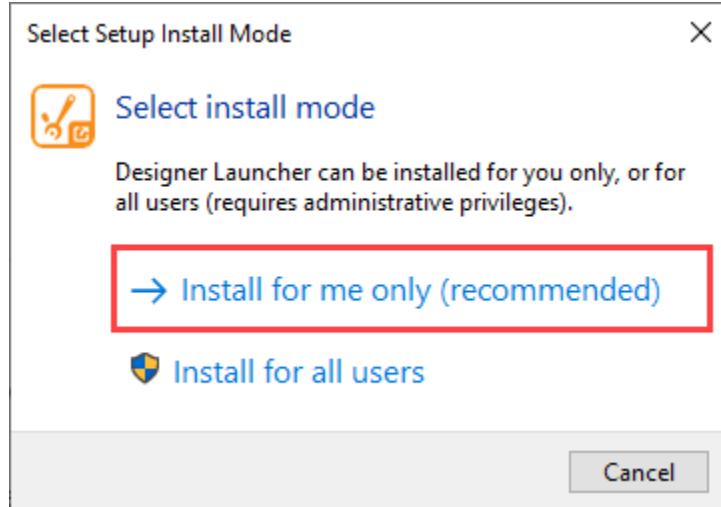
2. Your web browser will detect the operating system, and suggest the appropriate download for the Download button. Press the button to download the installer for the launcher.

The screenshot shows the Ignition Gateway interface for downloading the Designer Launcher. The main content area displays instructions for Windows users, including a screenshot of a file download dialog and three steps for setup. On the right, there's a sidebar for alternative launcher downloads.

3. Locate the installer on your local system (web browser typically store downloaded files in a "Downloads" folder), and run it.
4. Work through the install wizard. There are only a handful of options in the installer:

- Whether or not to create a desktop icon
- The installation directory for the launcher.

If this is a first time install, the Designer Launcher wizard has an additional step. It will prompt you to select the install mode. Installing locally installs the launcher in the user's AppData\Roaming folder whereas 'all users' installs it in Program Files.



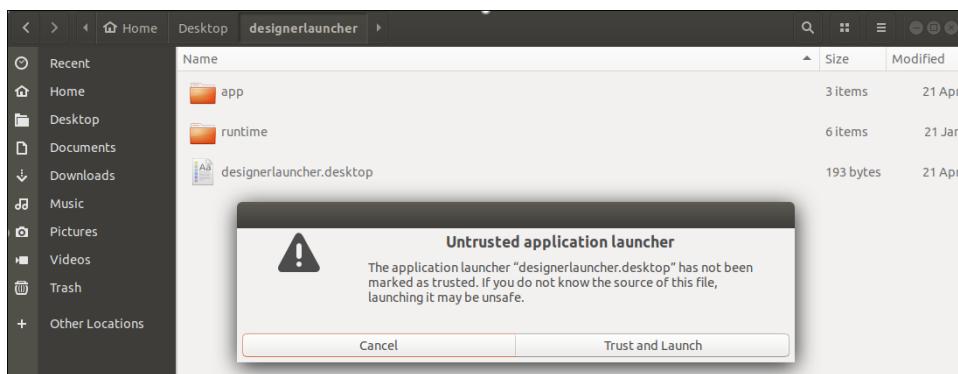
To install the Designer Launcher application, drag the Designer Launcher icon to the Applications folder.



Note: At this point, if you're presented with an error stating that the system is unable to open the installer, take a look at our [Knowledge Base Article](#).

Extract the contents of the '**tar.gz**' file to your desired install location.

Navigate to your install folder, and click the '**designerlauncher.desktop**' file. A message box may popup about the launcher application being an untrusted application. Click **Trust and Launch**.



5. Once the installation process reaches its end, click **Finish**.
6. The Designer Launcher will open, allowing you to start adding applications.

Upgrading Launchers

Whenever an Application is launched from a Launcher, the Launcher will check to see if it needs an update. If the Gateway containing launched application is several revisions ahead of the Launcher, the Launcher will provide a popup notifying you that an update is available.

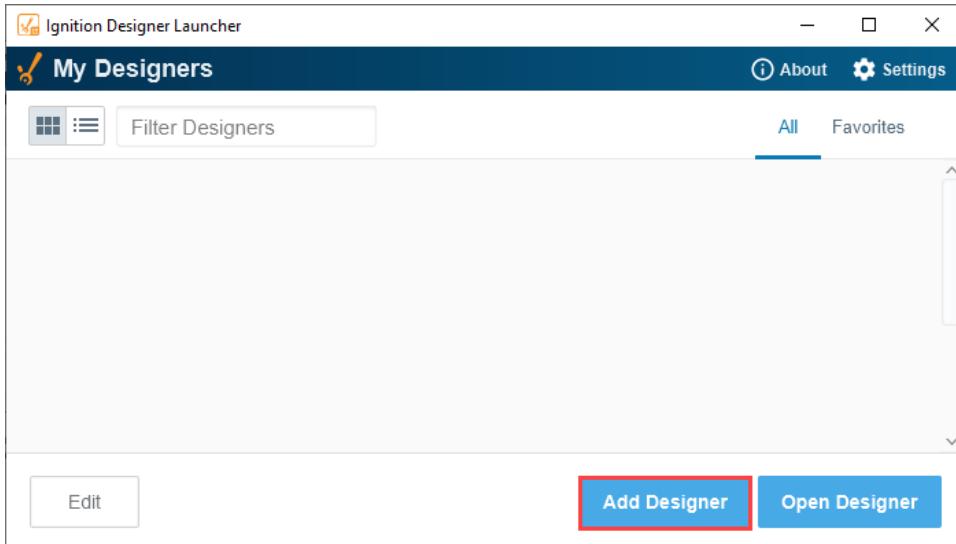
However manual upgrades can be performed. Simply download and run a new launcher installer from a more recent Ignition Gateway. During installation, simply set the installation directory to the same directory as the old launcher.

Adding Applications

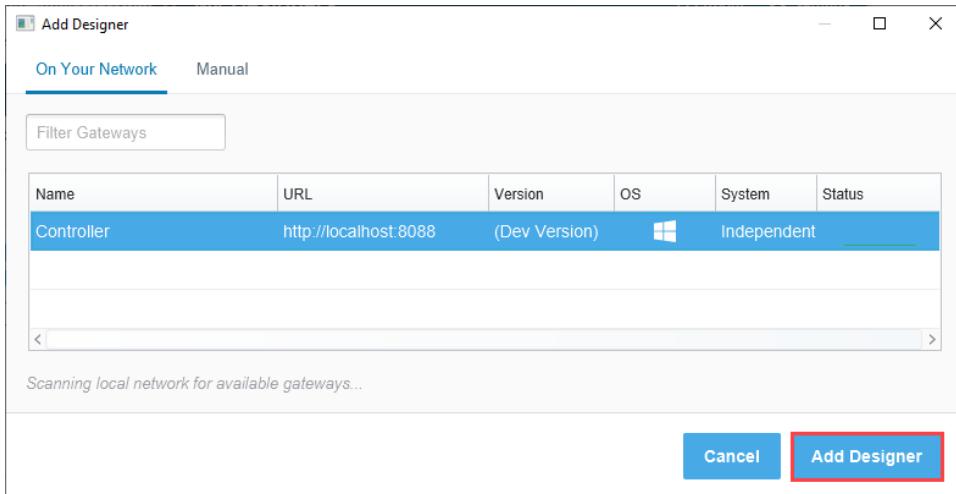
The process for adding Applications/Designers to the Designer Launcher is very similar to adding Applications to the Vision Client Launcher and Perspective Workstation. This section demonstrates how to add new Applications.

The following Designer Launcher example is used to show how to add applications for the Designer and Vision Client Launchers and the Perspective Workstation.

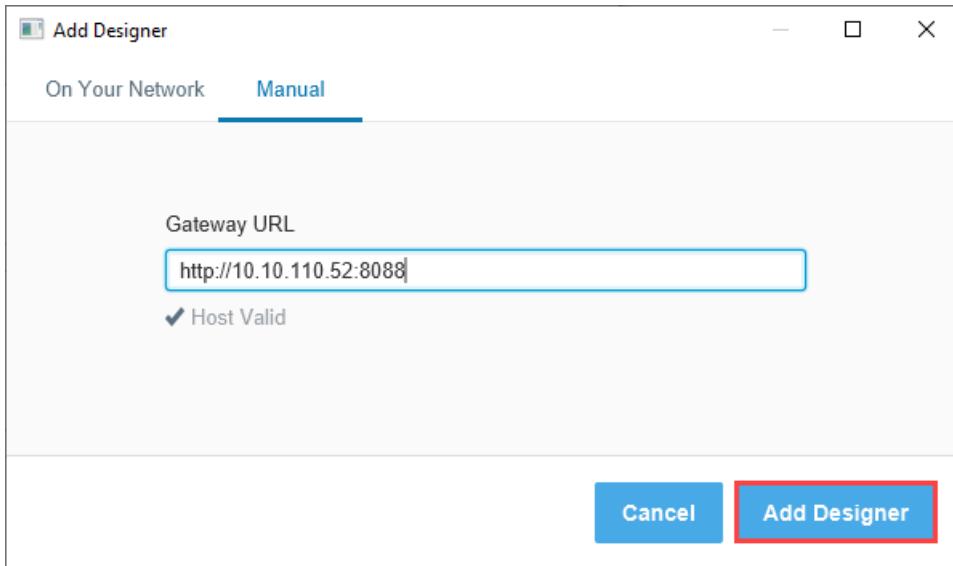
1. With the launcher open, click the **Add Designer** button.



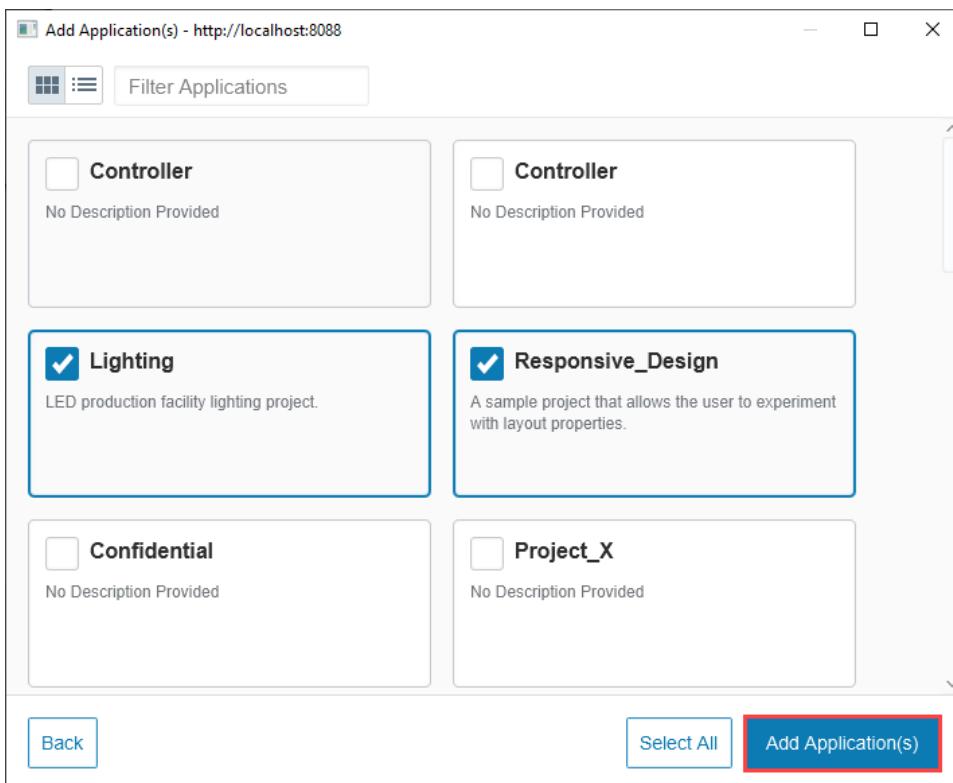
2. The launcher will browse and list all the available Gateways on your network. Select from any of the available gateways and press the **Add Designer** or **Select Gateway** button depending on which Launcher you are using.



If you don't see the Gateway you're looking for, but know the IP address, you can enter it manually. Click the **Manual** tab and enter the **Gateway URL**. It will inform you if you have a valid host name. If so, click **Add Designer** or **Select Gateway** depending what launcher application you are using.



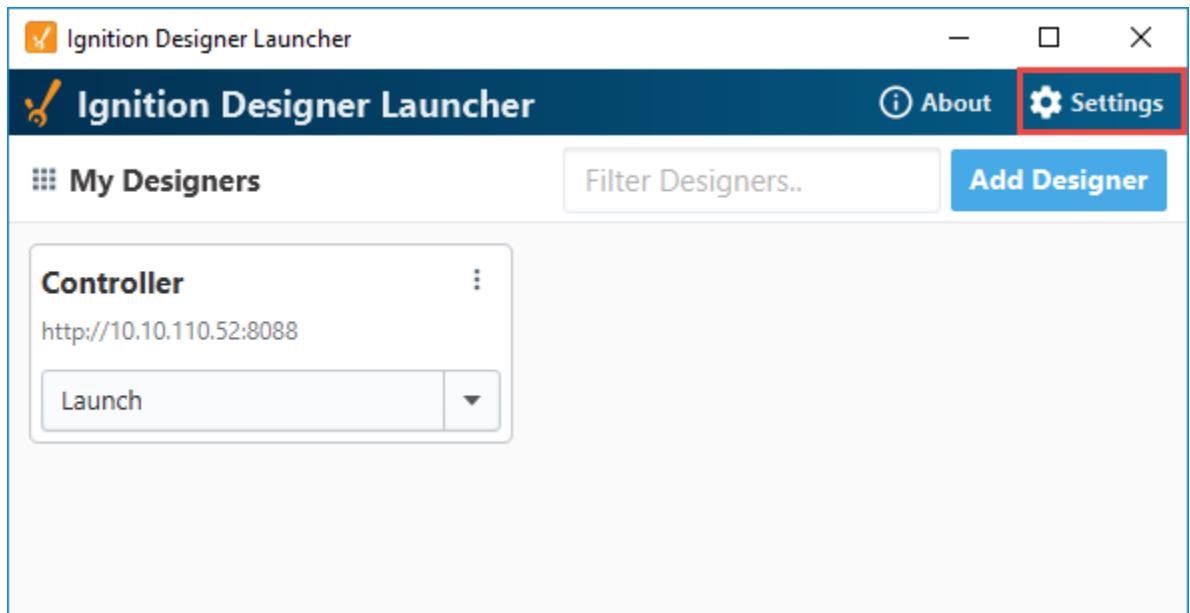
3. If you're adding an Application to the Vision Client Launcher or Perspective Workstation, you'll have this extra step where you can select from any of the available projects on the Gateway. Select the desired projects, then click **Add Application(s)**.



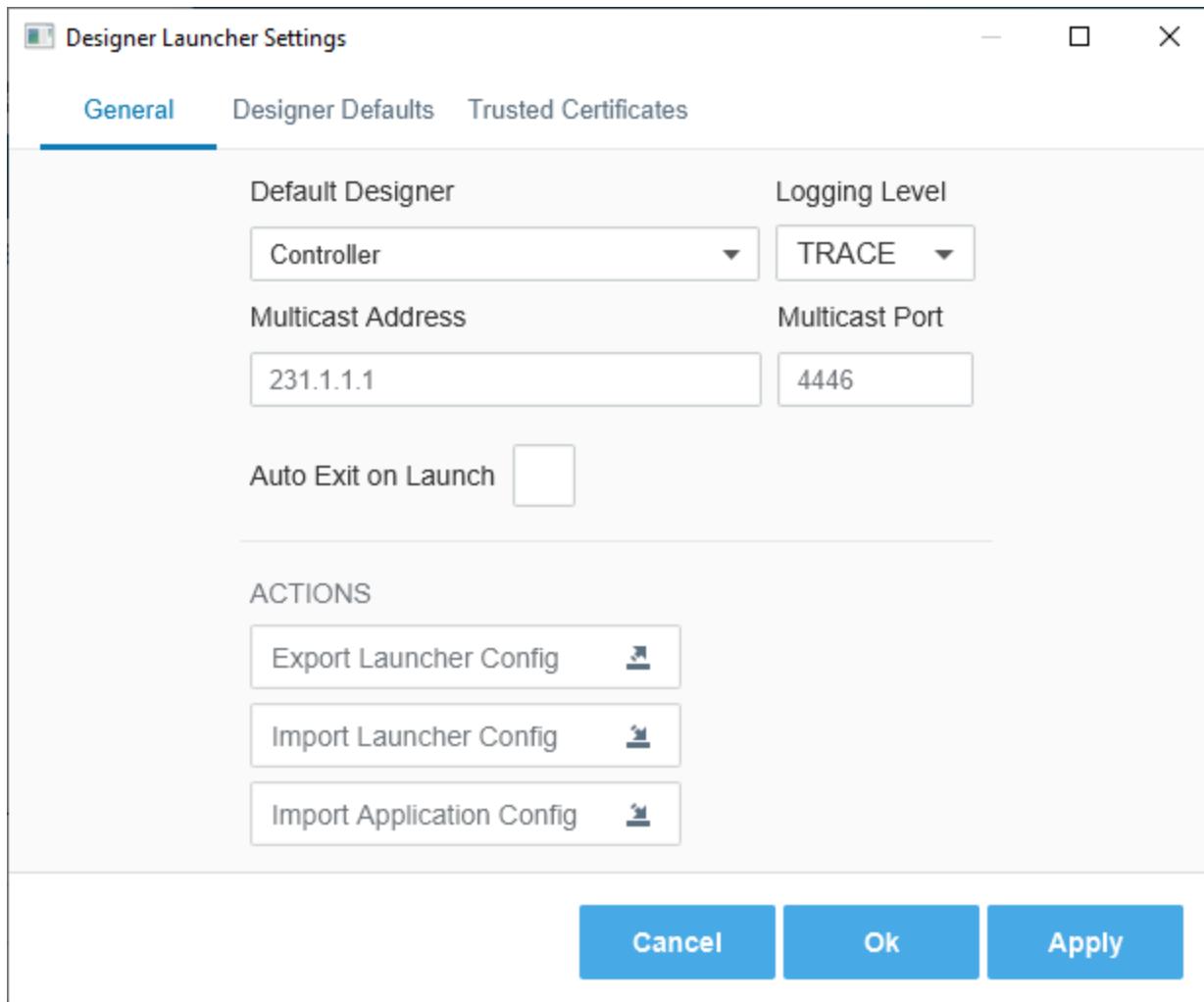
4. Once added, the new Applications will appear on the starting screen of the Launcher.

Launcher Settings

Each Launcher has its own settings. To access the settings on a Launcher, click **Settings** on the top right of any launcher window.



The Settings popup window will appear, listing settings for the Launcher to use. There are some shared settings, such as those under the **General** tab.



Property Settings

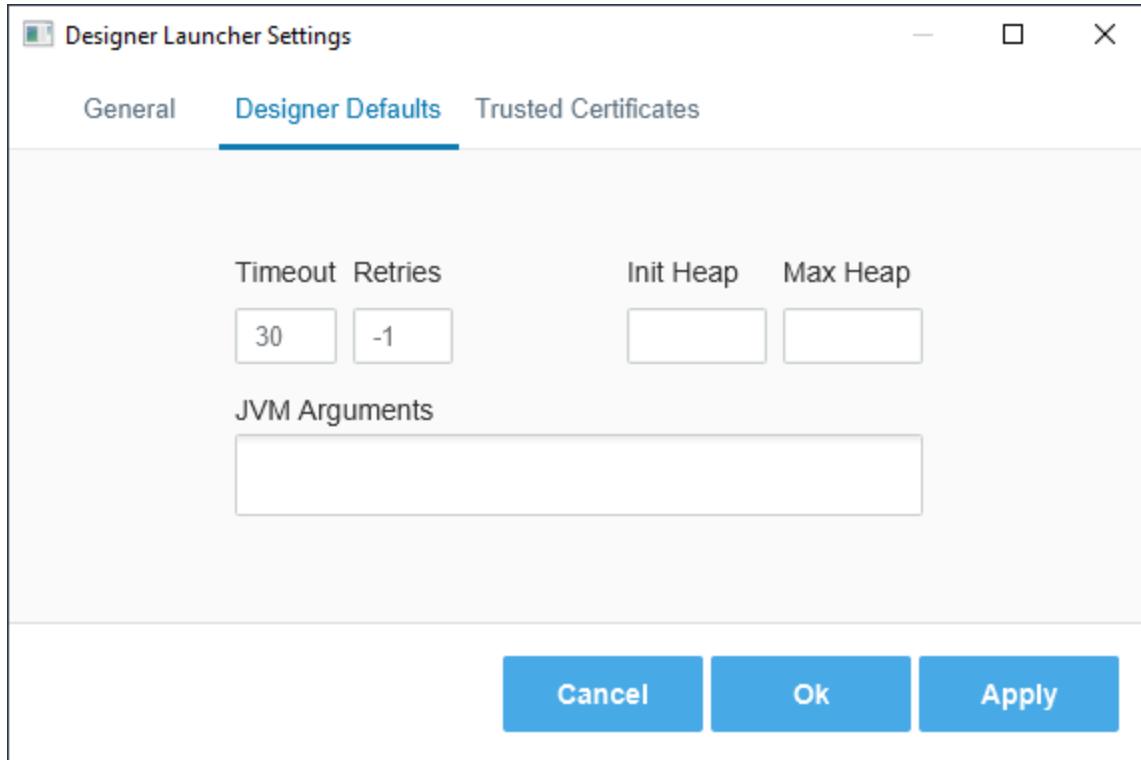
The following tables represent common property settings in the [Designer Launcher](#), [Vision Client Launcher](#) and [Perspective Workstation](#). There are a few settings that are unique to each launcher application and are addressed on the specific application launcher page.

General Property Settings

| General Settings - for the Design Launcher, Vision Client Launcher and Perspective Workstation | | |
|--|--|------------------------|
| Property Setting | Description | JSON Name |
| Default Application | When set, the Launcher will attempt to automatically launch the specified Application on Launcher startup. If left blank/null, an Application will not automatically start | default.application |
| Logging Level | Level of logging that will be used for the launcher. Useful for our support team, but can be used to help troubleshoot problems with the launcher. | logger.level |
| Multicast Address | The address that will be used to listen for multicast broadcasts from Gateways. | multicast.address |
| Multicast Port | The port that will be used to listen for multicast broadcasts from Gateways. | multicast.receive.port |
| Auto Exit on Launch | Automatically close the launcher window when an Application is started. | autoexit |

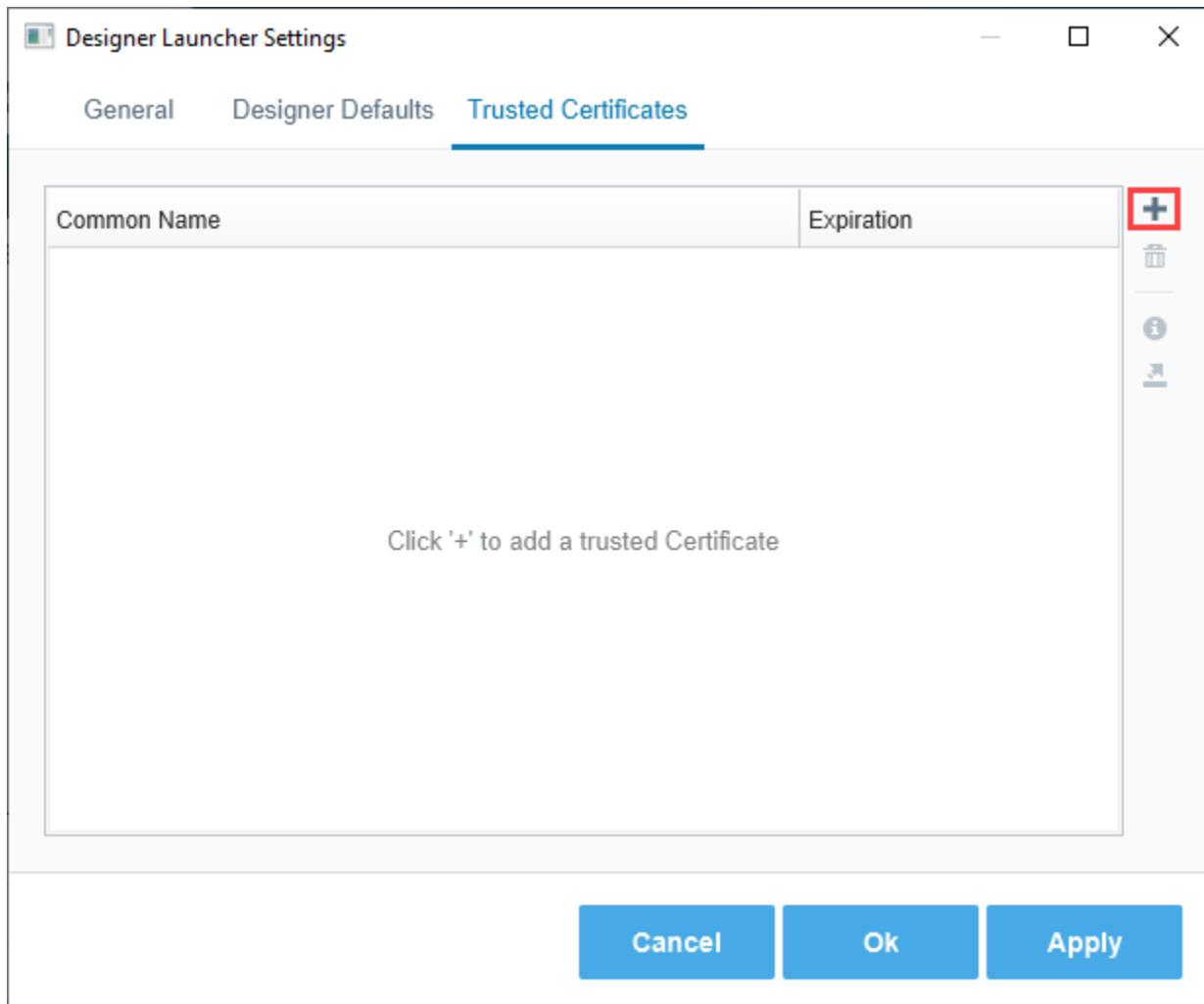
Default Property Settings

| Default Settings - for the Designer Launcher and Vision Client Launcher | | |
|---|--|---------------|
| Property Setting | Description | JSON Name |
| Timeout | Maximum number of seconds allowed for any Gateway communication. Any communication that exceeds this amount will cause the Launcher to abort and try again (if multiple retries are configured). | timeout |
| Retries | The number of times the Launcher will attempt to contact a gateway when launching an application. Available values are: <ul style="list-style-type: none">• -1 : Retry indefinitely, or until the launcher is manually closed.• 0 : Zero retries, or abort after the first failure.• 1 (or more): Determines the number of retries: i.e., a value of "5" means five retries. | retries |
| Initial Heap | Amount of heap memory to be allocated to the application on startup. | init.heap |
| Max Heap | Maximum amount of heap memory allocated to the application. | max.heap |
| JVM Arguments | Arguments to append to client startup. These should start with a '-D' and be separated by semi-colons. Mostly used by our support teams to aid with troubleshooting. | jvm.arguments |



Security Certificates

The launchers are able to trust certificates from each Gateway. Normally this is an automated process: if you launch an application from a host Gateway that requires the use of Security Certificates, the launcher will attempt to retrieve the certificate details, and ask you to trust the certificates. If you have certificates from the gateway, you can manually add them to the launcher from the Trusted Certificates tab of the Launcher's settings by clicking the Add icon.



When the launcher is aware of security certificates, you can use the **Trusted Certificates** screen to delete or export the certificates with the **Delete** icon and click the **Export** icon, respectively.

The directory where the Trusted Certificates are stored is under:

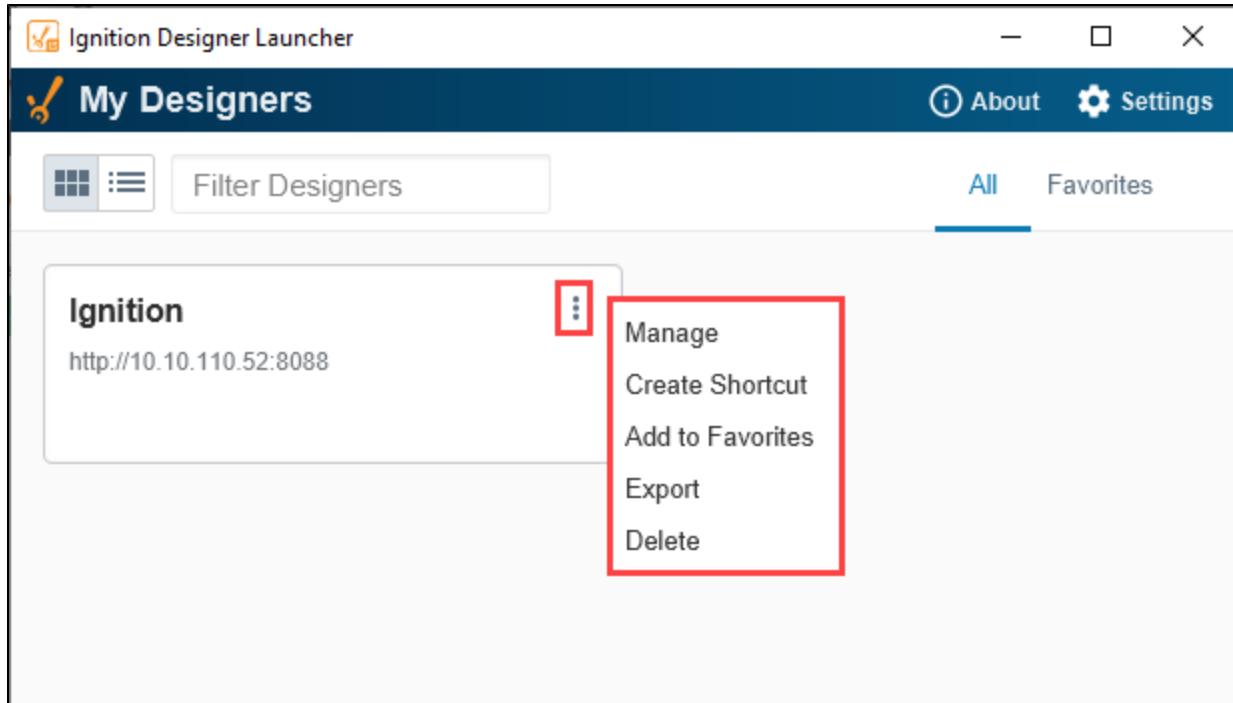
```
{user folder}\.ignition\clientlauncher-data\certificates
```

Certificates Signed by a Certified Authority

When the Gateway's SSL certificates are signed by a recognized Certificate Authority, no additional configuration is required on the launcher.

Options Menu

There is a "three dot menu" on the applications in each of the launchers and Perspective Workstation. When you click on the three dots, a dropdown list appears. There are five options available.



Launcher Options Table

| Option | Description |
|------------------|--|
| Manage | Allows you to configure/update property settings for your application. These properties are unique to each launcher application. |
| Create Shortcut | Creates a shortcut of the application and shows up as an icon on your desktop. |
| Add to Favorites | Creates a Favorite in your application and puts it under the Favorites tab located in the upper right corner. Your application is marked with a star icon denoting a Favorite. |
| Export | Exports the application launcher and configuration settings to a JSON file. It defaults to the application name and you provide the folder path. |
| Delete | Deletes/removes the application name from the launcher application. |

Note: To view the property settings under the **Manage** option, refer to the specific launcher application: [Designer Launcher](#), [Vision Client Launcher](#), and [Perspective Workstation](#).

Windows Silent Installations

The [Designer Launcher](#), [Vision Client Launcher](#) and [Perspective Workstation](#) can be installed on Windows systems from command line without any user prompts.

```
# Shows a simplified user interface
C:\Users\user\Downloads\VisionClientLauncherSetup.exe /CURRENTUSER /SILENT

# Shows no user interface at all.
C:\Users\user\Downloads\VisionClientLauncherSetup.exe /CURRENTUSER /VERYSILENT

# The following option installs silently without installing a desktop shortcut
C:\Users\user\Downloads\VisionClientLauncherSetup.exe /CURRENTUSER /VERYSILENT /MERGETASKS="!desktopicon"
```

Note:

Using ALLUSERS may trigger a check from User Account Controls, so you may need administrator access for the following command to offer a truly silent installation:

```
C:\Users\user\Downloads\VisionClientLauncherSetup.exe /ALLUSERS /VERYSILENT
```

Hosted Launcher Installers

Normally, each Ignition Gateway includes files for the various launchers. When you download a launcher from a Gateway, it simply streams its local launcher files. However, you can override this behavior, causing the Gateway to ignore its local launcher files and instead download launchers from the internet. For more information, see the [Gateway Configuration File Reference](#) page.

If you simply wanted to download one of our hosted launchers, without modifying your Gateway's configuration file, you can download them directly from your web browser:

```
https://files.inductiveautomation.com/launchers/VERSION/LAUNCHER
```

Where VERSION is the Ignition version (i.e., 8.1.0), and LAUNCHER is the name of the launcher. Launcher names are listed below.

| Launcher | Operating System | Launcher Name |
|-------------------------|------------------|-------------------------------|
| Vision Client | Windows | visionclientlauncher.exe |
| Vision Client | Linux | visionclientlauncher.tar.gz |
| Vision Client | macOS | visionclientlauncher.dmg |
| Perspective Workstation | Windows | perspectiveworkstation.exe |
| Perspective Workstation | Linux | perspectiveworkstation.tar.gz |
| Perspective Workstation | macOS | perspectiveworkstation.dmg |
| Designer | Windows | designerlauncher.exe |
| Designer | Linux | designerlauncher.tar.gz |
| Designer | macOS | designerlauncher.dmg |

For example, the following would download an 8.1.0 version of Perspective Workstation for Windows:

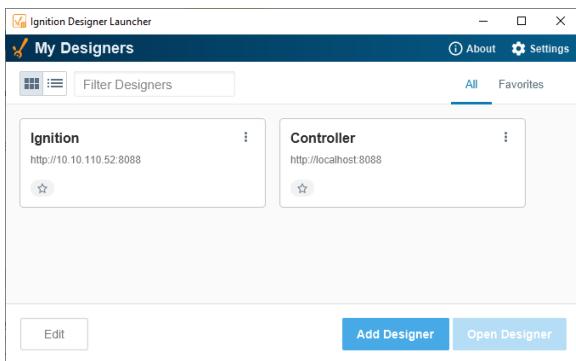
Example

```
https://files.inductiveautomation.com/launchers/8.1.0/perspectiveworkstation.exe
```

Designer Launcher

The Designer Launcher is the only means of launching a Designer, allowing you to make changes against a Gateway. The same Designer can access all the projects that are available on that Gateway, and you don't need to add the Designer multiple times for different Gateways.

The Designer Launcher is available on the **Home** tab of the Gateway Webpage. More information on installing the Designer Launcher can be found on the [Launchers and Workstation](#) page.



On this page ...

- [Designer Launcher Settings](#)
- [Designer Application Settings](#)
- [Designer General Settings](#)



The Designer Launcher

[Watch the Video](#)

Designer Launcher Settings

Settings for the entire Launcher can be found on the [Launchers and Workstation](#) page.

Designer Application Settings

Settings for applications in the Designer Launcher are listed below. To access the application settings, click on the "three menu dots" on the **My Designers** window and select the **Manage** option.

Designer General Settings

| Setting Name | Description |
|-----------------|---|
| Designer Name | Name of the Designer. |
| Gateway Address | The address to the Gateway in the format http[s]://host.port. |
| Timeout | The maximum number of seconds to allow for any Gateway communication. Any communication that exceeds this amount will cause the Vision client launcher to abort the communication and try again if configured. |
| Retries | How many times to attempt to contact a Gateway again if an error occurred during communication. Available values are: <ul style="list-style-type: none">• -1 : Retry indefinitely, or until the launcher is manually closed.• 0 : Zero retries, or abort after the first failure.• 1 (or more): Determines the number of retries: i.e., a value of "5" means five retries. If the number of retries is exceeded, then the launcher will attempt to launch the Fallback Application. |
| Initial Heap | The initial heap size (memory) for the Client. Defaults to the value specified on the Gateway Home page. |
| Maximum Heap | The maximum heap size (memory) for the Client. Defaults to the value specified on the Gateway Home page. |
| JVM Arguments | Additional JVM arguments to add to that specific application if desired. The defaults are created from the global jvm-args that exist at the time of the creation of the application. These can then be modified and the modifications will persist and never be updated again from the global setting. |

 Configure Controller

General

Designer Name

Gateway Address

Timeout Retries

Init Heap Max Heap

JVM Arguments

ACTIONS



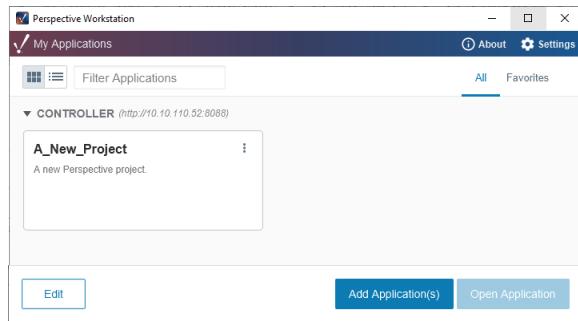


Perspective Workstation

This feature is new in Ignition version **8.1.0**
[Click here](#) to check out the other new features

The Perspective Workstation is a single application that acts as both a launcher, and desktop "wrapper" for Perspective Sessions. Perspective Workstation allows you to run your Perspective applications as first class desktop applications designed for HMI's, panel PCs and dedicated workstations.

Perspective Workstation has built-in features to run in Windowed mode and full screen Kiosk mode eliminating any distractions from the underlying operating system. There are also features to manage applications running on dedicated workstations with multiple monitors configured to run on specific monitors.



On this page ...

- [Managing Workstation Settings](#)
 - [Launch Modes - Windowed and Kiosk](#)
 - [Tab/Window Links](#)
 - [Multi-Monitor Support](#)
 - [Perspective Component Actions and Scripts](#)
- [Perspective Workstation Settings](#)
- [Perspective Workstation Application Settings](#)
 - [General Application Settings](#)
 - [Page Configuration Settings](#)
- [Perspective Workstation - On-screen Keyboard](#)
- [Redundancy](#)
- [Command Line Arguments](#)
- [System Requirements](#)

Note: Perspective Workstation will only be able to launch sessions against 8.1+ Ignition Gateways.



Perspective Workstation

[Watch the Video](#)

Managing Workstation Settings

The Perspective Workstation application settings are unique to each Perspective Session. To access Workstation settings, click on the "three menu dots" on the **My Applications** window and select the **Manage** option. The property setting tables in the [Perspective Workstation#Perspective Workstation Settings](#) section below shows the settings that can be applied to each session.

When you open the Manage tab and navigate through some of the screens, some of the settings will be pre-populated.

Launch Modes - Windowed and Kiosk

Perspective Workstation has built-in features to run either Windowed mode or in full screen Kiosk mode eliminating any distractions from the underlying operating system.

Tab/Window Links

Tab/Window Links has three settings to select choose from that can either allow users to open links in a web browser, in a window, or block users from accessing the operating system.

Multi-Monitor Support

Workstation supports launching across multiple monitors. A single application configured in Perspective Workstation can open multiple application windows (operating system windows). Each page is represented on a separate operating system window with each monitor representing a different Perspective page. Users can specify which pages to appear on each monitor on the Page Configuration window.

When configuring multi-monitor settings, Workstation automatically detects and shows you how many monitors are available on the Page Configuration page. The page configuration settings sets the page(s) that the session will start on. Users mount which page they want for each display. If the Page field is empty, nothing will appear. If a Page field contains a URL, a page will be mounted. A Primary display indicates where you will sign in and identify what window will present you with the sign-in dialog box. There is an Identify Display setting that you can set to provide an overlay with what display you are looking at to help you with your configuration.

Note: The default Page Configuration when adding an application is "/" meaning that we load the default page as configured in the Designer when creating a Page Configuration. For more information refer to [Page Configuration](#).

Perspective Component Actions and Scripts

Perspective resources have access to some Workstation only features, allowing you to develop a project that has some built-in Workstation controls.

The [Component Events and Actions](#) page highlights the Workstation Component Action, allowing you to switch the mode of a running Workstation Application.

The [system.perspective.workstation](#) leads to various system functions that further allow you to manipulate a running Workstation Application.

Perspective Workstation Settings

Settings for the entire Launcher can be found on the [Launchers and Workstation](#) page.

Perspective Workstation Application Settings

Settings for applications in Perspective Workstation are listed below. To access the application settings, click on the "three menu dots" on the [My Applications](#) screen and select the **Manage** option.

General Application Settings

| Setting Name | Description |
|------------------|---|
| Application Name | The name of the application that appears in the Launcher. |
| Launch Mode | The mode that the application will run in that determines the default launch behavior of the application. There are two options: <ul style="list-style-type: none">• Windowed - Launches the session in window mode.• Kiosk - Launches a session in full-screen mode. Prevents users unable to access the underlying filesystem and close the Perspective Workstation application. |
| Tab/Window Links | Determines what happens when a user clicks a link in the session that would open a new tab in a standard browser. <ul style="list-style-type: none">• Blocked - Nothing happens when the user clicks the link.• In Window - Opens the linked page within Workstation.• System Browser - Open links in the user's default web browser. |
| Description | A description that identifies what the application is for. The Description is displayed on the Application on the My Applications screen. |
| Gateway Address | The address to the Gateway in the format http[s]://host/port. |
| Project URL | The URL path to the Perspective page the project will load. |

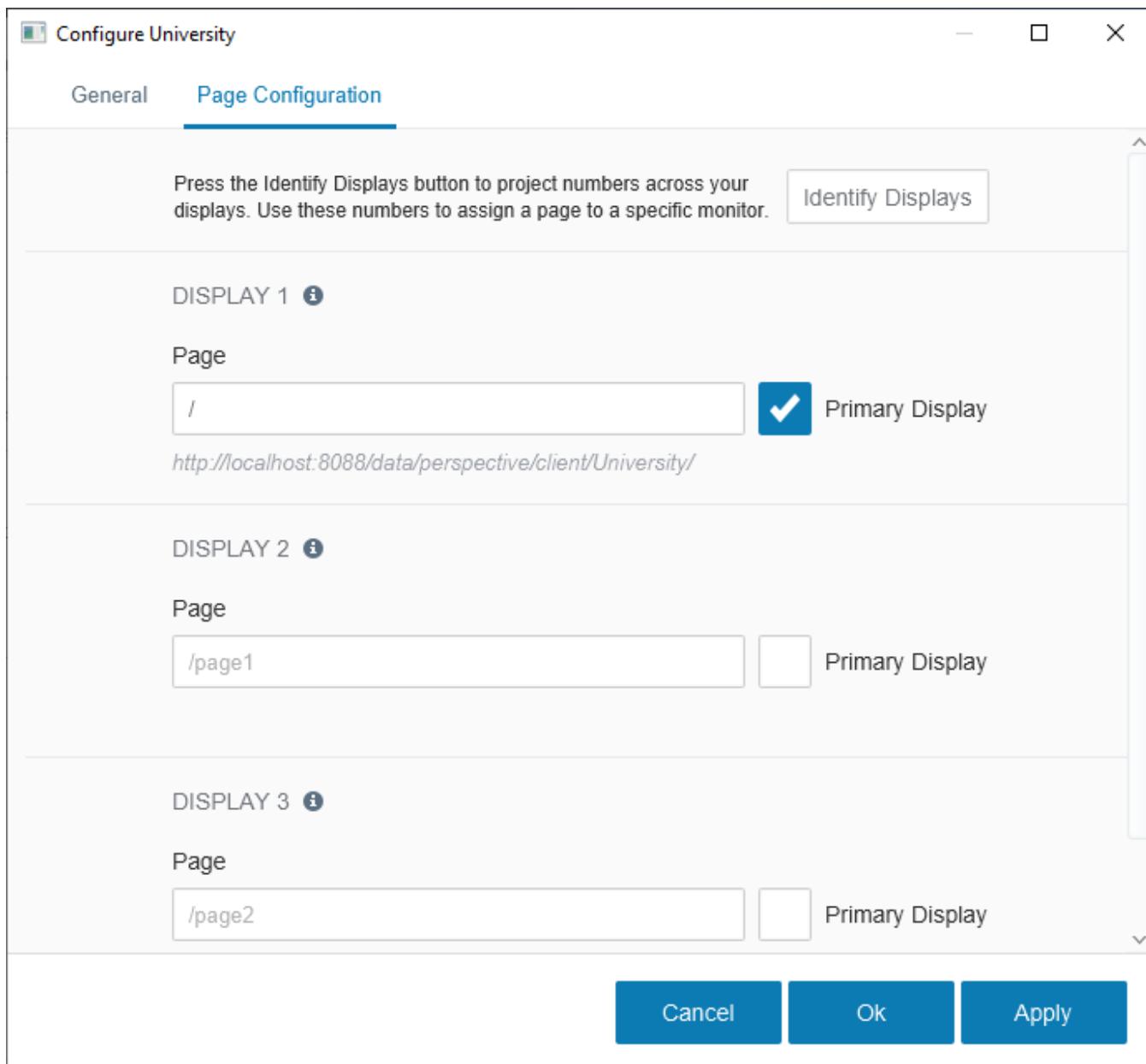
Configure A_New_Project

General Page Configuration

| | | |
|---|-------------|------------------|
| Application Name | Launch Mode | Tab/Window Links |
| A_New_Project | Windowed ▾ | Blocked ▾ |
| Description | | |
| A new Perspective project. | | |
| Gateway Address | | |
| http://10.10.110.52:8088 | | |
| Project URL | | |
| /data/perspective/client/A_New_Project | | |
| <i>http://10.10.110.52:8088/data/perspective/client/A_New_Project</i> | | |
| ACTIONS | | |
| Export Config  | | |
| Create Desktop Shortcut  | | |
| Cancel | | Ok |
| | | Apply |

Page Configuration Settings

| Setting Name | Description |
|-------------------|--|
| Identify Displays | Identifies display numbers across your displays. Use these numbers to assign a page to a specific monitor. |
| Page | The page within the Perspective project to show on the display. |
| Primary Display | If checked, logins and other operations will happen on this display. |



This feature is new in Ignition version **8.1.4**.
[Click here](#) to check out the other new features

Perspective Workstation - On-screen Keyboard

As of 8.1.4, Perspective Workstation now better integrates with your local operating system's on-screen keyboard. If an on-screen keyboard has been enabled in the local operating system, then the keyboard will appear automatically within a running Perspective session that was opened from Perspective Workstation when it is needed (such as the user clicking into a text field).

Note that the on-screen keyboard will not automatically appear when interacting directly with Perspective Workstation (configuring workstation apps, pages, importing certificates, etc.).

Redundancy

Perspective Workstation can take advantage of a redundant Gateway setup. Whenever a connection is established with a master Gateway, the backup Gateway IP address is automatically stored in the launcher configuration file. If the master Gateway cannot be contacted the next time the launcher is run, an attempt is made to contact the backup Gateway. If the backup cannot be contacted, the launcher switches between contacting the primary Gateway and the backup Gateway until one responds or the user closes the launcher.

Command Line Arguments

Perspective workstation can be called from command line. The following arguments are available when doing so.

| Argument | Description |
|--|---|
| application | The name of the workstation application to launch. |
| debugPort | Allows specifying a specific debug port for the launched application. Default is 9224. |
| launch.mode | Override the launch mode to launch the application in. Options are: <ul style="list-style-type: none">• WINDOWED• KIOSK |
| browser.tab.mode | Override the Tab/Window mode for the launched application. Available options are: <ul style="list-style-type: none">• BLOCKED• IN_WINDOW• SYSTEM |
| config.json | Allows you to point Perspective Workstation to a configuration file from command line. Doing so will start running an instance of workstation using the configurations in the file as a temporary override. The argument expects a path to a JSON export file, specifically the same file that's created by the Export Launcher Config button under workstation's Settings menu. |
| <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 10px;">Example<pre>"C:\Launchers\Persepctive Workstation.exe" config.json="C:\Users\MyUser\Desktop\workstation.json"</pre></div> | |

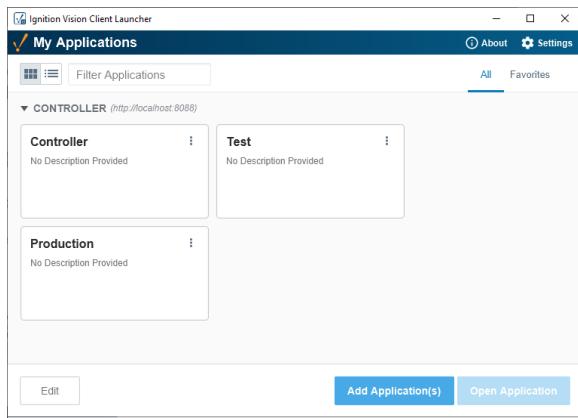
System Requirements

Workstation utilizes JxBrowser. As a result it will only run on 64-bit operating systems that are supported by JxBrowser (ARM OS's are not supported at this time): <https://jxbrowser-support.teamdev.com/docs/quickstart/requirements.html>

Vision Client Launcher

The Vision Client Launcher opens Vision Clients from an Ignition Gateway. Once your Vision projects are added to the Vision Client Launcher, they will be displayed under **My Applications** and organized by Gateway.

The Vision Client Launcher is available on the **Home** tab of the Gateway Webpage. More information on installing any of the Launchers can be found on the [Launchers and Workstation](#) page. This page describes the unique Vision Client Launcher settings, deploying the launcher and Command Line Arguments.



On this page ...

- [Vision Client Launcher Settings](#)
- [Vision Client Application Settings](#)
 - [Vision Client Launcher Application Settings](#)
 - [Vision Client Tag Override Settings](#)
- [Redundancy](#)
- [Command Line/Terminal](#)
 - [Command Line Arguments](#)



Launching a Client

[Watch the Video](#)

Vision Client Launcher Settings

Settings for the entire Launcher can be found on the [Launchers and Workstation](#) page.

Vision Client Application Settings

This section details settings for applications in the Vision Client Launcher. To access the settings, click on the "three menu dots" on the **My Applications** window and select the **Manage** option. The General tab will open. The table below shows the General settings that can be applied to each designer.

Vision Client Launcher Application Settings

| Setting Name | Description |
|-----------------------|--|
| Application Name | The descriptive name of the application. This is independent of the project name that corresponds to a project on the gateway. Instead, this is the name of the application as it is defined in the Launcher. |
| Gateway Address | The address to the Gateway in the format http[s]://host:port |
| Description | An optional description for the application that will be listed on the Application in the launcher. Defaults to the Project's description. |
| Vision Client Project | The name of the project containing Vision resources that the Application will attempt to launch. |
| Fallback Application | The name of the application (configured in the same launcher) to use if the number of retries has been exceeded. The fallback is only utilized if the Retries setting is greater than 0. |
| Image Path | This will allow the icon of the application to be set within the designer and it will be downloaded and displayed as the icon for the application as well as the shortcuts. If no path is set, the default icon is used. If this is set to a different icon path, that icon will be used. The image path notation is a filepath on the local system. |
| Window Mode | Controls the client mode. Available options are: |

| | |
|---------------|--|
| | <ul style="list-style-type: none"> • window : Launches the client in Windowed Mode • fullscreen : Launches the client in Fullscreen Mode |
| Screen Index | The screen index indicates which monitor to use. |
| Timeout | The maximum number of seconds to allow for any gateway communication. Any communication that exceeds this amount will cause the Vision client launcher to abort the communication and try again if configured. |
| Retries | <p>How many times to attempt to contact a gateway again if an error occurred during communication. Available values are:</p> <ul style="list-style-type: none"> • -1 : Retry indefinitely, or until the launcher is manually closed. • 0 : Zero retries, or abort after the first failure. • 1 (or more): Determines the number of retries: i.e., a value of "5" means five retries. If the number of retries is exceeded, then the launcher will attempt to launch the Fallback Application. <p>If the number of retries is exceeded, then the Launcher will attempt to launch the Fallback Application.</p> |
| Init Heap | The initial heap memory size for the Client. Defaults to the value specified on the Gateway for that project. |
| Max Heap | The maximum heap memory size for the Client. Defaults to the value specified on the Gateway for that project. |
| JVM Arguments | Arguments to append to client startup. These should start with a '-D' and be separated by semi-colons. |

Configure Controller

General Client Tag Overrides

Application Name
Controller

Gateway Address
http://localhost:8088

Description

Vision Client Project
Controller1

Fallback Application

Image Path
VisionIcon.ico

Window Mode Screen Index Timeout Retries Init Heap Max Heap
 window ▾ 0 30 -1 32M 256M

JVM Arguments

ACTIONS

Export Config

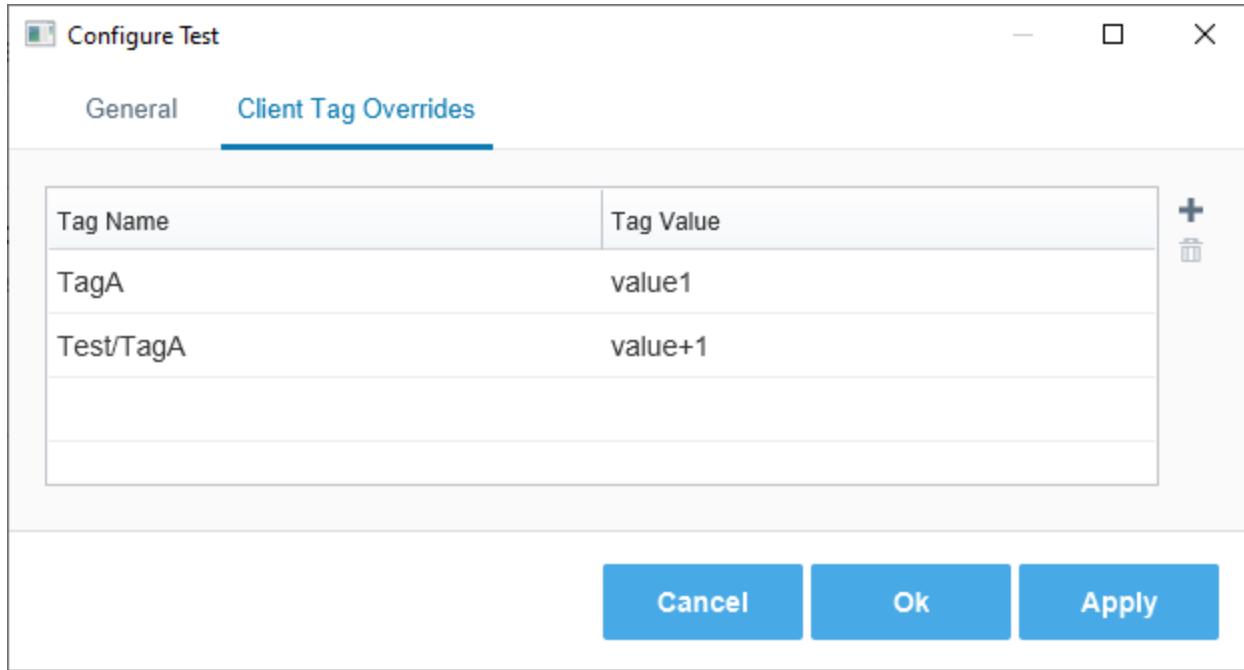
Create Desktop Shortcut

Cancel **Ok** **Apply**

Vision Client Tag Override Settings

Click on the **Client Tag Overrides** tab to create or update any Tags. The table below shows the Client Tag Override setting that can be applied to each designer.

| Setting | Description |
|----------------------|--|
| Client Tag Overrides | <p>Allows client tags to be overridden on the Vision Client. These are set using the table on the application configuration screen by adding rows with the names of the tags and the corresponding values.</p> <p>When applying the overrides from command line, the Tags must first be defined as launch parameters, then a value can be set on the parameters.</p> <p>Vision Client Tag overrides within folders and passing client tag overrides are supported by using a '+' icon as an escape character. This happens automatically if you are configuring overrides in the Client Launcher. To learn more about how to configure Client Tag Overrides, refer to Overriding Vision Client Tags.</p> |



For additional application settings, refer to the [Launchers and Workstation](#) page.

Redundancy

The Vision Client Launcher can take advantage of a redundant Gateway setup. Whenever a connection is established with a master Gateway, the backup Gateway IP address is automatically stored in the client launcher configuration file. If the master Gateway cannot be contacted the next time the client launcher is run, an attempt is made to contact the backup Gateway. If the backup cannot be contacted, the client launcher switches between contacting the primary Gateway and the backup Gateway until one responds or the user closes the launcher.

Command Line/Terminal

Clients can be launched from the Client Launcher via command/terminal. When called in this way, many of the application properties may be overridden for the one call. The overrides use the same property names as noted in the [Application Property Reference Table](#), under the "JSON name" column.

```
Windows:  
"C:\ClientLauncher\visionclientlauncher.exe" application=myproject window.mode=window  
  
Linux:  
../visionclientlauncher.sh application=myterminal window.mode=fullscreen screen=0
```

There are a few important notes when using the Command Line/Terminal to launch a project from the Vision Client Launcher.

- The Vision Client Launcher must be installed and have an application added for the Command Line/Terminal commands to work.
- The **application** argument requires the application name in the Launcher, not the project name. You can open the launcher to determine what the application name is. Adding new applications in the launcher uses the Project title by default.
- Applications may contain spaces in their name. However, when launching from command/terminal, spaces should be escaped with **%20**. For example, if our application was named **my project**, then we could all it with the following:

```
"C:\ClientLauncher\visionclientlauncher.exe" application=my%20project
```

Command Line Arguments

| Argument | Description |
|-------------|--|
| application | The name of the application to launch. |
| | Controls the client mode. Available options are: |

| | |
|-----------------------|---|
| window.mode | <ul style="list-style-type: none"> • window : Launches the client in Windowed Mode • fullscreen : Launches the client in Fullscreen Mode |
| screen | The screen index indicates which monitor to use. |
| fallback.application | The name of the application to use if the number of retries has been exceeded. The fallback is only utilized if the Retries setting is greater than 0. |
| timeout | The maximum number of seconds to allow for any gateway communication. Any communication that exceeds this amount will cause the Vision client launcher to abort the communication and try again if configured. |
| retries | <p>How many times to attempt to contact a gateway again if an error occurred during communication. Available values are:</p> <ul style="list-style-type: none"> • -1 : Retry indefinitely, or until the launcher is manually closed. • 0 : Zero retries, or abort after the first failure. • 1 (or more): Determines the number of retries: i.e., a value of "5" means five retries. If the number of retries is exceeded, then the launcher will attempt to launch the Fallback Application. <p>If the number of retries is exceeded, then the Launcher will attempt to launch the Fallback Application</p> |
| init.heap | The initial heap size (memory) for the client. |
| max.heap | The maximum heap size (memory) for the client |
| -Djavaws.launchparams | <p>Defines client tags that can be overwritten upon launch. The use of this argument alone only defines the client tags that will be overwritten. Setting a value on the tags can be done by an additional argument that utilizes the tag names delimited by a semicolon:</p> <pre>// Establishes the tag names -Djavaws.launchparams="Tag1;Tag2" // Sets values on the tags -Djavaws.launchparams.Tag1=10 -Djavaws.launchparams.Tag2=20 //An actual call would look like: "C:\ClientLauncher\visionclientlauncher.exe" application=myproject -Djavaws.launchparams="Tag1;Tag2" -Djavaws.launchparam.Tag1=10 -Djavaws.launchparam.Tag2=20</pre> |
| config.json | <p>Allows you to point the launcher to a launcher configuration file from command line. Doing so will start running an instance of the launcher using the configurations in the file as temporary overrides. The argument expects a path to a JSON export file, specifically the same file that created by the Export Launcher Config button under the launcher's Settings menu.</p> <pre>"C:\ClientLauncher\visionclientlauncher.exe" config.json="C:\Users\MyUser\Desktop\vision-client-launcher.json"</pre> |

Configuring Pre-Deployed Launchers

A Pre-Deployed configuration means that the launcher and its necessary files are bundled or organized in a way to make it portable and easily moved or copied from machine to machine.

The advantages of this configuration are:

1. The launcher's configuration remains the same no matter where you move the launcher to.
2. Necessary directories and launch files (including SSL certificates added to the `clientlauncher-data/certificates directory`) are copied to the local cache used when launching an Application.
3. The local cache for launching Applications remains decoupled from the launcher configuration, meaning that its footprint remains small and portable.

One caveat here is that a Pre-Deployed launcher has essentially global settings. Settings such as Application Configurations, Arguments, and even Launcher Settings are shared for all users. This means that if somebody changes a setting, every person/computer that launches from the launcher will be affected by it. That being said, [custom shortcuts](#) and even [locking down the launcher](#) can prevent this and ensure that the launched application is reliably launched.

Note: The examples on this page demonstrate configuring a Vision Client Launcher for pre-deployment, but similar steps can be followed for the Designer Launcher and Perspective Workstation.

On this page ...

- [Windows](#)
- [Linux](#)
- [macOS](#)
- [Custom Shortcuts](#)
 - [Windows](#)
 - [Linux](#)
 - [Mac OS](#)
- [Locking the Launcher Configuration](#)
- [Redirect Client Launch Cache](#)

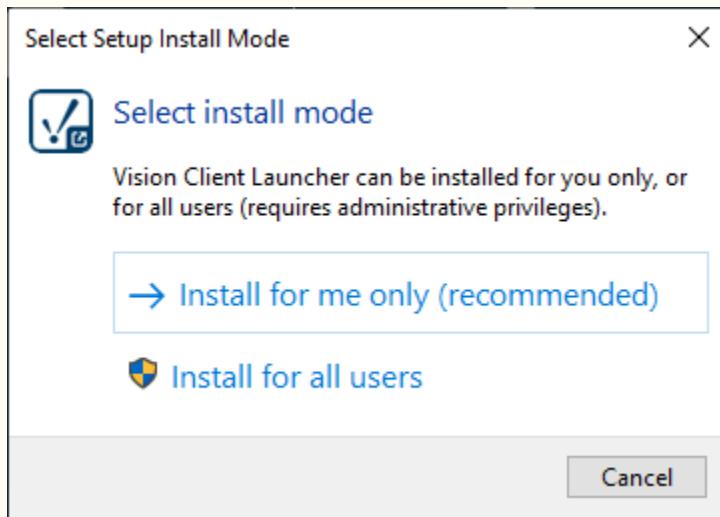
Windows

You must have a copy of the launcher installer you're going to configure for pre-deployment, which you can obtain from an Ignition Gateway. In this example we'll use the Vision Client Launcher's installer: `visionclientlauncher.exe`. In addition, you need to have a clean (empty) `clientlauncher-data` directory. The directory is normally located at: `C:/Users/MY_USER/.ignition/clientlauncher-data`. This folder holds the settings that the launcher will use, so it's best to start with a clean slate.

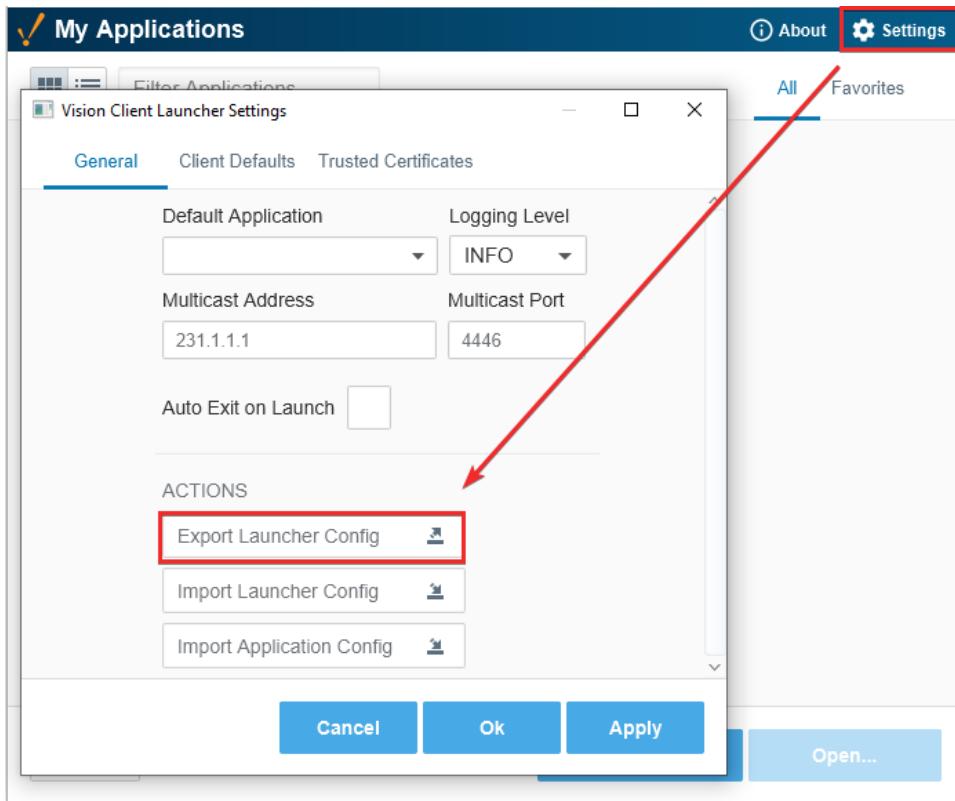
1. Install the Vision Client Launcher on a system.

Note:

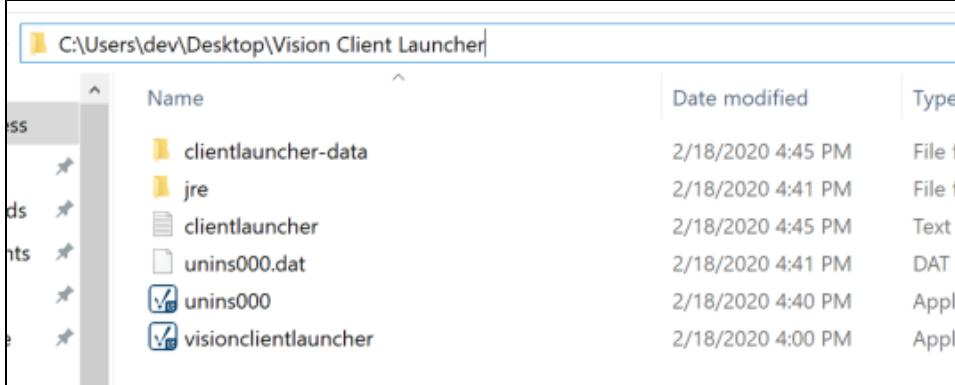
While running through the install process, Windows will ask if you're installing just for yourself or all users. Ultimately, the deployment process will require that you install for you only (the recommended option).



2. Work through the install until you're asked to choose an installation directory. Choose a directory that's easy to access, or create a new folder on your desktop to use as the installation directory. Wherever you choose, you will need to find this installation directory later.
3. Continue with the installation process until it's finished. When prompted to create a desktop shortcut, opt out, since we're ultimately going to be moving the launcher to separate systems.
4. Inside of the launcher's installation, create a directory named `clientlauncher-data`. We'll use this directory later to store configurations for the applications on the launcher.
5. Open the launcher and configure it to your specifications. This includes adding Applications as well as making changes to the launcher's configuration.
6. Once you've finished configuring the launcher, take an export of the launcher. You can export the launcher's settings by clicking on the **Settings** icon, and then clicking the **Export Launcher Config** button.



7. You'll be asked to pick a directory to save the JSON launcher configuration file. Navigate to that `clientlauncher-data` we created in step 5, and place the file in there.
8. Use the default name for the resulting file - in this example it would be named `vision-client-launcher.json` - whereas a file from the Designer Launcher would be named `designer-launcher.json`, and files from Perspective Workstation would be called `workstation.json`.
9. At this point, you'll have a directory that looks something like the following:



If so, then you're ready to deploy. Simply take the installation directory and move it to another system. Once on another system, the launcher will rely on the bundled configuration file and otherwise operate normally.

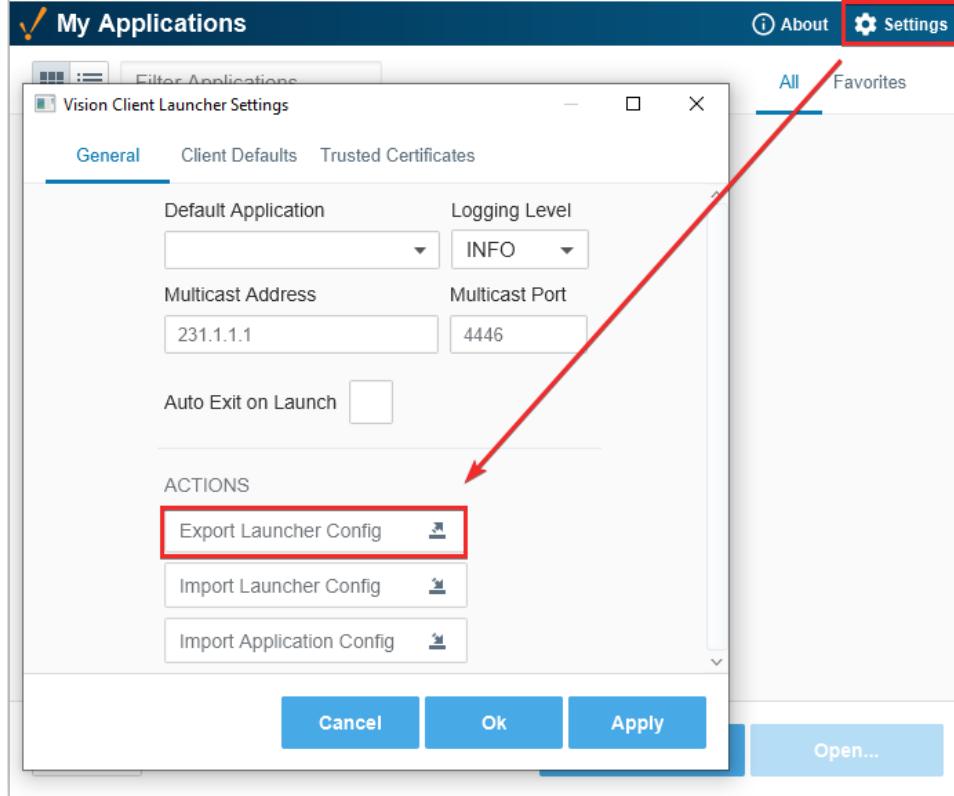
Linux

You must have a copy of the launcher files you're going to configure for pre-deployment, which you can obtain from an Ignition Gateway. In this example we'll use the Vision Client Launcher: `visionclientlauncher.tar.gz`.

In addition, you will need to have a clean (empty) `~/.ignition/clientlauncher-data` directory. This directory holds the settings that the launcher will use, so it's best to start with a clean slate.

1. Unarchive the `.tar.gz` file to somewhere that you can easily access (the desktop is ideal).
2. Make a directory inside the top level launcher folder called `clientlauncher-data`.
3. Open the launcher and configure it to your specifications. This includes adding Applications as well as making changes to the launcher's configuration.

- Once you've finished configuring the launcher, take an export of the launcher. You can export the launcher's settings by clicking on the **Settings** icon, and then clicking the **Export Launcher Config** button.



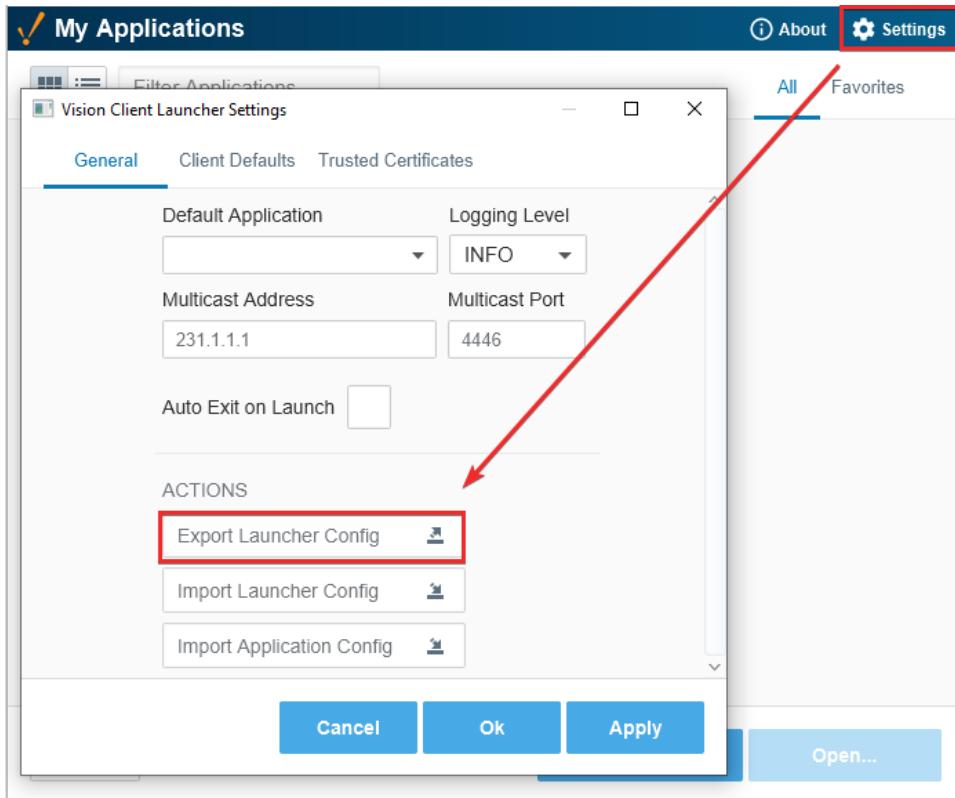
- You'll be asked to pick a directory to save the JSON Launcher Configuration file. Navigate to that `clientlauncher-data` we created in step 2, and place the file in there.
- Use the default name for the resulting file – in this example it would be named `vision-client-launcher.json` – whereas a file from the Designer Launcher would be named `designer-launcher.json`, and files from Perspective Workstation would be called `workstation.json`.
- That's it. You can now take the directory containing all of the launcher files and move it to another system.

macOS

You must have a copy of the launcher files you're going to configure for pre-deployment, which you can obtain from an Ignition Gateway. In this example we'll use the Vision Client Launcher: `visionclientlauncher.dmg`

In addition, you will need to have a clean (empty) `~/.ignition/clientlauncher-data` directory. This directory holds the settings that the launcher will use, so it's best to start with a clean slate.

- Open the DMG and copy the `Launcher.app` file somewhere that you can easily access (the desktop is ideal).
- Make a directory inside the top level launcher folder called `clientlauncher-data`.
- Open the launcher and configure it to your specifications. This includes adding Applications as well as making changes to the launcher's configuration.
- Once you've finished configuring the launcher, take an export of the launcher. You can export the launcher's settings by clicking on the **Settings** icon, and then clicking the **Export Launcher Config** button.



5. You'll be asked to pick a directory to save the JSON Launcher Configuration file. Navigate to that `clientlauncher-data` we created in step 2, and place the file in there.
6. Use the default name for the resulting file - in this example it would be named `vision-client-launcher.json` - whereas a file from the Designer Launcher would be named `designer-launcher.json`, and files from Perspective Workstation would be called `workstation.json`.
7. That's it. You can now take the directory containing all of the launcher files and move it to another system.

Custom Shortcuts

When a shortcut to an application is created, a launchable link is placed on the desktop of the user. This link uses the configuration of the application that is saved in the launcher. This ensures that updates to the launcher are respected even if the link was created before the changes. That being said, you can override the following settings for the shortcut itself:

- "window.mode"
- "timeout"
- "screen"
- "retries"
- "init.heap"
- "max.heap"
- "fallback.application"
- "custom.jre.path"

The values for these settings can be found elsewhere in this User Manual: general launchers settings are on the [Launchers and Workstation](#) page, while each of the three launcher pages list settings unique to particular launcher.

Windows

1. Create a shortcut.
2. Right-click and select **properties**.
3. Add additional arguments to the end of the Target field:

```
...application=test window.mode=fullscreen
```

Linux

1. Create a shortcut.

2. Right-click and select **properties**.
3. Add additional arguments to the end of the command, but before the &":

```
... -Dapplication=test -Dwindow.mode=fullscreen
```

Mac OS

1. Create a shortcut.
2. Right-click or CRTL-click and select **Show package contents** on the shortcut.
3. Inside of Contents/Resources look for the file "launcher.properties".
4. Add additional arguments *on their own lines* as necessary.

Locking the Launcher Configuration

Sometimes you want to restrict user edits to a launcher's configurations and not allow users to do CRUD operations on their own.

To restrict this, set the `lock.configuration` property in the launchers JSON file to `true`. This will limit the ability of the user to modify the settings inside the GUI of the application. (Note that it doesn't eliminate editing the JSON directly.)

Redirect Client Launch Cache

You can also redirect the cache directory that is used by the launcher. Typically the user's cache is in their home directory in the ".ignition" directory. This holds all the resources that are required for launching a client as well as the resources for each launcher. Sometimes this is not desirable, for example in cases where you want a global cache that all users on a machine rely on or when you want the users client cache to reside in a different directory under AppData.

To change this directory, place a file named `redirect.txt` inside the `.ignition` directory in their user home directory. This file should have a single line in it which is an **absolute** path to the appropriate file.

Ignition Modules

With its modular architecture, the Ignition platform enables you to build a customized system using modules as the building blocks. Modules integrate seamlessly into the system and provide things like design workspaces, reporting, alarm configuration, interface to SQL databases, new drivers, and much more. Newly installed modules immediately integrate with existing modules.

Core Modules

Core modules are commonly used modules included in a typical installation. Most systems have at least one of these modules installed. Each core module encapsulates a core SCADA or HMI principle such as data-logging or process visualization. These modules are the foundation upon which projects are built.

| Module | Description |
|---------------------------------|--|
| Perspective | Create beautiful, mobile-responsive industrial applications that run natively on any device type such as smartphones, tablets, touchscreens, laptops, and desktop computers. Automatically adapts to fit screens of any size and orientation. |
| Vision | Create and maintain an interactive, accurate Human-Machine Interface (HMI) for your site. Launch an unlimited number of powerful Ignition Vision clients anywhere on the network. Display charts, graphs, trends, HMI screens, and more. Provides a simple method of visualizing and presenting data to your users. |
| Symbol Factory | Symbol Factory module is included with either Vision or Perspective. Choose from this library of over 4000 vector graphics to enhance your application. This clipart has an emphasis on industrial symbols, and images can be resized with no pixelation or distortion. |
| Reporting | Create dynamic, database-driven reports and deliver them on schedule or display them on your Vision client or Perspective session. Data from any SQL database can be presented in a report. Reports can also be scheduled to run automatically, and be delivered in a number of ways such as emails, or saving to a network drive. |
| SQL Bridge (Transaction Groups) | Bridge the gap between your OPC data and SQL databases. Log data, call stored procedures, and synchronize data bi-directionally. SQL Bridge includes a highly efficient SQL-based Tag historian and transaction manager. |
| Tag Historian | Turn your SQL database into a high-performance time-series Tag historian. |
| Alarm Notification | Enables you to manage alarm notifications and send notifications via email. With Alarm Notification you can configure the logic for how, why and when alarm notifications are delivered. |
| OPC UA | Connects to most major PLCs and serves as a cross-platform OPC-UA server with an open, plugable driver system. |
| Enterprise Administration | Provides a secure and intuitive way to manage many Ignition installations from one location. |
| Sequential Function Charts | Streamlines the creation, organization, and visualization of robust logic systems. |
| SECS /GEM | Enables Ignition projects and third-party applications to communicate with semiconductor fab equipment. |
| SMS Notification | Enables SMS text alerts to be sent for alarms (required Alarm Notification Module). |

Additional Modules

The following modules provide Ignition with additional functionality.

| Module | Description |
|--------|--|
| Voice | Enables both alarm notification and acknowledgment via a phone call. |

On this page ...

- [Core Modules](#)
- [Additional Modules](#)
- [Third-Party Modules](#)
- [User-Created Modules](#)
- [Modular Architecture and Software Stack](#)



INDUCTIVE
UNIVERSITY

About Ignition's Modules

[Watch the Video](#)

| Notification | |
|---------------------------|---|
| Twilio Alarm Notification | Adds integration with Twilio via an SMS Alarm Notification Profile. Send SMS alarm notifications without a physical device. This module unlocks the system.twilio functions. |
| OPC COM | Provides the ability to connect to legacy ("classic") COM-based OPC-DA servers. |
| OPC COM Tunneller | Simplifies connecting to remote OPC-COM servers. |
| Web Browser | Adds a web-browser component to the Vision Module . |
| WebDev | Enables you to directly program against the web server inside the Ignition Gateway. |
| Serial | The Serial Modules add serial comm support and system.serial functions to python scripts. <ul style="list-style-type: none"> The Serial Support Client Module allows the system.serial functions to be accessed from client based scripts, as well as connect to serial devices plugged into the computer running the client. The Serial Support Gateway Module allows the system.serial functions to be accessed from Gateway based scripts, as well as connect to serial devices plugged into the Gateway server. |
| HASP | If you anticipate that the installation might move from server to server frequently, you may want to consider purchasing a USB / (HASP) license key to ease transition to new servers. This also makes things more convenient when the server is being deployed in an area without an active internet connection. |

Third-Party Modules

In addition to the modules provided by Inductive Automation, third-party companies have developed their own modules using our Module Software Development Kit.

Our strategic partners [Sepasoft](#) and [Cirrus Link](#) have developed the following modules:

- MES - [Sepasoft](#) offers a suite of [Manufacturing Execution Systems \(MES\) modules](#) that allow you to track OEE, implement a SPC system, track resources, and much more. The core purpose of these modules is to spot deficiencies in your process, and increase profitability by detecting problem areas, or catching problems before they become an issue.
- MQTT - Modules from [Cirrus Link](#) bring the power of [MQTT](#) into Ignition. These modules expose massive amounts of data-points as tags, and allow Ignition access to data that would otherwise be lost.

Additional third-party modules can be found on our [Third-Party module showcase](#) page.

User-Created Modules

You can create something custom for yourself or help make Ignition better by creating your own modules. Refer to the [SDK documentation](#) for the resources necessary to develop your own module.

Modular Architecture and Software Stack

Ignition platform has a modular architecture. Modules are software applications that are built and integrated into the platform to offer additional functionality. The modules are similar to applications for a smartphone in how they are seamlessly integrated and provide additional capabilities.

Most of the main features of Ignition are provided by different modules such as [Perspective](#), [Vision](#), and [SQL Bridge](#).

The Ignition software stack is shown in the illustration below. You can see that the HMI/SCADA/MES module layers are built on the Ignition platform. Here are the different software layers in Ignition's modular architecture:

- OS Layer – the Operating System Layer**
Provides basic computing resources such as the file system and access to the network.
- Platform Layer**
Provides all the basic functionalities such as connecting to devices and databases, licensing, launching clients, managing all Ignition modules over the web, and more.
- Core Module Layer**
Provides the core modules that enable real-time and historical data access, trends, and control.
- Third-Party Module Layer**
Additional modules provided by Strategic Partners and other developers to further extend Ignition's capabilities.

- **User Created Application Layer**

The resulting project created for your organization. Developed internally, or by a third-party.



In This Section ...

Vision

The Vision module is a tool for creating and maintaining an interactive, accurate Human-Machine Interface (HMI) for your site. Many other modules, as well as platform level features, seamlessly integrate with the Vision module, providing a simple method of visualizing and presenting data to your users.

Windows

[Vision Windows](#) are the basic building blocks for all of your HMI screens. There are three basic window configurations that define how a window behaves:

- **Main Windows** - A [main window](#) is one that is set to start maximized (taking up all available screen space (minus space used by any docked windows).
- **Popup Windows** - A [popup window](#) is one that appears (pops up) when the user performs an action such as clicking the mouse, pressing a function key, or touching a button (if using a touchscreen). Popup windows usually remain on top of the current window until closed, enabling users to quickly choose options or settings before returning to the previous window.
- **Docked Windows** - A docked window is set to a static location on the screen. Docked windows are often used to hold navigation trees or status information that needs to remain on the screen at all times.

By changing a window's properties, you can transform any window into various configurations, with each behaving differently based on those settings. Passing custom parameters into windows allows you to create the window once, and then re-use your screens multiple times within the same project. You get to choose what windows are available on startup and how your navigation is configured. The following is an example of a Vision module screen.

On this page ...

- [Windows](#)
- [Navigation](#)
- [Components](#)
- [Bindings](#)
- [Graphics](#)
 - [Scalable Vector Graphics](#)
 - [Other Images](#)
- [Templates](#)
- [Vision Client Disconnections](#)



INDUCTIVE
UNIVERSITY

Window Types

[Watch the Video](#)



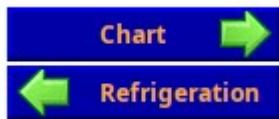
Navigation

A large number of [navigation](#) options exist in the Vision module. For example, docked windows can be set up with navigation trees, tab strips, or menu bars. Components such as buttons can be used to navigate to other windows. A graphic or photograph of a map can be customized with clickable zones. Many of the common options are available as project templates that you can take advantage of when first creating your project. Here are some examples:



INDUCTIVE
UNIVERSITY

- Back/forward buttons



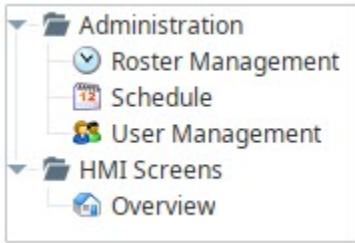
Navigation Strategies

[Watch the Video](#)

- Navigation tab strips

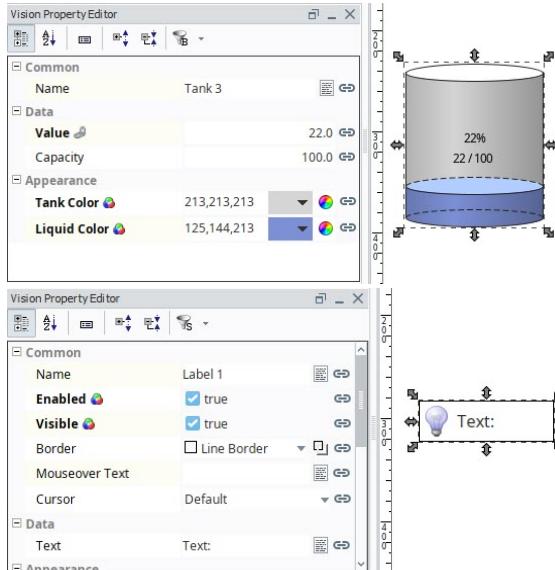


- Multi-tier navigation



Components

Components are building blocks for your project. The Vision module has a variety of built-in components such as displays, buttons, charts, and other elements that display information. Each component has multiple properties that control its appearance, behavior, and data. For example, a [Tank](#) has a level, capacity, and a liquid color, while a [Label](#) has text, font, and an image. You can enhance components with [custom properties](#) to create additional functionality.



Component Overview

[Watch the Video](#)

Bindings

A [binding](#) is a mechanism that allows a property on a component to change based on a change to a value elsewhere in Ignition. For example, with binding, the liquid level displayed in a tank graphic can be bound to the realtime liquid level in a tank. The value of a Tag could be bound to a linear scale, a meter, or a label on your window. The power of bindings comes from the variety of binding types.

Click on the following links for complete information about binding types:

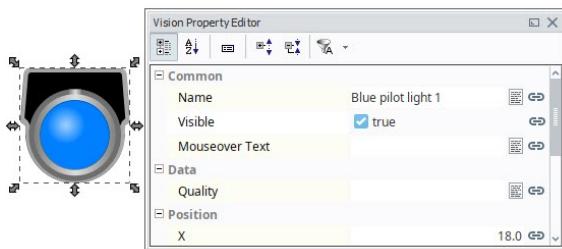
- [Properties - Property Binding, Cell Update Binding, Component Styles](#)
- [Tags - Tag Binding, Indirect Tag Bindings in Vision, Tag History Binding](#)
- [Expressions - Expression Binding](#)
- [Databases - DB Browse Binding, SQL Query Binding, Named Query Bindings](#)
- [Functions - Function Binding](#)

Graphics

In addition to standard components, the Vision module supports the use of [SVG](#), [PNG](#) and [JPEG](#) images on Vision Windows. You can create your own images and import them into your project or use Ignition's [2D Drawing tools](#) to create graphics.

Scalable Vector Graphics

[Scalable Vector Graphics](#) (SVG) have several advantages over other graphic types. Because they are vector graphics, they can be scaled without a loss of clarity or resolution. Additionally, you can drill into an SVG to change individual parts of the image. To use an SVG in Ignition, simply drag the file directly onto the window in which you want it to appear.



[Images \(png, jpg, gif\)](#)

[Watch the Video](#)

Other Images

You can use the built-in SVGs from Symbol Factory which contains hundreds of ready-to-use graphics, or use the raster image library with an Image component to get a jump start on your project.

Templates

Components and images can be combined to create [Vision Templates](#). These are re-usable objects that can be configured once and used throughout your project. Templates work under a principle of inheritance. When a change is made to a template, that change is inherited by each instance of that template. Most templates use one or more custom properties (Template Parameters) to tie data from a window to the internals of the template.

The [Cloud Templates Browser](#) portal in the Designer gives you access to pre-built templates. Ignition community members can also share their own templates in the Cloud Templates.



[About Templates](#)

[Watch the Video](#)

Vision Client Disconnections

If a Vision Client loses its connection to the Ignition Gateway, a message like the one shown below will be displayed immediately on your screen:

Gateway Connection Lost



Searching for active Gateway

Gateway is not available | <http://localhost:8088>

Unable to connect to Gateway http://localhost:8088: Connection refused: connect

[Logout](#)

The Vision Client will continuously attempt to reconnect to the Gateway. When the connection between the Vision Client and the Gateway is re-established, your Vision Client will resume its normal operation. If the project associated with the Vision Client is somehow deleted, the Client will fail to reconnect because its source project is missing. A project can be manually deleted or overwritten by a Gateway restore.

[In This Section ...](#)

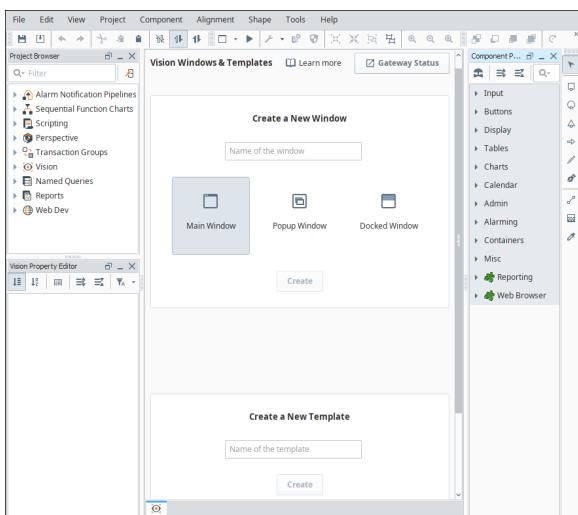
Vision Designer Interface

Vision Designer Workspace

The Vision Designer Interface is where the bulk of the designer's work is done. The Vision Designer Interface provides some built-in functionality to help you get started designing your project whether you are logging into your project for the first time or the 50th time. From the Vision Windows & Templates Welcome tab, you can easily create Main windows, Popup windows, Docked windows and Templates. It even shows you the recently modified windows, so picking up where you left off the last time you logged into your project, is right at your fingertips.

You can also check the Gateway Status from the Designer Interface and see all the Vision Clients that are running along with the client details and stats.

When looking at a Vision specific element in the Designer, such as a window or template, the Designer is organized with some panels that are specific to the Vision Designer Interface, such as the **Property Editor** and **Component Palette**. Other elements of the workspace that are shared between spaces are discussed in the [General Designer Interface](#).



On this page ...

- [Vision Designer Workspace](#)
- [Component Palette](#)
- [Vision Property Editor](#)
 - [Filters](#)
 - [Binding Icon](#)
 - [Status Indication](#)
 - [Dropdown Lists in Properties](#)
 - [Common Properties](#)
- [Vision Menubar](#)
 - [File Menu](#)
 - [Edit Menu](#)
 - [View Menu](#)
 - [Project Menu](#)
 - [Component Menu](#)
 - [Alignment Menu](#)
 - [Shape Menu](#)
- [Tools Menu](#)



The Designer User Interface

[Watch the Video](#)

Component Palette

The Vision module comes with a host of useful components out of the box, such as buttons, text areas, dropdowns, charts, and many more, many of which are specialized for industrial controls use. The Component Palette is located on the right side of the Designer workspace. The basic workflow is to drag a component from the component palette and drop it into a container on a [window](#). From there, you can use the mouse to drag and resize the component into the correct position. While the component is selected, you can use the Property Editor panel to alter the component's properties, which changes the component's appearance and behavior.

Vision Property Editor

The Vision Property Editor is a dockable panel that appears in the Designer's central workspace, usually in the lower left corner. It displays the properties of the selected component. If more than one component is selected, it will show all properties that the current selection set have in common. Hovering your mouse over a property will display a tooltip that gives a description of the property, as well as its data type and scripting name. Alternately, you can click on the Show/Hide Description Area icon to bring up the description area which displays the same information for the currently selected property.

You can also change how the properties are sorted in the property editor. By default, they are sorted with the Categorized icon, with similar components grouped under different categorical headers. However, they can also be sorted in alphabetical order by clicking on the Alphabetical icon.

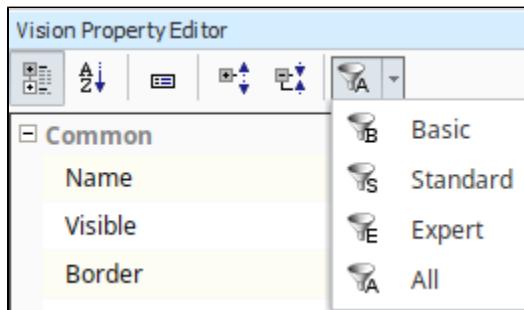
Filters

It is common for components to have many properties, so by default the Property Editor only shows the **Basic** properties. These are the properties that you'll most commonly want to set or bind for a given component. However, the property filter can be changed to show different sets of properties. The designer will remember your selection for future sessions.

- **Basic:** The Name property and any very commonly used properties. Most components only show two to four properties in Basic.
- **Standard:** Most of the common properties that a designer would want to use. Few or none of the Expert properties are in the Standard list.
- **Expert:** The properties that are most commonly used with more advanced features of the component. Few or none of the Standard properties are in the Expert list.
- **All:** All properties



Most users find it best to set the property filter to All, so they can see all of the properties available to them at all times.



Binding Icon

To the right of most properties is the Binding  icon. Click this icon to modify the property binding that is driving that property. You can only use this button when the window workspace is not in Preview mode. Some properties cannot be bound because their datatype is not supported by the binding system. You can still use scripting to affect these properties.

Status Indication

The name of a property in the **Property Editor** conveys important information about that property:

- A **blue name** indicates that the property is a [custom property](#).
- A **bold name** with a Link  icon next to the property indicates that the property is bound using a [property binding](#).
- A **bold name** with a Color Palette  icon indicates that the property is being affected by the [component styles](#) settings.
- A **red bold name** with a Warning  icon indicates that the property is double-bound. This means that two things, a property binding and the styles settings are both trying to drive the property value. This will result in errors as the two systems fight each other to write to the property.

Dropdown Lists in Properties

Some of the properties you will encounter on components will have a dropdown list instead of a field to type into. The property description will say it is an integer value, and in most of these cases you can still create a binding on that property. These dropdown lists are an enumeration, meaning each element in the dropdown has an integer value. In all cases, the first value in the list is 0, the second is 1, the third is 2, and so on. You can use this knowledge to create a dropdown list on-screen for your operators that matches the list. In this case, you would just bind this property to the Selected Value of the dropdown.

Common Properties

Every component has properties arranged into categories based on what it has available (i.e., Common, Behavior, Data, Appearance, Layout, etc.). Each component has a different list of properties to effect how it behaves, but every component has the **Common** group of properties located at the top of the list. These **Common** properties will behave the same for all components. Here's a list for each Common property and when it might be used.

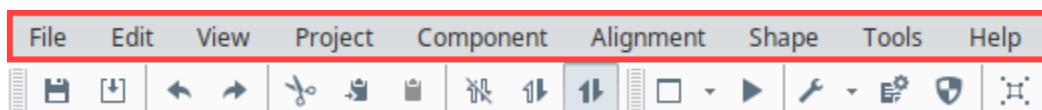
| Function | Description |
|----------|--|
| Name | The name of the component. This string is used to identify your components in the Project Browser. This is especially important for Bindings and Scripting. Binding is allowed on this property, but it is recommended to never bind this property . Binding it can break your scripts, bindings, and cause errors. |
| Enabled | This Boolean controls whether a component can be interacted with. Most commonly used with data entry components to allow the user to see the value, but not change it. |
| Visible | This Boolean controls whether the component is shown on the window. You can bind this property to show/hide the component based on any logic you want (i.e., security, process step, etc.). |

| | |
|----------------|--|
| Border | The border that surrounds the component. There is a dropdown to select from a list of common borders, and a button to the right to manually edit a border from several different options with a second tab that shows Titled Borders. When binding this property, note that this is a complex data type. It is a Java Border data type, not a string or an enumeration. The common ways to make this property dynamic are to bind it with an Expression binding type or to set it through a script, but using the Expression binding is preferred. If you are using an Expression binding, you must use the toBorder() expression function to return the correct data type. If you are using a script, you need to make sure you use the Java Border data type. See the Java documentation for more information on setting a border through scripting. |
| Mouseover Text | The text that is displayed when a user moves the mouse over the component. This string is commonly used to provide your operators more information about an object (i.e., showing the PLC address of an on-screen value, or telling the operator exactly what will happen when a button is pressed). HTML is allowed in this property. |
| Cursor | The mouse pointer image to use when the operator moves the mouse over the component. This int property corresponds to one of the options in the list. Selecting 'default' means the operating system decides what pointer to use. |

| Value | Cursor |
|-------|-----------|
| 0 | Default |
| 1 | Crosshair |
| 2 | Text |
| 3 | Wait |
| 4 | SW Resize |
| 5 | SE Resize |
| 6 | NW Resize |
| 7 | NE Resize |
| 8 | N Resize |
| 9 | S Resize |
| 10 | W Resize |
| 11 | E Resize |
| 12 | Hand |
| 13 | Move |

Vision Menubar

There is a menubar at the top of the Designer Workspace that provides functionality that you can interact with when working in the Vision workspace. Each menu has a host of functions as it relates to that menu. The other menus that are shared between Vision and Perspective are discussed in the [General Designer Interface](#).



File Menu

See [General Designer Interface](#).

Edit Menu

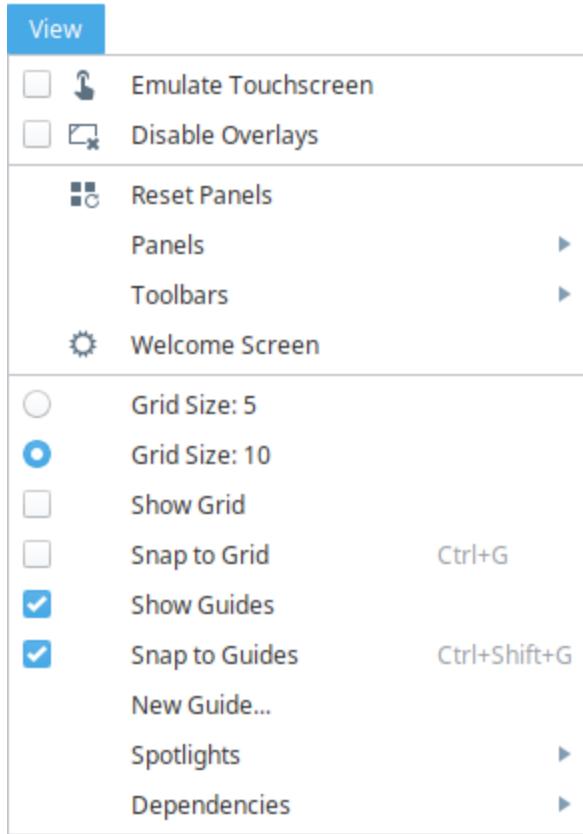
The **Edit Menu** is also similar to other applications edit menus in that it provides much of the basic copy/paste functionality. You can also right click on an item to access this menu.

| Edit | | |
|---|----------------------------|------------------|
|  | Undo | Ctrl+Z |
|  | Redo | Ctrl+Y |
|  | Cut | Ctrl+X |
|  | Copy | Ctrl+C |
|  | Paste | Ctrl+V |
|  | Duplicate | Ctrl+D |
|  | Paste Immediate | Ctrl+I |
|  | Cancel Paste | Escape |
|  | Find/Replace | Ctrl+F |
|  | Select All | Ctrl+A |
|  | Select Same Type | Ctrl+Shift+A |
|  | Select Same Type In Window | Ctrl+Alt+Shift+A |
| Group Rename | | |
|  | Delete | Delete |

| Function | Description |
|----------------------------|---|
| Undo and Redo | Can be used to revert to the previous state, essentially removing the last change, or redoing it again after having been removed. This has a large queue that can be traversed, but does not include every change (i.e., Tag edits cannot be undone). |
| Cut/Copy /Paste /Duplicate | These functions much the same as they do everywhere else. Most things in the Designer can be copied and pasted elsewhere, from individual components on the window to entire folders of windows. The difference is that when using Paste with an object on a window, it will instead create a paste action, and allow you to move the mouse and select where you want to paste it, clicking the mouse to confirm. Cancel Paste will cancel the paste action, while paste immediate will bypass the paste action, and instead immediately paste the object from where it was cut or copied from. |
| Find /Replace | Brings up the Find and Replace interface to allow you to find specific objects within the project. See also: Find and Replace |
| Select All | Selects All siblings in the same container as the currently selected component. |
| Select Same Type | Selects all components of the same type in the same container as the currently selected component. |
| Select Same Type in Window | Selects all components of the same type as the currently selected component, regardless of what container they are in. |
| Group Rename | Renames a group of components to a prefix with a number afterwards. For example, if your prefix is Button, it will rename all selected components Button (1), incrementing the number each component. |
| Delete | Deletes the currently selected component. This can also be done using the delete key. |

View Menu

The **View Menu** allows you to manipulate how various objects look or act in the Designer.

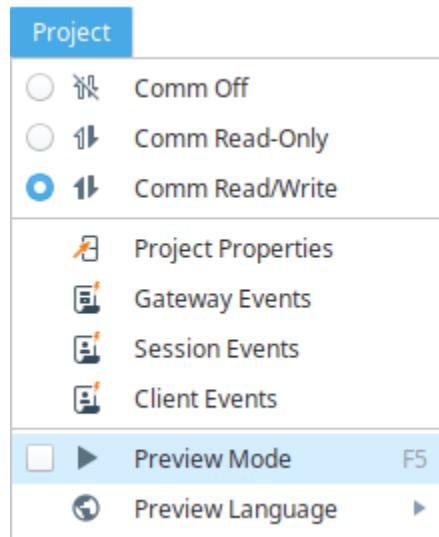


| Function | Description |
|---------------------|---|
| Emulate Touchscreen | Simulates Touchscreen mode in the Designer to be able to test it without having to open a client. |
| Disable Overlays | Disables the red or gray overlays on components because of a bad Tag or binding, but only in this Designer session. |
| Reset Panels | Resets panels (Project Browser, Tag Browser, etc.) to the default panel configuration. |
| Panels | Allows you to enable or disable certain panels within the Designer. |
| Toolbars | Allows you to enable or disable certain toolbars within the Designer. |
| Welcome Screen | Takes you to the welcome screen in the Designer, or reopen it if it had been closed. |
| Grid Size | Allows you to select a grid size of 5 or 10. |
| Show Grid | Toggles the grid on and off. |
| Snap to Grid | Changes click-and-drag behavior to snap components to grid lines. This works even when Show Grid is off. |
| Show Guides | Shows any guide lines. |
| Snap to Guides | Changes click-and-drag behavior to snap components to any created guidelines. |
| New Guide | Adds a guide line to the current window. |
| Spotlights | Puts a highlighted border around components that have the selected spotlight. Bound objects will get a green highlight, objects with scripting will get a blue highlight, and invisible objects will get a pink highlight. If a component has multiple highlights, and both are enabled, it will alternate the colors throughout the highlight. |
| Dependencies | Shows the binding dependencies (as arrows) based on the selected component or components. Show Supporters will show all components that the currently selected component is bound to, Show Dependents will show all components that are bound to the |

currently selected component, and Show All will show all of the bindings, regardless of the selected components.

Project Menu

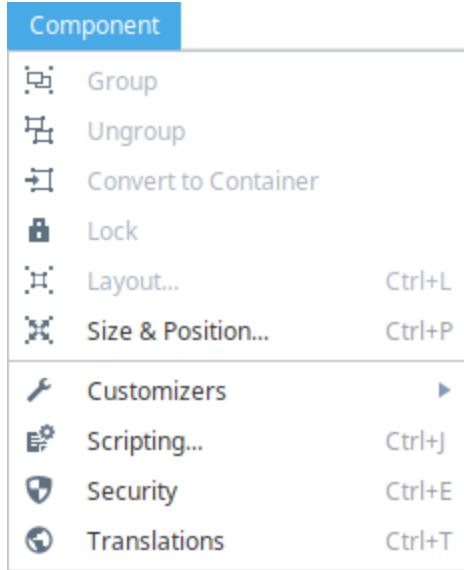
The **Project Menu** is where many project specific settings can be changed.



| Function | Description |
|------------------|--|
| Designer Comms | The Comm settings allow you to select the level of communication the Designer can have with the Gateway. By default, this is set to Comm Read-Only, which will make any information coming from the Gateway read only, but this can be changed to Comm Off which will prevent Gateway communication, or Comm Read/Write, which will allow both read and write communications between the Gateway. The default that the Designer opens at can be changed in the Project Properties. See also: Communication Modes |
| Properties | Opens up the Project Properties window, allowing project settings to be changed. See also: Project Properties |
| Event Scripts | Opens up the appropriate event script window, either client or Gateway. These can also be accessed from the Project Browser. See also: Client Event Scripts and Gateway Event Scripts . |
| Preview Mode | Puts the Designer into Preview Mode, allowing you to interact with it like a client. See also: Previewing the Project |
| Preview Language | Determines the language that the Designer will revert to when in Preview Mode. See also Localization in Vision |

Component Menu

The **Component Menu** offers many of the same selections for the selected component that right clicking on that component would contain.



| Function | Description |
|----------------------|---|
| Group | Only available when multiple components are selected. Grouping will place the currently selected components into a group. Ungroup will remove the grouping. See also: Working with Components |
| Ungroup | Only available when a group is selected. This option removes the group (and any custom properties that are on the group) and places all items from that group into the object the group was in. |
| Convert to Container | Only available when a group is selected. Converts the selected group to a container. See also: Container |
| Lock | Locks or unlocks the selected component's size and position. |
| Layout | Set layout constraints for the selected component. |
| Size and Position | Change the size and position of the currently selected component. |
| Customizers | Allows you to select any of the available customizers for the currently selected component. |
| Scripting | Brings up the scripting window for the currently selected component. |
| Security | Opens up the Security Settings Panel, allowing security to be placed on the selected components. |
| Translations | Brings up the Translatable Terms Panel, showing any translations for the selected component. |

Alignment Menu

The **Alignment Menu** options allow you to adjust the alignment of components relative to other components.

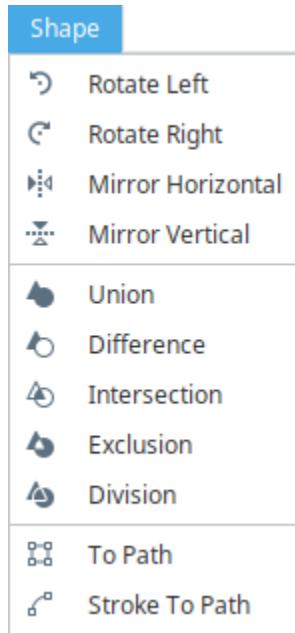
Alignment

| | |
|---|-----------|
|  Move to Front | Home |
|  Move to Back | End |
|  Move Forward | Page Up |
|  Move Backward | Page Down |
|  Align Left | |
|  Align Right | |
|  Align Top | |
|  Align Bottom | |
|  Align Centers Horizontal | |
|  Align Centers Vertical | |
|  Align as Row | |
|  Align Row and Normalize | |
|  Align as Stack | |
|  Align Stack and Normalize | |
|  Center Horizontally | |
|  Center Vertically | |

| Function | Description |
|--------------------------|--|
| Move to Front | Move the selected components to the front of the z-order. |
| Move to Back | Move the selected components to the back of the z-order. |
| Move Forward | Move the selected components forward in the z-order relative to any overlapping components. |
| Move Backward | Move the selected components backward in the z-order relative to any overlapping components. |
| Align Left | Align the left edges of a group of components. |
| Align Right | Align the right edges of a group of components. |
| Align Top | Align the top edges of a group of components. |
| Align Bottom | Align the bottom edges of a group of components. |
| Align Centers Horizontal | Aligns all of the selected components horizontally on their centers. |
| Align Centers Vertical | Aligns all of the selected components vertically on their centers. |
| Align Centers | Aligns all of the selected components either vertically or horizontally on their centers. |
| Align as Row | Aligns all of the components on their centers as a row, and will add padding between them that you can select. Normalizing them will change the size of all of the components to the first selected component. |
| Align as Stack | Aligns all of the components on their centers as a stack, and will add padding between them that you can select. Normalizing them will change the size of all of the components to the first selected component. |
| Center Horizontally | Centers the currently selected components horizontally. |
| Center Vertically | Centers the currently selected components vertically. |

Shape Menu

The **Shape Menu** allows for manipulation of shape or path objects.



| Function | Description |
|----------------|--|
| Rotate | Rotates the currently selected shape 90 degrees either right or left. |
| Mirror | Flips the component either vertically or horizontally. |
| Union | Alters the first shape to be the combination of all selected shapes. |
| Difference | Alters the first selected shapes by removing the last selected shape from them. |
| Intersection | Alters the first shape to become a new shape consisting of the area they share. |
| Exclusion | Alters the first shape to become a new shape consisting of the area they do not share. |
| Division | Cuts the first shape into multiple shapes along the borders of other shapes. |
| To Path | Converts a shape to a simple path object. |
| Stroke To Path | Converts the selected shape into a new shape defined by its stroke. |

Tools Menu

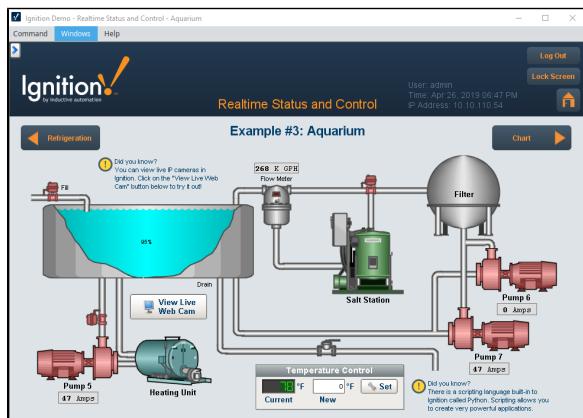
The Designer comes with many tools that allow you to manage and test various resources within a project. Each of the tools have their own interface and are accessed within the Tools menu on the menu bar of the Designer. For more information, see [Designer Tools](#).

Vision Windows

Windows are the key to your HMI/SCADA application. A window is the basic building block of any Vision project, where each window can contain any number of components that can display Tag values, run scripts, write values to the database, accept user input and more. When you publish your project, these windows are loaded into the Vision Client where any number of windows can be opened at one time.

Your windows are brought to life through the property bindings and event handlers on your components. They can be designed to fit any need, from simple screens showing basic information, to complex diagrams outlining an entire plant floor with various controls. Despite their abilities, using windows is relatively simple so that even new users can get started creating windows right away. The possibilities are endless when designing windows for your project.

In this Realtime Display example, you can see how windows can be used to display and interact with data and Tags.



Window Anatomy

While there is only one type of [window object](#), windows have various properties that determine how they behave within the client. When these settings are configured in specific ways, they create certain categories or [types of windows](#): **Main Windows** act like a typical HMI screen and take up all available space, **Popup Windows** are often opened by a component in a Main Window and appear to float on top of the Main Window, and **Docked Windows** stick to one side of the screen and are typically always open. These types of windows all provide different functionality to a project which, when combined create the basis for a Vision project that displays relevant information while remaining intuitive and user friendly.

Root Container

Inside a window there is always a **Root Container**. The Root Container is where you place all components in the window. It's a normal [Container component](#) except that it cannot be deleted or resized, and is always set to fill the entire window. The root container will be the root of all components that go onto the window.

Window Name and Title

Windows have both a **Name** and a **Title**. The name is used within the Project Browser to differentiate the windows from each other and to form part of the path to the window. Windows can be renamed by right clicking on the window object and selecting rename or by pressing F2. Each window must have a unique path, so windows can have the same name as long as they are not in the same folder.

The Title property is a property within the property editor and works a little differently than the name. By default, Ignition assigns the Title property the same name as the window type that is created (i.e., Main Window, Docked Window, or Popup Window). These window titles are used for the titlebar of a window, but are also used when viewing currently opened windows. In the Client, the Windows menubar command will display a list of all currently opened windows, as well as allow you to switch between which one is in focus. The list of opened windows displays the title of the window, and not the window name or path, so it is also important to have good titles for your window.

On this page ...

- [Window Anatomy](#)
 - [Root Container](#)
 - [Window Name and Title](#)
 - [Titlebar and Border](#)
- [Creating a Window](#)
 - [Right Click in the Project Browser](#)
 - [Using the Welcome Window](#)
 - [From the Menubar](#)
- [Organizing Windows](#)
- [Window Right-Click Menu](#)
 - [Exporting Window Example](#)
 - [Importing Window Example](#)
- [Navigation Strategy](#)



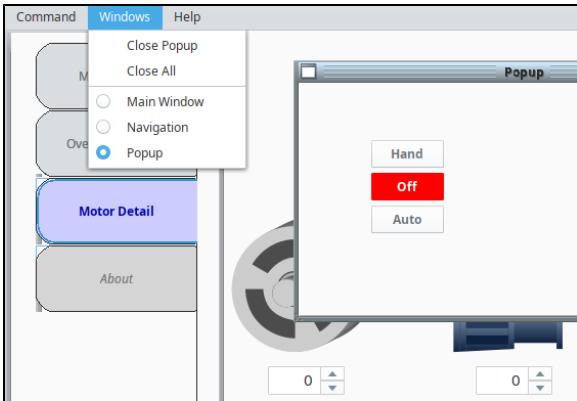
Anatomy of a Window

[Watch the Video](#)



Locate All Opened Windows In Client

[Watch the Video](#)



Titlebar and Border

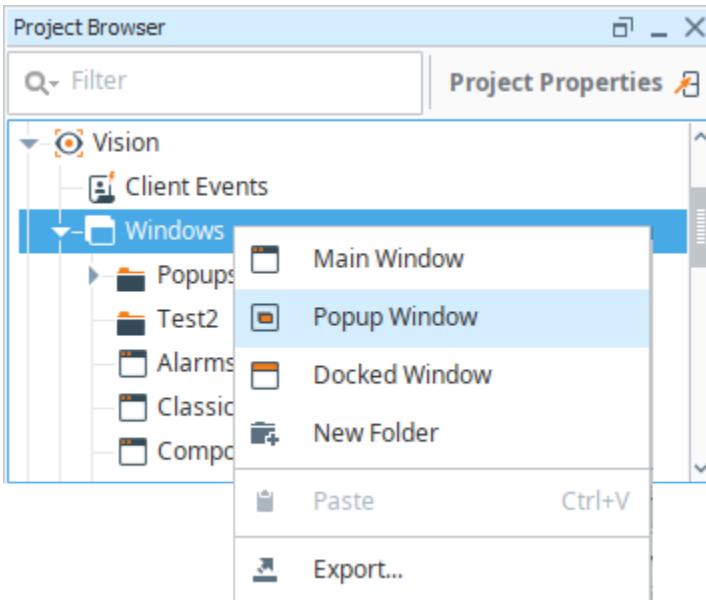
A window can display a **Titlebar** and/or a **Border**. A titlebar allows you to drag a window around the workspace, and contains the window's close, maximize and restore buttons. The border of a window also lets you resize the window when it is floating or docked. Whether or not the titlebar and border are displayed depends on the property values set for your Titlebar and Border properties. A window typically displays both a titlebar and border when it is floating, but only a titlebar when maximized. It is often desirable to remove titlebars and borders on maximized windows.

Creating a Window

Creating windows is easy. There are three ways to create a new window:

Right Click in the Project Browser

Probably the most common method is to right click within the Windows section of the Project Browser and select one of the window types to create a window. While you can create each type of window, it is important to remember that the only difference is the configuration of the window properties.



Using the Welcome Window

The Welcome Window that is available when the project is first opened has a few quick start options available on it. One of these options is the ability to create a new window.

Create a New Window

Name of the window

Main Window



Popup Window



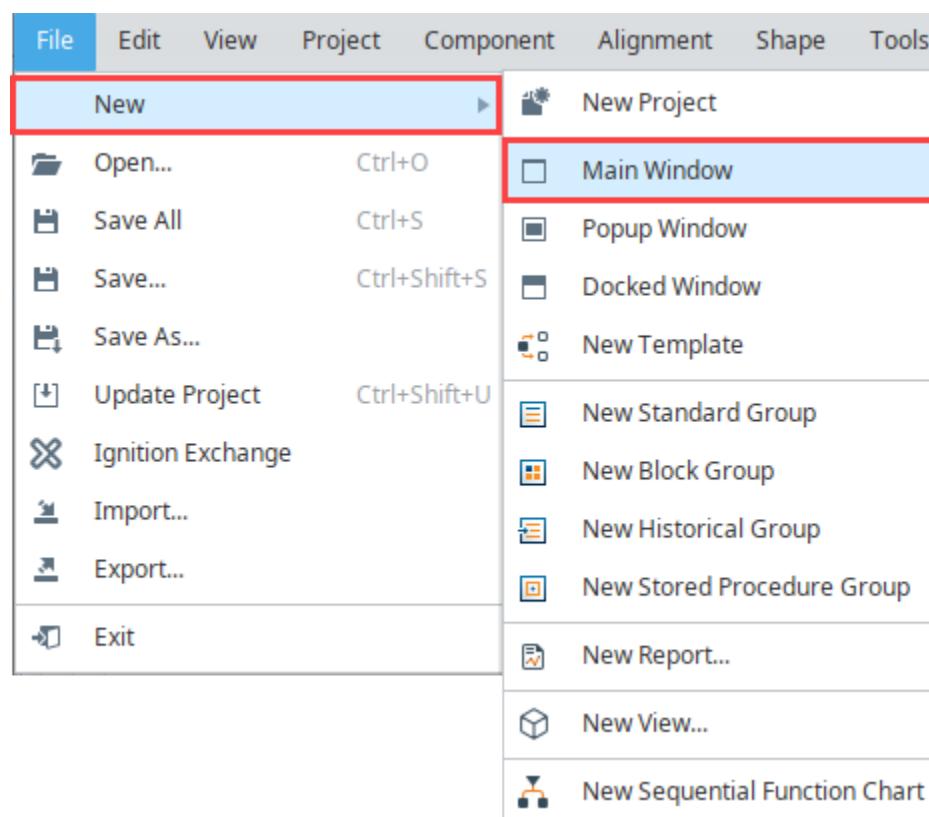
Docked Window



Create

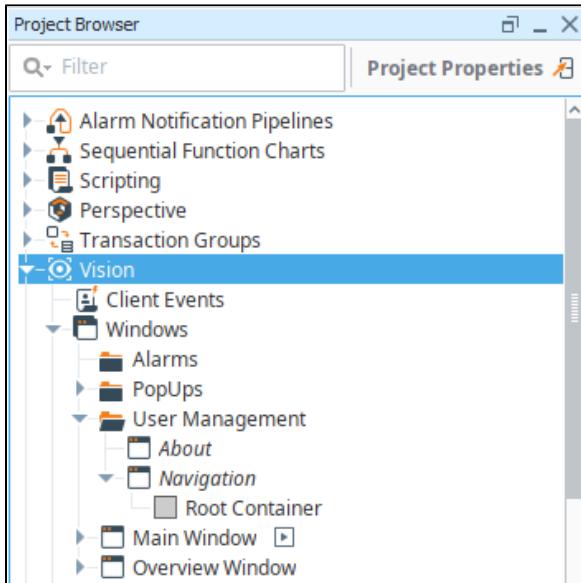
From the Menubar

In the menubar of the Designer, the **File** menu has a **New** option that allows you to create a new window regardless of where you are in the project.



Organizing Windows

You can create folders to organize your windows. A window's name must be unique among the windows in its folder, but you can have the same window name in multiple folders. The window name and folder path are very important, they are used as references by other windows. You can create as many folders as you want and nest them as deep as you need for your project. To rearrange a window, just click and drag the window where you want to place it.



INDUCTIVE
UNIVERSITY

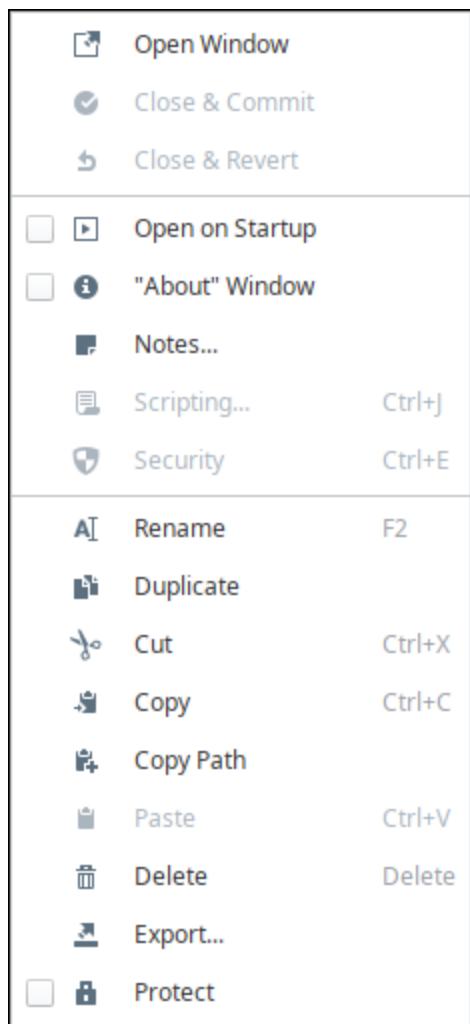
Open Static Window(s) on Startup

[Watch the Video](#)

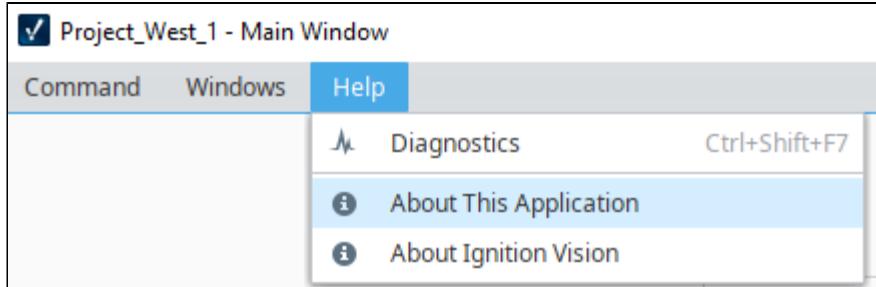
Note: If you have a security requirement to open a different startup window depending who is logged in, you can create a client startup script to open a dynamic set of windows. To learn more, refer to [Open Dynamic Windows on Startup](#).

Window Right-Click Menu

For a full list of properties that can be set on windows, refer to [Vision - The Window Object](#). Windows also have right click menu of options for additional functionality.



| Function | Description |
|-----------------|---|
| Open Window | Opens the selected window. |
| Close & Commit | Commits any changes to your workspace and closes the window. |
| Close & Revert | Reverts any changes that were made since the window was last opened or saved. |
| Open on Startup | <p>One of the most useful properties is the Open on Startup property, which when enabled will automatically open the window when the client first starts up. This makes it easy to open a static set of default windows that everyone can see after logging in to the project. Multiple windows can be set to open on startup, though it is recommended that only a single main window is set to open on start, as opening multiple at once will cause them to be hidden behind one main window.</p> <p>All windows that Open on Startup have a little box with a Right Arrow icon next to the window name.</p> |
| About Window | <p>An "About" Window relays information to the user that may be important, such as instructions on how to use the project, or information about the projects creator. To specify a window as the About window, right click on the window in the Project Browser. Then click the About Window checkbox. The window will have a Information Bubble icon displayed next to its name.</p> <p>In the client, the window will be displayed when a user selected Help > About This Application.</p> |



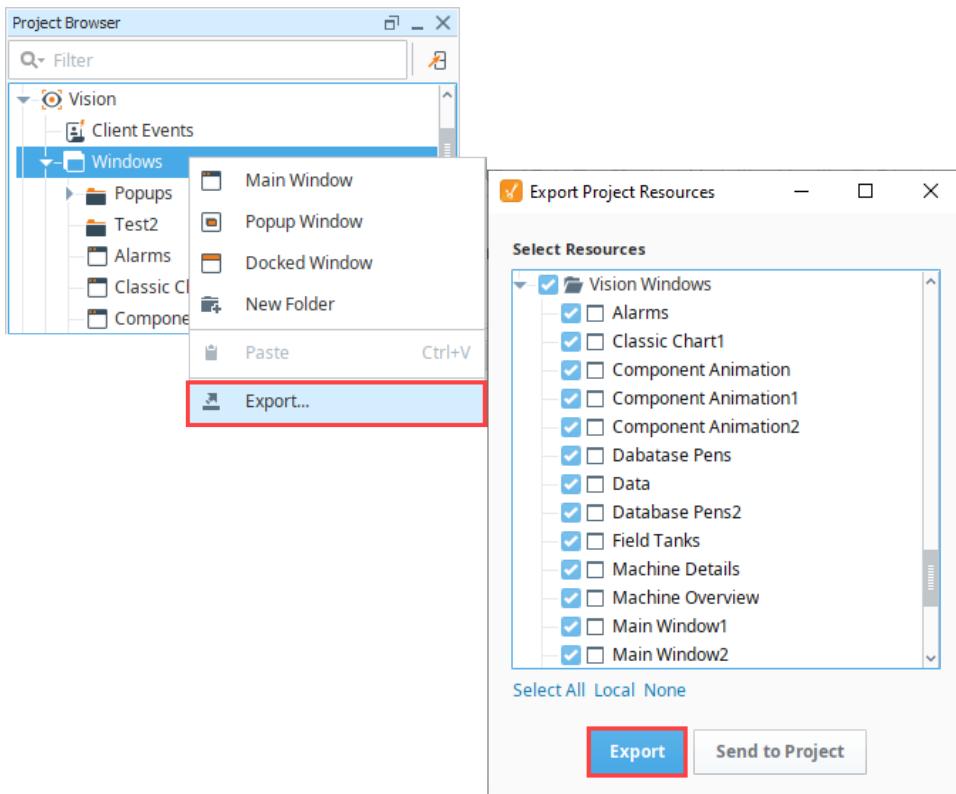
| | |
|-----------|---|
| Notes | Windows can also have notes attached to them. The notes provide a way for a windows designer to provide some documentation on what the window is doing and how the various components interact with one another. Any windows that have notes will have a small Document icon next to the window name. |
| Scripting | The Scripting option takes you to the Component Scripting for that window. For more information, refer to Component Events and Script Builders in Vision . |
| Security | The Security options displays Security Settings for role-base security. For more information, refer to Security in Vision . |
| Rename | To rename a window, select this option then enter a new name. |
| Duplicate | Duplicates the selected window. |
| Cut | Cuts the selected window onto the clipboard. |
| Copy | Copies the selected window onto the clipboard. |
| Copy Path | Copies the path of the selected window into the clipboard. |
| Paste | Pastes the content in the clipboard into the selected context. |
| Delete | Deletes the current selection. |
| Protect | Locks the individual project resource from inside the Designer. |
| Export | Exports the window as a project resource file which can then be imported into other projects. See the following sections for examples of Export and Import. |

Exporting Window Example

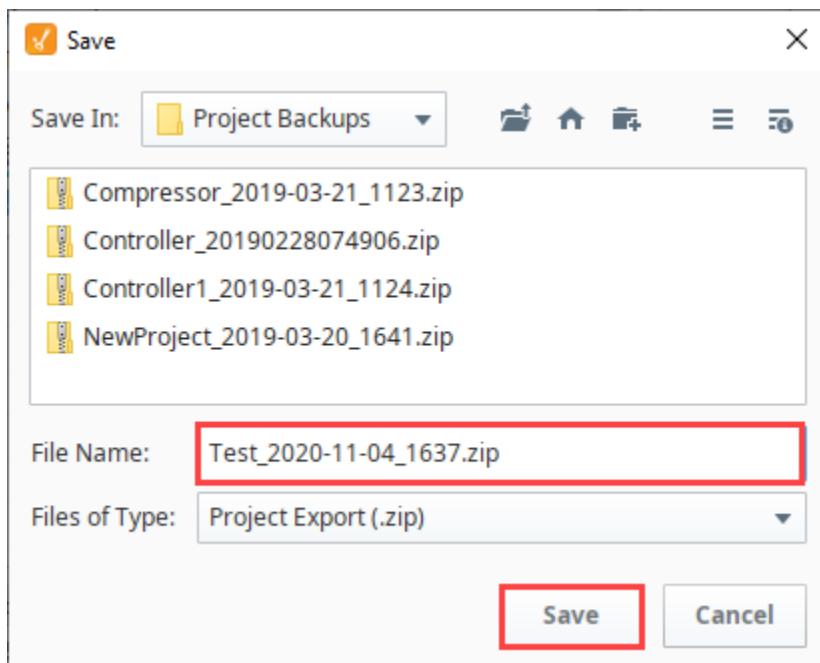
In the Designer, you can export and import windows from one project to another project using external files or sending it directly to a project on the same gateway.

1. You can export windows in two ways.

- a. To export multiple windows, right click on the folder of windows. and select either the **Export** option or **Send to Project** option. The window export works similarly to the [project export](#), the difference being that it automatically highlights only that window to export from the list of project resources.



- b. To export one window, right click on an individual window and click **Export**, then choose the **Export** or **Send to Project** option.
2. If you choose the **Export** option, the **Save** window is displayed. You can save the windows with the existing project name (not recommended if you are only exporting part of a project), or type a new name in the File Name field.

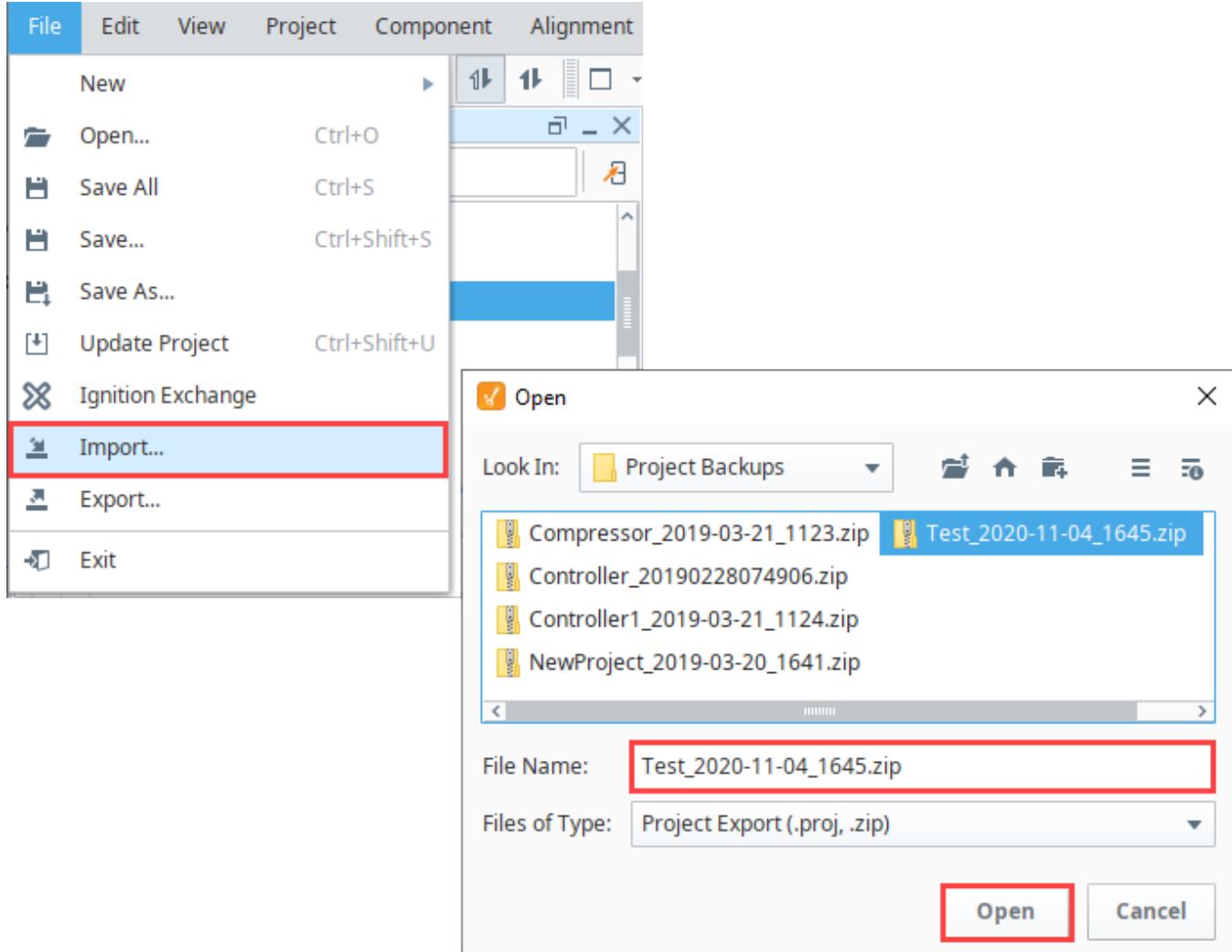


3. Click **Save** to save the windows as a project export file.

Importing Window Example

Importing the individual windows can be done by right clicking **File** from the top menubar and selecting **Import**.

Browse to the folder that contains the .zip file you want to import, and click **Open**.



Navigation Strategy

Setting up a [navigation strategy](#) allows you to navigate between different windows in the runtime Client. While we have a few examples of the most common navigation strategies, it is certainly not an exhaustive list as most users tend to combine multiple strategies to create a project that fits their needs.

A typical navigation strategy for a Vision project is as follows:

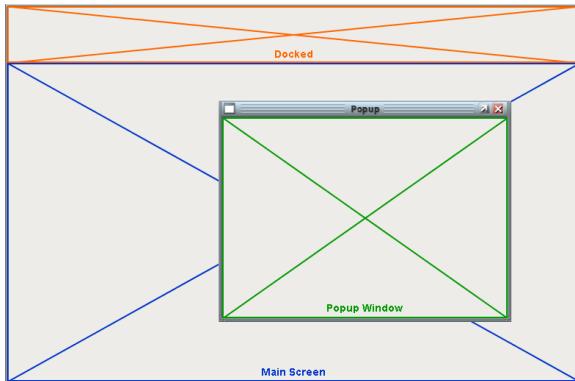
- Have a **Docked** window or two, usually docked North and/or West.
- Have a single **Main** window visible at a time.
- Use swap navigation to swap between the **Main** windows. This ensures that only one main window is open at a time.
- Use standard open navigation to open various **Popup** windows as necessary.

This style of project is so common, that the default operation of the **Tab Strip** component expects it. When it is in its default automatic operation, it expects that each tab represents a main window, and will automatically swap from the current screen to the desired screen. Additionally, the `[System]/Client/User/CurrentWindow` Tag is calculated based on this strategy: its value is the name of the current maximized window. This navigation strategy is used in the [Ignition Online Demonstration](#) that you can download from our website.

[In This Section ...](#)

Window Types

There are three Vision [window](#) types: **Main** windows, **Popup** windows, and **Docked** windows. You can create windows from the **File > New** menu or by right clicking on the Windows object in the Project Browser. By changing a window's properties, you can transform any window into various configurations, with each behaving differently based on those settings.



On this page ...

- Main Windows
- Popup Windows
- Docked Windows
 - Docking Settings



Window Types

[Watch the Video](#)



It is important to understand that just because a certain type of window was created does not mean that it must always be that type of window. A windows type is determined by its settings, so changing its settings to match a different window type will change that window to a new type.

Main Windows

A **Main** window is one that is set to start maximized, and has its Border and Titlebar display policies set to 'When Not Maximized' or 'Never.' This will make the window take up all available space (minus space used by any "docked" windows). This makes the window act much like a typical "HMI screen." There can be many main windows in a project, but only one should be open at any time since they would all overlap.

Popup Windows

A [popup window](#) is a window whose Dock Position is set to Floating and is not maximized. Its Border and Titlebar display policies are typically set to 'When Not Maximized' or 'Always,' so that they can be manipulated by the end-user. These windows are often opened by components in a main window, and are meant to be on top of the screen. To this end, they should have their [Layer](#) property set to a number higher than zero so they don't get lost behind the main window. Popups can be set to open at a specific position on the screen using window's [Location](#) property. Popup windows can also be [parameterized](#) so they can be made once and used for multiple similar applications, dynamically changing the content on the screen based on a parameter that gets passed in.

Docked Windows

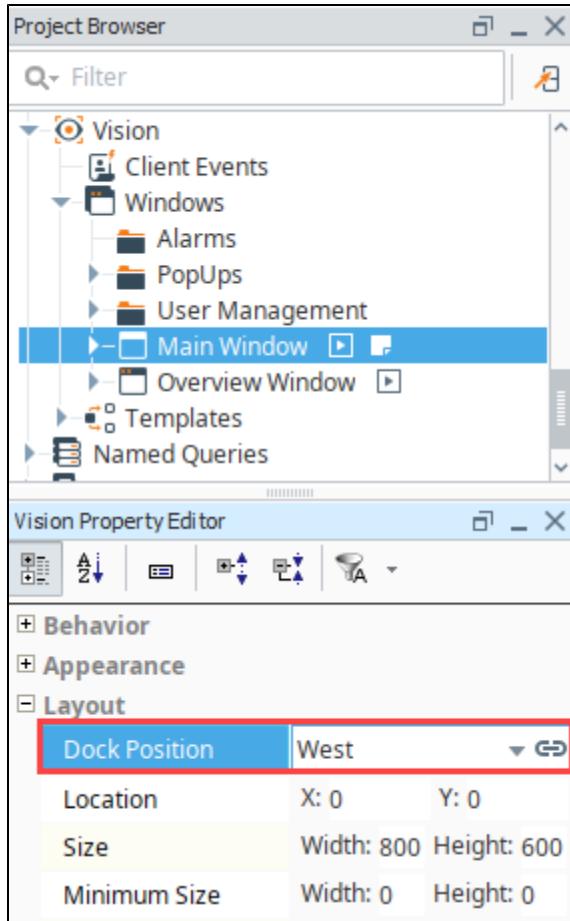
A **Docked** window is one whose Dock Position is set to anything but Floating. Docked windows are locked to the edges of the Client and fill all the space on that edge (i.e., West Docked fills the left side of the Client). It will also typically have its Border and Titlebar display policies set to Never. This makes the "docked" window appear to be joined seamlessly with the current main window. These screens are usually tall and skinny or short and wide, depending on the side 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.

Setting which side the window is docked on is done through the window's **Dock Position** property.



Docked Windows - Order Precedence

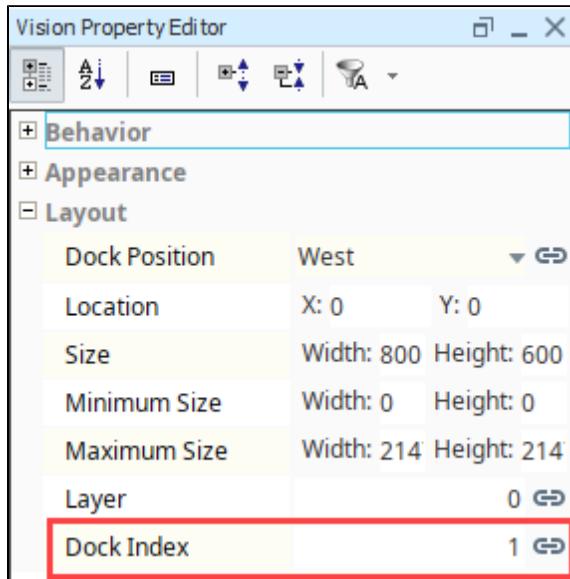
[Watch the Video](#)



Docking Settings

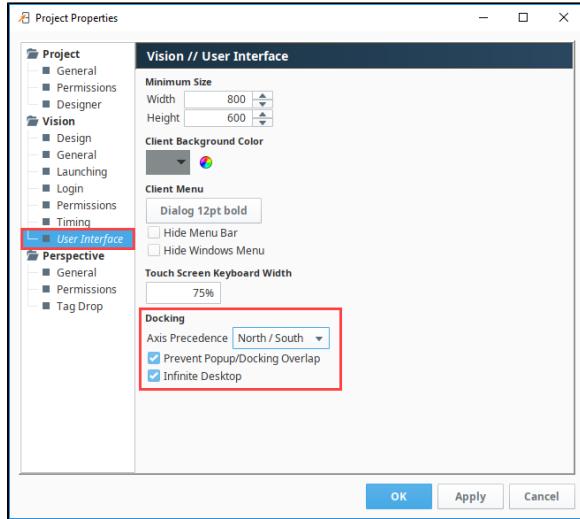
Having multiple docked windows means you need to decide how you want them to appear in relation to each other. For that, Ignition has several settings.

The Dock Index is a property on the window that determines the order of docked windows if multiple windows are docked to the same side. The window with the lowest Dock Index will appear closest to the edge on that side, whereas the highest Dock Index will appear closest to the middle of the client.



The remaining three settings are located in the [Project Properties > Vision > User Interface](#) section:

- **Axis Precedence** - Project wide property that determines which sides get to extend to the edge of the window, North and South or East and West.
- **Prevent Popup/Docking Overlap** - When set to true, then floating (popup) windows will not overlap with docked windows.
- **Infinite Desktop** - When set to true, then the desktop area will be expanded if windows are dragged out of frame.



Related Topics ...

- [Vision Project Properties](#)

Popup Windows

A popup window is typically a window that "floats" on top of the main window. It can be resized and moved around by the user, all while the main window is still open in the background. Popup windows are great for displaying additional information about a selected item on the screen, for example a details screen about one particular component. Popup windows are often opened by components in a main window, and are meant to be on top of the screen. They are used to view setpoints, zoom into a specific area, and more.

The great thing about Popup windows is they can also be parameterized so they can be reused. One popup window design can be reused for many components as long as the proper information is passed to the popup window.

On this page ...

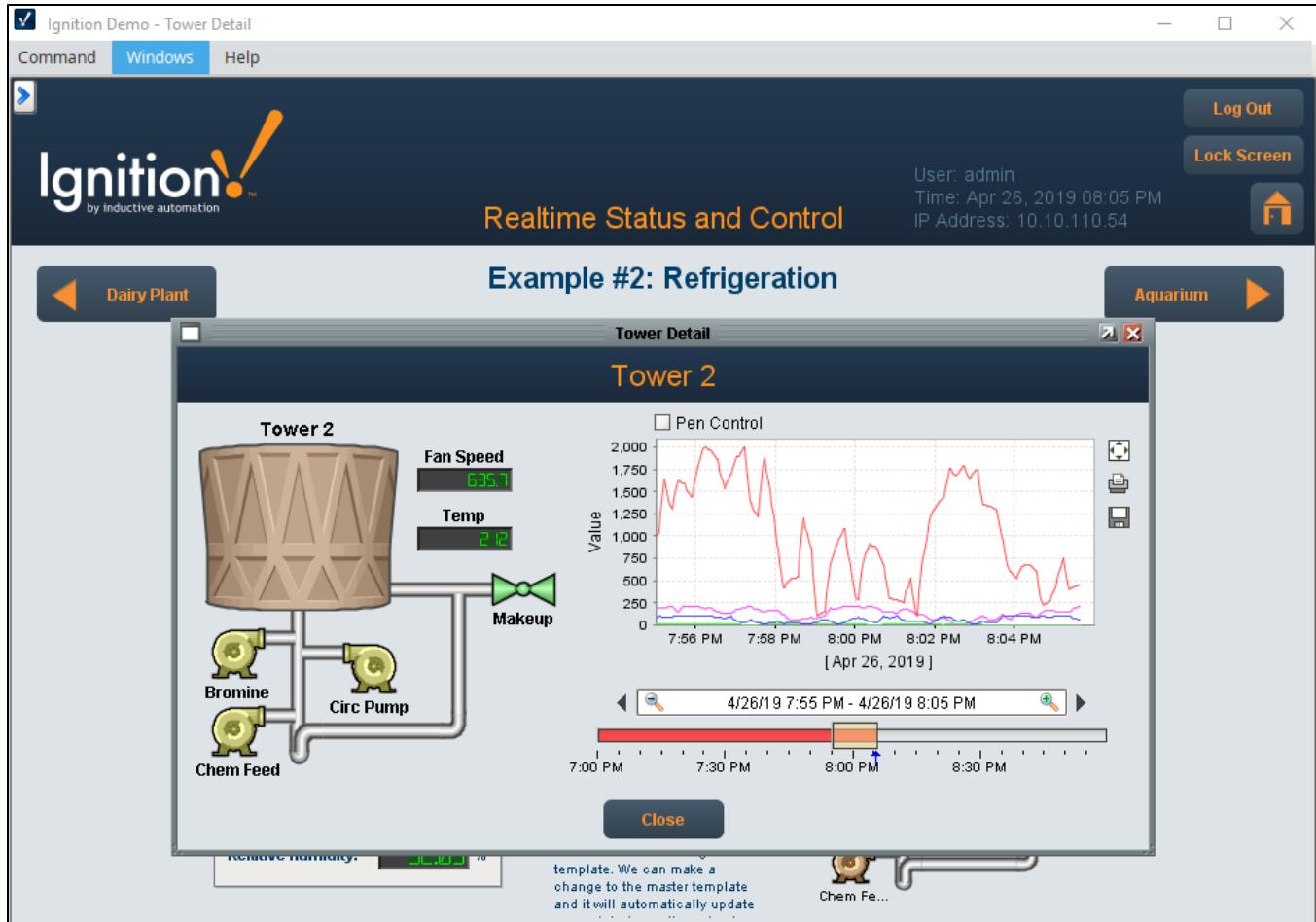
- [Creating a Popup Window](#)
- [Opening a Popup Window](#)
- [Popup Window Properties](#)
 - Layer
 - Location
- [Parameterized Popup Windows](#)
- [Multiple Instances of a Popup Window](#)



INDUCTIVE
UNIVERSITY

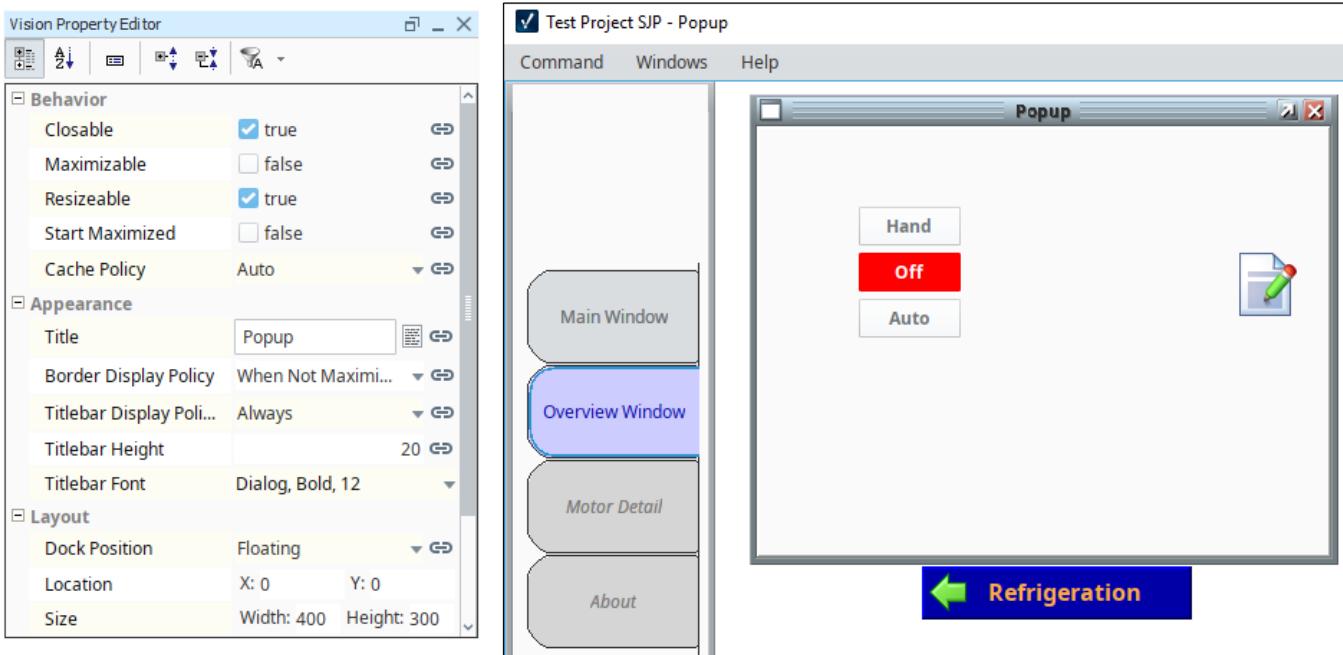
**Open Popup
Window**

[Watch the Video](#)



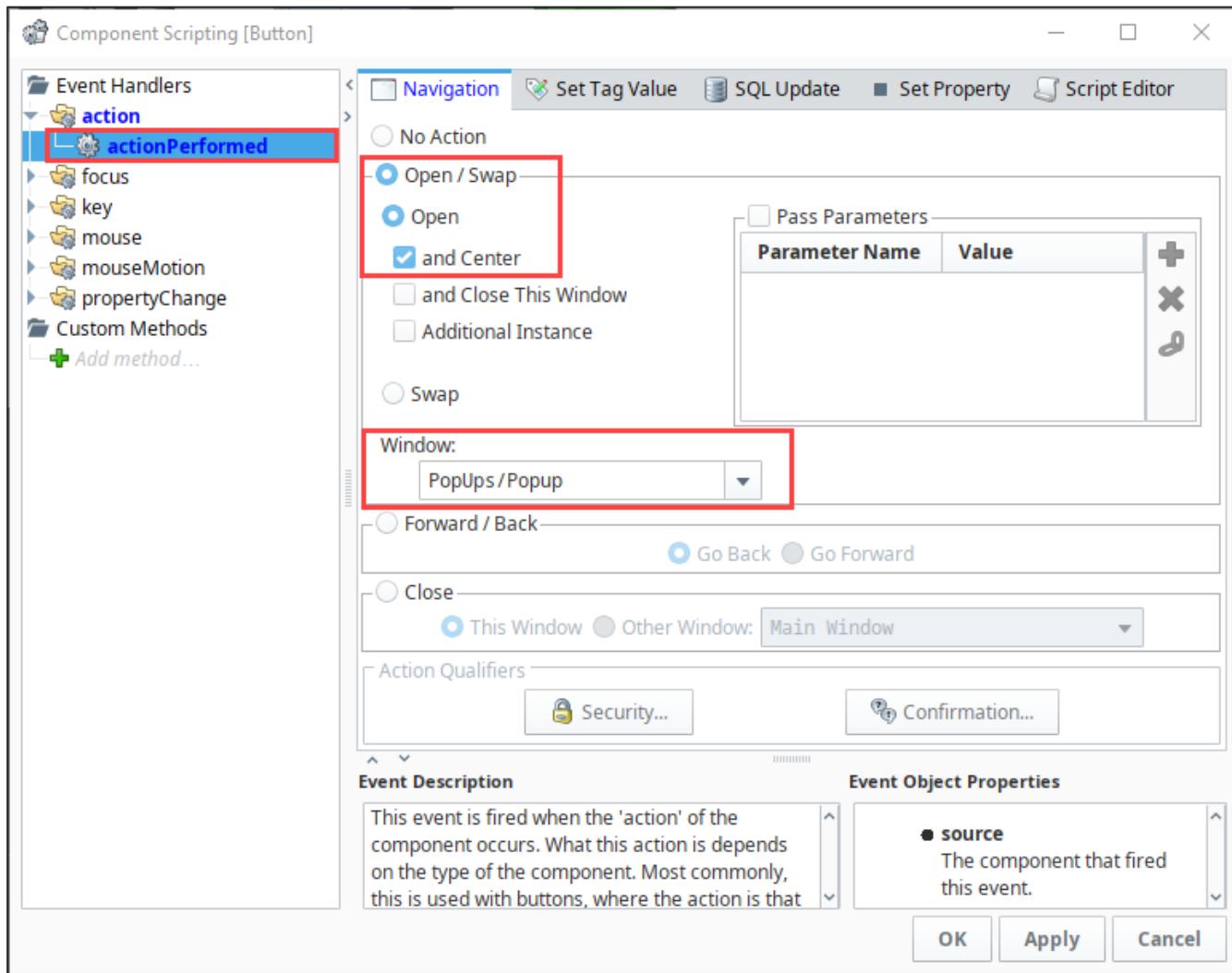
Creating a Popup Window

Before you can open a popup window, you have to create it. Like main windows and docked windows, popup windows are simply windows that have specific settings. In particular, popup windows are floating windows that are not set to start maximized. In the Designer, when adding a new window, selecting the popup window option creates a window with these presets. Once you have your popup window created, you can make it as big or small as you want. You can also set properties in the Property Editor to make it closeable, resizable, change the title, and display the title bar and border in the window.



Opening a Popup Window

In any window, you can add a script to any component to open your popup. This is easiest to do from a component like a button on the main window using the [Navigation Script Builder](#). Simply select the Open action and the window that you want to open. Clicking on the button will then open the popup window that was selected. Alternately, you can use one of the many [scripting functions](#) that open a window.

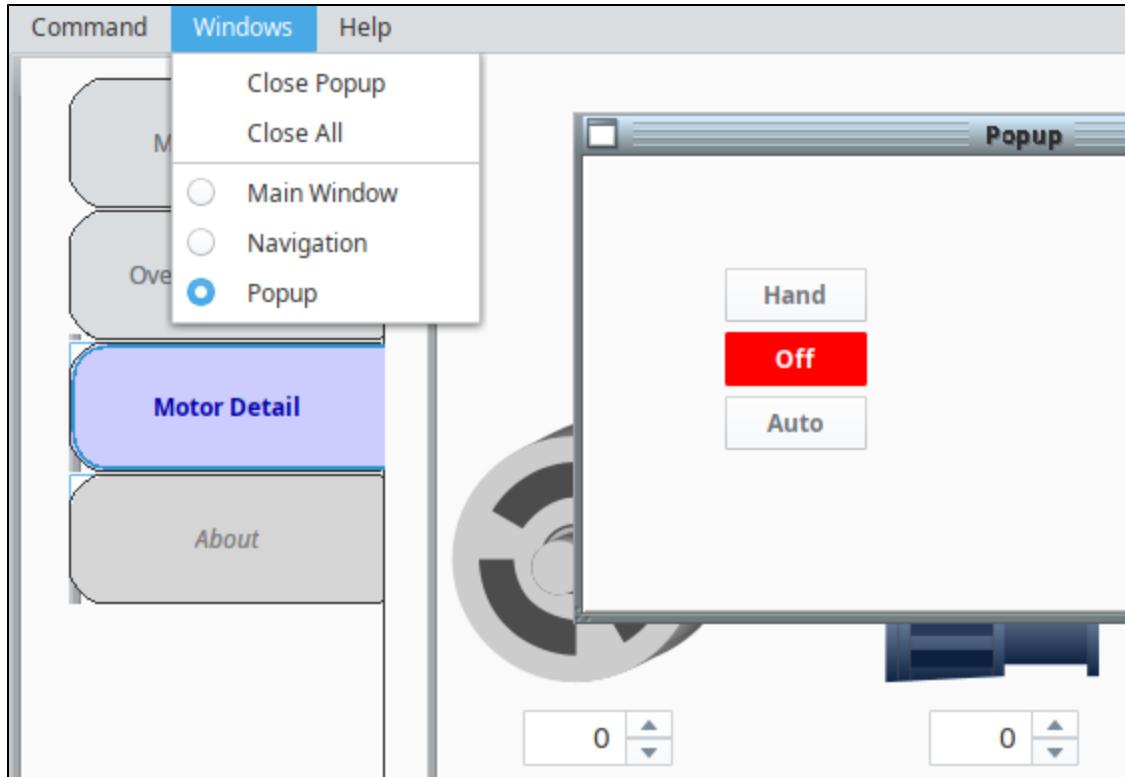


Popup Window Properties

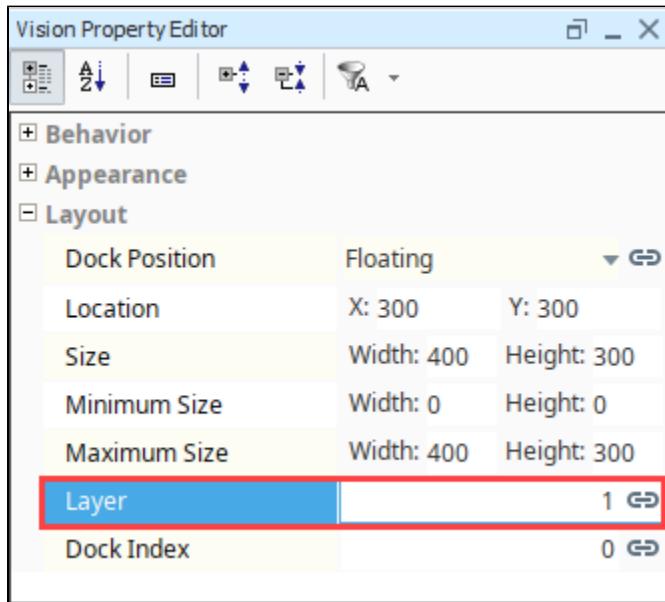
There are a few properties of the Window that are useful to popups.

Layer

The Layer property of windows controls the z-order of the windows. Windows with a higher Layer will always be on top of windows with a lower Layer, regardless of which window is in focus. This is useful for keeping popup windows at the forefront. By default, all windows have a Layer of 0, but we can change this so that popups always remain on top. If popups have a layer that is the same as the main window, clicking on your main window makes it look like the popup window disappears, but it's actually behind your main window. The **Windows Menu** will show you all the open windows in your Client, with the popup still being open.

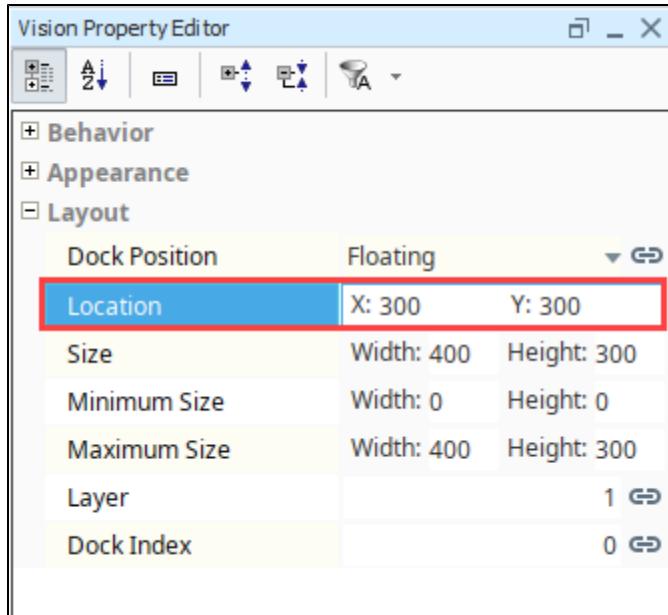


The Layer property is located on the window object itself, in the Property Editor. Simply set it to a higher value so that the popup is always on top.



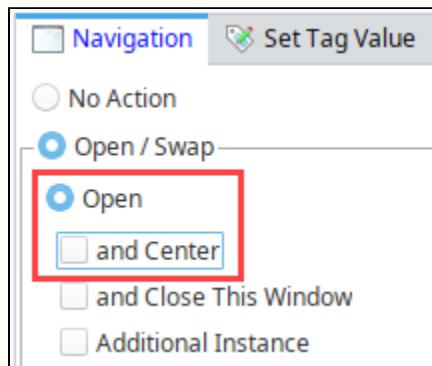
Location

Popup windows can also be given a specific location to open up at, when not being automatically centered by the script. In the Vision Property Editor, go to **Layout > Location** and provide a specific X and Y position (in pixels).



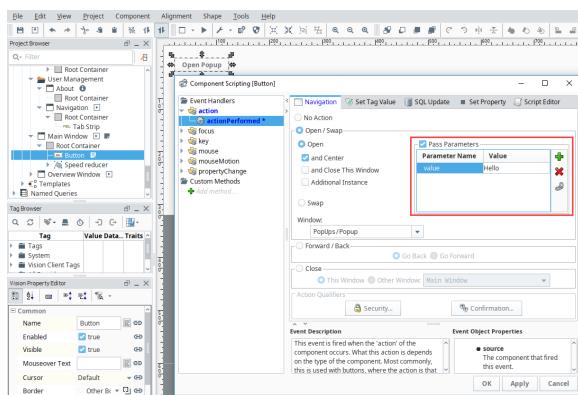
Note:

When opening a popup window to a specific location, ensure the **Open** and **Center** option is unchecked so that it doesn't override the location coordinates.



Parameterized Popup Windows

A [parameterized popup window](#) lets you pass information from one window to another window. You can make a single popup window, change what it does and what it points to from a parameter(s) that gets passed into the receiving window using Custom properties. Parameters can range from simple integers and strings, to properties on the window that is opening the popup, and even entire [UDT](#) custom properties.



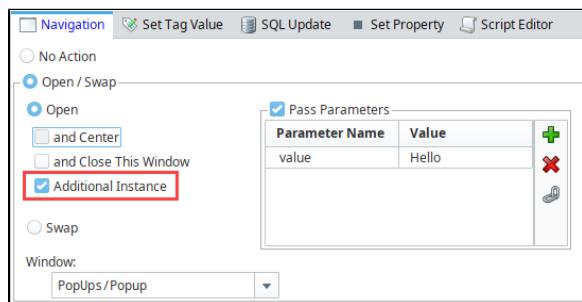
Parameterized Popup Window

[Watch the Video](#)

Multiple Instances of a Popup Window

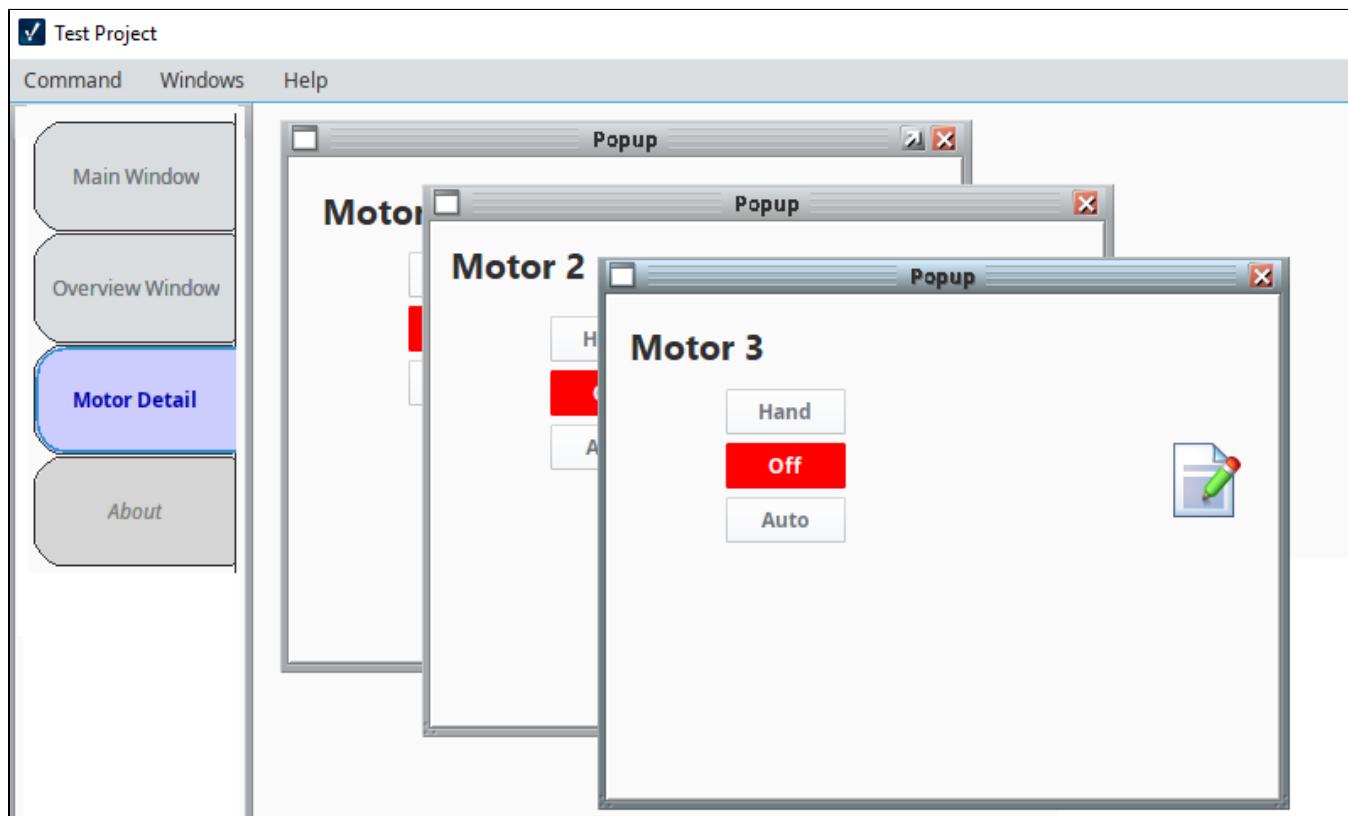
By default, the client only opens a single instance of a popup window, but you can change this behavior. For example, suppose you have four different Tanks passing all the same parameters with the only difference being the individual Tank number. In order to see all four instances of your tanks, you need to configure component scripting to display Additional Instances. This is done by selecting the **Additional Instance** option when setting up the navigation scripting action.

Alternately, the `system.nav.openWindowInstance` can be used within a more complex script instead.



Popup Window - Multiple Instances

[Watch the Video](#)



[In This Section ...](#)

Parameterized Popup Windows

A parameterized popup window lets you pass parameters from one window into a popup window, where the receiving popup window could then use that parameter to display relevant information. This also allows you to maintain a single window that can be used to display similar information. For example, suppose you have two compressors: Compressor 1 and Compressor 2. Imagine clicking on one of the compressors on the main window and a popup displays the diagnostic information about that specific compressor. Instead of creating a popup window for each compressor, you can create a single popup and use indirection with the passed parameter to display a different compressor's information depending on which was selected.

Passing Parameters to a Popup Window

To pass parameters from one window to a popup window, the receiving popup window must have [custom properties](#) that receive the passed parameters. When the event on the parent window is called, the parameters are passed to the receiving Popup Window's custom properties on its root container. The component's properties on the receiving window can use the root container's custom properties to address their bindings.

The following examples explain how to setup a main window and a popup window to pass compressor numbers to the popup window in order to display relevant information about each compressor.

On this page ...

- [Passing Parameters to a Popup Window](#)
 - [Setting up the Popup Window](#)
 - [Setting up the Main Window](#)
 - [Passing a UDT to a Popup](#)

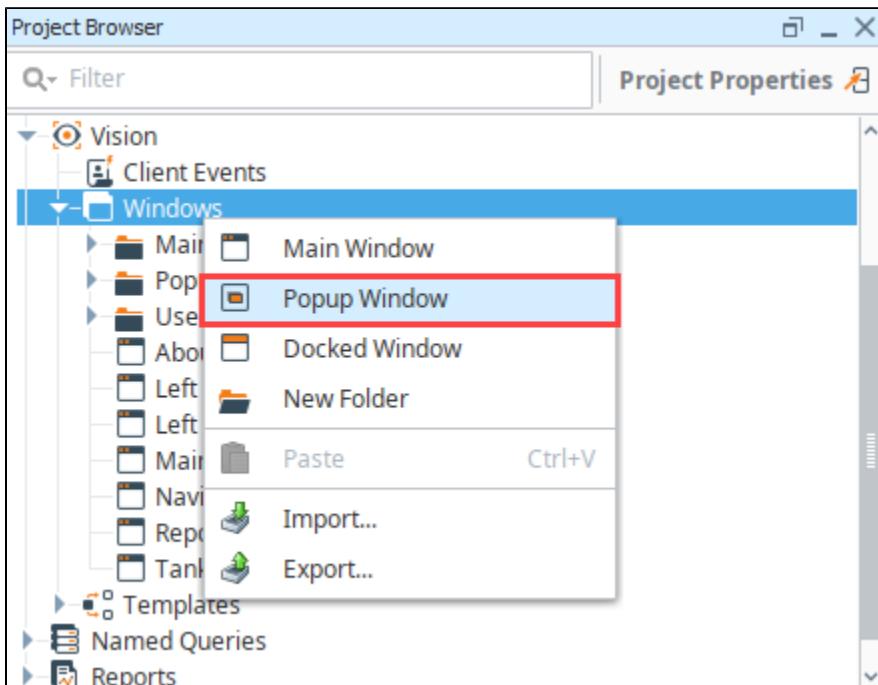


Parameterized Popup Window

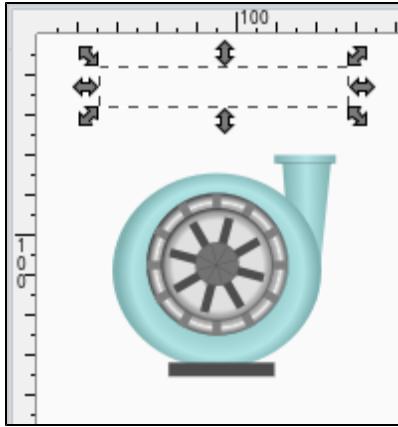
[Watch the Video](#)

Setting up the Popup Window

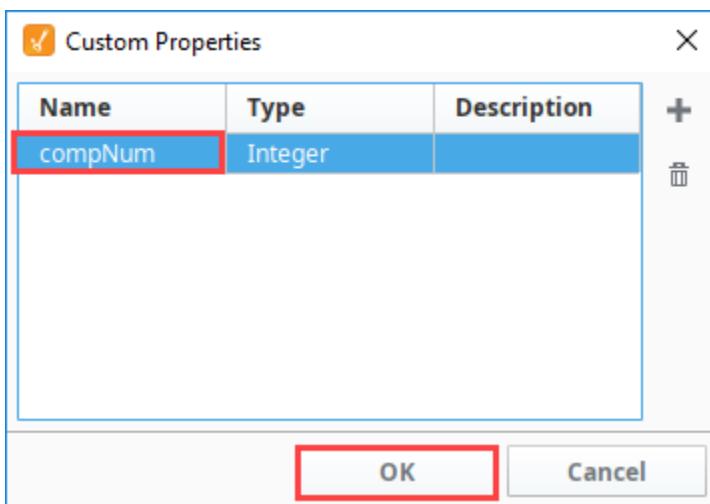
1. Right click on a folder in the Project Browser and select Popup Window to create a new popup.



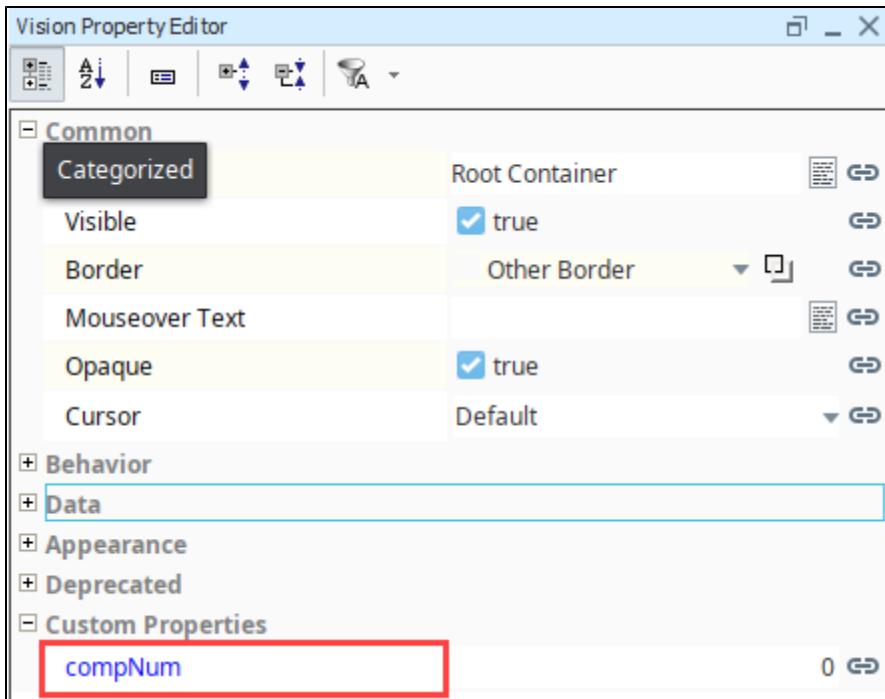
2. Drag a **Compressor** image from Symbol Factory.
3. Drag a **Label** component from the component palette to your window.



4. Create a [custom property](#) on your popup window that will receive the passed parameters. Right click on your window and select **Customizers > Custom Properties**. The Custom Properties window is displayed.
5. Click the **Add**  icon to add a property.
6. Specify a **Name** for the Custom Property, such as **compNum**, and click **OK**.



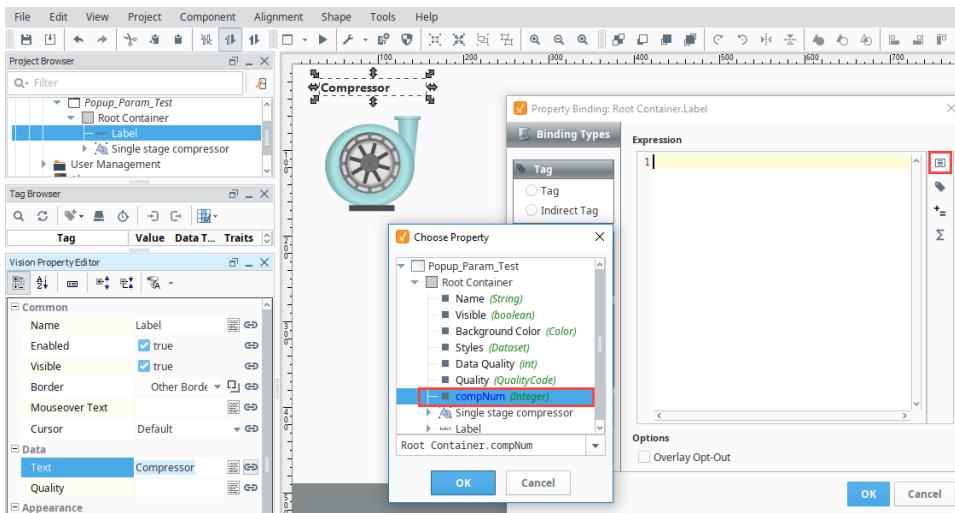
The custom property is created and displayed in blue at the bottom of the Property Editor.



Caution: Do not bind these custom properties to anything, leave them unbound so you can pass values into them without any other values to override them.

- Let's use an expression on the Label to show what compressor number we are on.

- Select the **Label** and click the binding icon for the **Text** property.
- Select **Expression** for the binding type.
- Click the **Insert Property Value** icon in the Expression window and choose the **compNum** custom property on the root container for the compressor popup window as shown in the image below. Click **OK**.

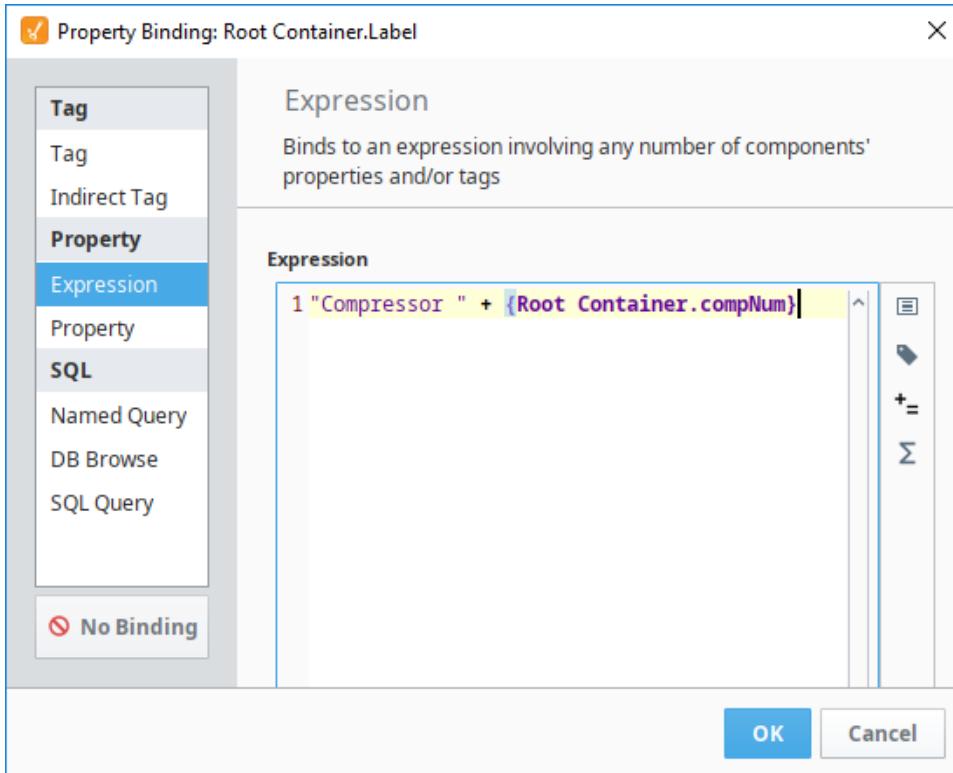


- Now, let's update the Expression using the script below to show the word "**Compressor**" before the number in the label. Update the Expression as follows.

Script to change the compressor number

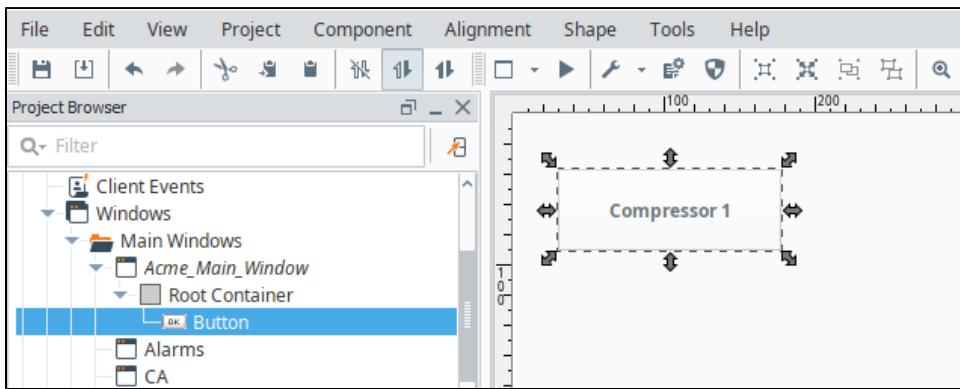
```
"Compressor " + {Root Container.compNum}
```

- The expression should look as follows in the Property Binding window. Click **OK** to save the property binding.



Setting up the Main Window

1. In a Main Window (parent window) drag a **Button** from the component palette to your window. Type "Compressor 1" into the text property.

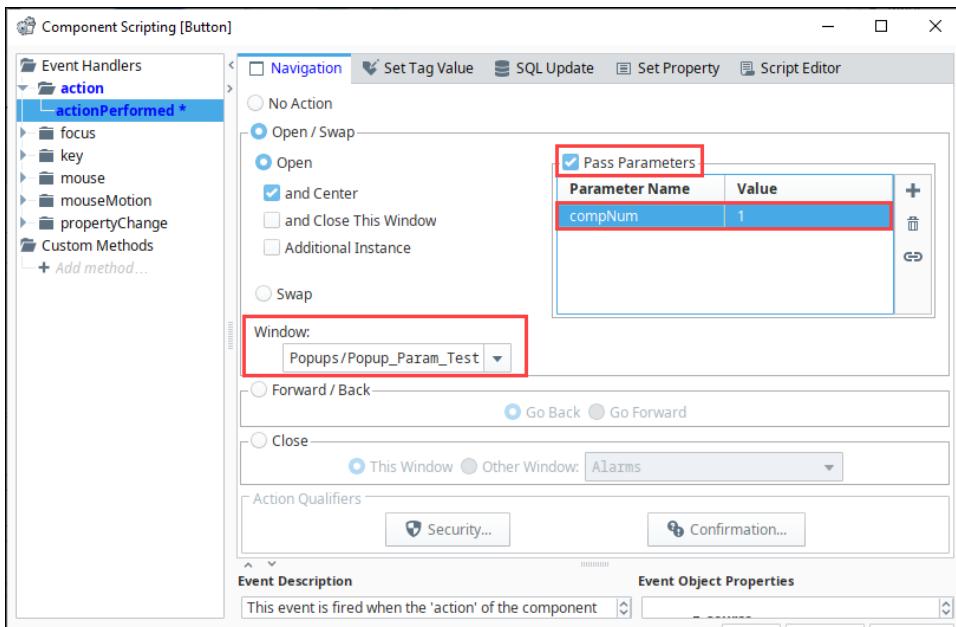


2. Let's add a script to the button which opens the popup that we created earlier. We can then pass in a value to designate that this button should be opening Compressor 1.

- a. Right click on the **Compressor 1** button and select **Scripting**.
- b. Under Event Handlers, select **actionPerformed**.
- c. Click the **Open / Swap** radio button.
- d. Under Window, use the dropdown list to select the path to your Popup Window (i.e., **Popup_Param_Test**).
- e. Check the **Pass Parameters** check box, and click the **Add** icon to add a parameter.
- f. Click the new row under **Parameter Name** and a dropdown list will appear. Select the custom property **compNum**.

Note: Ignition will automatically check the Root Container of the window selected in the **Window** dropdown. If you do not see the **compNum** parameter, it may have been created on wrong component, so check the Root Container of the Compressor Popup window.

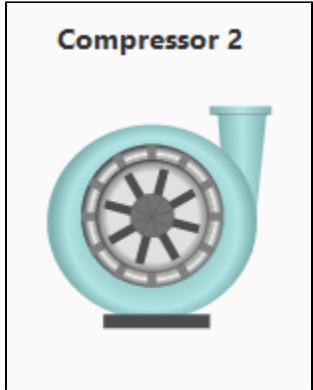
- g. Enter "1" in the **Value** column because the button will be for Compressor 1.
- h. Click **OK** to save the script.



3. Now, create a second compressor button. A quick way to do this is duplicate (**Ctrl D**) the Compressor 1 button so it inherits the script. Update the Text property to **Compressor 2**.
4. Right click on the new **Button** component and select **Scripting**. Update the parameter Value being passed in from a "1" to a "2".
5. Click **OK** to save the script.

6. Test it out by putting the Designer in **Preview Mode**. Click one of the Compressor buttons, then navigate back and click the other Compressor button. While these buttons are opening the same popup, they display different information because they are using the parameter that we passed in for indirection. In this example, we just used a label, but the parameters can be used in things like **indirect Tag**

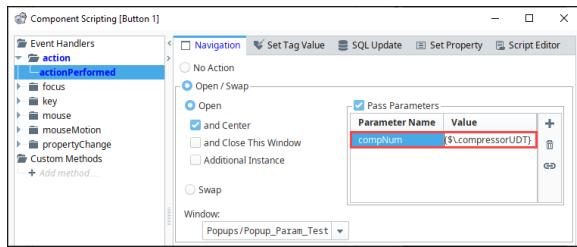
[bindings](#) or scripts to pull in various Tag bindings.



Passing a UDT to a Popup

In addition to the basic types, parameters can be a complex [UDT type](#). This works much the same as passing in basic values, where the popup window has a custom property on the root container, and a parameter is passed in when opening the window.

The difference is that the custom property on the popup window needs to be a UDT that has been previously defined, and the value being passed in when opening the window needs to be an entire UDT instance. This gives the popup access to every Tag within the UDT, which can be useful when making popups that show all the details of a certain area which has a UDT.



Parameterized Popup Window and UDTs

[Watch the Video](#)

Navigation Strategies in Vision

Navigation Strategy

Setting up a navigation strategy allows you to navigate between different windows in the runtime Client. Ignition provides several different types of runtime navigational strategies you can choose from when designing your project including several [Vision project templates](#) to help you get started. Before selecting the proper navigation strategy or template for your project, there are several things to consider. These considerations will help you determine the best navigation strategy to use for your project and your users. Once you address these considerations, then you can choose the best navigation strategy from the types below. This will also help you decide if you want to use a Vision Project Template to quickly for your project.

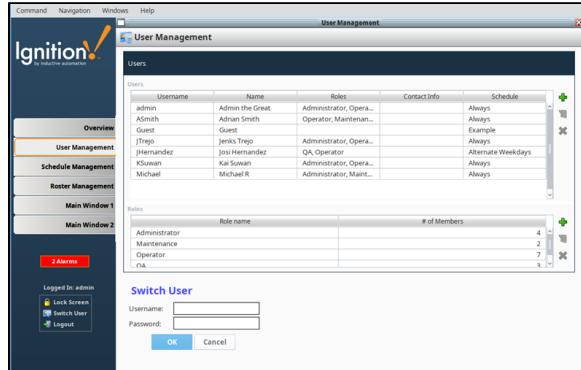
- Does your project have a lot of windows?
- How complex is your project structure?
- Is your project structure organized?
- What types of things are you doing?
- Do you want to use navigation windows or fill the screen?

Types of Navigation Strategies

To help you select the right strategy that fits your project structure, here is a brief description of each navigation strategy that Ignition provides. Keep in mind your project structure, size, organization, and types of things you are doing while you are reviewing these strategies so you can select the best runtime strategy for your project.

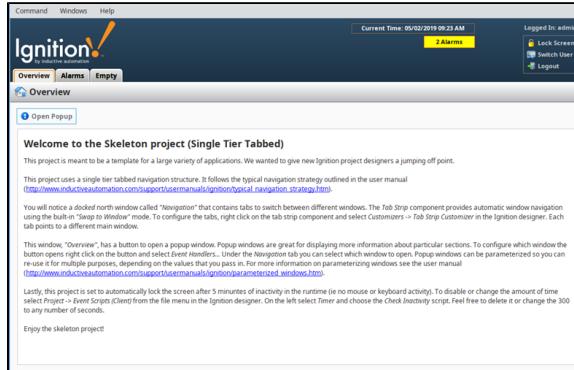
- **Tab Strip Navigation** is a simple strategy used for small structures regardless of how organized your project is. It lends itself perfectly to only having a few windows and showing all of them on a navigation window. Having too many tabs does not work well with the Tab Strip because of size limitations. You want your users to see all the navigation tabs immediately on the first screen. The Tab Strip works by clicking to swap one main window for another.
- **Two Tier Navigation** is similar to the Tab Strip, but is good for small and regular size project structures where windows are grouped. It contains a second level of tabs allowing you to navigate around various areas of your project. This strategy has a docked window that contains tabs that are always open to do navigation, and the main window which fills the rest of the space.
- **Tree View Navigation** is excellent for large project structures. You can view the entire project structure at a glance allowing you to navigate to any structure within the multi-tier Tree View component.
- **Back and Forward Buttons** are perfect if you have a small process with ordered steps. It is one big main window that has Back and Forward buttons to step through each process step or operation one right after the other.
- **Drill Down Navigation** is ideal if you have different geographical locations, whether it's in a local facility or facilities sprinkled around the world. The project opens with an overview that has areas that correspond to specific locations/areas in your facility. With the Drill Down strategy, you can select a specific area representing the facility, and the client swaps windows to display details pertaining to that specific area.
- **Menubar** is ideal for maximizing the usable screen space, while still having the ability to navigate to any window at any time by selecting from a list of windows.
- **Retargeting** enables navigation between multi-project operations: a simple script can push the user between many different projects, even on different gateways.

Tab Strip Navigation



Tree View Navigation

Two Tier Navigation



Back and Forward Button Navigation

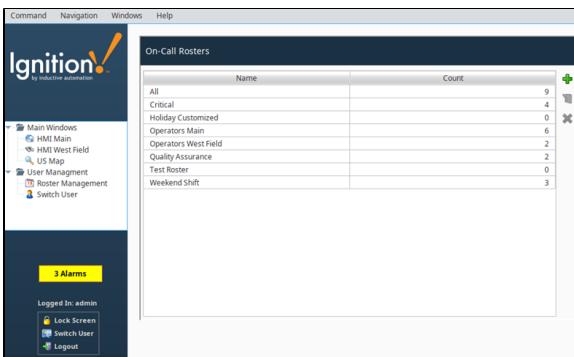
On this page ...

- **Navigation Strategy**
 - Types of Navigation Strategies
 - Tab Strip Navigation
 - Two Tier Navigation
 - Tree View Navigation
 - Back and Forward Button Navigation
 - Drill Down Navigation
 - Menubar Navigation
- **Navigation Operations - Swapping vs. Opening**
 - Opening
 - Swapping
- **Common Navigation Mistakes**
 - Multiple 'Main' Windows
 - Swapping a Main Window with a Docked Window

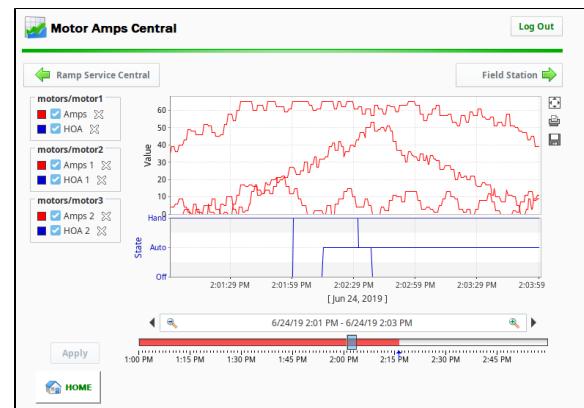
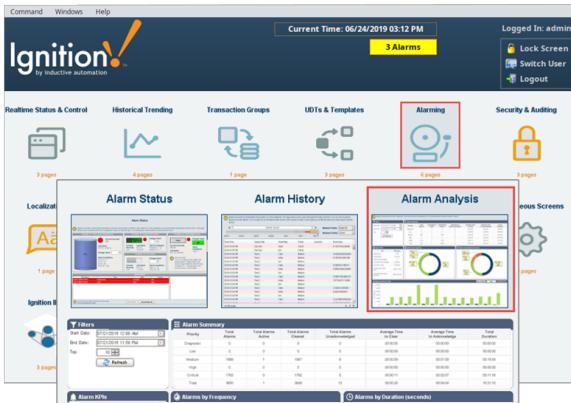


Navigation Strategies

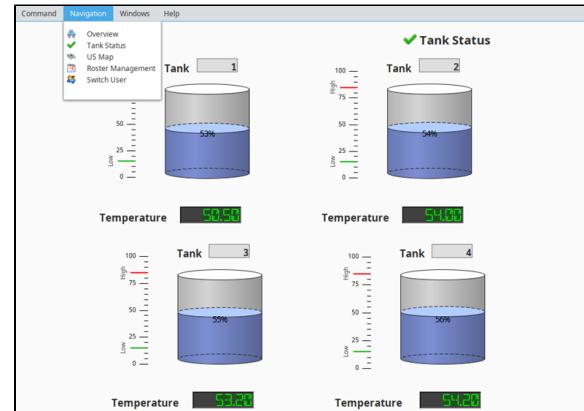
[Watch the Video](#)



Drill Down Navigation

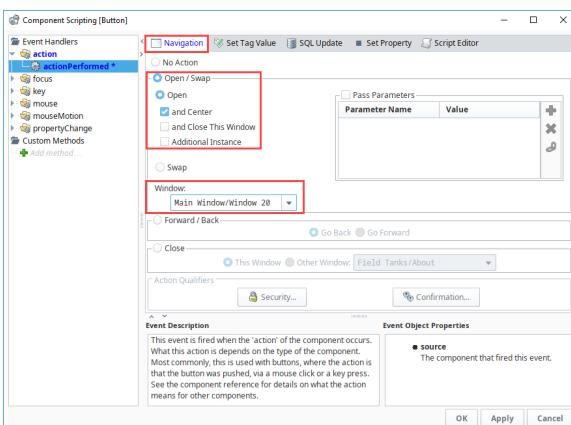


Menubar Navigation



Navigation Operations - Swapping vs. Opening

Any time you open a window, you have to use one of the two navigation operations: **Swapping** or **Opening**. These operations can be performed on any type of window, but are usually reserved for specific cases. Navigating between different windows typically involves some sort of scripting, but the [Navigation Script Builder](#) makes this a simple task. Otherwise, you can use the specific scripting functions to completely customize navigation.



Opening

Opening and by extension closing are the basic window navigation options. Opening a window opens the window at the same size it was in the Designer, unless the Start Maximized property is **true** or the Dock Position is **not Floating**. This is typically reserved for opening popup windows. They have a scripting function that can open and a function that can close.

- `system.nav.openWindow`
- `system.nav.closeWindow`

INDUCTIVE UNIVERSIT

Swapping vs. Opening

[Watch the Video](#)

INDUCTIVE UNIVERSIT

Navigation Functions

[Watch the Video](#)

Swapping

When Ignition swaps a window, it closes the current main window and then opens another window in its place. This is typically reserved for moving between main windows, as it performs the close action automatically. There are two different scripting functions that can be used to swap windows, depending on what needs to be done.

- [system.nav.swapWindow](#)
- [system.nav.swapTo](#)

Common Navigation Mistakes

Multiple 'Main' Windows

The most common mistake that will cause windows to stay open unintentionally is to implement a swapping navigation system using the `system.nav.openWindow()` function on main windows instead of `swapTo`. When you do this, the next time the `swapTo` function is called, it may swap from a window that is hidden behind the current 'Main' window and look like nothing happened. It is easy to check the client's Windows menu to see what windows are currently open. If there are more windows listed there than you can currently see, there is a problem in your navigation logic that is failing to close windows properly.

Swapping a Main Window with a Docked Window

Another common mistake that will cause windows to stay open unintentionally is to implement a swapping navigation system using the `system.nav.swapTo()` function on windows that are docked. This will cause your docked windows to be 'swapped in' as a maximized window instead of its usual size. When you do this, the client will not see it as a main window and next time the `swapTo` function is called, it may not have space on screen to open the main window. Logging out and back in to the client (or restarting it) is the only solution to this. Identify the offending button or script that is swapping the docked window and change it accordingly.

Related Topics ...

- [Vision Project Templates](#)

In This Section ...

Navigation - Tab Strip

The **Tab Strip** component provides a simple navigation strategy used for small project structures having only a few windows. It allows users to see all the navigation tabs on the first screen of the client. It is most commonly used in a docked window to provide automatic window navigation. The Tab Strip works by clicking on a tab to swap one main window for another. The Tab Strip has two navigation modes:

- **Swap Windows** - the Tab Strip automatically calls `system.nav.swapTo()` with the name of the selected tab for easy navigation from one window to another.
- **Disabled** - the Tab Strip doesn't do anything when a tab is pressed. Users can customize tabs using property bindings or by responding to the `propertyChange` scripting event.

A Tab Strip is an effective primary navigation strategy, particularly when you don't have many items to choose from.

On this page ...

- [Tab Strip Navigation Example](#)



Navigation - Tab Strip

[Watch the Video](#)

Tab Strip Navigation Example

Tab Strip navigation is simple to setup. In the following example, we'll setup a small project that has a few windows which are visible on the navigation tabs.

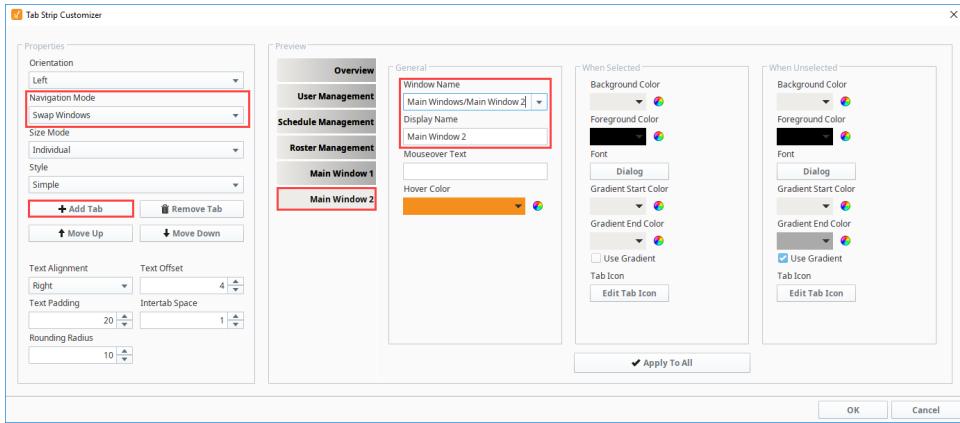
1. Add a Tab Strip component to a window, typically a docked window.
2. Right-click on the Tab Strip component, choose **Customizers > Tab Strip Customizer**.
3. In the Tab Strip Customizer you can specify which window to open with each tab. Notice the **Navigation Mode** property which is set to **Swap Windows** as shown in the screenshot below.
4. To create a new tab, click **Add Tab**. If you have a tab already selected, clicking the **Add Tab** button creates a Tab with the same colors and font as the selected tab.



Main Windows already created

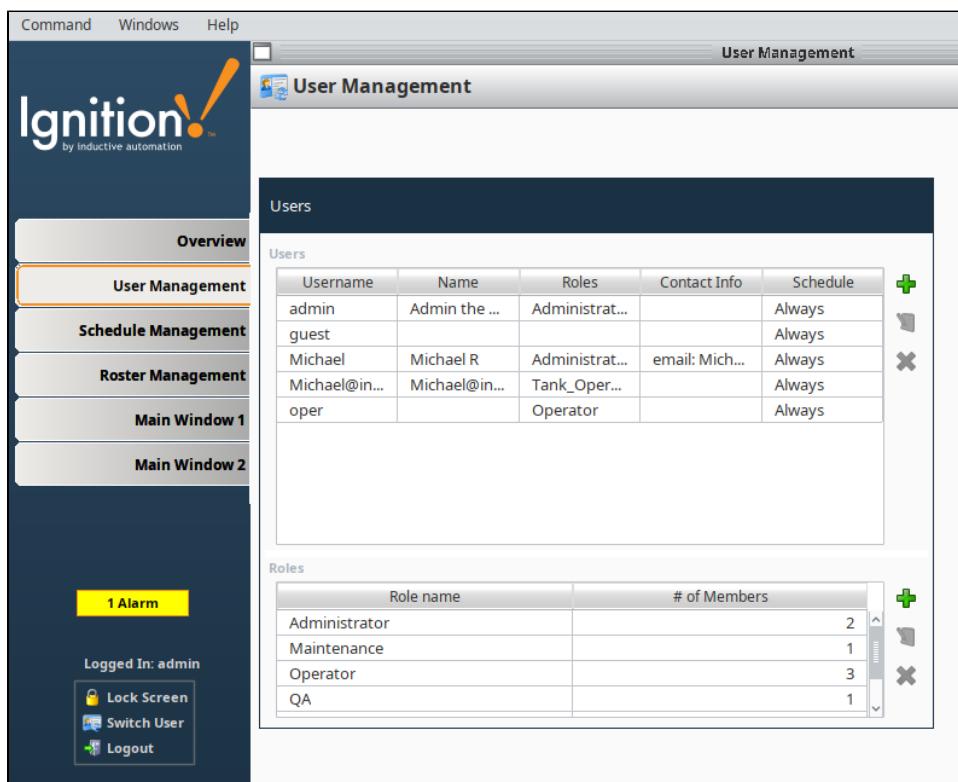
This step assumes you already have your main windows created in your Project Browser.

- a. Under **Window Name** dropdown list, select the window you want to open. Note, it is the full path from the window and not just the name (i.e., **Main Windows/Main Window 2**).
- b. Enter the **Display Name** for your new tab (i.e., **Main Window 2**).
- c. With the **Move Up**, **Move Down**, and **Remove Tab** buttons you can move tabs up and down on the tab strip, and remove a tab if it is no longer required.
- d. You can also set the **Background** and **Foreground** colors when to show when a tab is selected and unselected.
- e. When you're finished, press **OK**.



5. Save your project.

6. Open your project in the **Client**, and see that each tab navigates to a different window. As you can see, the **Tab Strip** works by clicking on a tab to swap one main window for another.



Navigation - Two Tier

Two Tier Navigation is similar to the [Tab Strip navigation](#) strategy. It's good for small and regular size project structures where windows are grouped, and lets you organize your main windows into different sections making navigation easy for users. It uses two levels of tabs to navigate around various areas of your project. Once you select a first tier tab, a different set of tabs appear in the second tier to switch between different windows.

This works a bit differently than the default Tab Strip navigation, as the first tier Tab Strip will actually not do any window swapping, which will instead be left up to the second tier of tabs. The Two Tier approach has a docked window that contains multiple sets of tabs. One set is used as a higher level of grouping of windows, and is used to conditionally swap out another set of tabs based upon user selection.

In the image below, the top tier of tabs contains the HMI Screens and Administration tabs. Clicking on **HMI Screens** makes a second tier of tabs appear (the set containing **Overview**, **Alarms**, and **Empty**). Clicking on the **Administration** tab would make a different set of sub tier tabs appear. Thus, the top tier of tabs isn't directly responsible for any sort of window navigation. Rather, it's used to make other sets of tabs appear.



On this page ...

- [Two Tier Navigation Example](#)



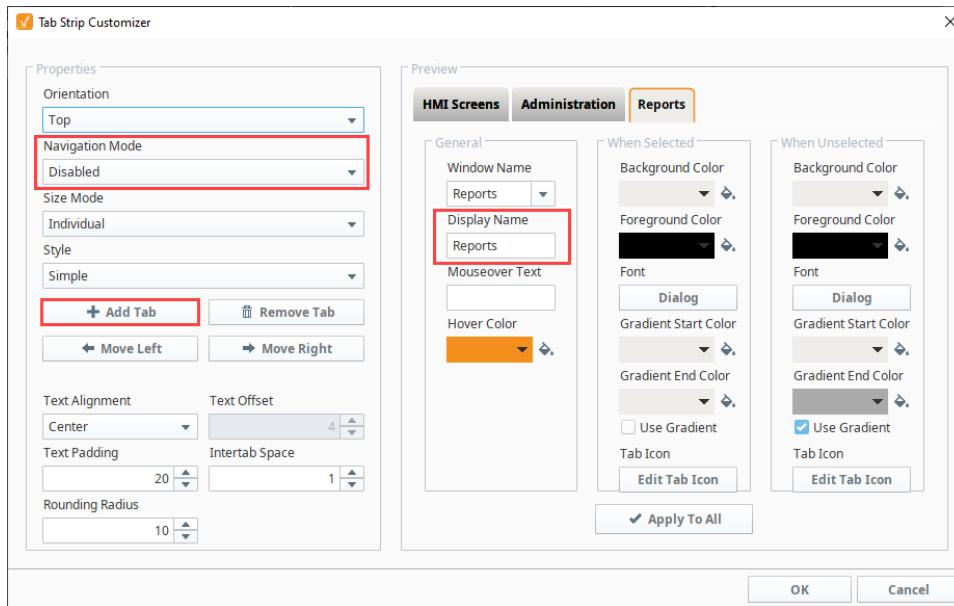
Navigation - 2 Tier

[Watch the Video](#)

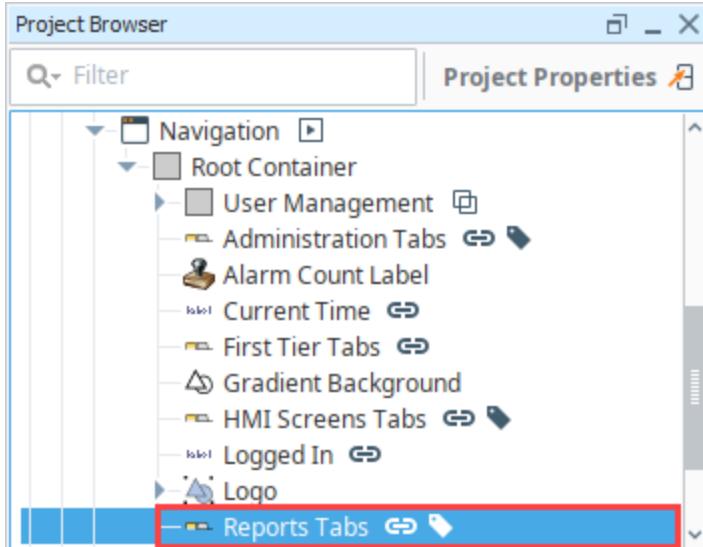
Two Tier Navigation Example

In this example, we are using the 2-Tier Tab Nav project template which is selectable upon project creation. This comes with two tiers of tab strips and several default tabs. We will add one tab on the first tier and two tabs on the second tier.

1. Right click on the **First Tier Tabs** in the Project Browser to add another tab. Select **Customizers > Tab Strip Customizer**.
2. To create a new tab, click **Add Tab**, and position it on the tab strip anywhere you like.
3. Enter your **Window Name** and **Display Name**. Make sure the Navigation Mode is set to **Disabled** since this first tab strip is not swapping to any windows.



4. Now let's create a second tier category of tabs for the Reports tab. The easiest way to do this, is copy the second tier of tabs from either the HMI Screens or Administration tabs in the root container of the Navigation folder of the Project Browser. **Paste** it in the root container of the Navigation folder giving it a unique name identifying what it is (i.e. Reports Tabs).

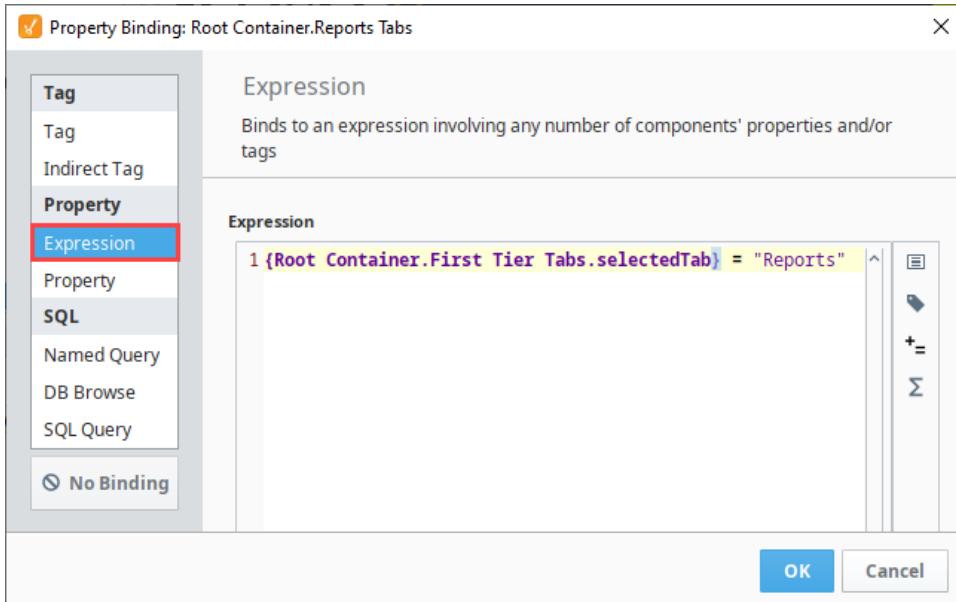


5. Right click on the **Reporting Tabs Tab Strip** of the Designer and select the **Customizers > Tab Strip Customizer**.
6. Add Tabs like you would for a normal Tab Strip, but only add tabs that fit that category of windows. In this step, we are adding two tabs: Report 1 and Report 2. Click **OK** to create the new tabs.

Note: This example assumes you already have some main windows created in your Project Browser (i.e., Report 1 and Report 2).

7. The second tier tabs can be shown or hidden depending on what tab is selected in the first tier.
 - a. Select a second tier Tab Strip (i.e., Reports).
 - b. Go to the **Property Editor**, select the **Visible** property and set it to true, and then select its binding icon.
 - c. Select the **Expression** binding and set up an expression to be true when the appropriate first tier tab is selected, as shown in the image below. Click **OK**.

```
{Root Container.First Tier Tabs.selectedTab} = "Reports"
```



8. **Save** your project.
9. Open your project in the **Client**, click on the various tabs to see your first and second tier tabs switch between the different windows.

The following images show the first and second tier tabs for **HMI Screens** and **Reports** tabs.

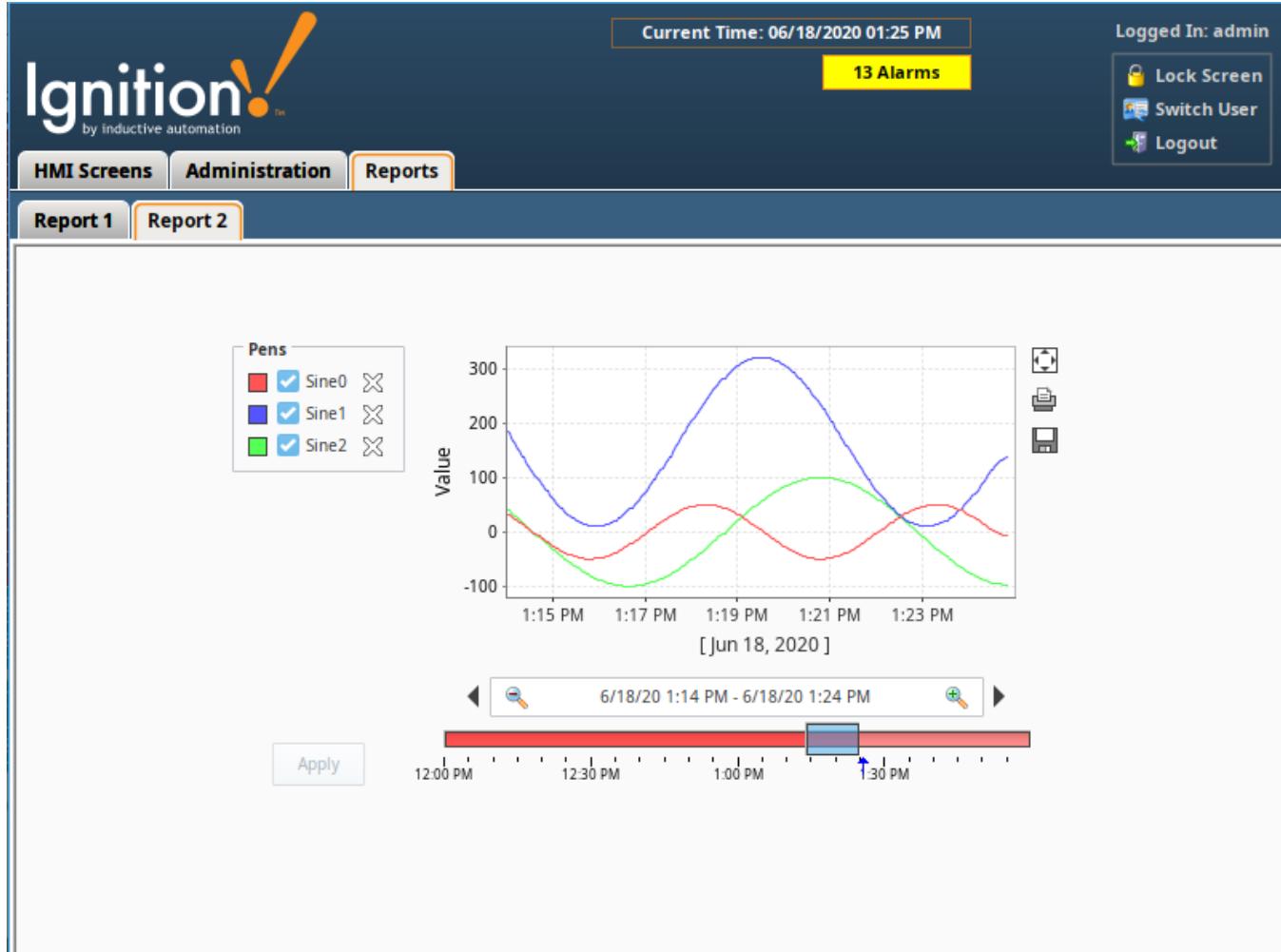
HMI Screens Tab

The screenshot shows the Ignition HMI Screens interface. At the top, there's a header bar with the Ignition logo, the current time (06/18/2020 01:28 PM), and a red button indicating 12 Alarms. On the right, it shows the user is logged in as admin with options to Lock Screen, Switch User, or Logout. Below the header, there are tabs for HMI Screens, Administration, and Reports, with Overview and Alarms selected. The main content area is titled "Alarms" and displays a table of active alarms. The table has columns for Active Time, Display Path, Current State, and Priority. The data is as follows:

| Active Time | Display Path | Current State | Priority |
|-------------------|---|-------------------------|----------|
| 6/10/20, 7:11 PM | Speed/High Speed | Active, Unacknowledged | Critical |
| 6/10/20, 7:12 PM | Tank Level 2/Low SP2 | Active, Unacknowledged | Critical |
| 6/10/20, 7:12 PM | Writeable/WriteableInteger1/Low Tank Level | Active, Unacknowledged | Critical |
| 6/10/20, 7:11 PM | Tank 100 | Active, Unacknowledged | High |
| 6/10/20, 7:11 PM | Tank 100 | Active, Unacknowledged | High |
| 6/10/20, 7:11 PM | Turbine Number 200 located at Livermore, CA | Active, Unacknowledged | High |
| 6/10/20, 7:11 PM | Turbine Number 150 located at Folsom, CA | Active, Unacknowledged | High |
| 6/10/20, 7:11 PM | Turbine Number 100 located at Folsom, CA | Active, Unacknowledged | High |
| 6/10/20, 7:11 PM | Turbine Number 300 located at Fresno | Active, Unacknowledged | High |
| 6/18/20, 1:24 PM | Sine/Sine1/High Level | Active, Unacknowledged | High |
| 6/10/20, 7:11 PM | High Temp/High Temp | Active, Unacknowledged | Medium |
| 6/10/20, 7:11 PM | F Temp/Alarm | Active, Unacknowledged | Low |
| 6/18/20, 12:49 PM | Sine/Sine2/Low Level | Cleared, Unacknowledged | Critical |

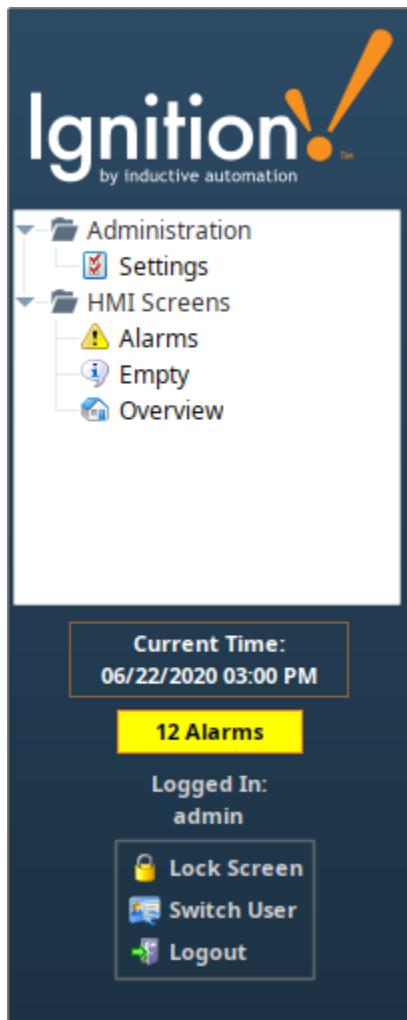
At the bottom left are buttons for Acknowledge and Shelve. To the right are search and filter icons.

Reports Tab



Navigation - Tree View

The **Tree View** navigation strategy is excellent for large project structures. It uses a typical navigation strategy again with a docked west window that contains a Tree View to navigate around to various areas. Users can double click on an item in the tree view and it will swap out one main window for another. The list is fairly compact, and can contain folders, helping you to group similar windows just like you would in the project browser.



On this page ...

- [Tree View Navigation Example](#)



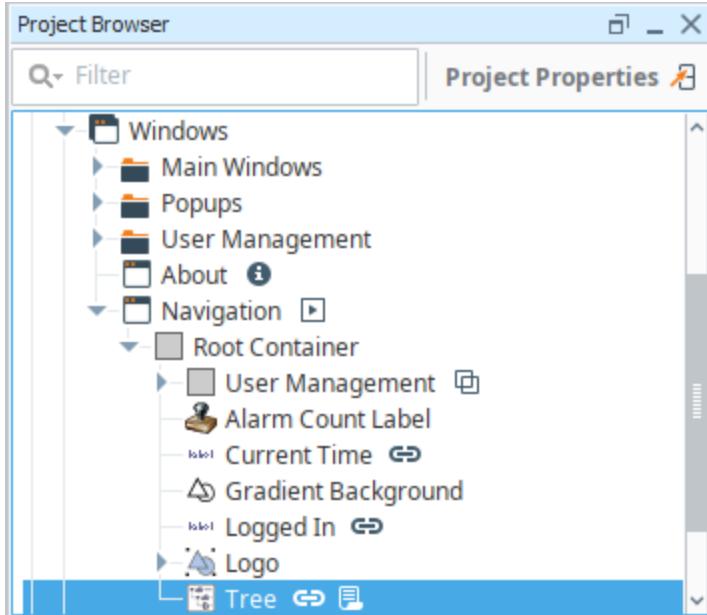
Navigation - Tree View

[Watch the Video](#)

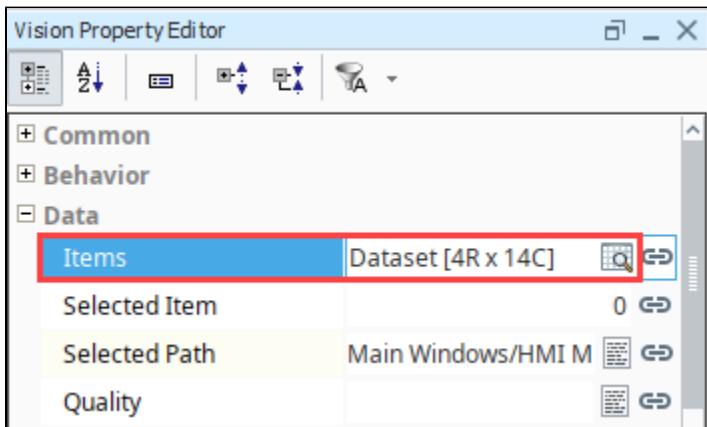
Tree View Navigation Example

In this example, we are using the Tree View Nav project template which is set up when the project is created. By default, Tree View navigation comes with several default folders to help get you started. This simple example adds one new main window.

1. Once you create your project and set the Tree View as your navigation strategy, open the **Project Browser**, and expand the **Main Windows** folder. Click the **Navigation** folder to open the skeleton project, then select the **Tree** property.



2. From **Property Editor**, find the **Items** property and click on the **Dataset Viewer** icon.



3. This brings up a **dataset editor**. You will see a number of columns that identify how each tree view item is displayed. Each row corresponds to a node in the tree view. The **windowPath** column is the window that we want to navigate to. The **path** is the folder that the window will display in the Tree View.

Note: This example assumes you already have your new window created in your Project Browser.

4. Let's add a new main window under Administration. To add a new Main Window (i.e., Reports) we first need to add a row. Click the **Add Row** icon and populate the fields manually using the data from the previous row, or you can also copy the entire dataset into a notepad where you can manually manipulate the data to add a row, and then paste it back into the dataset editor. Click **OK**.

Note: Do not use the Tree View Customizer to edit any of your data. Use only the Dataset Editor, otherwise it will overwrite that item's dataset.

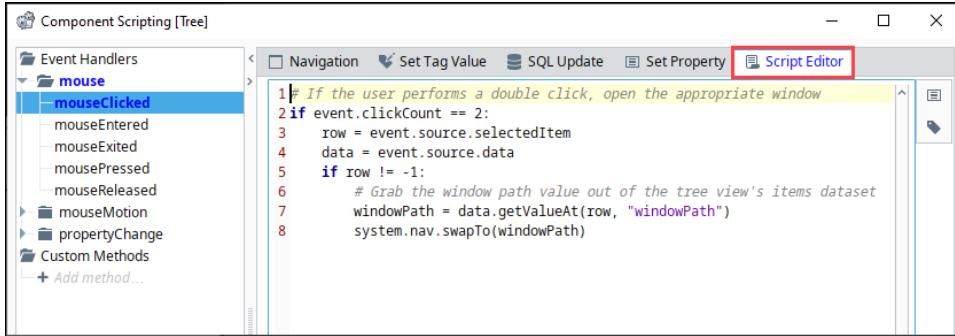
| Dataset Editor | | | | | | | | | | | | |
|-----------------------|----------------|----------|---------------------------------|------------------------|------------------|---------|--------|--------------|------------------|--------------------|----------------|--------------------|
| windowPath | path | text | icon | background | foreground | tooltip | border | selectedText | selectedIcon | selectedBackground | selectedBorder | selectedForeground |
| Main Windows/Overview | HMI Screens | Overview | Builtin/icons/16/home.png | color(255,255,255,255) | color(0,0,0,255) | | | Overview | Builtin/icons... | color(250,21... | | |
| Main Windows/Alarms | HMI Screens | Alarms | Builtin/icons/16/warning.png | color(255,255,255,255) | color(0,0,0,255) | | | Alarms | Builtin/icons... | color(250,21... | | |
| Main Windows/Budget | HMI Screens | Budget | Builtin/icons/16/about.png | color(255,255,255,255) | color(0,0,0,255) | | | Budget | Builtin/icons... | color(250,21... | | |
| Main Windows/Settings | Administration | Settings | Builtin/icons/16/preferences... | color(255,255,255,255) | color(0,0,0,255) | | | Settings | Builtin/icons... | color(250,21... | | |
| Main Windows/Reports | Administration | Reports | Builtin/icons/16/calculator.png | color(255,255,255,255) | color(0,0,0,255) | | | Reports | Builtin/icons... | color(250,21... | | |

5. You can then add a script that will use the newly added **windowPath** to open the correct window when a user double clicks on a node. Right click on the Tree View component and select **Scripting**.

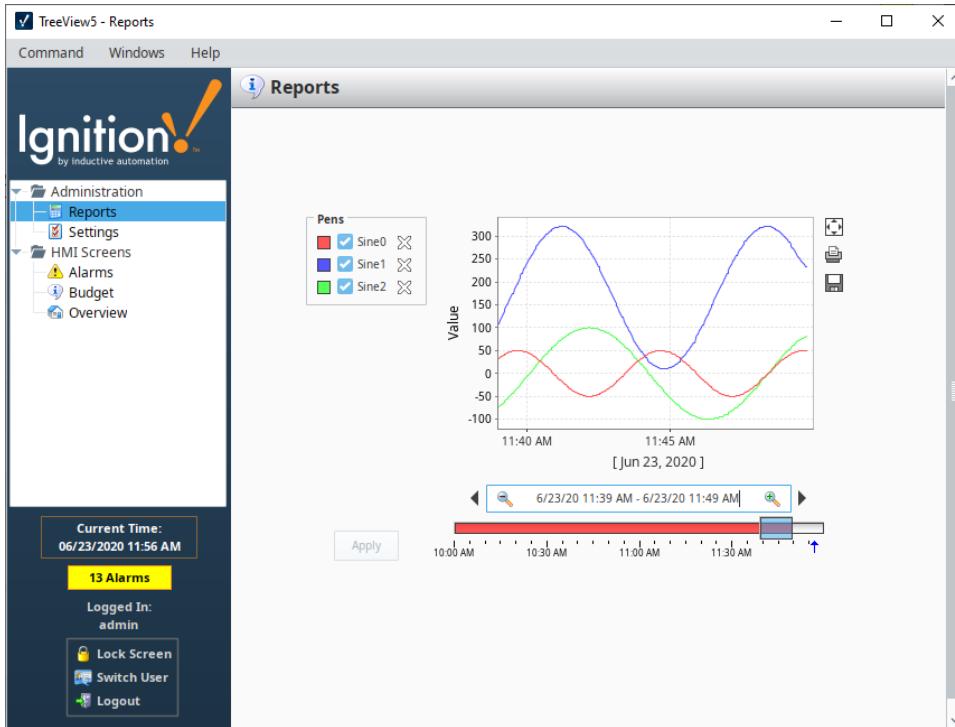
6. Select the **mouseClicked** event handler, and add the following script to the Script Editor tab.

mouseClicked code

```
# If the user performs a double click, open the appropriate window
if event.clickCount == 2:
    row = event.source.selectedItem
    data = event.source.data
    if row != -1:
        # Grab the window path value out of the tree view's items dataset
        windowPath = data.getValueAt(row, "windowPath")
        system.nav.swapTo(windowPath)
```

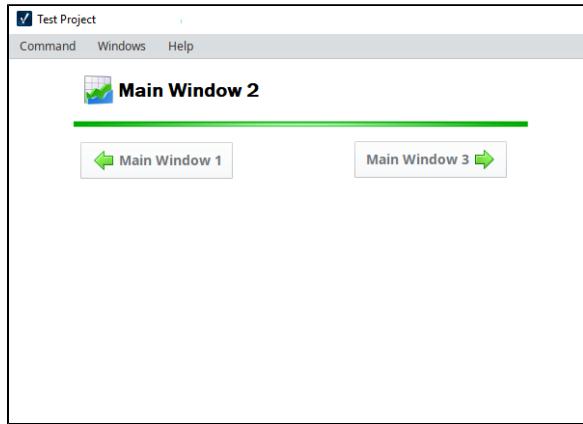


7. Save your project and launch the **Vision Client** and test out the Tree View navigation.



Navigation - Forward and Back Buttons

Another navigation strategy in Vision is to set up Forward and Back Buttons to navigate between different windows. This strategy is perfect for a small business process with ordered steps. It does not have a docked window, tree view, or tabs to navigate around. It is one big main window and has buttons to navigate forward and back from one Main Window to the next in the list.



On this page ...

- [Forward and Back Buttons Example](#)



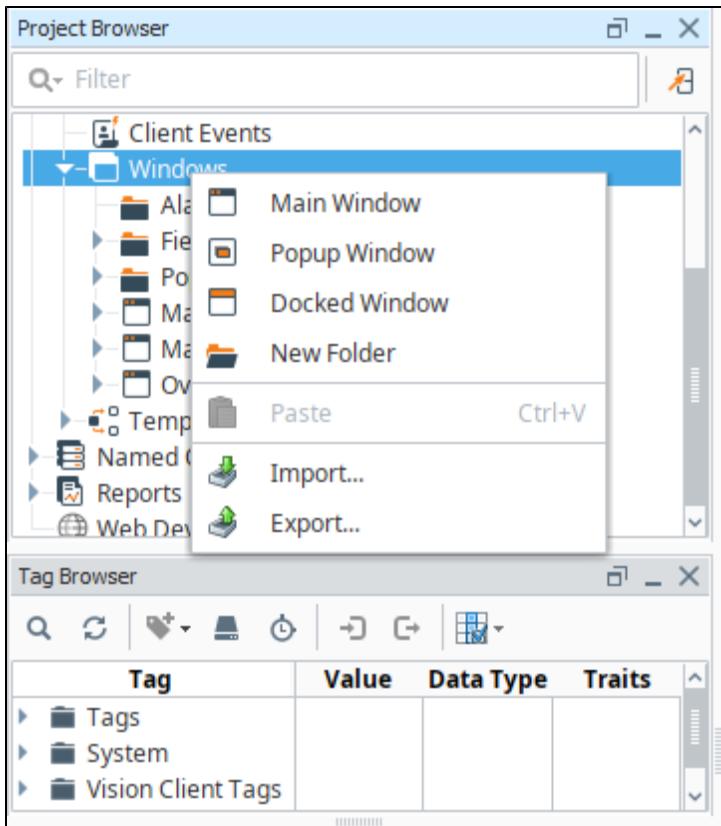
Navigation - Back and Forward Buttons

[Watch the Video](#)

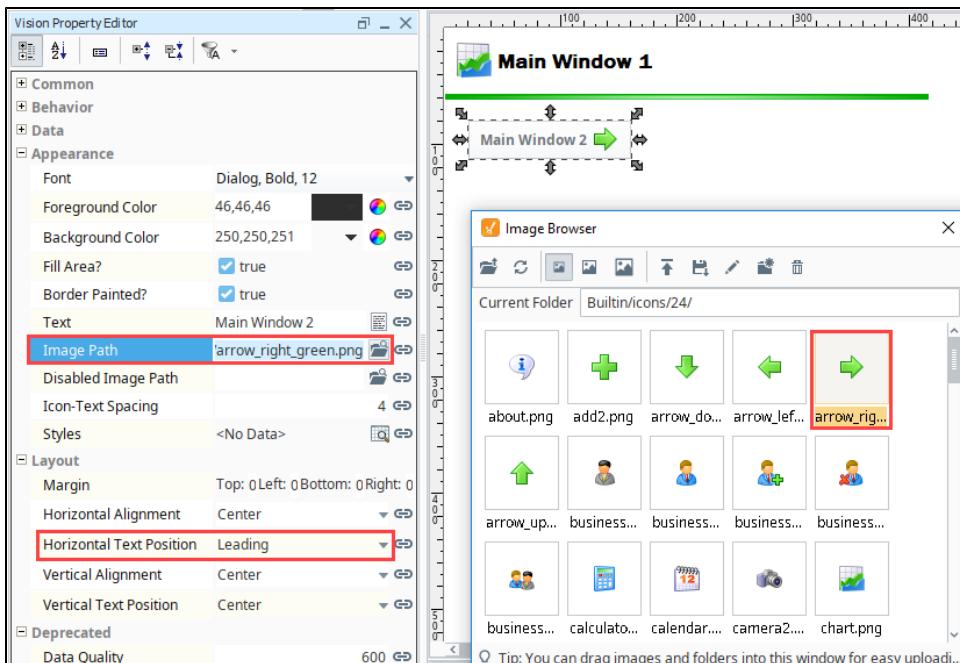
Forward and Back Buttons Example

In this example, we'll use the forward and back buttons on a main window to navigate between different windows in a project.

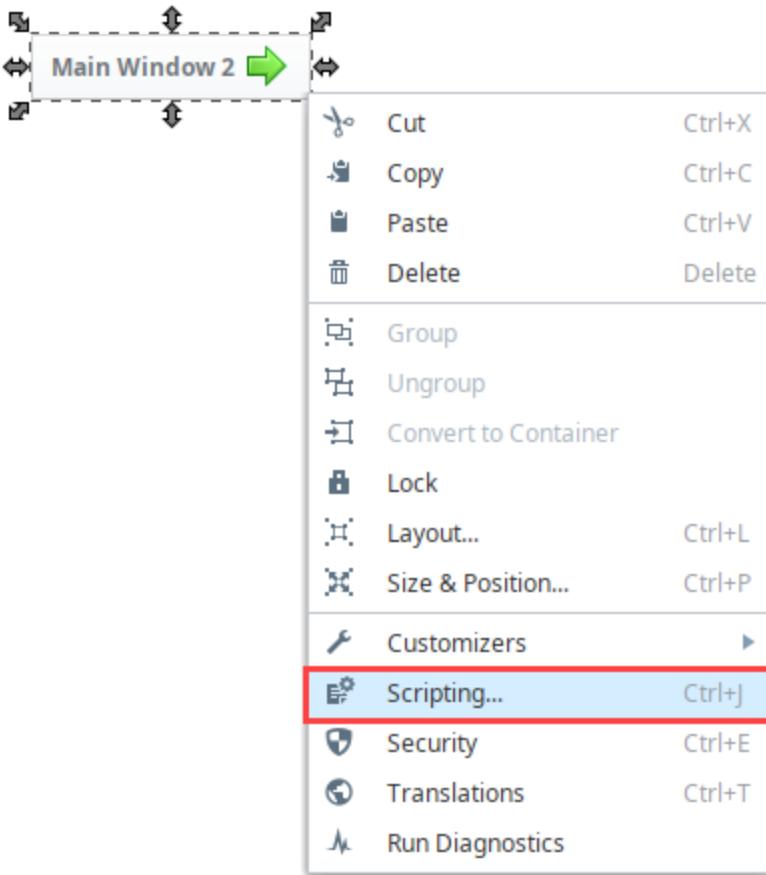
1. In the **Project Browser**, right click on the Main Windows folder and create a new **Main Window**. Enter a name for your window to whatever best describes the window (i.e., Main Window 1). Add a label to the window for clarity (i.e., Main Window 1) so when you navigate through different windows, you know precisely what window you're viewing. Repeat this step to create Main Window 2 and Main Window 3.



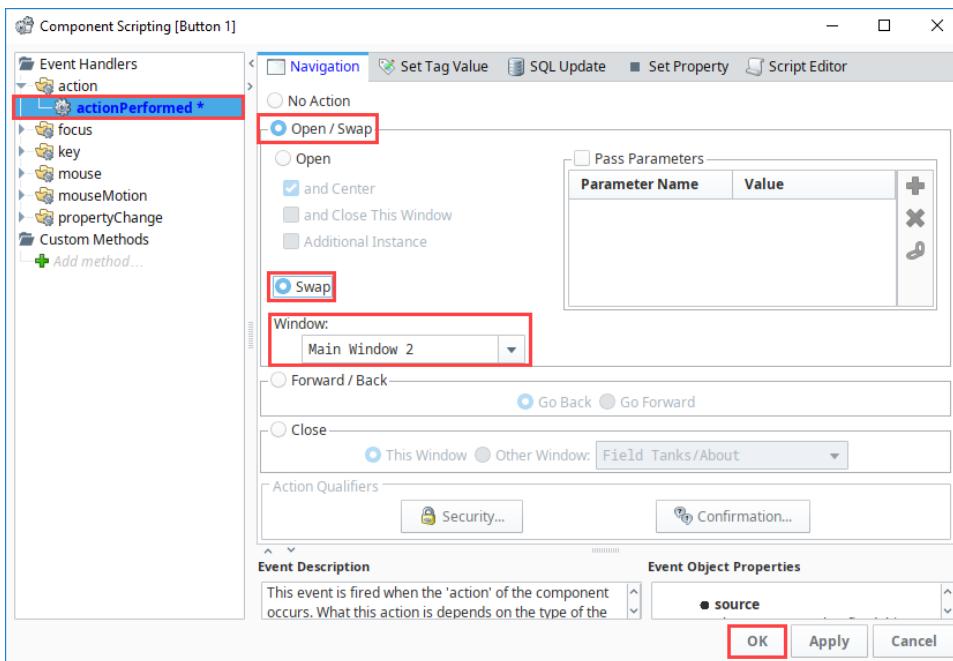
2. From the **Component Palette** in Designer, drag a **Button** component to your Main Window 1. Label the button, **Main Window 2**.
3. Next we'll add a right green arrow to the Button component. With the Button component selected, click the folder to the right of the **Image Path** property . This opens the [Image Management Tool](#).
4. Open the Built-in/icons/24/ folder and select a right green arrow. Close the Image Management Tool.
5. Set the **Horizontal Text Position** property to **Left** in the Property Editor.



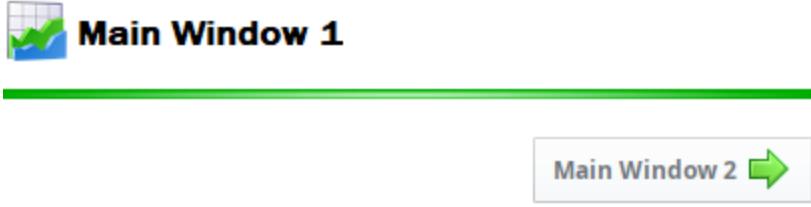
6. Now you need to tell the Main Window 2 button what to do when a user clicks on it. Right click on the **Main Window 2** button, select **Scripting** . The component scripting dialog box will open.



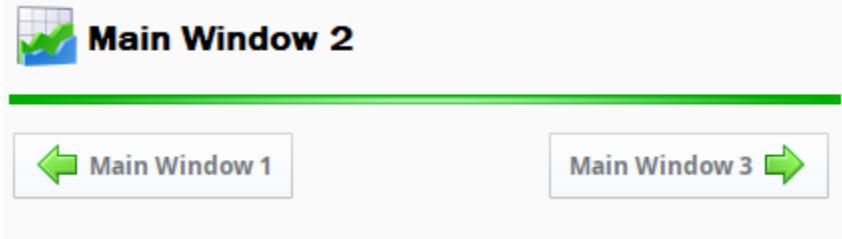
7. Under the Event Handlers, open the **action** folder and select **actionPerformed**.
 - a. Click the **Open / Swap** and **Swap** radio buttons. The Swapping function builds a simple script to go back and forth between different windows. The Swap function ensures only one main window will be open at a time.
 - b. From the **Window dropdown box**, select **Main Windows/Main Window 2**.
 - c. Click **OK**.



8. **Save and Publish** your project.
9. Open your **Client**. Click on the **Main Window 2** button and Main Window 1 will be swapped out with Main Window 2.



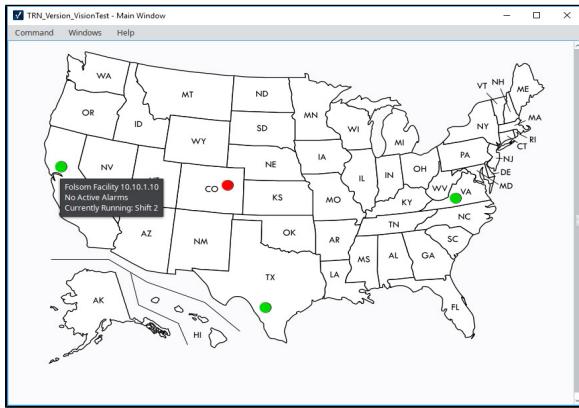
10. Repeat Steps 2 and 3 to create Main Windows 2 and 3, respectively. On Main Window 2 make sure you have a button for both Main Window 1 and Main Window 3. Set your window path for the Main Window 1 button to **Main Windows/Main Window 1** (refer to step 4). For the Main Window 3 button, set the window path to **Main Windows/Main Window 3**.
11. In this image, Main Window 2 is open. By clicking either button, it opens the respective window. Click on the **Main Window 1** button, and Main Window 2 swaps out and opens Main Window 1. Click on the Main Window 3, and Main Window 2 swaps out and opens with Main Window 3. You'll notice that you always have one main window open.



Navigation - Drill Down

Another popular navigation strategy is to drill down into various areas of your project using a map. The Drill Down navigation strategy is ideal if you have different geographical locations. A good example is to have a main window that has an image representing a plant or factory. The image can have any type of drawing tool component, such as an rectangle, circle, etc., that overlays the image. When the user selects one of the overlay components, the Client swaps windows with a window that displays information that pertains to the selected area of the plant.

It is a very simple navigation strategy to setup and is popular because it lets users select different areas on an image and drill down to access specific information about that geographic location.



On this page ...

- [Drill Down Navigation Example](#)



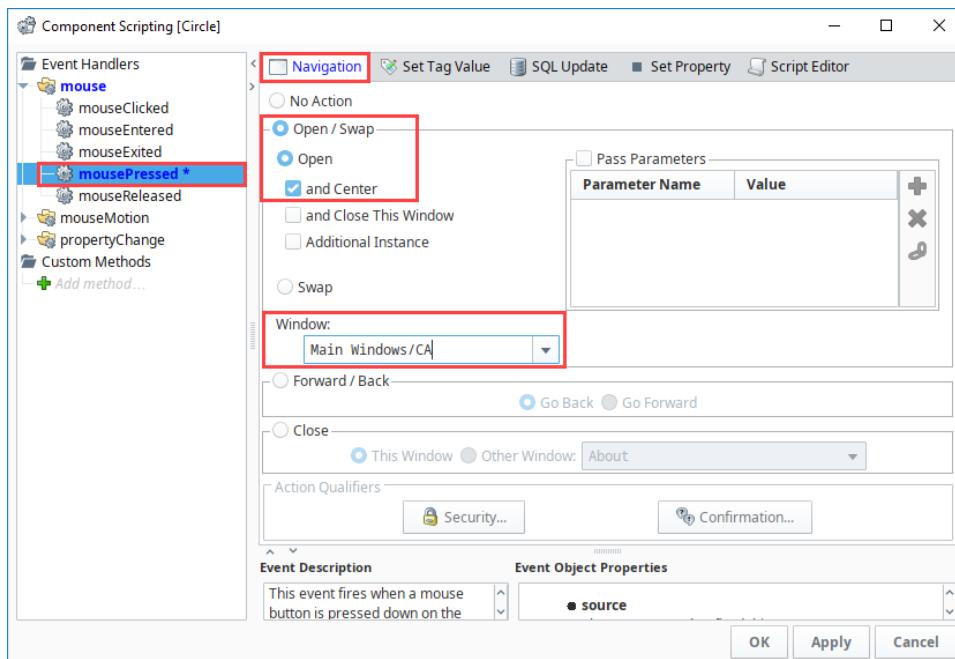
Navigation - Drill Down

[Watch the Video](#)

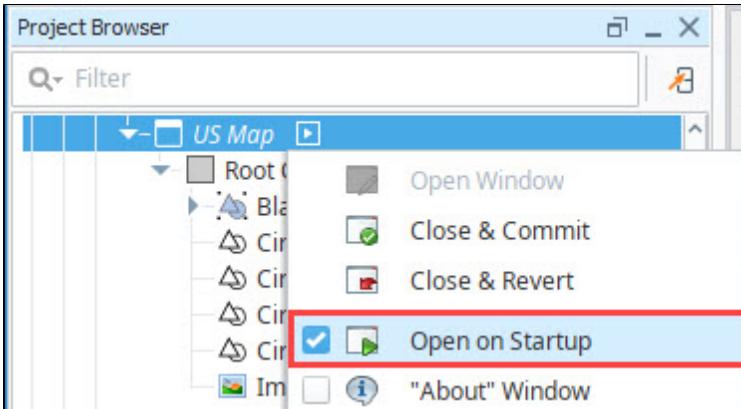
Drill Down Navigation Example

This example demonstrates how to use a US map and configure it to get information about different geographical facilities sprinkled across the US.

1. Drag an **image** on to a Main Window. It can be any type of image including a photo. This example uses an image of a US Map.
2. Add a **Drawing Tool** shape such as a rectangle, circle, polygon over an area on your image to identify the location. In the example, we use different color circles.
3. Right click your drawing tool and select **Scripting**.
4. Select the **mousePressed** event handler, and with the **Navigation tab** selected.
 - a. Select the **Open/Swap** radio button.
 - b. From the **Window** drop-down box, select the window that relates to the selected area on the map.
 - c. Click **OK**.



- In the Project Browser, select your US Map window and set it to **Open on Startup**.



Area windows already created

This example assumes you already have your area windows created in your Project Browser.

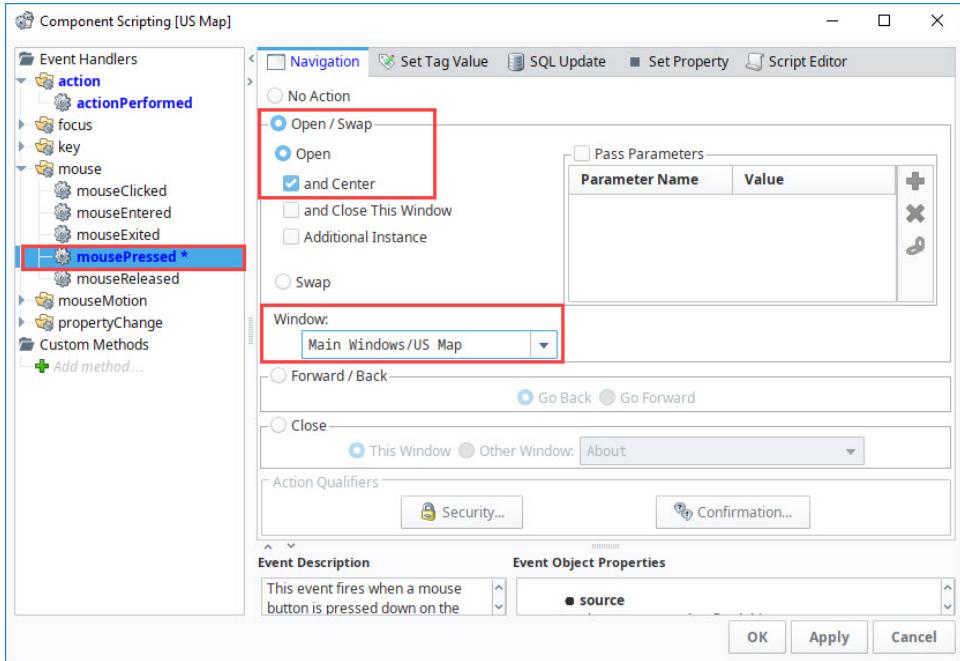
- On your area window, add a **Button** component so you can navigate back to the main window containing the overview map. (If you have multiple area windows, copy and paste this button on to each window).

- Open** your area window.
- Add a **Button** component.
- Right click on the **Button** component and select **Scripting**.

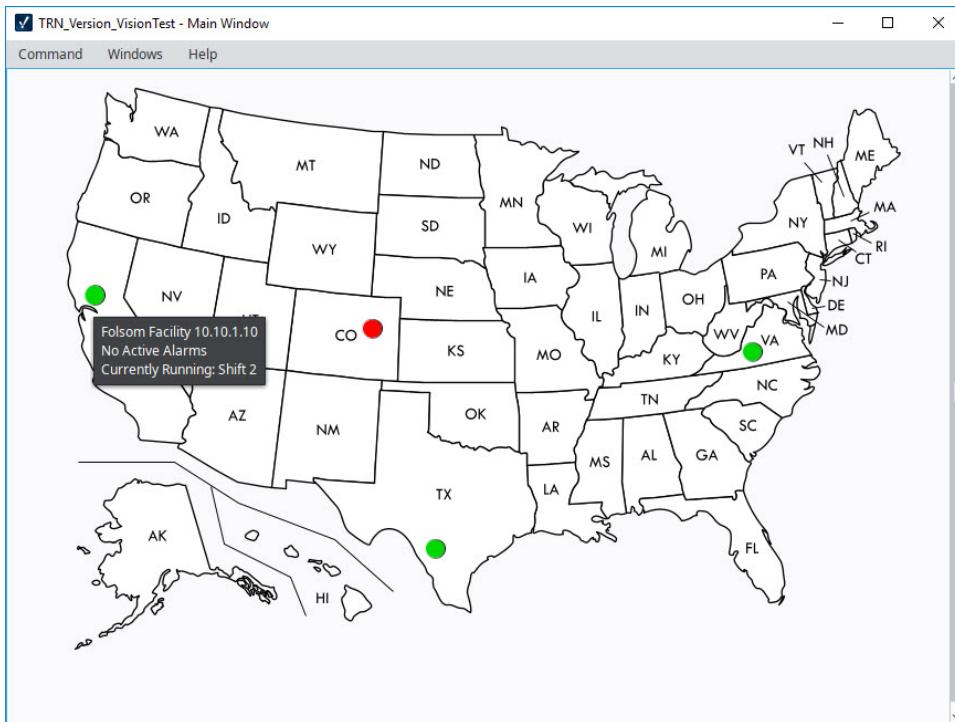


- Select the **mousePressed** event handler, and with the **Navigation tab** selected:

- Select the **Open/Swap** radio button.
- From the **Window** drop-down box, select the window that is your overview map.
- Click **OK**.



8. In **Preview Mode**, test your window navigation by switching between windows.
9. **Save** your project.
10. Now, try it out by opening your **Client**, and clicking on a shape to navigate to the area selected. A new window will open each time you select a designated area on the map. You'll notice that in the following image, the **Mouseover Text** property was used to display the location information when you hover over one of the circles.

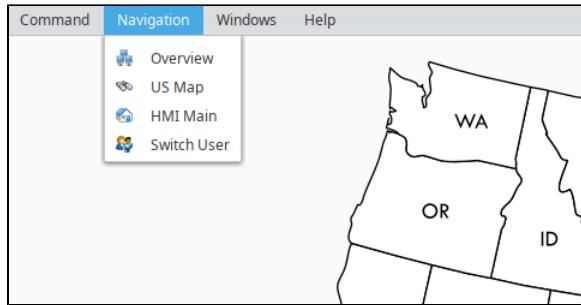


11. Click on the **US Map** button to go back to the US Map.



Navigation - Menubar

You can set up a special menu within the [Menubar](#) that allows you to navigate throughout the project using the scripting functions. They can be simple, like swapping to a window, or be more complex in how they navigate around the project. The benefit of using the Menubar for navigation is that it keeps navigation tucked away instead of using up valuable screen space.



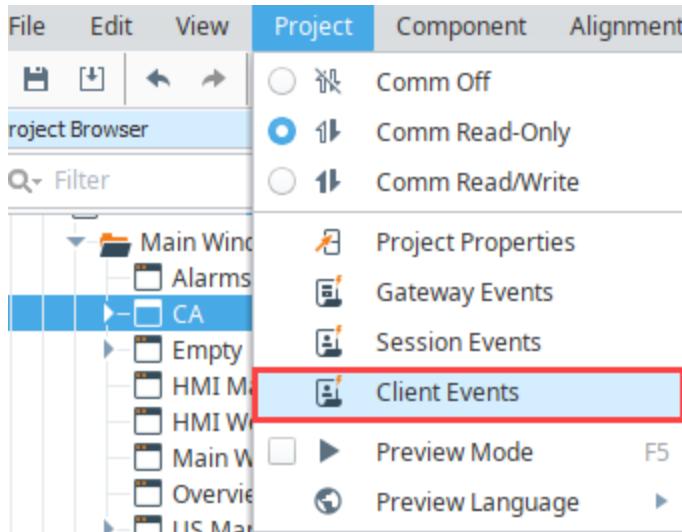
On this page ...

- [Menubar Navigation Example](#)

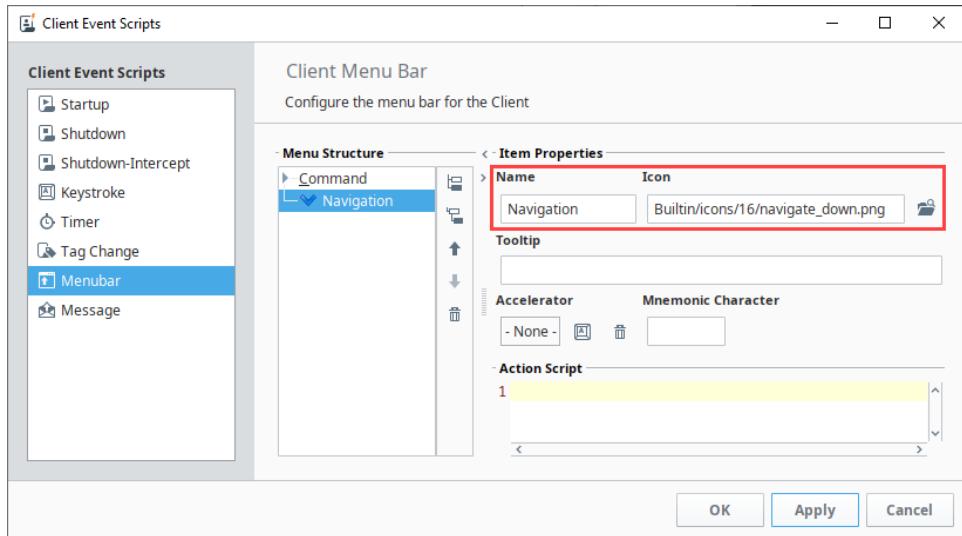
Menubar Navigation Example

In this example, we'll setup a new menu option or sibling on the menubar as well as a child option. This example assumes you have these additional windows created.

1. In the menubar of the Designer, click on **Project** then select **Client Events**.



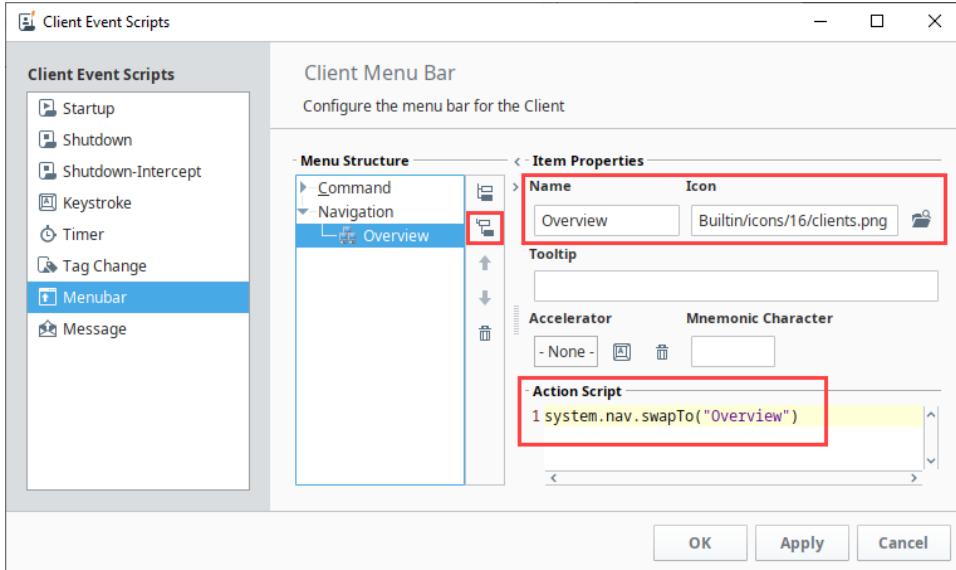
2. This opens the screen below in the [Client Event Scripts](#) space. Click on the **Menubar** under Client Event Scripts.
3. Select the **Add Sibling** icon to add a Menu Item. Update the Name to a new menu option (i.e., Navigation). You can also add a path to an icon if desired.



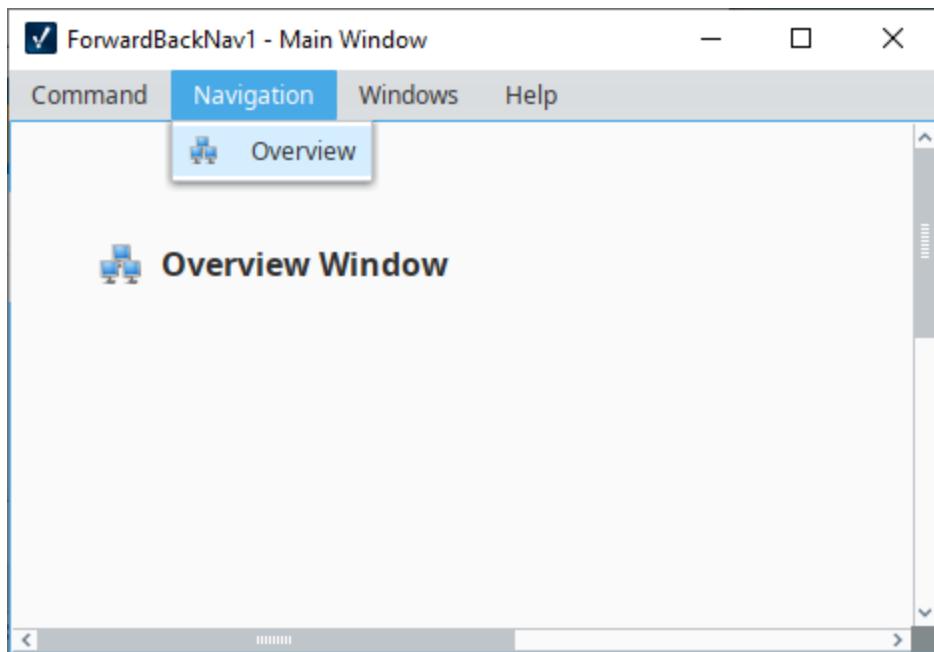
4. Click the **Apply** button.
5. Click the **Add Child** icon to add a new option under the Navigation menu.
 - a. Give the menu item a name that is appropriate for the window it will be navigating to (i.e., Overview).
 - b. Add a script that will swap to the window.

Code Snippet - Menubar navigation

```
system.nav.swapTo( "Overview" )
```



6. Repeat step 5 to add as many windows as needed. New groups of windows can even be nested within the parent Navigation Menu.
7. Click **OK** to save your new menu structure.
8. **Save** your project.
9. Now, open the **Client** to navigate from one window to another using the menubar structure. Notice that Navigation menu option is located in the menubar because it was created as sibling. All the other menu options were created as children under Navigation.
10. By default, there are three commands under the Command menu option: Logout, Lock Screen, and Exit.



Navigation - Retargeting

Retargeting is a special form of navigation which involves navigating to an entirely different project.

Retargeting is accomplished through scripting, usually as a response to a button press or other component event. The [system.util.rettarget\(\)](#) function allows you to 'retarget' the Client to a different project. You can have it switch to another project on the same Gateway, or another Gateway entirely, even across a WAN. This feature makes the vision of a seamless, enterprise-wide SCADA application a reality.

The retarget feature will attempt to transfer the current user credentials over to the new project / Gateway. If the credentials fail on that project, the user will be prompted for a valid username and password. Once valid authentication has been achieved, the currently running project is shut down, and the new project is loaded.

You can pass any information to the other project through the parameters dictionary. All entries in this dictionary will be set in the global scripting namespace in the other project. Even if you don't specify any parameters, the system will set the variable `_RETARGET_FROM_PROJECT` to the name of the current project and `_RETARGET_FROM_Gateway` to the address of the current Gateway.

Retargeting can be as simple as 1 line of code, just make sure you are using the project name (no spaces allowed), and not the title. See the [retarget](#) function in the appendix for more information.

A typical retargeting strategy actually combines this strategy with one or more other navigation strategies. A simple landing project could be made that all users would have access to, which would allow users to do some basic user management functions, as well as a screen with a button that retargets out to other projects, with each project having a specific purpose, or targeting a specific area of operations. The buttons that retarget to these projects can be hidden or shown based on the user, allowing you to build in an extra layer of security to your projects. Additionally, each of the other projects would utilize navigation strategies that best suit those areas.

On this page ...

- [Retargeting Navigation Example](#)



INDUCTIVE
UNIVERSITY

Retargeting

[Watch the Video](#)

Retargeting Navigation Example

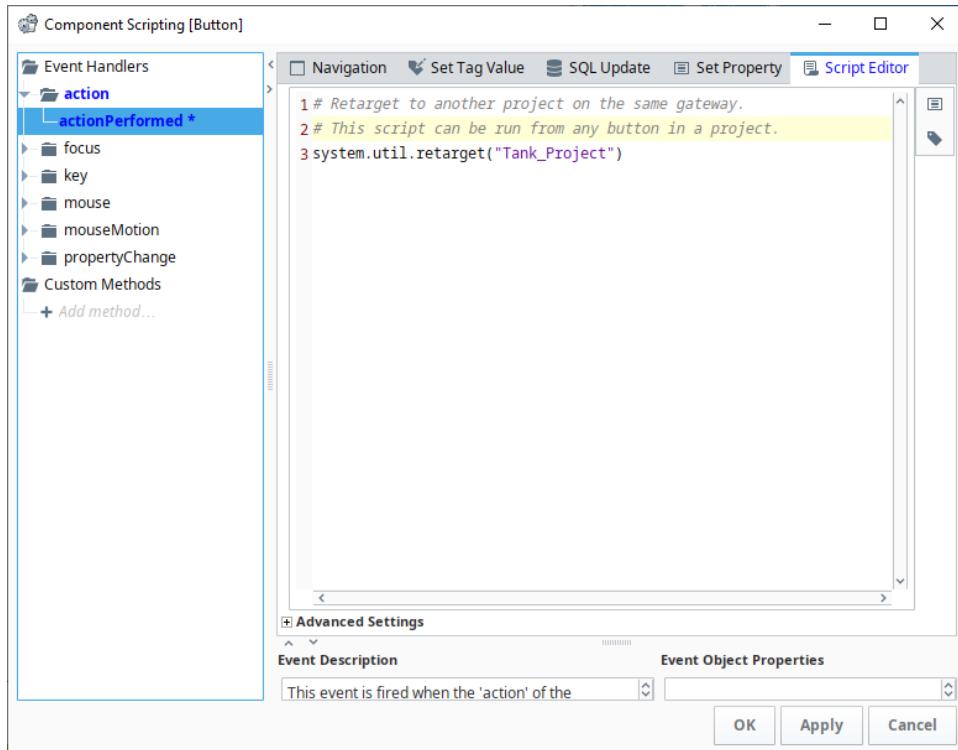
In this example, we have two projects that we will setup to retarget to each other.

1. On a blank window, add a **Button** component, and either change the button text or add a label that informs the user that the button retargets to a new project.
2. Right click on the Button and select the **Scripting** option. On the **actionPerformed** event, add this script to the script builder, and click **OK**.

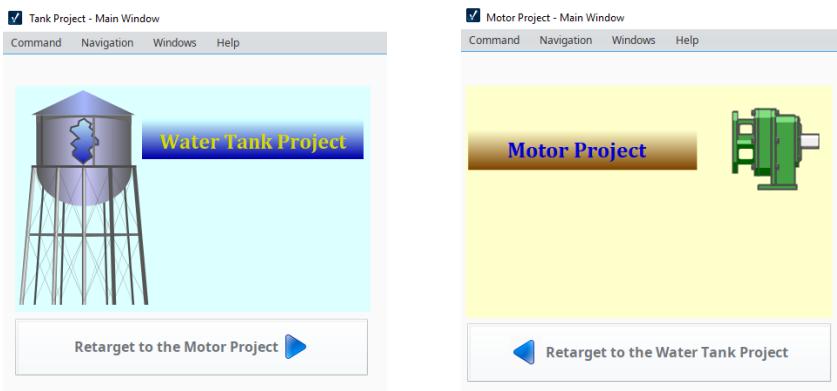
Python - Retargeting to Another Project

```
# Retarget to another project on the same gateway.  
# This script can be run from any button in a project.  
system.util.rettarget("My_Other_Project")
```

The script below was added to the button in the **Tank** project. A similar script was also added to the **Motor** project.

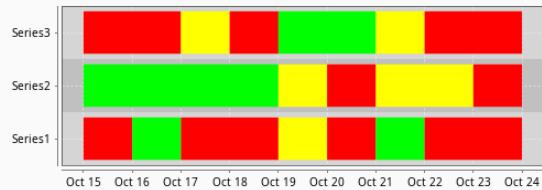
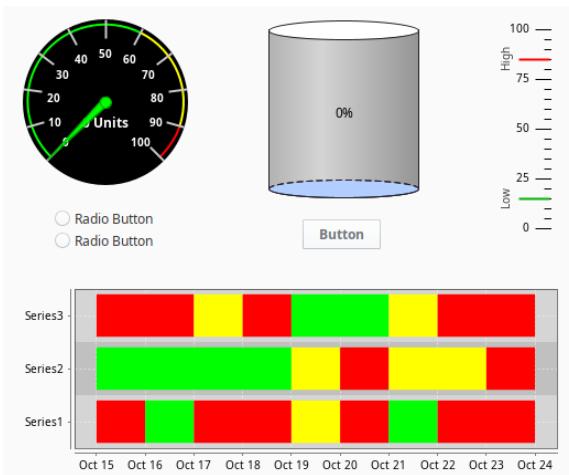


3. This can be repeated with more buttons and scripts that lead to other projects. **Save** your projects.
4. Open the **Client**, and use the retargeting feature to navigate between two different projects. Press the **Retarget** buttons to navigate between the **Tank** project and **Motor** project.



Working with Vision Components

The Vision Module comes with a host of built-in components that you can select from for use in your project. One thing that you'll find when working with components is there are a few different ways to manipulate and layout components on a window when working in the Designer. Here is a small sampling of the components available in Vision. See the [Vision Components](#) Appendix page for a complete list of components.



This section introduces you to how to work with components so you can learn how to quickly select, move, resize, duplicate, and group components during the design process.

On this page ...

- Selecting Components
 - Mouse Selection
 - Tree Selection
- Component Properties
 - Data Types
 - Dataset Editor
- Manipulating Components
 - Resizing
 - Moving
 - Duplicating
 - Rotating
 - Size and Position
 - Component Grouping
 - Component Layout
 - Relative Layout
 - Font Scaling
 - Anchored Layout



Component Overview

[Watch the Video](#)

Selecting Components

There are a number of different ways to select components within a window, each of which have their own advantages.

Mouse Selection

Using the mouse is the most common way to select components. Click the Selection  icon, then click on a component to select it. If the component you want to select is obscured by other components, hold down **Alt** and keep clicking, the selection will step down through the z-order.

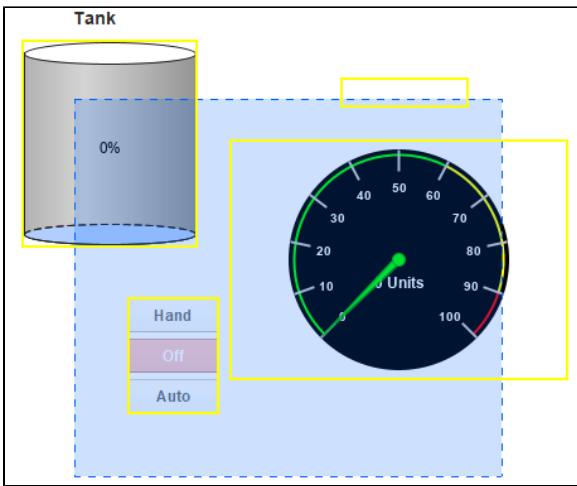
You can also select components using window-selection. Click-and-drag within a container to draw a selection rectangle. If you drag the window left-to-right, it will select all components that are completely contained within the rectangle. If you drag the window right-to-left, it will select all components that the rectangle touches.

Lastly, you can start dragging a window selection and then hold down the **Alt** key to use touch selection. This will draw a line as you drag, and any components that the line touches will become selected. As you're using these techniques, components are given a yellow highlight border.



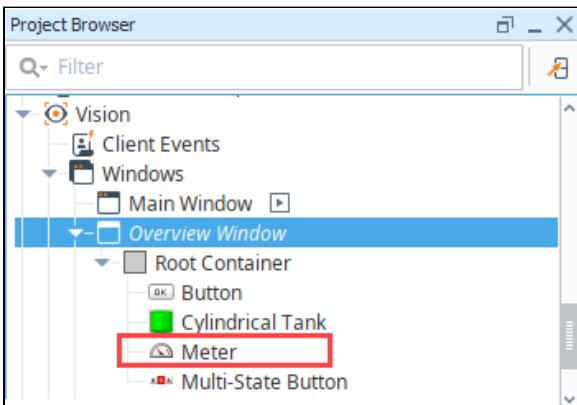
Selecting Components

[Watch the Video](#)



Tree Selection

By selecting nodes in the **Project Browser** you can manipulate the current selection. This is a handy way to select the current window itself, which is hard to click on since it is behind the Root container. However, you can click to it, using **Alt-click** to step down through the z-order. It is also the only way to select components that are invisible.



Component Properties

Each component has a unique set of properties that can be set and modified within the **Property Editor**. A property is simply a named variable with a distinct type that affects something about the component's behavior or appearance. You can also create your own **custom properties** on the component, which act like variables that can store any information that you want on the component.

Data Types

There are a wide variety of datatypes across all of the Vision Module's components. Each property has a distinct type, which dictate what values will be allowed. Below are the common data types.



Component Properties

[Watch the Video](#)

Numeric Types

| | |
|-------------|--|
| Bool ean | A true/false value. Modeled as 0/1 in Python. Technically, 0 is false and anything else is true. |
| Short | A 16-bit signed integer. Can hold values between -32,768 to 32,767, inclusive. |

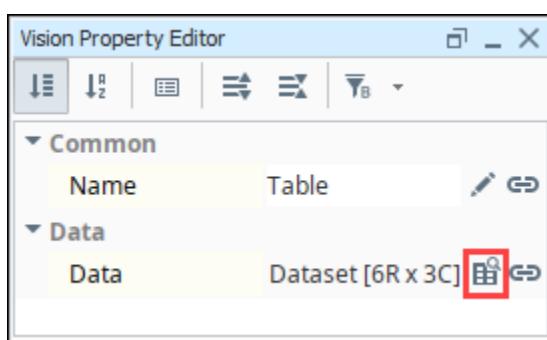
| | |
|-------------|---|
| Integer/int | A 32-bit signed integer. Can hold values between -2,147,483,648 to 2,147,483,647 inclusive. |
| Long | A 64-bit signed integer. Can hold values between -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 inclusive. |
| Float | A 32-bit signed floating point number in IEEE 754 format. |
| Double | A 64-bit signed floating point number in IEEE 754 format. |

Non-Numeric Types

| | |
|---------|---|
| String | A string of characters. Uses UTF-16 format internally to represent the characters. |
| Color | A color, in the RGBA color space. Colors can easily be made dynamic or animated using Property Bindings or Styles. |
| Date | Represents a point in time with millisecond precision. Internally stored as the number of milliseconds that have passed since the "epoch", Jan 1st 1970, 00:00:00 UTC. |
| Dataset | A complex data structure that closely mimics the structure of a database table. A Dataset is a two-dimensional matrix (also known as a table) of data organized in columns and rows. Each column has a name and a datatype. |
| Font | A typeface. Each typeface has a name, size, and style. |
| Border | A component border is a visual decoration around the component's edges. You can make a border dynamic by using the Style Customizer , the <code>toBorder()</code> expression function, or scripting with the Java border object . |

Dataset Editor

The Dataset Editor icon  appears next to the binding icon for the Data property. Clicking the icon brings up the Dataset Editor window where you can view and make changes to the raw data. Note, that any changes will be overwritten the next time your binding polls.



With the Dataset Editor you can add and delete columns and rows, delete all rows, and copy information to or from the clipboard. When adding columns you have multiple formats to choose from: string, date, integer, double, float, etc.

Dataset Editor

| Date | Value1 | Value2 |
|--------------------|--------|--------|
| 5/7/20, 4:26:17 PM | 67 | 109 |
| 5/7/20, 4:26:45 PM | 23 | 85 |
| 5/7/20, 4:26:46 PM | 15 | 77 |
| 5/7/20, 4:26:47 PM | 41 | 62 |
| 5/7/20, 4:26:47 PM | 18 | 34 |
| 5/7/20, 4:26:47 PM | 26 | 79 |

Column Name: ---- Column Type: ----

OK Cancel

The Dataset Editor icons and their corresponding actions are shown in the table below.

| Icon | Action |
|------|------------------------|
| | Add row |
| | Delete selected rows |
| | Add a column |
| | Delete selected column |
| | Delete all rows |
| | Add to clipboard |
| | Paste from clipboard |

Manipulating Components

Manipulating components can be done with both the mouse and the keyboard. You can move components around, resize them, and rotate them.

Resizing

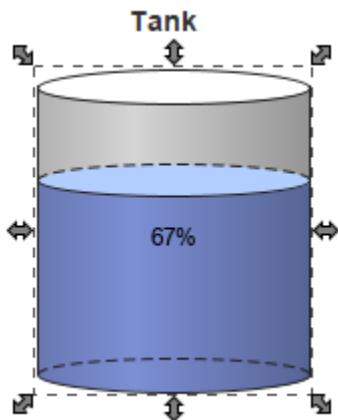
When you select the component you want to resize, they'll get eight resize-handles displayed around the edge of the selection. These handles look like double-sided arrows around the perimeter. Use the mouse to drag them to change the size of the components in the selection. To maintain the selection's aspect ratio, hold down **Ctrl** as you resize. To resize around the center of the current selection, hold down **Shift**. These can be used at the same time.



Manipulating Components

[Watch the Video](#)

You can also resize the current selection using the keyboard. To nudge the right or bottom edge of the selection in or out, use **Shift** combined with the arrow keys, which resizes by the nudge distance, which defaults to one pixel at a time. To nudge the top or left edge of the selection, use **Ctrl-Shift** combined with arrow keys. To resize faster, hold the **Alt** key as well, to move the component the alt nudge distance, which defaults to ten pixels at a time.



Moving

To move the component, simply drag it anywhere within the container's bounds. You can also move whatever is currently selected by holding down **Alt** while dragging, regardless of whether or not the mouse is over the current selection. This is important because it is the primary way to move a Container component. (Normally, dragging in a container draws a selection rectangle inside that container).

While a component is selected, you may also use the keyboard's arrow keys to move a component around the nudge distance. Just like resizing with the arrow keys, to move the alt nudge distance, use the **Alt** key.

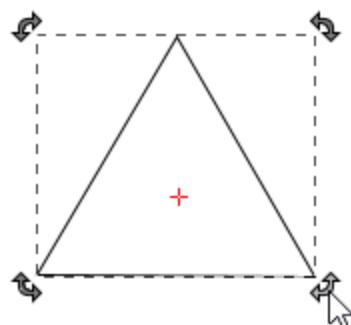
Duplicating

Components can be easily duplicated by dragging them as if you were going to move them and holding down the **Ctrl** key. This will drop a copy of the component at the desired drop location. It is often useful to also hold down **Shift** key as you do this to ensure exact alignment. You may also use the **Ctrl-D** shortcut to quickly duplicate a component in place.

Rotating

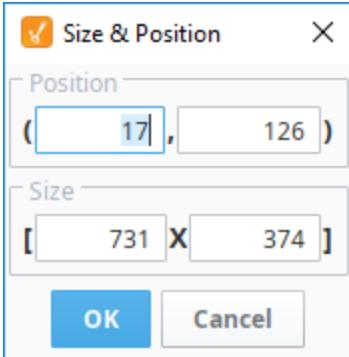
Shapes can be rotated directly using the selection tool. Other components cannot be rotated in this manner. To rotate a shape, first select it using the selection tool so that you see the resize handles around it. Then simply click on it once again and you'll see the rotation handles appear. Clicking (but not double-clicking) on selected shapes toggles back and forth between the resize handles and the rotation handles.

Once you see the rotation handles, simply start dragging one to rotate the shape or shapes. Holding down the **Ctrl** key will snap your rotation movements to 15° increments. When the rotation handles are present, there is also a small red crosshair handle that starts in the middle of the selection. This is the rotation anchor: the point that the selection will rotate around. You can drag it anywhere you'd like to rotate around a point other than the center of the shape.



Size and Position

Components can also be positioned and resized with the Size and Position window. This window allows you to type in an exact pixel size of the component as well as x/y coordinates that the component will be at (with the upper left point of the component moving to that point). To access the size and position window, right click on the component and select Size and Position.



Component Grouping

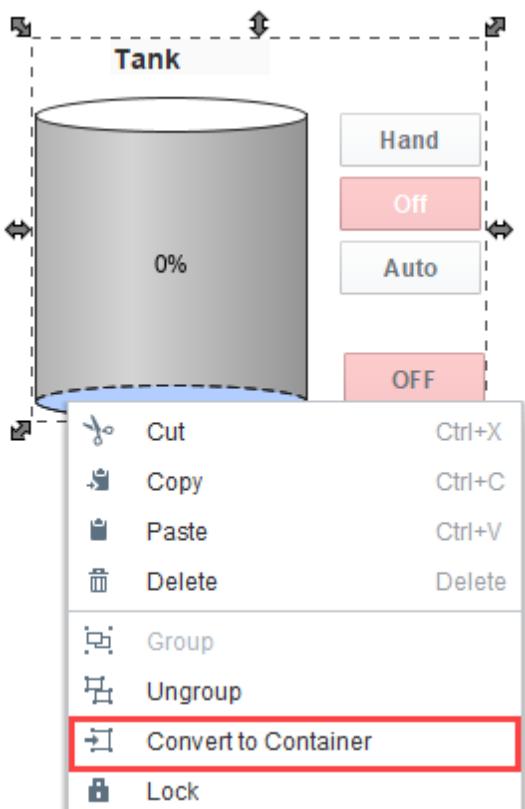
Shapes and components can be grouped together so that they act like a single component in the Designer. Grouping components is very similar to putting them in a Container. In fact, it is the same thing as cutting and pasting them into a perfectly-sized container and then putting that container into group mode, with one exception. If the group contains only shapes and no other kinds of components, it will be a special shape-group that has the ability to be rotated and has some other shape-like properties.

When components or shapes are in a group, clicking on them in the Designer will select the group instead of the shape. If you double-click on a group, it will become "super-selected", which will allow you to interact with its contents until you select something outside of that group.

Groups can contain other groups, creating a nested structure. Groups themselves are also components, meaning that you can add custom properties to groups, bind them, and so on.

Difference between a Container and a Group

It is helpful to use Groups and Containers to organize the components on your window. You can select multiple components and right-click to convert them into a group, then right-click again to convert that group to a container.



Layout works differently for groups. The layout setting for components and shapes inside a group is ignored. All members of a group act as if they are in relative layout with no aspect ratio restrictions. This special group-layout mode is also active when resizing a group inside of the Designer, whereas traditional (container) layout doesn't take effect in the Designer.



Component Grouping

[Watch the Video](#)

Component Layout

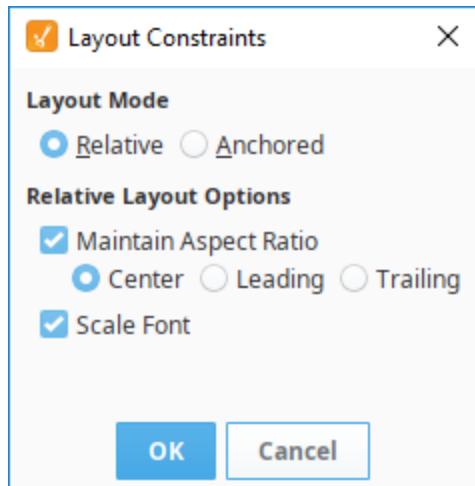
Layout is the concept that a component's size and position, relative to its parent container's size and position, can be dynamic. This allows the creation of windows that resize gracefully using either **Anchored** or **Relative** layouts and can optionally keep the original aspect ratio.

This is a very important concept because of the web-launched deployment of Vision clients - they often end up being launched on many different monitors with many different resolutions.

This is also important for components that have user-adjustable windows like popup windows. Imagine a popup window that is mostly displaying a large table or chart. If you're running on a large monitor, you may want to make the window bigger to see the table or chart easier. Of course, this is only useful if the table or chart actually gets larger with the window.

Changing a component's layout is as simple as right-clicking on the component and opening the Layout dialog box. You can also alter the default layout mode that gets assigned to new components. See [Designer/Window Editing Properties](#).

There are two layout modes, and they are set on a per-component basis. Both affect the component's size and position relative to its parent container. The root container's size is dictated by the window size. To edit the layout of a component, right-click on the component and select **Layout** from the menu. the Layout Constraints window displays showing all the default settings. These default settings can be altered in the Project Properties.



Layout Modes

- **Relative**

This mode makes a component's size and location relative to its parent's size and location. When the parent changes size, the component changes accordingly. This creates components that auto-scale.

- **Anchored**

This mode makes the edge of a component's two axes (horizontal and vertical) anchored to the edge or edges of its parent.

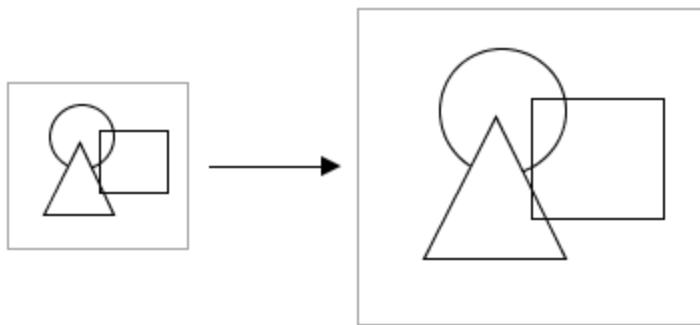
Relative Layout

Relative Layout is the default mode. This is a simple and effective layout mode that simply keeps a component's size and position constant relative to its parent container, even when the parent container grows or shrinks. More precisely, it remembers the component's position and size as a percentage of its parent's bounds at the last time the window was saved. Relative Layout also has the option of scaling a component's font appropriately.

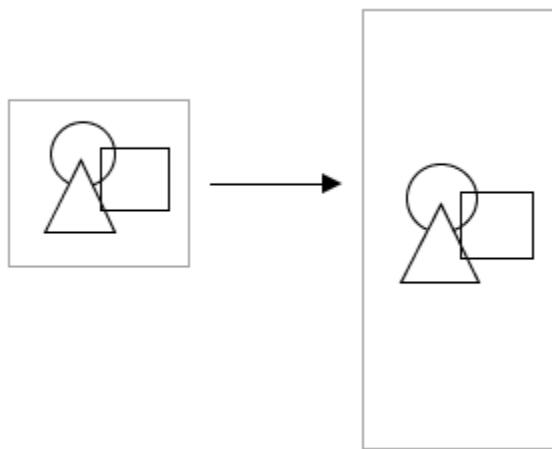


Component Layout

[Watch the Video](#)



Note, that Relative Layout mode respects aspect ratio. So if the parent component is distorted, the contents will not be. The extra space is distributed evenly on both sides of the contents.



Relative Layout Options

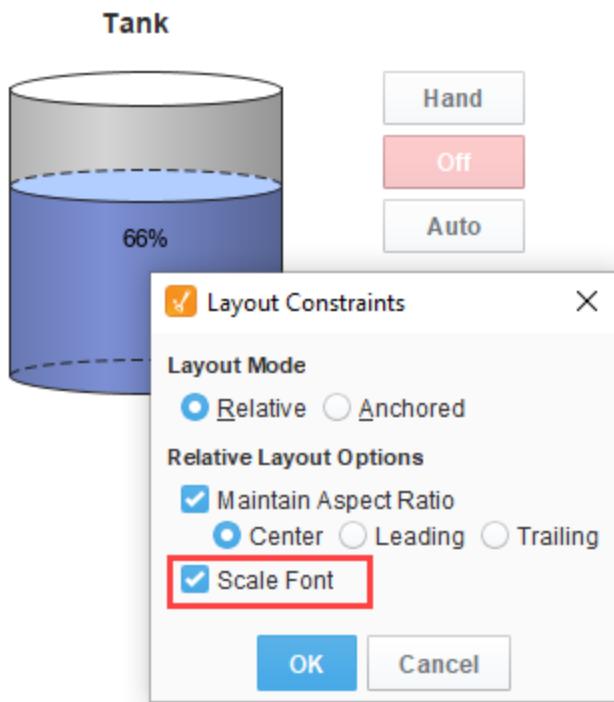
- **Maintain Aspect Ratio**
If selected, the component's original aspect ratio is preserved. Otherwise, it can stretch tall or wide.
- **Center**
When maintaining the aspect ratio, centers the component with respect to its parent.
- **Leading**
When maintaining the aspect ratio, aligns the component with the paper leading edge.
- **Trailing**
When maintaining the aspect ratio, aligns the component with the parent's trailing edge.
- **Scale Font**
If selected, the component's font will scale along with its size as the relative layout adjusts the component. This will override other font size settings. If this setting is applied to a Group, then all components in the group will use this setting.

Font Scaling

By default, font scaling is enabled on all components, but it can behave differently on some components so it's good to test it out before putting it into production. You can change the default Component Layout settings in **Project Properties** under **Vision > Design > Relative Layout Options**.

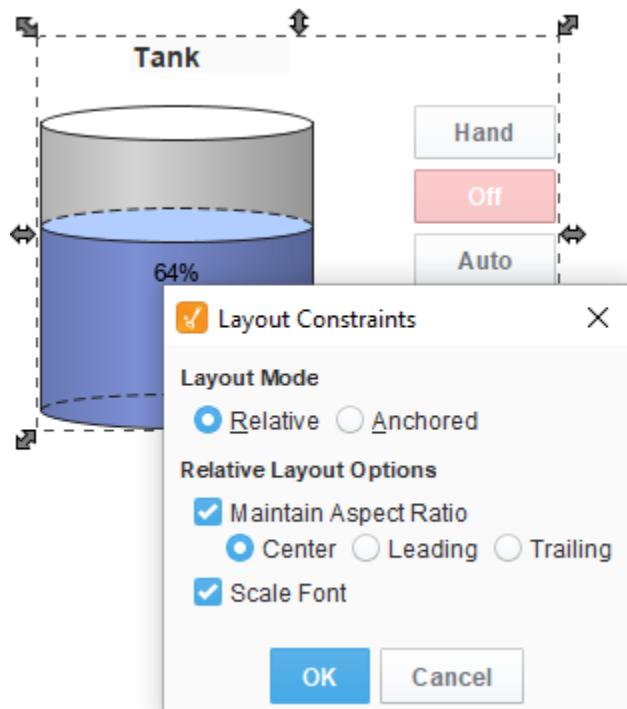
Font Scaling on Individual Components

When individual components are dragged into a window, the following default Layout settings for each component are applied. All components on the window default to font scaling.

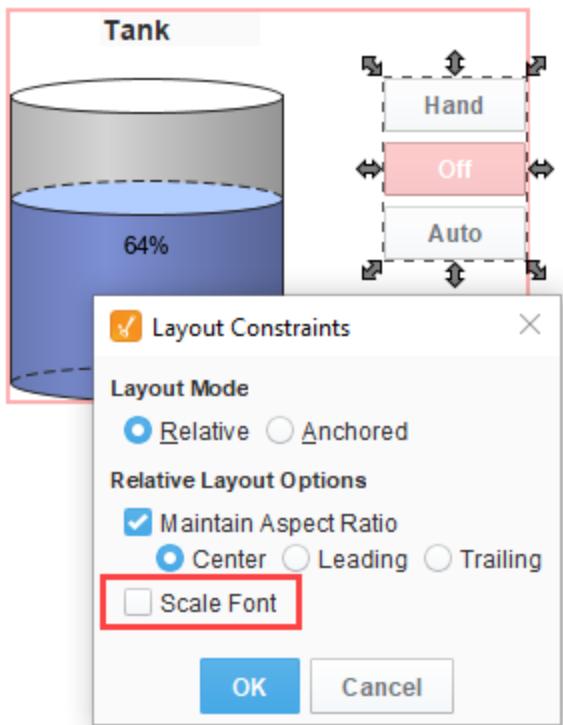


Font Scaling on Groups

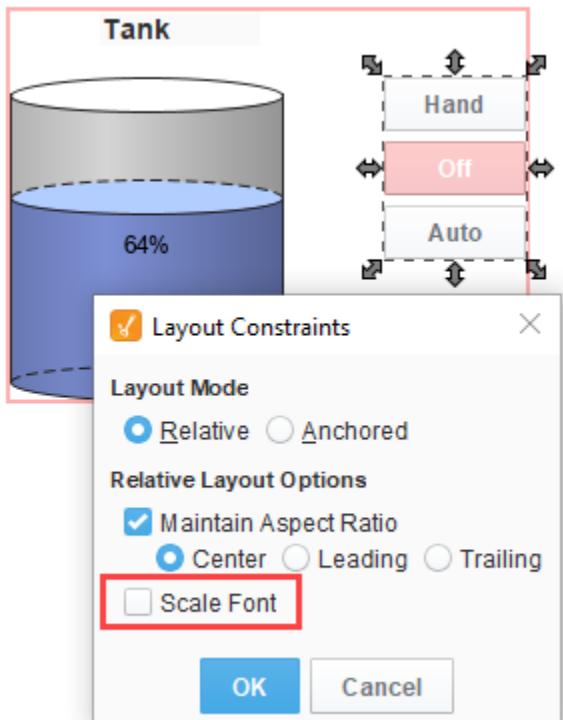
When individual components are grouped together, the default Layout settings are applied to the new group component, and to each component in the group. All the components in the group default to font scaling.



Font scaling on individual components in a group can be disabled by selecting the group, and double clicking on a single component. You will get a red outline around the group, then you can select the individual component(s), and disable font scaling. **Note:** The individual component font scale setting takes precedence within a group.



When you remove the group (right-click and select **Ungroup**), all the individual components within that group will get reset to the default Layout settings including font scaling, even if font scaling was set differently.



Font Scaling on Containers

You can convert a group to a container to change the way scaling works. Ignition remembers the last Layout Settings you made to each individual component in that group prior to the conversion to a container (i.e., if any of the components had font scaling disabled, once the group is converted to a container, those same components will still have font scaling disabled).

Anchored Layout

Anchored Layout lets you specify various "anchors" for the component. The anchors dictate how far each of the 4 edges of the component stay from their corresponding edges in the parent container.

Anchored Layout Options

- **North/South**

If one of these is selected, the distance between that edge of the component and that edge of the container is preserved. If both are selected, the component will stretch its **height** to maintain both distances.

- **West/East**

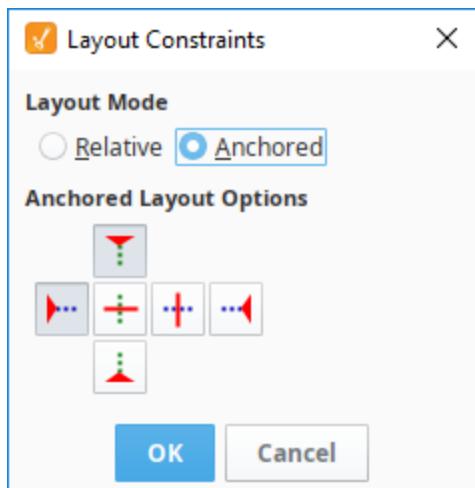
If one of these is selected, the distance between that edge of the component and that edge of the container is preserved. If both are selected, the component will stretch its **width** to maintain both distances.

- **Center Vertically**

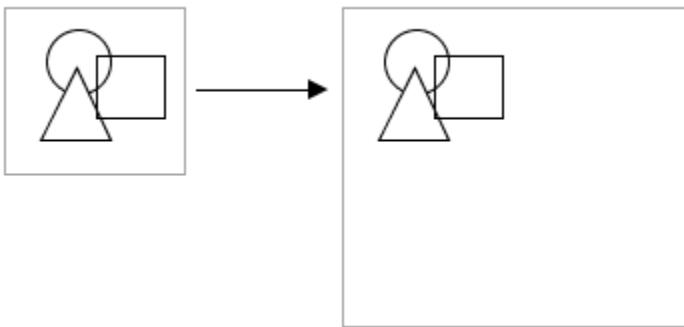
When selected, both top and bottom buttons will be deselected. This option maintains the height of the component and centers it **vertically** in the container.

- **Center Horizontally**

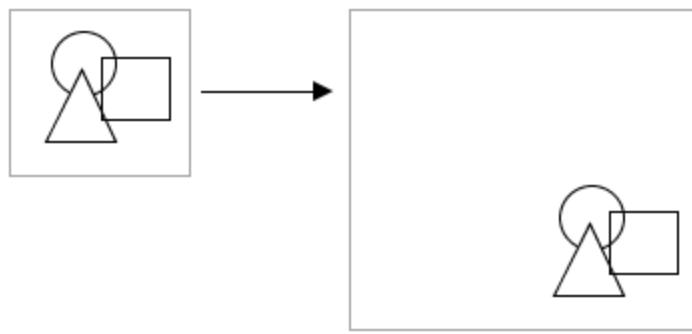
When selected, both left and right buttons will be deselected. This option maintains the width of the component and centers it **horizontally** in the container.



For example, if you anchor top and left, then your component will stay a constant distance from top and left edges of its parent. Since you didn't specify an anchor for the right or bottom sides, they won't be affected by the layout.

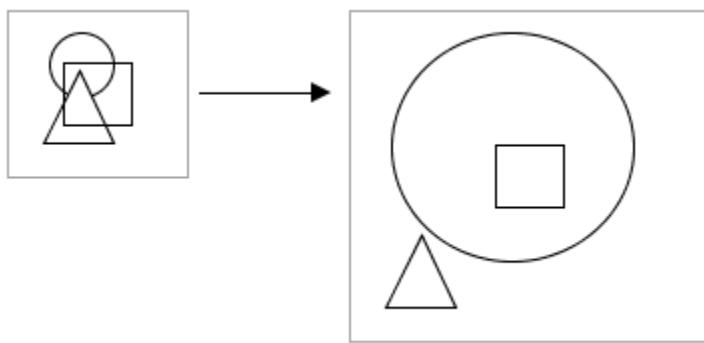


If you anchor bottom and right instead, the components will again stay the same size (since you didn't specify an anchor for their other edges, but they will stay a constant distance from their parent's right and bottom edges).



Of course, you can mix and match the various modes for the different components in a given container. The following image shows the following:

- The **square** uses a horizontal and vertical centering anchor. It is centered, and stays the same size.
- The **triangle** is anchored south and west.
- The **circle** is anchored north, west, south, and east. This means that its edges are all anchored and stay a fixed distance to each of its parent's edges, so it grows.



[In This Section ...](#)

Creating Vision Components

Adding Components to a Window

There are four primary methods for adding Vision components to a window:

1. Select the component in the palette, and then clicking and dragging on the window.
2. Drag a component's icon from a palette onto a container.
3. Drag one or more Tags onto a window.
4. Adding shapes using the drawing tools or SVGs.

The Component Palette

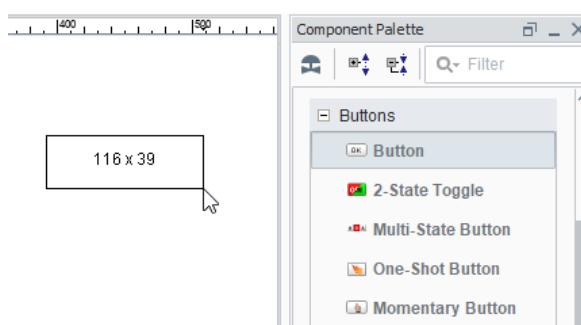
There are two styles of component palette in Ignition Vision: the tabbed palette and the collapsible palette. These palettes work in the same way, but the tabbed palette docks to the north or south edge of the [workspace](#), and the collapsible palette docks to the east or west edge. By default, the collapsible palette is visible in the window workspace. To switch palettes, navigate to the **View > Panels** menu, and select either **Component Palette - Tabbed Palette** or **Component Palette - Collapsible Palette**.

On this page ...

- [Adding Components to a Window](#)
 - [The Component Palette](#)
 - [Creating Components Using Click and Drag](#)
 - [Creating Components by Dragging from the Palette](#)
 - [Creating Components Using Tags](#)
 - [Creating Components Using Shapes](#)
 - [Custom Palette](#)

Creating Components Using Click and Drag

Components can be created on the window by first selecting them in the component palette, and then clicking and dragging on the window space. Draw a rectangle in the container to specify where the component should be placed and what size it should be.



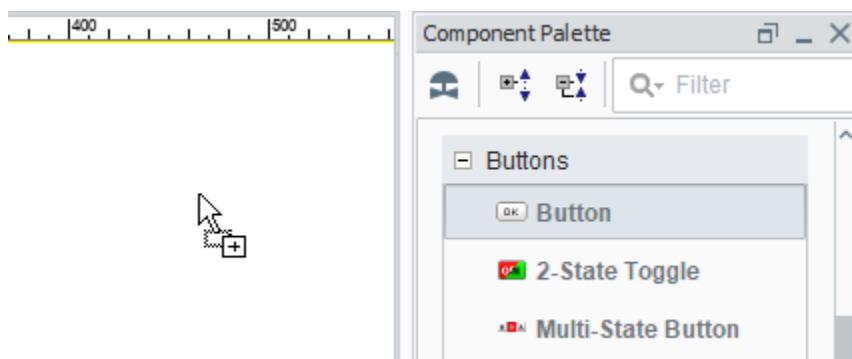
**INDUCTIVE
UNIVERSITY**

**Creating
Components**

[Watch the Video](#)

Creating Components by Dragging from the Palette

Components can be created by dragging them from the component palette to the window. The component will be placed where they were dropped at its default size. Once on the window, the component can be resized using its resize handles.



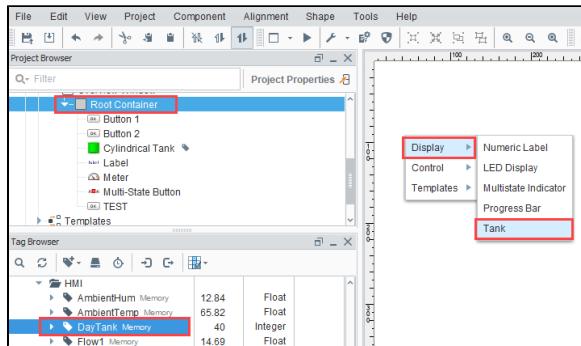
Creating Components Using Tags

Components can also be created by simply dragging a Tag onto a container. Depending on the data type of the Tag, you will get a popup menu prompting you to select an appropriate type of component for that Tag. This technique is great for rapid application design as it does two things for you:

- The component is created at the position you dropped it.
- A variety of property bindings are created automatically.

**INDUCTIVE
UNIVERSITY**

Tags are used in windows to power property bindings on components. The easiest way to make some components that are bound to Tags is to drag and drop some Tags onto your window.



In the example above, we dragged the DayTank Memory Tag onto the window and were given the option of Display, Control, or Templates. Within the display components, we were given the option of displaying the tag in a Numeric Label, LED Display, Multistate Indicator, Progress Bar, or Tank component.

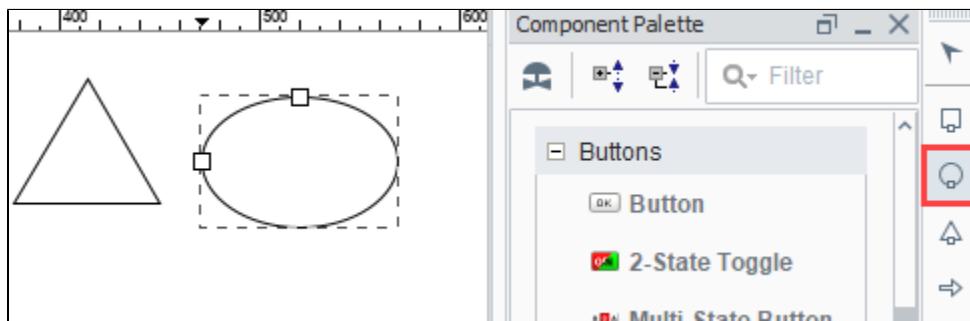
The bindings depend on what kind of Tag was dropped and what kind of component was created. For example, let's suppose you have a Float8 point that represents a setpoint, and you want to set it. Drop the Tag onto a container and choose to control it with a Numeric Text Field. The following bindings will be set up automatically:

- The text field's **doubleValue** property gets a bidirectional Tag binding to the Tag's **Value** property.
- The text field's **minimum** and **maximum** properties get Tag bindings to the Tag's **EngLow** and **EngHigh** properties, respectively.
- The text field's **decimalFormat** property gets a Tag binding to the Tag's **FormatString** property.
- The text field's **toolTipText** property gets a Tag binding to the Tag's **Tooltip** property.

It is important to realize that multiple property bindings are created when creating components this way. These bindings not only use the Tag's value, but much of the Tag's metadata as well. Using the Tags metadata in this way can greatly improve a project's maintainability. For example, if you decide that the setpoint needs 3 decimal places of precision, you can simply alter the Tag's **FormatString** to be `#,##0.000`, and anywhere you used that Tag will start displaying the correct precision because of the metadata bindings.

Creating Components Using Shapes

All of the shapes that you can draw using the shape tools are themselves components. As such, they have properties, event handlers, names, layout constraints, and all of the other things that you'll find on other components.



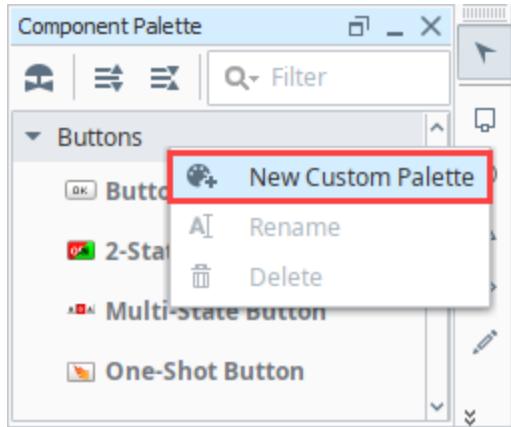
Custom Palette

Custom palettes are like expanded copy/paste clipboards. You can put customized components or groups of components into a palette for quick access.

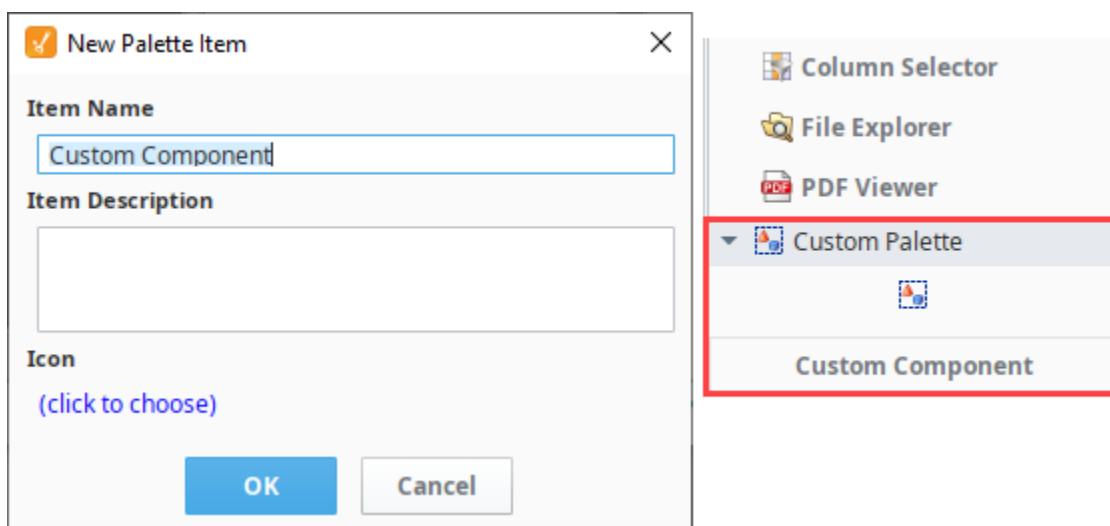
To create a custom palette, right-click on a tab in the tabbed palette or a header in the collapsible palette, and choose New Custom Palette. Your custom palette will appear as the last palette. You can rename it by right clicking on the palette. Your custom palette has one special icon in it, the Capture icon. To add components to your palette, select them and press the capture button. This effectively does a copy, and stores the captured components as a new item in the clipboard. You can then use that item much like a normal component, and add multiple copies of it to your windows.

Tag Binding - Drag and Drop

[Watch the Video](#)



You can assign your custom component a name and it will appear under the Custom Palette. Note that these are simple copies, and are not linked back to the custom palette. Re-capturing that palette item will not update all uses of that item across your windows.



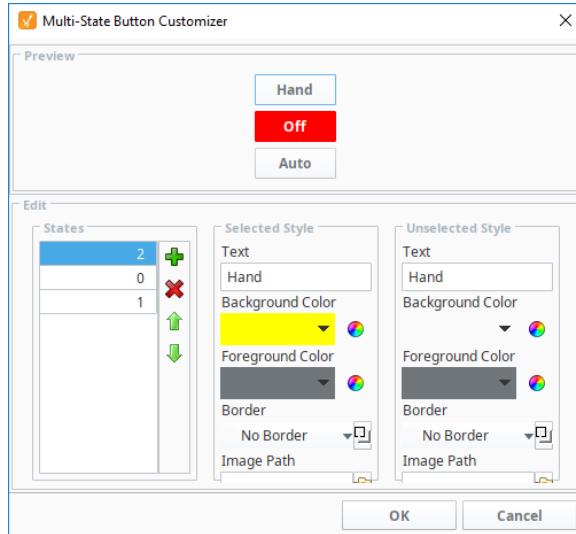
Vision Component Customizers

The Vision module provides a number of customizers to configure components in ways that are more complex or detailed for basic properties.

The two main customizers are the Component Customizer and the Style Customizer. These two customizers are used repeatedly for many different components. For special purpose components like the [Easy Chart](#), [Table](#), [Tab Strip](#), and [Multi-State Button](#), they have their own special customizers for you to create your own custom properties.

Component Customizers

To use a customizer, right-click on the component, choose **Customizers**, and select the customizer for the component you are working with. You can also select the component and click the **Customizer** icon in the Vision Main Toolbar at the top of the window. The following is an example of the customizer for the Multi-State Button Component.



On this page ...

- [Component Customizers](#)
- [Custom Properties](#)
- [Style Customizer](#)
 - [Configuring the Style Customizer](#)
 - [Example 1](#)
 - [Example 2](#)
 - [Value Conflict](#)

Expert Tip

Often, a Customizer works as a user-friendly user interface to one or more expert properties. For example, the [Easy Chart Customizer](#) modifies the contents of the pens, tagPens, calcPens, axes, and subplot dataset properties. This means you can also use Property Bindings and scripting to modify the values of these expert properties at runtime, giving you the ability to dynamically perform complex manipulations of components.

Custom Properties

In addition to the component's basic property settings, you can also create your own custom properties to enhance and add functionality to a component. You can use the custom properties like any other properties - with data binding, scripting, styles, and so on. Custom properties are important for passing parameters from one window to another, especially with a [popup window](#). Properties on the window's Root Container are special in that they double as a window's parameters. For example, when you click on a Button component to open a popup window, it can pass a set of values into the window, which then get set to the custom property on the Root Container for use on that window.

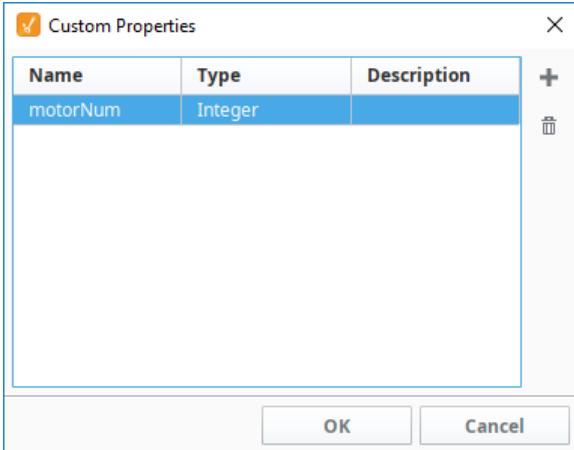
1. To configure a custom property, right click on the component, and select **Customizers > Custom Properties**.
2. Click the plus icon to add a row. Enter the **Name** (i.e., motorNum) of the custom property and data **Type**. Click **OK**.



Custom Properties

[Watch the Video](#)

3. In the **Property Editor**, scroll to the bottom of the panel to see your custom property in blue.



Custom Properties can be any of the basic property types, but can also be a [UDT Definition](#). When specified, The UDT option will create a property shape that matches the shape of the UDT, allowing each member in to UDT to be represented as a separate property in the resulting custom property.

Style Customizer

Many components support the Style Customizer which lets you define a set of visual styles that change based on a single property. Typically, you'll have a property (often a custom property) on your component that you want to use as a driving property, usually a discrete state, and you have multiple visual properties, like the font, border, foreground color, visibility, and so on that you want to change based on that one driving property. The Style Customizer enables you define these relationships all at once, and lets you preview them as well. Without styles, you would have to go to every property and bind them all individually.

 INDUCTIVE
UNIVERSITY

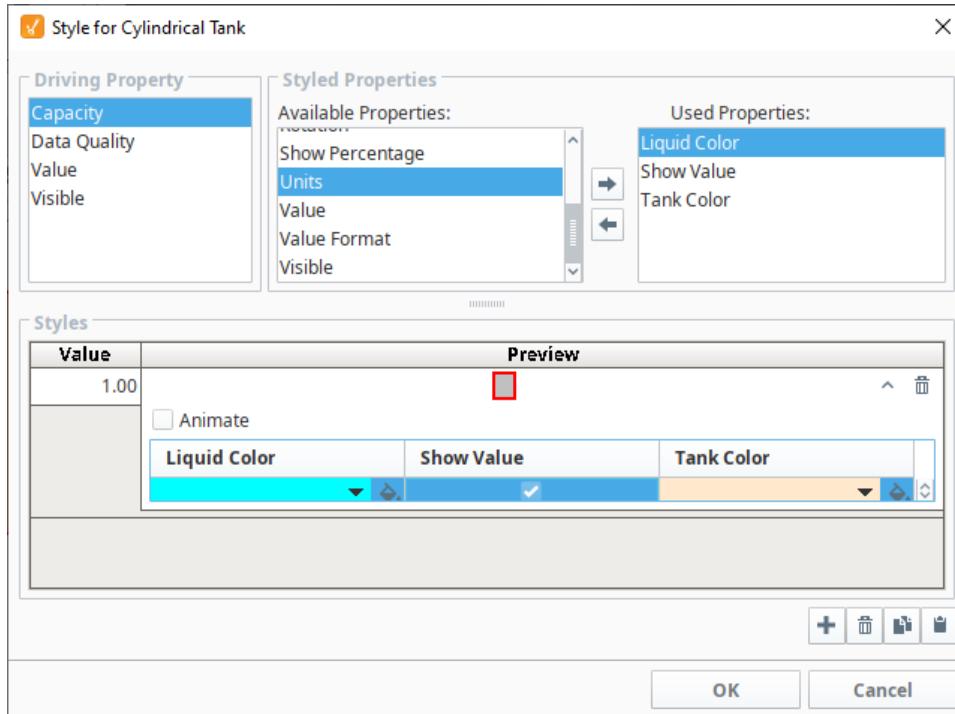
Component Styles

[Watch the Video](#)

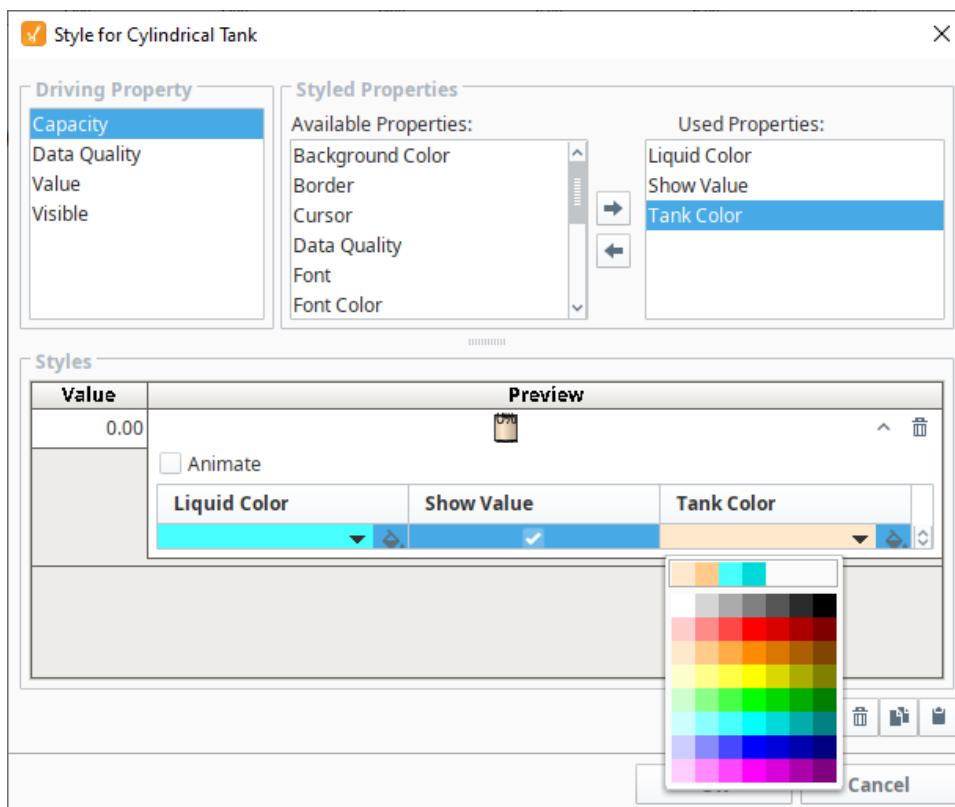
Configuring the Style Customizer

Some components have styles already setup and others do not. The following example involves a Cylindrical Tank component that already has a styles defined. This example shows you how to change styles using the Style Customizer for the Cylindrical Tank.

1. Drag in a **Cylindrical tank** from the [component palette](#) on to your window.
2. Right click on the Cylindrical Tank **component** and scroll down to **Customizers > Style Customizer**. There are four driving properties that can have styles configured: Capacity, Data Quality, Value, and Visible.
3. For this example, click on **Capacity > Liquid Color** and then the **Add Property**  icon. Repeat this step for the **Show Value** and **Tank Color** Properties.



4. Next under Styles, click the Add icon.
5. Click the Expand icon to see the color palette for the Liquid Color.

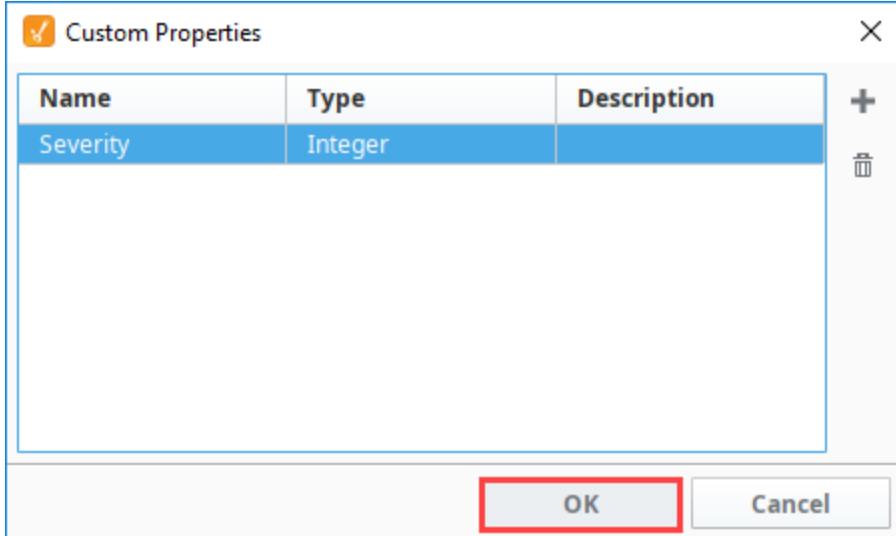


6. Choose a color from the palette. Repeat this step for the **Show Value** and **Tank Color** Properties.
7. Then click the **OK** to save your updates.

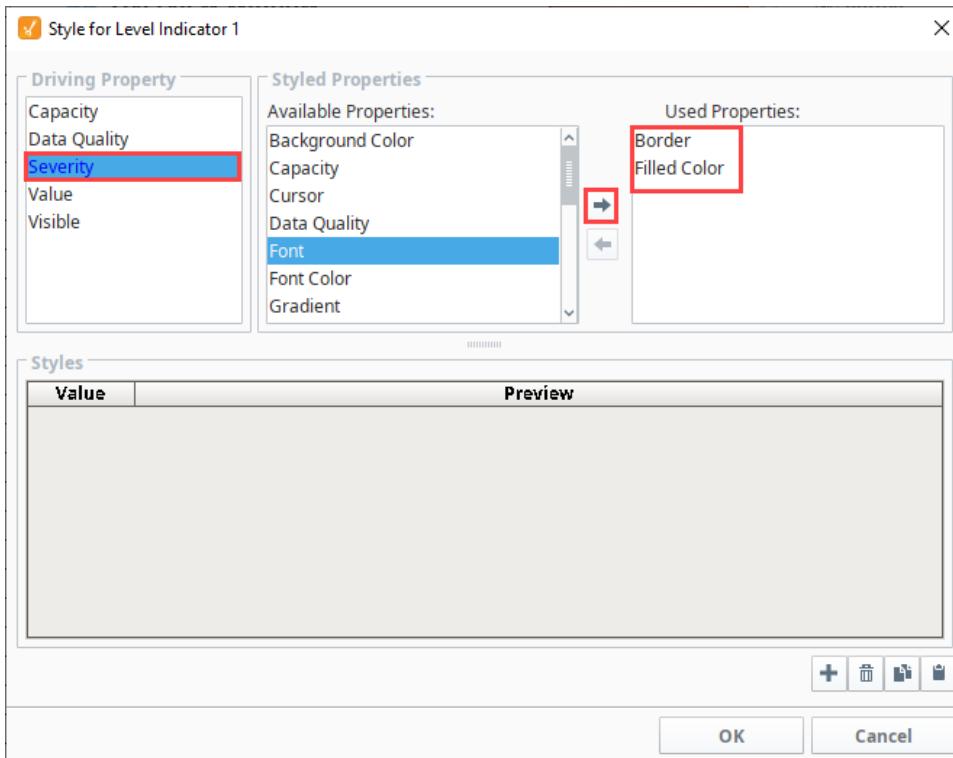
Example 1

In this example, we have a **Level Indicator** component that is displaying the level in a tank. Let's say that you want to have its appearance change based on the alarm state of the tank's temperature. We'll do this using a custom property.

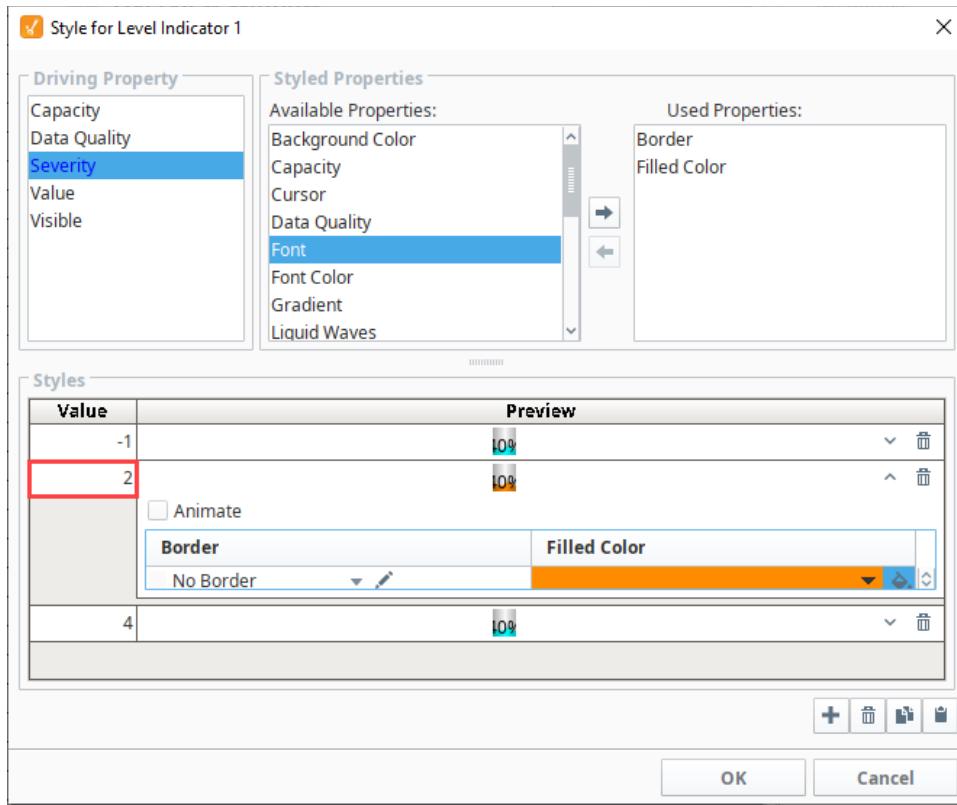
1. In the Vision window, right click on the component and choose **Custom Properties**.
2. Click the Add  icon.
3. Name the new property **Severity** and set it to an Integer type. Click **OK**.



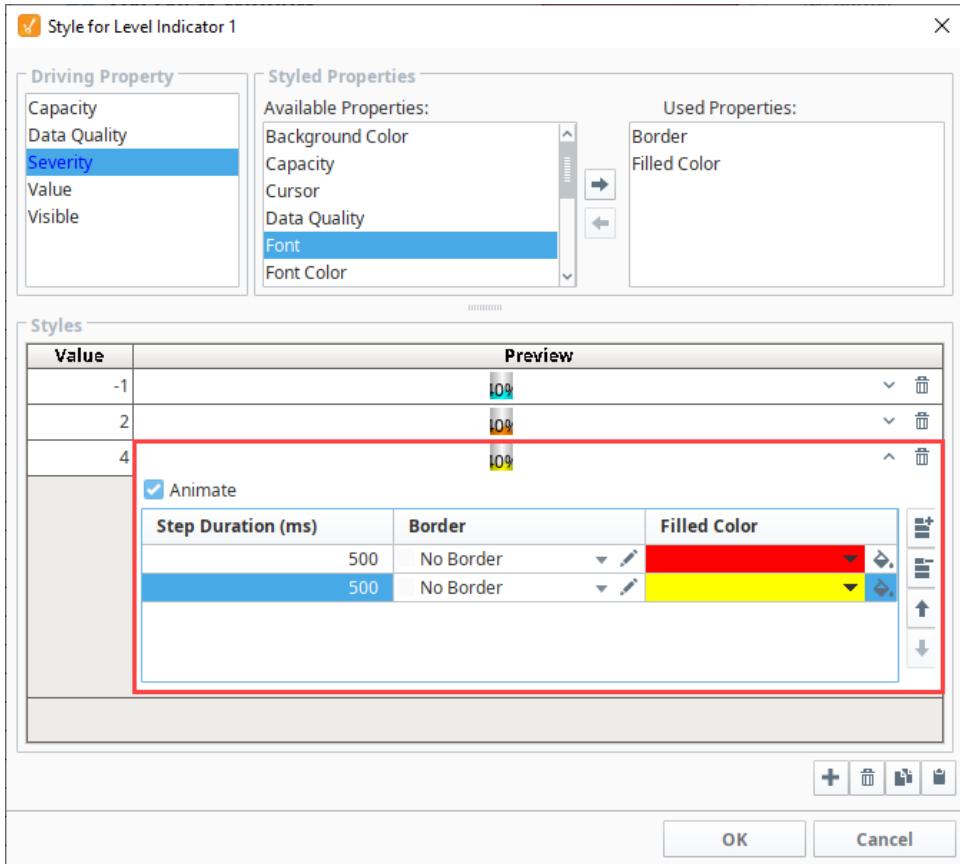
4. Right click on the component and choose **Style Customizer**.
5. Choose your **Severity** property as the driving property, and the **Border** and **Filled Color** properties as the styled properties.



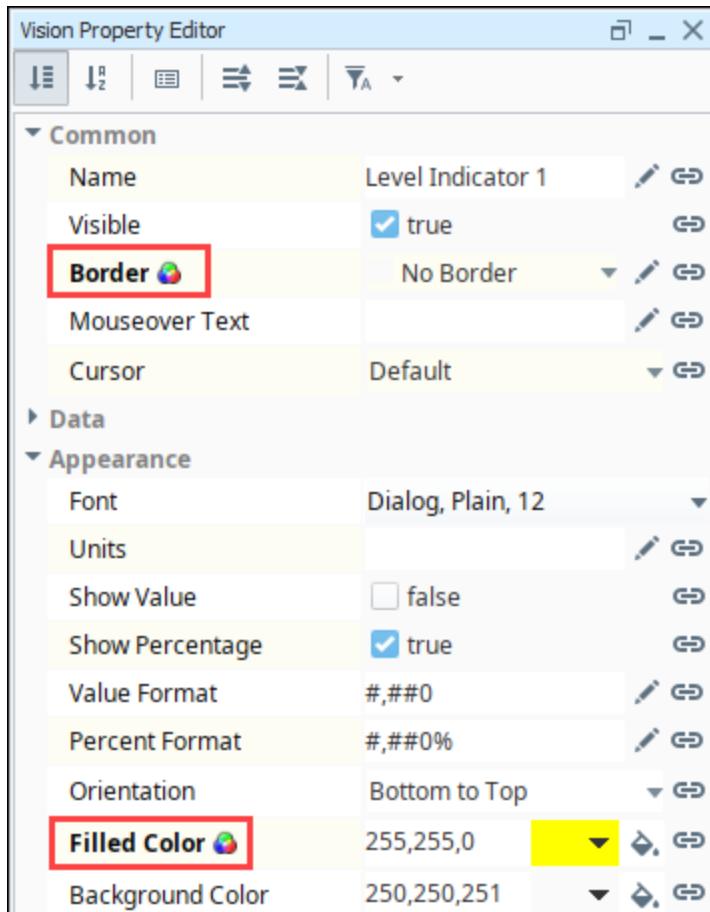
6. Under Styles, click the Add  icon three times.
7. Now create three styles for the three alarm states you want to show. For the first style, enter a value of -1 (not an alarm) and don't change anything else.
8. For the second, enter a value of 2 (medium alarm). Set the filled color to orange.



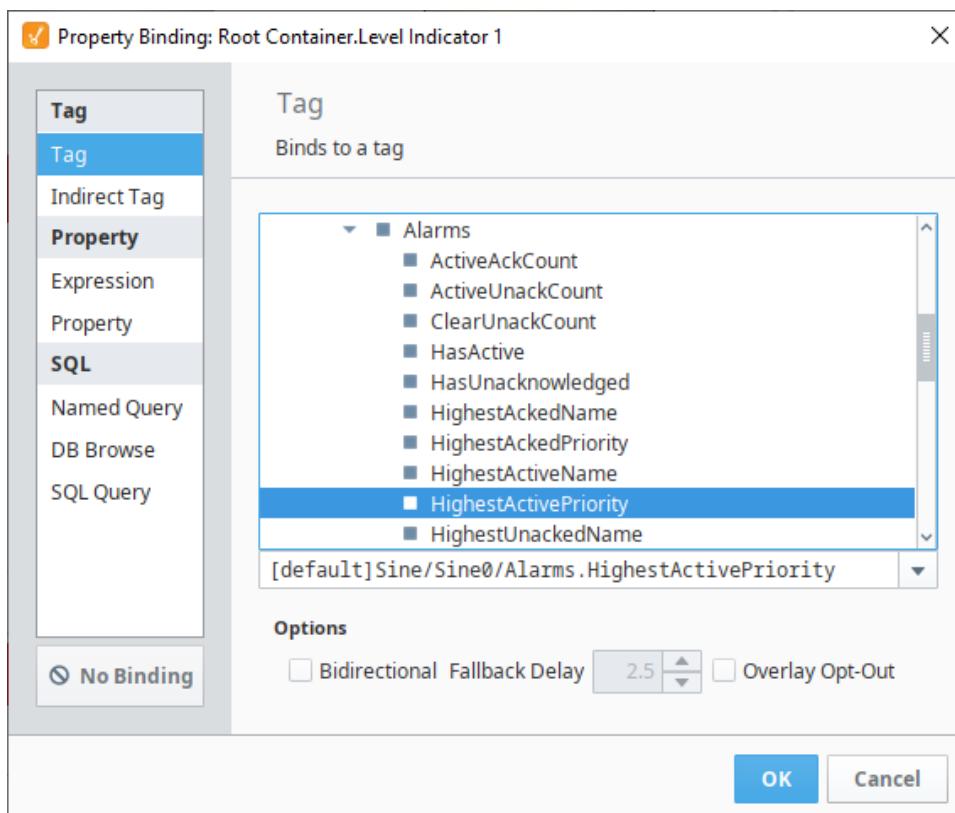
9. For the third style, enter a **Value** of 4 (high alarm).
 - a. Click the **Expand** icon.
 - b. Select the **Animate** checkbox.
 - c. Click the **Add** icon.
 - d. Set the **StepDuration** to 500 for both frames.
 - e. For the first frame set the **FilledColor** to red.
 - f. For the second frame, set the **FilledColor** to yellow.



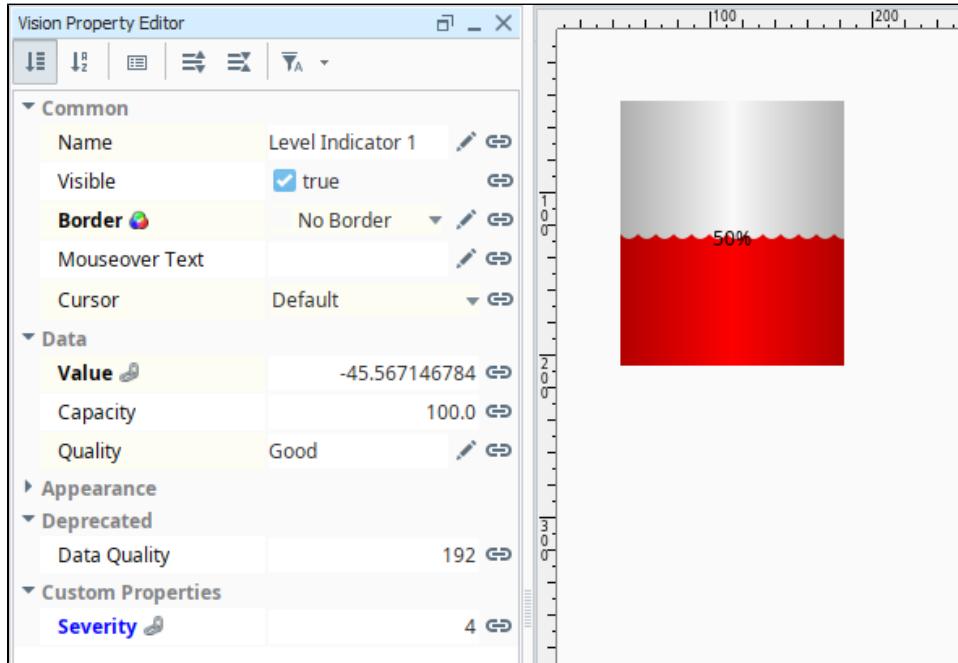
10. Click **OK**. Notice that the styled properties you chose are now bold and have the **Styles** icon next to them. This is to help remind you that those properties are being driven, so if you change their values directly, your changes will be overwritten.



11. In the Property Editor, click on the **Binding** icon for the **Severity** custom property. Bind it to the tank temperature tag's **Alarms.HighestActivePriority** property.



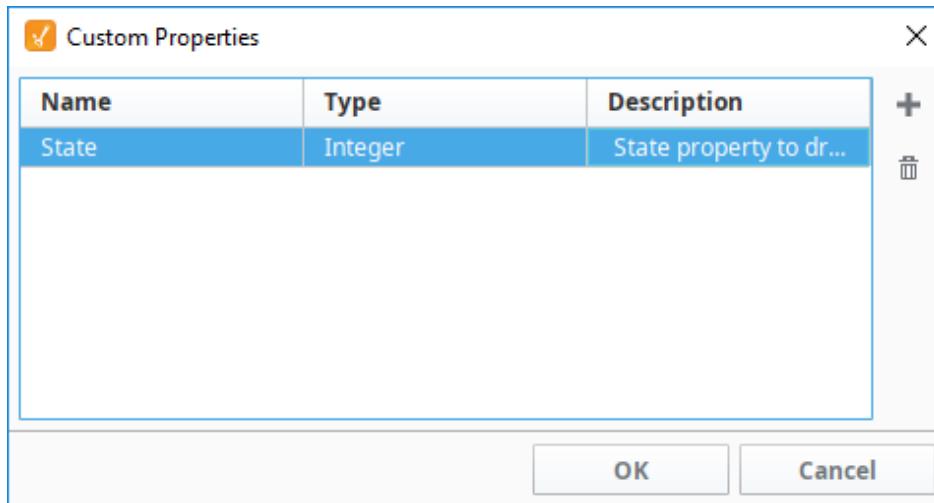
12. Now, when the alarm state for the tank's temperature changes, the color of the appearance of the indicator will change based on the settings in the Style Customizer. In this image, the indicator flashes red and yellow because the high alarm was triggered.



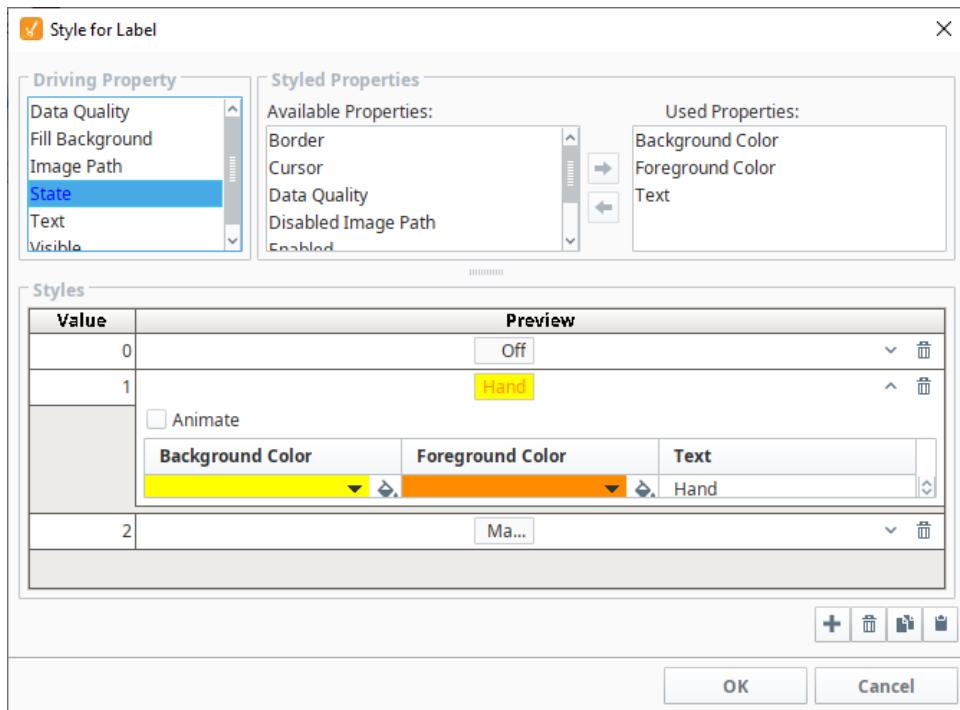
Example 2

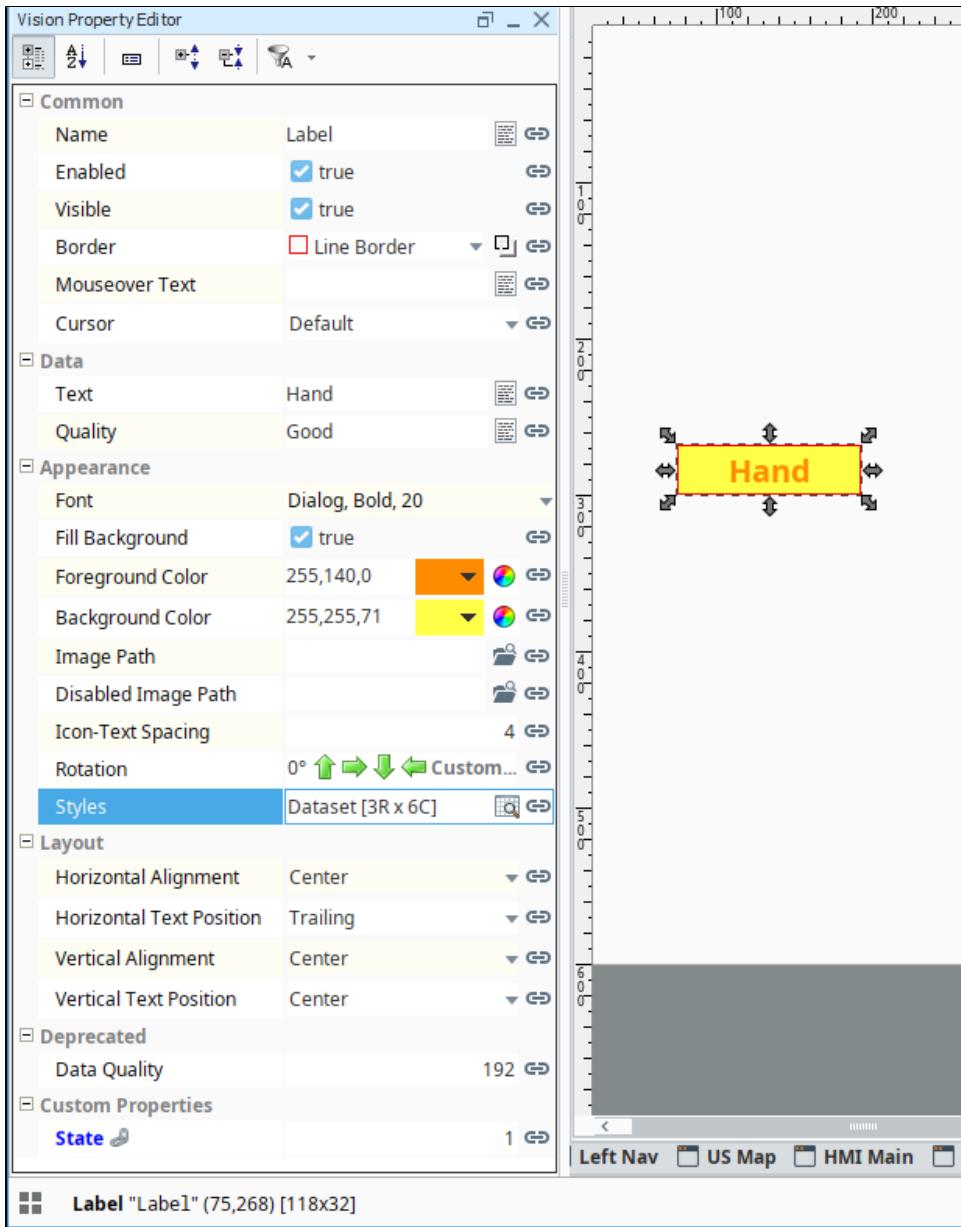
Let's look at another example that uses the Custom Properties and the Styles feature together. For example, the [Label](#) component seems pretty plain at first: it just displays a string. You can use its foreground color, background color, and border to make it look interesting.

1. Drag a Label component onto a window.
2. Right click on the Label component and choose **Custom Properties**.
3. Click the Add icon.
4. Name the new property **State** and set it to an **Integer** type. Click **OK**.



5. Bind that property to a discrete state Tag coming out of a PLC.
6. Next use the **State** property to drive its Styles configuration to make the component look different and display different text based on the state being 0, 1, or 2 (maybe for a Hand/Off/Auto indicator).



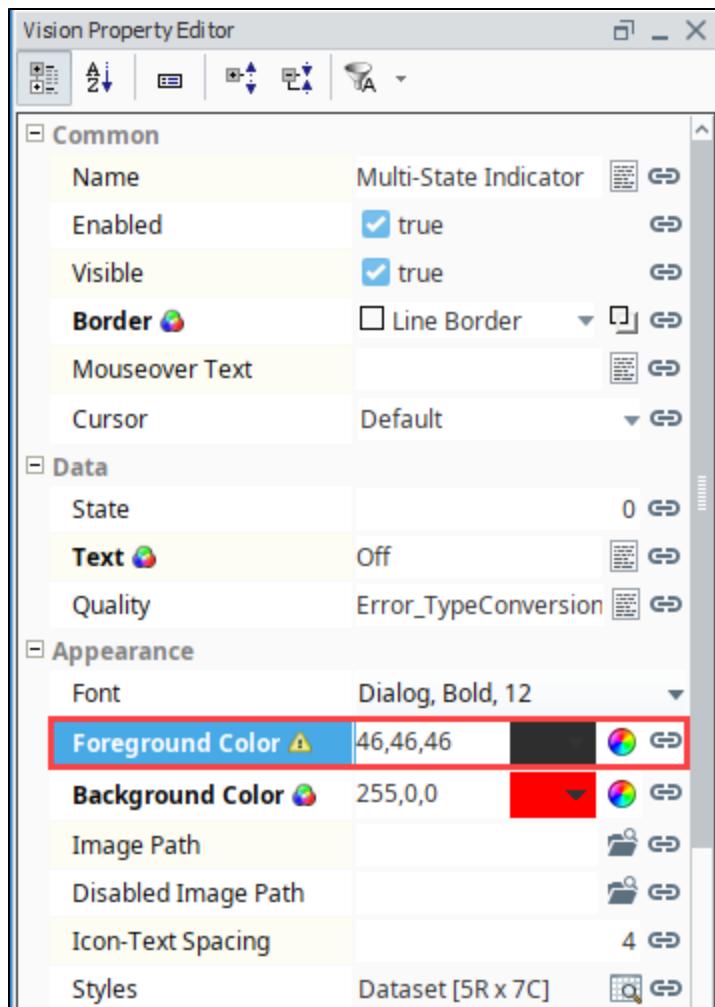


We could have used the [Multi-State Indicator](#) component from the very beginning, but understanding this example will let you create your own types of components by combining the existing components in creative ways.

Some components like the [Easy Chart](#), [Table](#), [Power Table](#), [Tab Strip](#), [Multi-State Button](#), and [Multi-State Indicator](#) have default styles already setup, but you can modify them however you like. If you don't like the default styles, change them. They are there to simply help you get started.

Value Conflict

You can bind a property that is already being used by a style, but a warning icon will appear on the property, and the property name turns red in the [Property Editor](#). This means there is a conflict between the binding on the property, and the style on the component. As a general practice, only the style or binding should write to the property, not both.



Drawing Tools

Drawing Tools Overview

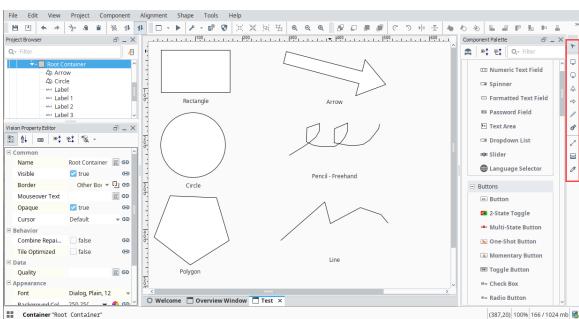
Vision comes with its own set of drawing tools so you can draw your own vector graphics on a window. Using the drawing tools you can create your own shapes such as lines, rectangles, circles, and more. These shapes are components with their own set of properties. Shapes or graphics such as lines, rectangles, and circles can be created using the 2D drawing tools in Vision. All SVG (Scalable Vector Graphics) images are importable in Vision and are made up of these basic shapes.

Using Drawing Tools

By default, the drawing toolbar is always located on the right side, but you can drag it to anywhere on your window that you prefer. At the very top of the toolbar is a **Selection** tool that allows you to select various components on a window. You can use the Selection tool to change the component's size and position as well as to configure the component. Below the Selection tool are all the tools that draw graphics. 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. Once you draw a graphic, and want to drag a different graphic tool on to your window, click on the Selection tool. When a drawing tool is active, a toolbar will appear in the top menubar that has specific settings and actions for that tool.

Types of Drawing Tools

There are multiple drawing tools that each fulfill a different purpose. Some, like the selection tool simply allow you to select different components, while others like the rectangle tool allow you to create shapes. When you create a shape it has a default Fill Paint color of white. After a shape is created, you can change its Fill Paint color, Stroke color, and Stroke Style properties. All shapes can be treated as paths and be used with [composite geometry functions](#) to alter or create other shapes.



On this page ...

- [Drawing Tools Overview](#)
- [Using Drawing Tools](#)
- [Types of Drawing Tools](#)
 - [Selection Tool](#)
 - [Rectangle Tool](#)
 - [Circle Tool](#)
 - [Polygon Tool](#)
 - [Arrow Tool](#)
 - [Pencil Tool](#)
 - [Line Tool](#)
 - [Path Tool](#)
 - [Gradient Tool](#)
 - [Eyedropper Tool](#)
- [Shape Size, Position, and Angle](#)



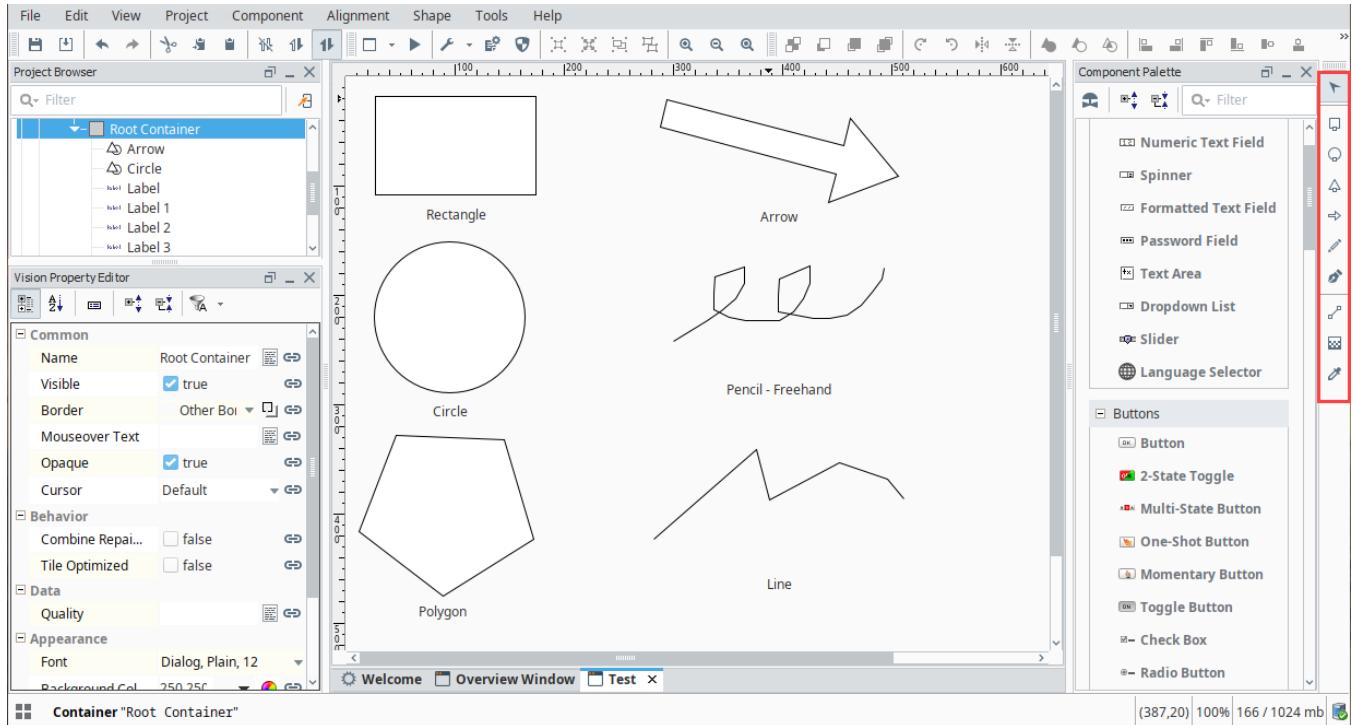
INDUCTIVE
UNIVERSITY

Drawing Tools Overview

[Watch the Video](#)

Selection Tool

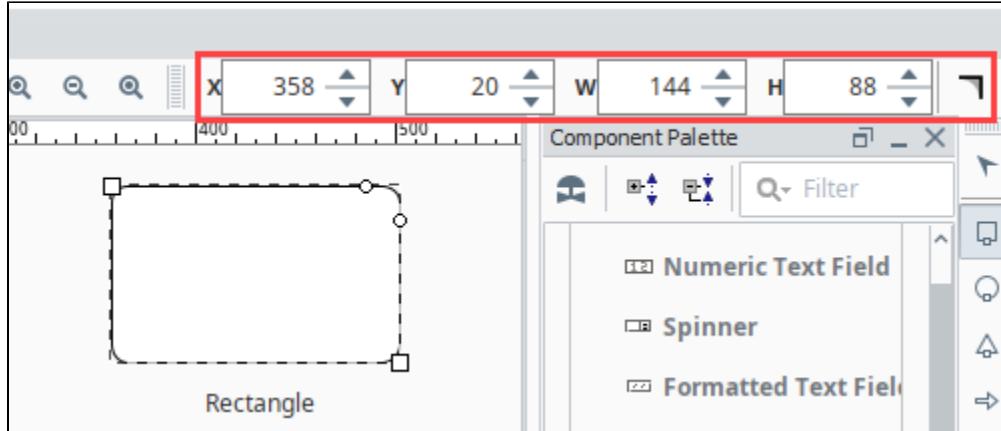
The **Selection** tool is active by default. When this tool is active, you can select shapes and components. Selected components can be moved, resized, and rotated. For more on using the Selection tool to manipulate components and shapes, see [Manipulating Components](#).



Rectangle Tool

The **Rectangle** tool creates and edits rectangle shapes. To create a rectangle, select the tool and then click and drag inside a window to create a new rectangle. Hold down **Ctrl** to make it a perfect square, and the **Shift** key to make it grow from the center point. Once a rectangle is created, you can use the square handles to change the rectangle's height and width. This is important because it is the only way to resize a rotated rectangle and let it remain a rectangle. If you resize a non-orthogonally rotated rectangle using the Selection tool, it will skew and become a parallelogram. If you double-click on the rectangle so the tool is active, you can change the rectangle's width and height using the tool-specific handles. From the toolbar, you can also change the rectangle's location (in pixels) on the window using the X and Y axes.

There are also small circle handles on the rectangle that allow you to alter the rectangle's corner rounding 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's toolbar () to return a rounded rectangle to be a standard, straight-corner rectangle.

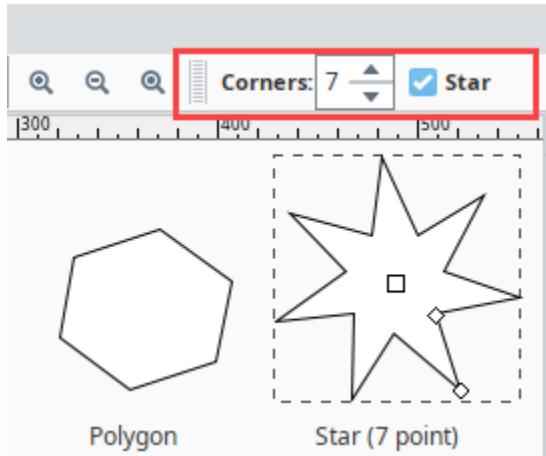


Circle Tool

The **Circle** tool creates and edits circles and ellipses. It is used in much the same way as the rectangle tool. While it is the active tool, you can click and drag inside a window to create a new ellipse. Hold down **Ctrl** to make it a perfect circle, and the **Shift** key to make it grow from the center point. When an ellipse is selected, use the width and height handles to alter the shape. You can also use the ellipse toolbar at the top of the Designer to change the width and the height as well as the X and Y axes.

Polygon Tool

The **Polygon**  tool is used to create polygons and stars. Use the polygon toolbar at the top that becomes visible when this tool is active to alter the settings of the shape that is created when you drag to create a polygon. This tool can be used to make any polygon with three corners (a triangle) or more. On the Polygon menu you can specify the number of corners for the polygon. Once created, you can use the center square handle to move the polygon around, and the diamond handles to alter the size and angle of the polygon. Hold down **Ctrl** to keep the polygon's rotation an even multiple of 15°. For a star shape, specify the number of corners (points) and select the Star check box. A second handle that is between each corner will appear on the polygon allowing you to make a star shape.



Arrow Tool

The **Arrow**  tool is used to create single or double-sided arrow shapes. When it is active, simply drag to create a new arrow. Use the checkbox on the arrow 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 arrow head. When changing the arrow's direction, you may hold down **Ctrl** to snap the arrow to 15° increments.

Pencil Tool

The **Pencil**  tool is used to draw freehand lines and shapes. When this tool is active, 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, then you will create a closed path, otherwise, you'll create an open path (line).

On the pencil toolbar, there are options for simplification and smoothing, as well as a toggle between creating straight line segments or curved line segments. The simplification parameter is a size in pixels that will be used to decrease the number of points used when creating the line. Points will be in general as far apart as this setting. If you find the line isn't accurate enough, decrease this setting. If you choose to create curved segments, then the segments between points will be Bézier curves instead of straight lines. The smoothing function controls how curvy these segments are allowed to get.

Line Tool

The **Line**  tool can be used to draw 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 then double-click, press enter, or make a vertex inside the origin box to end the path.

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.

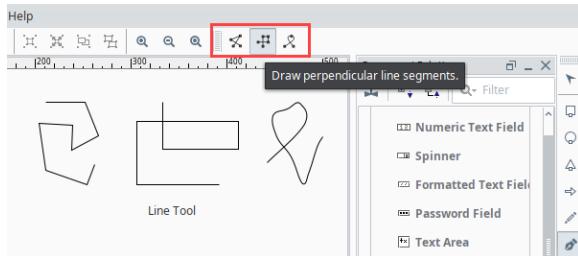
On the line toolbar, you can choose between three different type of line settings:

- straight-line segments
- perpendicular-line segments
- curve-line segments



Drawing a Line

[Watch the Video](#)

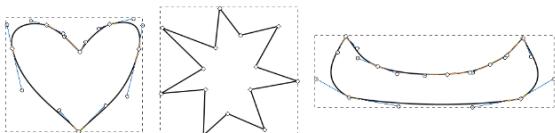


Perpendicular-line segment is just like a straight-line segment except that each segment is restricted to either horizontal or vertical.

The curve-line segment will create a [Bézier curve](#) path by attempting to draw a smooth curve between the previous two vertices and the new vertex.

Path Tool

All shapes and paths can be edited directly by using the **Path** tool. This tool lets you directly modify the nodes in the path, adding new nodes, removing nodes, and toggling segments between straight or curved. For more information, see [Shape Geometry](#).

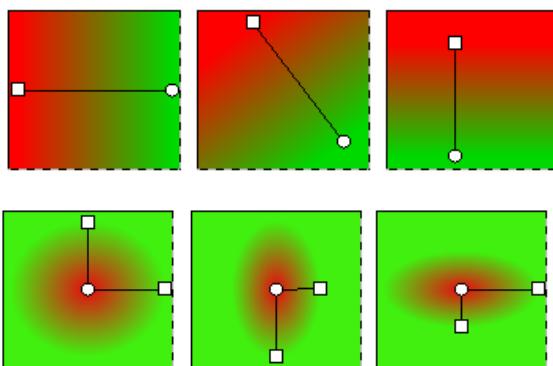


Editing Shape Paths

[Watch the Video](#)

Gradient Tool

The **Gradient** tool is used to affect the orientation and length of any gradient paints. They work hand-in-hand with the [Fill Paint](#) property. Gradients smoothly blend any number of colors that can be positioned along a straight line or form an ellipse across the shape. A **Linear** gradient uses a horizontal line drawn across the width of the shape by default. By switching to the gradient tool, the horizontal line can be changed to move in any direction by dragging the handles. The **Radial** gradient uses a 45° angle drawn over the shape which starts at the center and moves out. Just like the Linear gradient, the Radial gradient can also be changed by dragging the handles around.



Gradients

[Watch the Video](#)

Eyedropper Tool

With the **Eyedropper** tool you can set a selected shape(s) and/or component(s) foreground /background or stroke/fill colors by pulling the colors from somewhere else in the window. Select the component you want to change, and then activate the eyedropper tool. When this tool is active, left-click to set the selection's fill or background color, and right-click to set the selection's stroke or foreground color.

Remember to turn off the Eyedropper tool when you're finished by clicking the Selection tool, otherwise, you will continue to change colors on your component each time you do a mouse click. This tool works on most components as well as shapes. For example, right clicking will set the font color on a Button component, or left-clicking will set the background color.



Eyedropper Tool

Shape Size, Position, and Angle

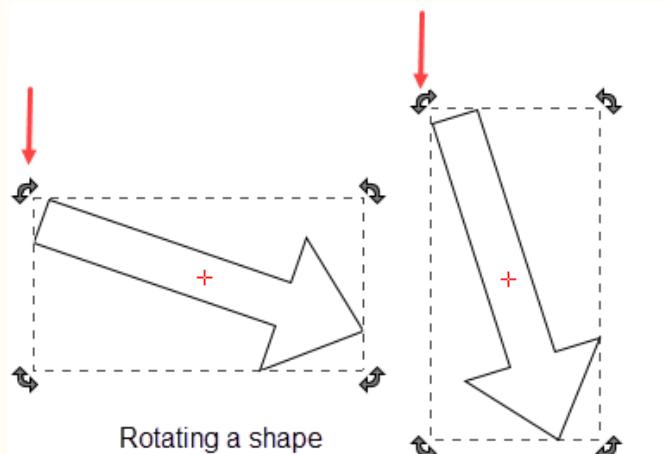
Shapes are different from other components in that they have properties that determine their size and position that can easily be bound. These properties are called X, Y, Height, and Width. The values of these properties are always relative to the shape's parent container's width and height, even in a running Client where that container may be a wildly different size due to the layout mechanism.

For example, let's say that you have a shape that is located at $x=100$, $y=100$, and was 125 by 125 inside a container that is 500 by 500. If you want to animate that shape so that it moves back and forth across the screen, you'd set up a binding so that X changed from 0 to 375. (You want X to max out at 375 so that the right-edge of the 125px wide shape aligns with the right edge of the 500px container). Now, at runtime, that container might be 1000 by 1000 on a user's large monitor. By binding X to go between 0 and 375, the true X value of your shape (whose width will now be 250px due to the relative layout system), will correctly move between 0 and 1750, giving you the same effect that you planned for in the Designer.

Another ability unique to shapes is the ability to be rotated. Simply click on a selected shape and the resize controls become rotate controls. There's even an Angle property that can be edited directly or bound to something dynamic like a Tag.

Note:

Use caution when binding the rotation. When you change a shape's rotation, its position also changes. The position of any shape is the top-leftmost corner of the rectangle that completely encloses the shape. Because of this effect, if you wish to both dynamically rotate and move a component, special care must be taken since rotation alters the position.



If you want to both dynamically rotate and move a component, special care must be taken since rotation alters the position. You don't want your position binding and the rotation binding both fighting over the position of the component. The way to both rotate and move a shape is as follows:

1. Bind the rotation on your shape as you wish.
2. Create a shape (for example, a rectangle) that completely encloses (in other words, it's bigger than) your shape at any rotation angle.
3. Set that rectangle's visible property to false.
4. Select your shape and the rectangle and group them.
5. Bind the position on the resulting group.

If you follow these steps, you can animate both the rotation and position of a shape.

In This Section ...

Shape Geometry

Shape Paths

Once you draw shapes using the drawing tools in Vision, it's possible to alter and edit shapes after they've been created. By default, all shapes have a white fill color. Editing the paths of your vector

graphic shapes is done by using the **Path**  tool. Simply select any shape or line while the Path tool is active to start editing. If the shape is already a path, you can switch to the Path tool by double-clicking on the shape.

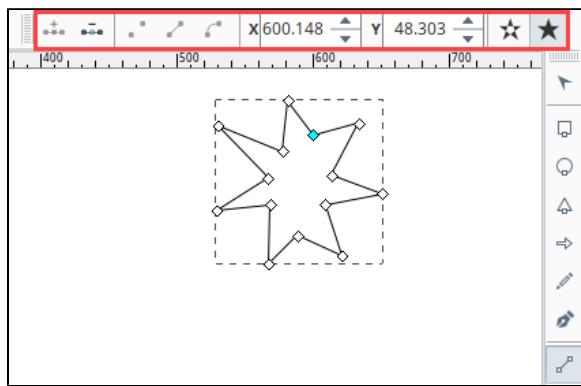


You can convert any shape into a general path by selecting the **To Path**  function under the **Shape** menu. Shapes will also implicitly turn into paths if they are altered in a way not supported by the underlying shape. For example, if you stretch a rotated rectangle, thereby skewing it into a parallelogram, it will become a path automatically.

Editing a Shape Path

Each point on the path is represented by a diamond-shaped handle when the path editor is active. These handles can be dragged to move them around. They can also be selected by clicking on them or dragging a selection rectangle to select multiple points. This allows groups of points to be altered simultaneously.

To change a line segment between open, straight, and curved, select the **Path**  tool and use the toolbar functions that become visible. Points can also be added and removed using the functions on the Path Editor toolbar.



Filling a Shape

Filled shapes have two fill settings that control whether or not holes in the shape should be filled. To remove the fill entirely, simply set the **Fill Paint** property in the Property Editor to **No Paint** .

On this page ...

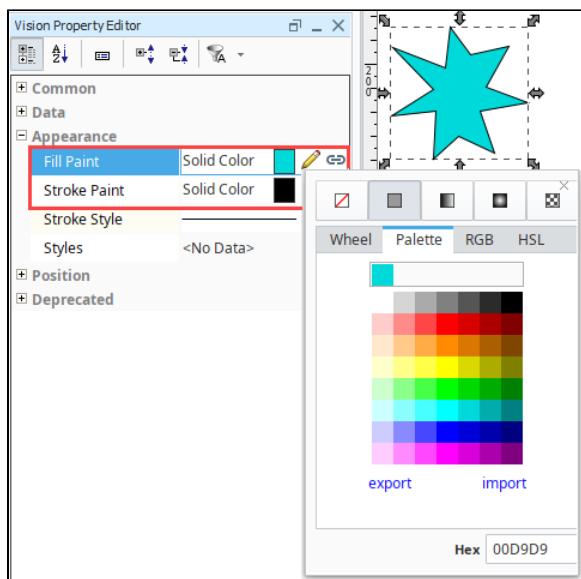
- **Shape Paths**
 - [Editing a Shape Path](#)
 - [Filling a Shape](#)
- **Bézier Curve**
 - [Making Bézier Curves](#)
- **Creating and Editing Shapes Using Constructive Area Geometry**
 - [Union](#)
 - [Difference](#)
 - [Intersection](#)
 - [Exclusion](#)
 - [Division](#)



INDUCTIVE
UNIVERSITY

Shape Geometry

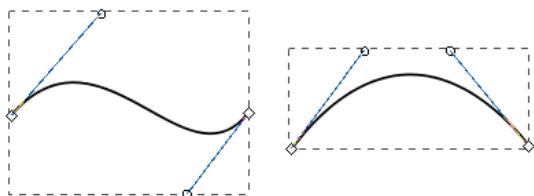
[Watch the Video](#)



When editing paths directly, it is often useful to be zoomed in on the path. Don't forget that you can zoom in on a location by holding down **ctrl** and using your mouse wheel to zoom in on a particular area without having to zoom in and then scroll. Also, if you press your mouse wheel in, you can pan around your window.

Bézier Curve

A **Bézier curve**, also sometimes called a quadratic curve, is a type of curved line used in vector graphics that connects two points, allowing you to create smooth vector graphic shapes. A Bézier curve is configured using four points: the two end-points and two control points. The curve starts along the line between the an endpoint and the first control point, and then curves to smoothly meet the line between the second control point and the next endpoint.



Making Bézier Curves

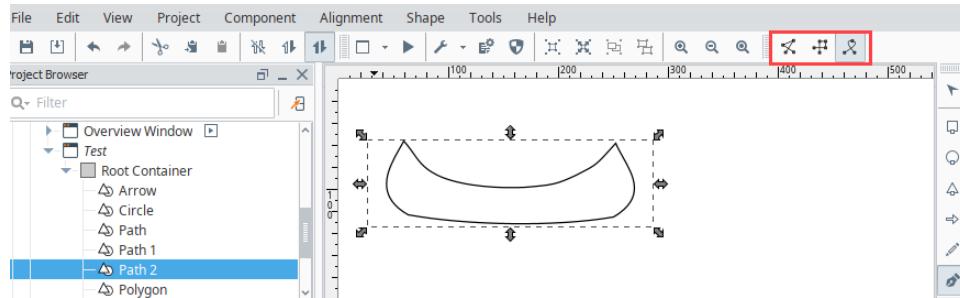


Bezier Curves

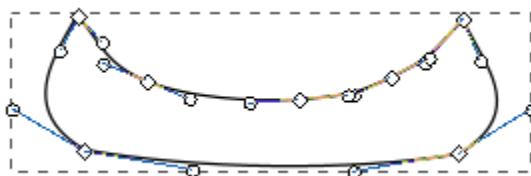
[Watch the Video](#)

Curves are made using the [Line tool](#).

1. In the Designer, select the **Line**  tool icon on the toolbar on the right side of your window. When the Line tool is active, a toolbar will appear in the top menubar. There are three different type of line settings: straight-line segments, perpendicular-line segments, and curved-line segments.
2. Select the curve-line segment on the menubar. Click on the window to begin drawing your image. Each time you want to make a curve, click on the screen. The curve-line segment will draw a smooth curve between points creating a smooth vector graphic shape.

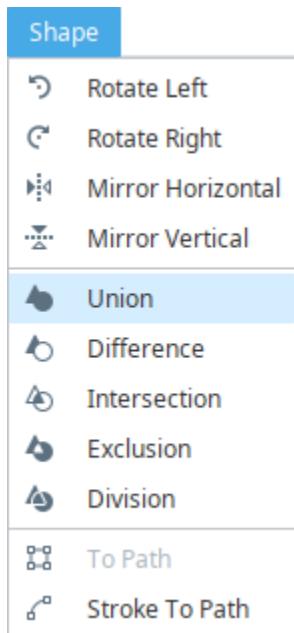


3. The line tool can make lines as well as shapes. To complete the line, simply click a second time on your final location. To make the line into a shape, click on the starting point as your final location.
4. Using the Path  tool in the drawing toolbar, you can see a vector graphic shape showing where the points meet and the smooth curves between each point. If you want to edit or alter the shape after you create it, use the Path tool and drag the circles or diamonds to change the shape.



Creating and Editing Shapes Using Constructive Area Geometry

Editing paths directly can be a bit awkward. Using Constrictive Area Geometry is usually a much easier and more intuitive way to get the shape that you want. These functions are accessed from the **Shape** menu and operate when two (or more) shapes are selected.



 **INDUCTIVE
UNIVERSITY**

Editing Shape Paths

[Watch the Video](#)

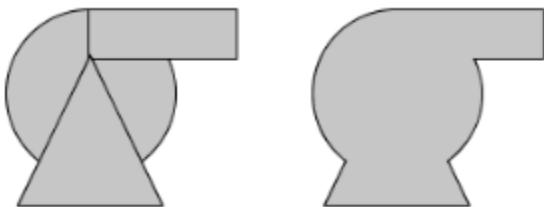


Selection Order Matters

The order that you select the shapes is important for many of these functions. Typically, the first shape you select is the shape you want to retain, and the second shape is the shape that you want to use as an "operator" on that first shape.

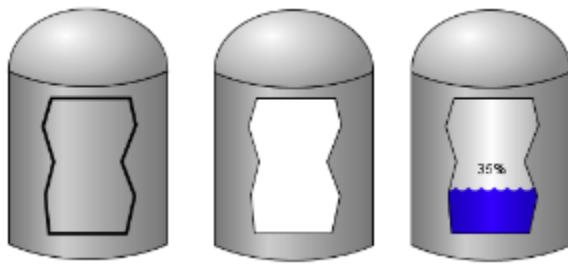
Union

The **Union** function combines two or more paths into one. The resulting shape will cover the area that any of the shapes covered initially. The example shows how the union of a circle, rectangle, and triangle can be unioned together to create a basic pump symbol. Creating the symbol using this method took a few seconds, whereas attempting to draw this shape by hand using paths would be quite frustrating.



Difference

The **Difference** function can be thought of as using one shape as a "hole-punch" to remove a section of another shape. The example shows how a zigzag shape drawn with the line tool can be used to punch a cutaway out of a basic tank shape. The level indicator is added behind the resulting shape to show how the area where the zigzag shape was is no longer part of the tank shape.



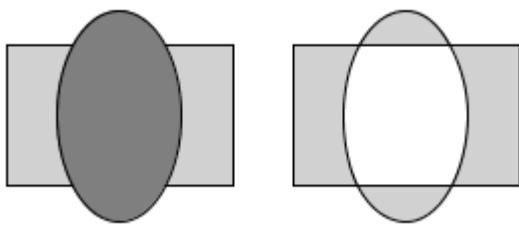
Intersection

The result of an **Intersection** function will be the area only where where two shapes overlap. The example shows how the "top" of the tank in the difference example was easily made using two ellipses.



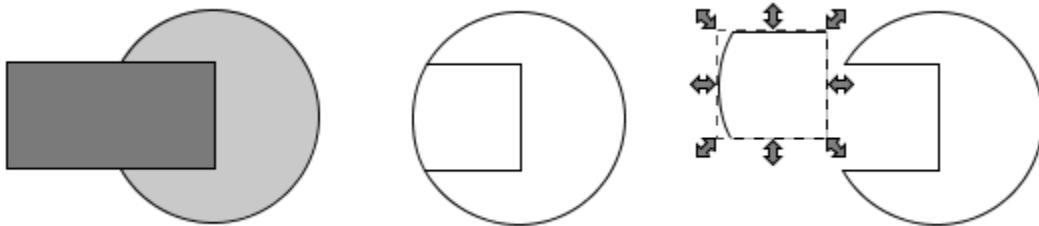
Exclusion

The **Exclusion**  function, sometimes called X-OR, creates a shape that occupies the area covered by exactly one of the source shapes, but not both.



Division

The **Division**  function divides or cuts one shape up along the outline of another shape. This works the same as an intersection for the first shape selected, and a difference for the second.



Fill and Stroke

All shapes have three properties that affect how they look: **Fill Paint**, **Stroke Paint**, and **Stroke Style**.

- **Fill Paint:** Determines the interior color of the shape.
- **Stroke Paint:** Represents the color of the shape's outline.
- **Stroke Style:** Determines the thickness, corners, and dash properties of the shape's outline.

Fill Paint

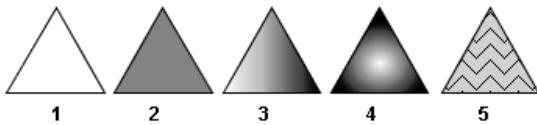
Editing Paints

Both the Fill and Stroke paints can be a variety of different types of paints. To edit a shape's fill or stroke paint, you can either use the paint dropdown in the Property Editor table by clicking on the **Edit**

icon or open up the dedicated **Fill and Stroke** panel from the **View** menu.

Paint Types

The top of the paint editor is a selection area that allows you to choose between the five different types of paints.



1. **No Paint** when used as a fill paint, then the interior of the shape will be transparent. If used as the stroke paint, then the paint's outline will not be drawn.
2. **Solid Color Paint** is equivalent to the Color type used elsewhere throughout the component library. A solid color is any color, including an alpha (transparency) level.
3. **Linear gradient** smoothly blends any number of colors along a straight line across the shape. Each color is called a Stop. Each stop is represented as a drag-able control on a horizontal preview of the gradient 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.
4. **Radial gradient** are similar to linear paints except that the colors emanate from a point creating an ellipse of each hue. Radial paints are configured in the same way as linear paints.
5. **Pattern paint** uses a repeating pixel-pattern with two different colors. You can pick a pattern from the dropdown or create your own using the built-in pattern editor.

Gradients

Gradient Paint Bounds

The two gradient paints, **Linear** and **Radial**, are more than a list of colored stops, they also need to be placed relative to the shape. The same gradient may look wildly different depending on how it is placed against the shape. By default, a Linear gradient will run horizontally across the width of the entire shape, but this is readily changed. By switching to the **Gradient** Tool located on the drawing tools toolbar, you can drag handles around to change the orientation of the gradient. You can even make the gradient larger or smaller depending on how big you want it to be.

On this page ...

- **Fill Paint**
 - [Editing Paints](#)
 - [Paint Types](#)
- **Gradients**
 - [Gradient Paint Bounds](#)
 - [Gradient Cycles](#)
 - [Setting a Gradient](#)
- **Stroke Style**



INDUCTIVE
UNIVERSIT

Fill and Stroke

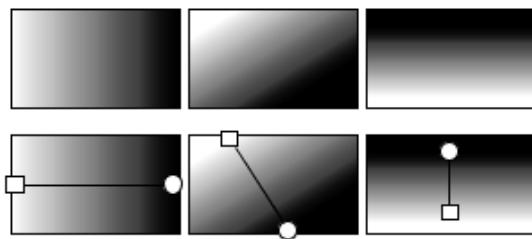
[Watch the Video](#)



INDUCTIVE
UNIVERSIT

Gradients

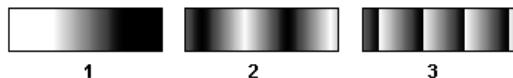
[Watch the Video](#)



Gradient Cycles

The two gradient paints (Linear and Radial) both have a cycle mode that you can change by right-clicking within the preview strip.

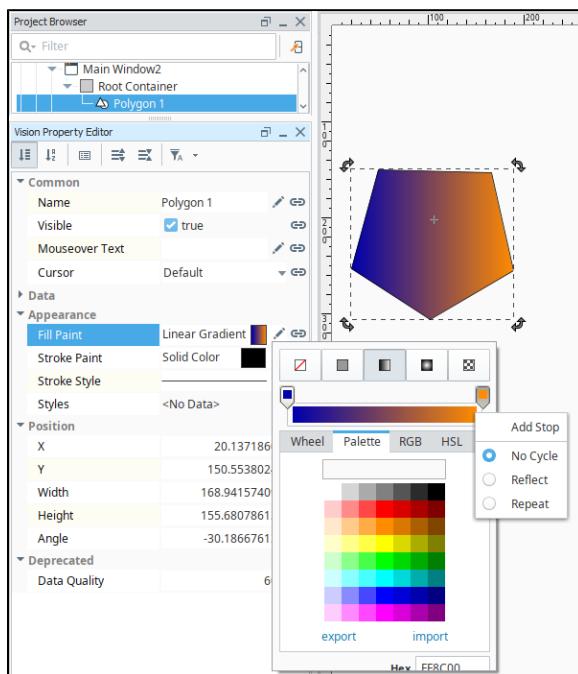
The cycle modes are illustrated below: **No Cycle**, **Reflect**, and **Repeat**.



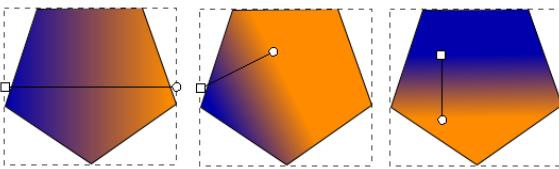
1. **No Cycle** - The first and last stops are repeated forever after the edge of the gradient bounds.
2. **Reflect** - Beyond the bounds of the gradient, it will be reflected and drawn in reverse, and then reflected again, creating a smooth repetition.
3. **Repeat** - Beyond the bounds of the gradient, it will be repeated forever.

Setting a Gradient

1. In the **Designer**, select your component.
2. In the **Property Editor** under **Appearance**, click the **Edit** icon.
3. Select either the **Linear** or **Radial** gradient.
4. You will see two stops: white and black. Click on each **Stop** and choose a different color.
5. If you want to add an additional stop, right click on the color bar and select **Add stop**. You can also add / remove **Stops**, and select your desired cycle: **No Cycle**, **Reflect** or **Repeat**.

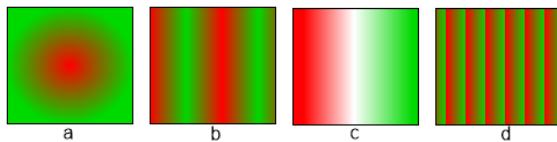


6. Close the **Color Selector** by clicking the X in the upper right corner.
7. With your component still selected on your window, and click the **Gradient** tool in the toolbar. You'll notice a line on your component. Now, you can drag the line's handles to change the orientation and lengthen or shorten the gradient.



8. Here are a few examples of what you can do with gradients:

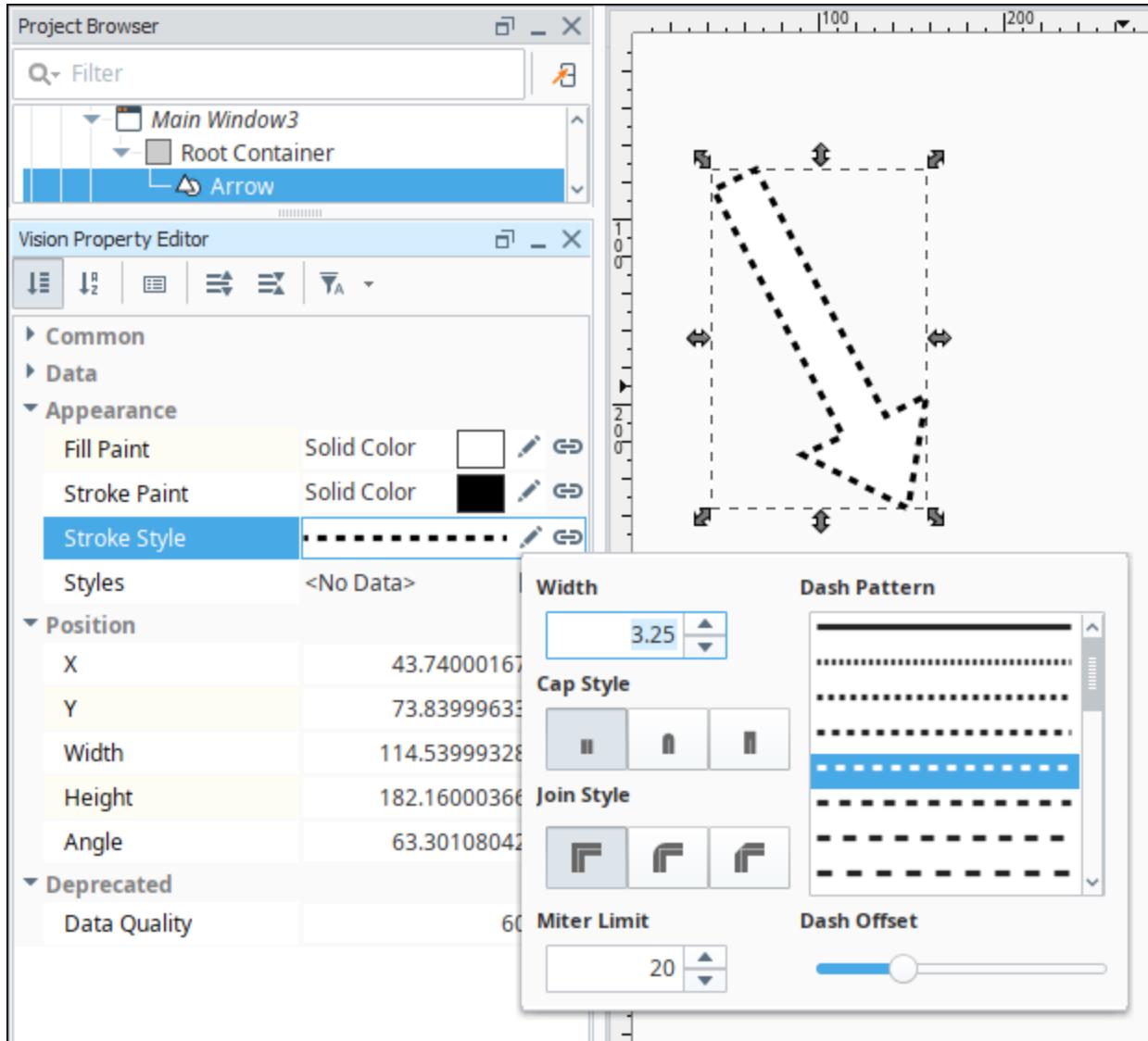
- a. **Radial** - No cycle
- b. **Linear** - Reflective
- c. **Linear** - with a 3rd Stop
- d. **Linear** - Repeat



Stroke Style

A shape's stroke paint is only half the story. The **Stroke Style** is also an important component of how an outline is drawn. Primarily the style controls the thickness of the line drawn, but it also can be used to create a dashed line. The setting for thickness is specified in pixels, and creating a dashed line is as easy as picking the style from the list. The effect of the 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 readily on thick lines.

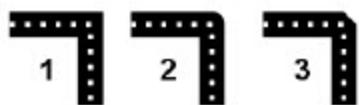
You can access the **Stroke Style** in the Property Editor under **Appearance**.



Cap style is a setting that controls what happens at the end of a line segment. You can either have the line simply be terminated with no decoration (#1), round-off the end with a semi-circle (#2), or cap the end with a square (#3).



Join style is a setting that affects how a line is drawn where two segments meet (a corner). The default setting is called a miter join (#1), where the stroke is extended into a point to make 90-degree angle. The other options are rounded corners (#2) or beveled edge corners (#3).



Miter Limit style joins can become a problem for very sharp angles. With a sufficiently sharp angle, the miter decoration can become extremely long. To control this, there is a miter length setting to limit the length of a miter decoration. The illustration below shows the same miter join with two different miter length settings. The first drawing illustrates the length of the miter join.

77

Images and SVGs in Vision

Using SVGs

Ignition can import SVGs (Scalable Vector Graphic) into a Vision window. Once imported, SVGs can be modified and styled. To use an SVG in your project, simply drag the SVG file directly onto the window you want the SVG to appear on. The SVG becomes a new polygon component on the window.

Sometimes the way the SVG imports may result in the SVG appearing very small, in which case you can manually expand the SVG to your desired size.

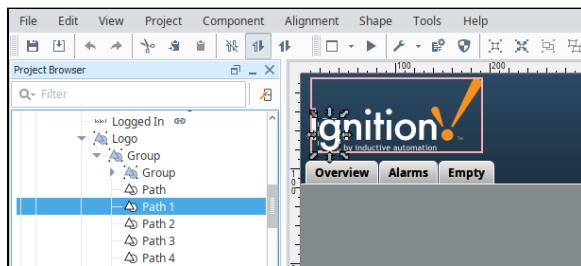
Note:

Some elements, attributes, and properties in an SVG are not supported.

The Vision module uses the Apache Batik library to handle SVGs, so a list of supported elements, attributes, and properties can be found on [Apache Batik's website](#).

SVGs as Grouped Components

All SVG images are made up of a group or groups of several (and often many!) paths. These paths are Ignition's [Drawing Tools](#), and are the basic building blocks of all SVGs in Ignition. You can select each path individually from the Project Browser, or by double-clicking on an SVG then single clicking on an object inside it.



On this page ...

- [Using SVGs](#)
 - [SVGs as Grouped Components](#)
- [Coloring an SVG](#)
 - [Coloring SVG Parts](#)
 - [Coloring SVG Example](#)
- [SVG Tinting](#)
 - [Tinting Example](#)
- [Using Images](#)
 - [Using the Image Management Tool](#)



Scalable Vector Graphics (SVGs)

[Watch the Video](#)

Coloring an SVG

With SVGs, one useful HMI technique is to color the SVG (Scalable Vector Graphic) to show the state of whatever the SVG represents. Whether you are bringing in a vector graphic from the Symbol Factory or importing an SVG from your computer, you can easily color the SVG to suit your needs. There are two ways of coloring an SVG: coloring an individual piece of the SVG, or placing a tint over the whole SVG.

Coloring SVG Parts

Individual pieces of the SVG can be pulled out and colored, by finding the path that corresponds with the part of the SVG that you want colored and applying a color to it. This can be done for a single piece of the SVG, or multiple different pieces. These colors can even be made dynamic by setting up a binding on the Fill Color. Since the property is expecting a value with a data type of [Color](#), you can either set up an expression binding which uses the `color()` function to create a color object, or use Ignition's built in [Number to Color Translator](#), which will automatically be made available to you when selecting a binding that will typically return a value, such as a Tag or Property binding.

Coloring SVG Example

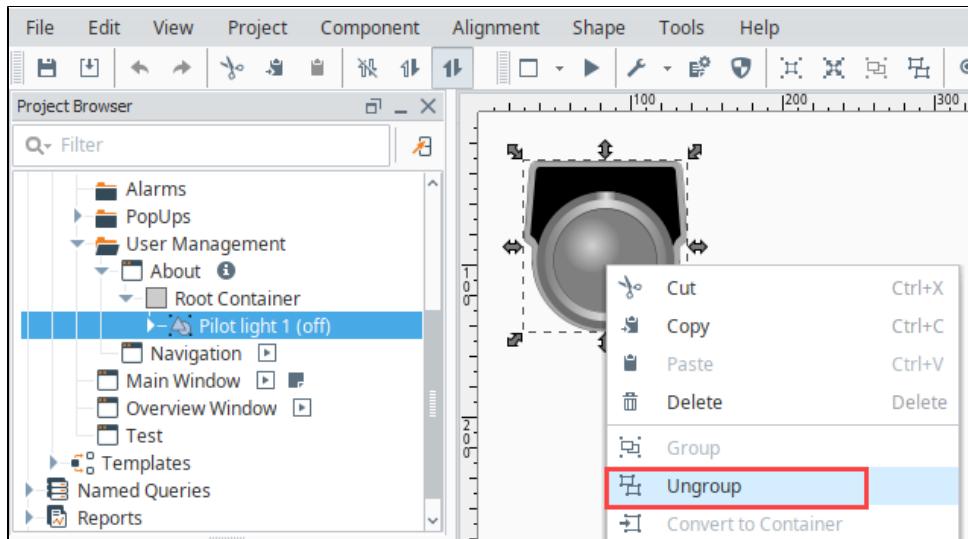
In this example, we selected an individual piece in a push button symbol and added a color to that area.

1. Place a push button image or any image from Symbol Factory onto the window.
2. Right click on the image and select **Ungroup**.

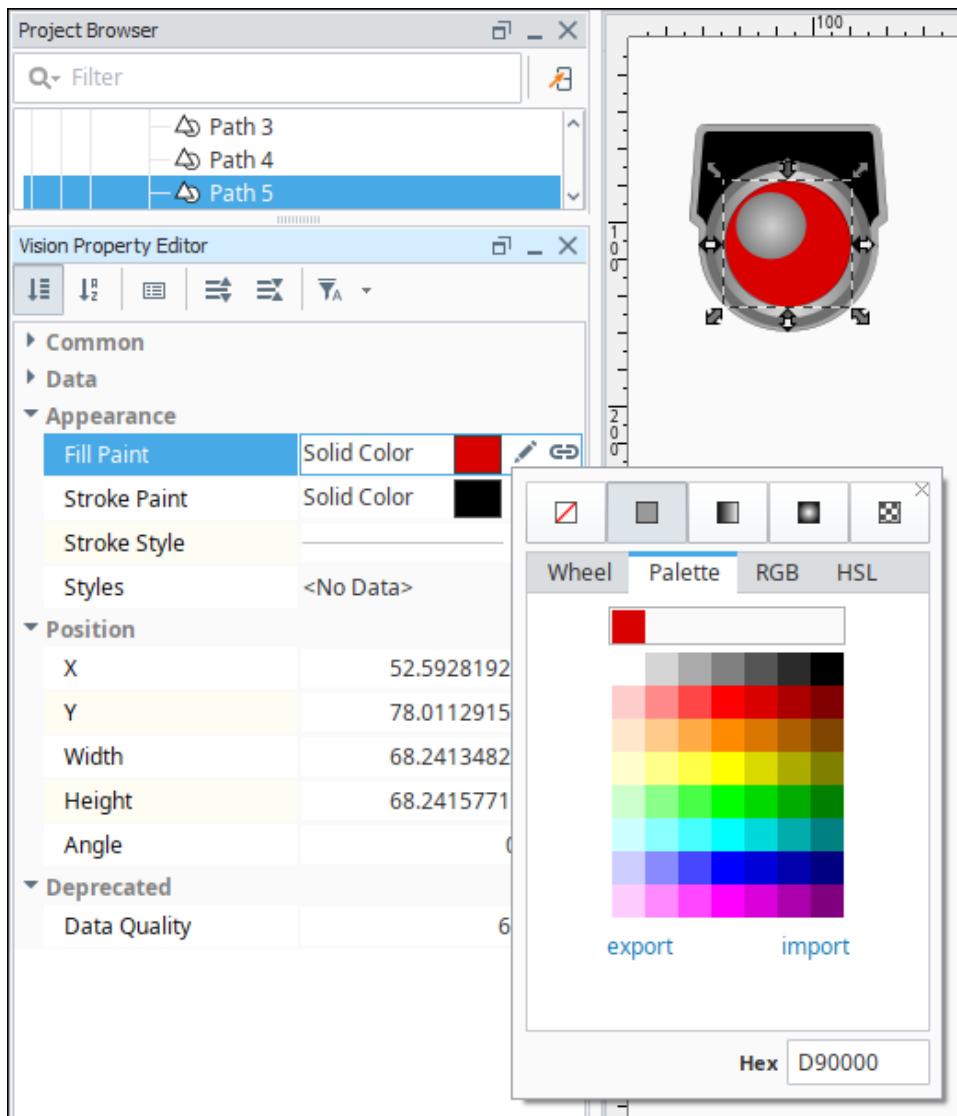


Color SVG Dynamically

[Watch the Video](#)

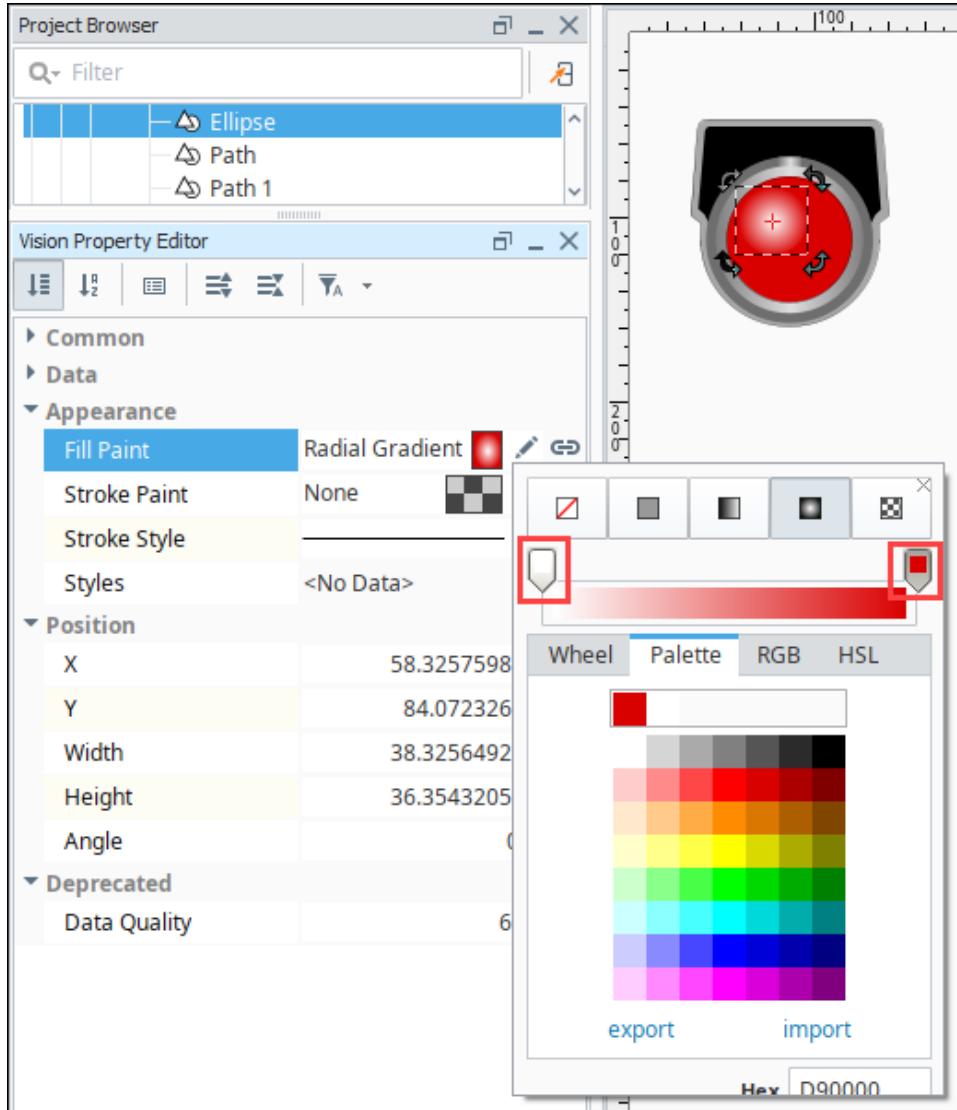


3. Right click in the center of the image and select **Ungroup** again.
4. Click on the larger inner circle in the image, then select the Fill Paint color you want. We chose red #D90000.

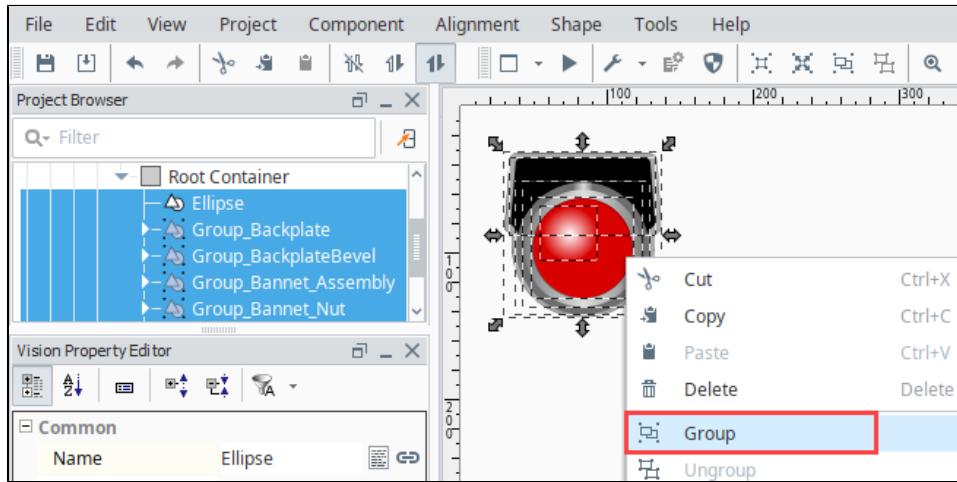


5. To get color in the "highlight" section of the graphic, click on the highlight. You'll see that the Fill Paint is a Radial Gradient. Click on the **Edit** icon to change the colors in the gradient.

6. Click on the left side stop and select white. Click on the right side stop and select red.



7. To regroup all the parts of the SVG, drag a box around them with your mouse. Then right click and select Group.



Alternately, you can create a style using the [Style Customizer](#) to change the color of an SVG path based on a driving property on that path. This is typically done by setting up a custom property to use as the driving property, and then binding the custom property to the property or Tag that will drive the color change.

SVG Tinting

Because an SVG is typically composed of many smaller shapes, it is difficult to color the entire object, as it would require changing the color on every shape within the SVG. So instead of changing the color of the entire SVG, we can make a new shape that is the same shape and size as the SVG, and make it opaque, so that its color acts like a tint on top of the SVG. A clever way to do this is to duplicate the SVG, union together the new SVG, so that all shapes combine into one shape, and then change the color.



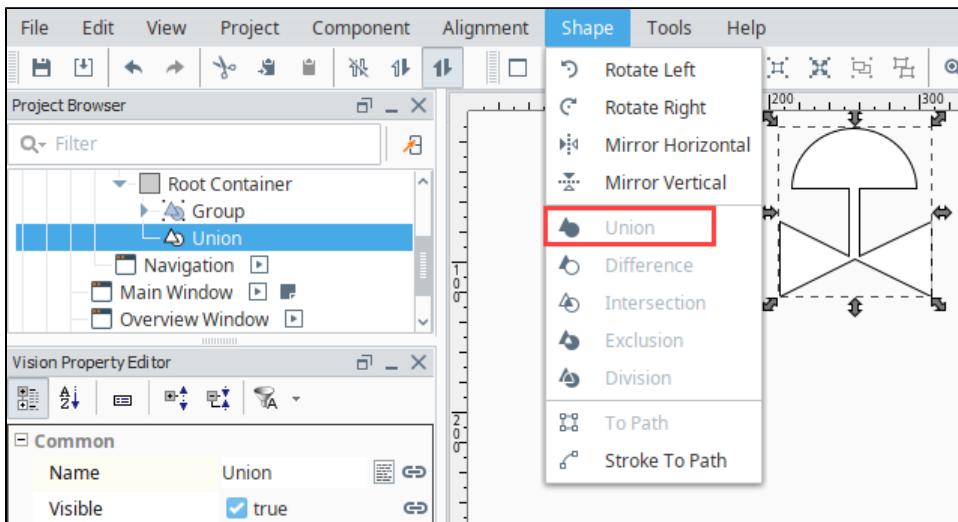
Tint SVG

[Watch the Video](#)

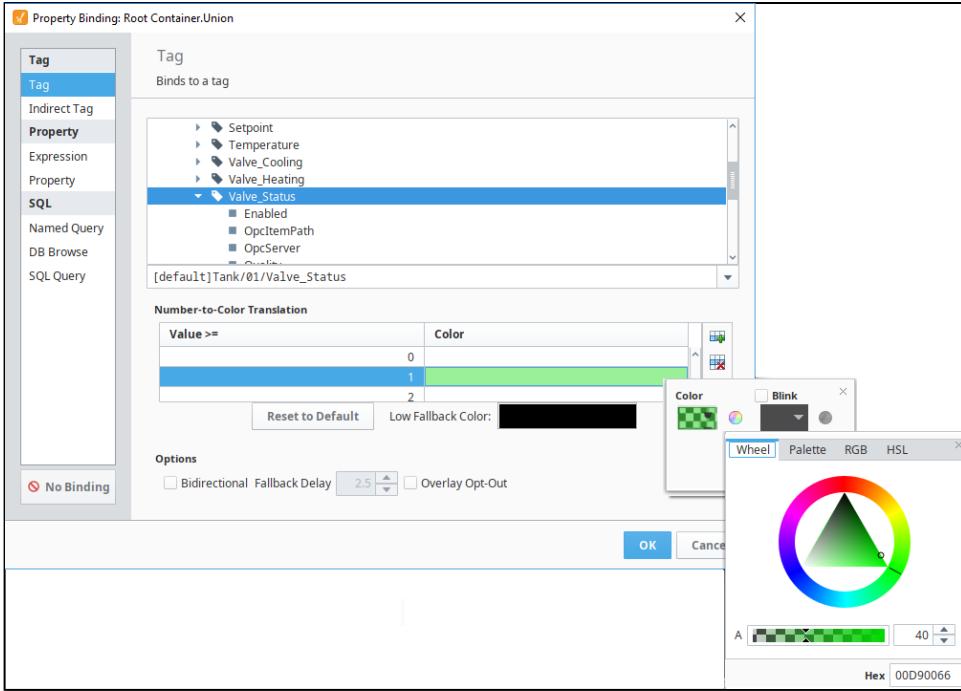
Tinting Example

Let's say you chose one of the many grayscale symbols, such as the **3-D Valve** symbol from the **Valves** category, and you want to tint the valve green when the valve is open, red when the valve has a fault, and keep it gray when the valve is closed. Suppose you have a Tag called **ValveStatus**, that is 0 for closed, 1 for open, and 2 for faulted.

1. Drag the symbol onto the screen.
2. Duplicate the symbol by selecting it and choosing **Duplicate** from the **Edit** menu, or pressing **CTRL-D**.
3. Now, select the duplicate symbol, which will be above the original.
4. Click the **Union** icon in the toolbar or find the **Union** item under the **Shape** menu.
This will combine the duplicate SVG into a single shape.



5. Remove the outline by setting the **Stroke Paint** property to **No Paint**
6. Click on the **Binding** icon next to the Fill Paint property.
7. Select the **Tag** binding type.
8. Navigate to the Tag you want to use. For this example, we used a **Valve_Status** Tag.
9. In the Number-to-Color Translation, double click the color next to the Value 0 and select white.
10. Click the Add New Translation icon. Set the Value 1 at 40% opaque green.
11. Click the Add New Translation icon. Set the Value 2 at 40% opaque red.



12. Click **OK** to save the binding.
13. On the window, place another copy of the original symbol.
14. Select the colored symbol and select **Alignment > Move to Front**.
15. Next place the colored symbol on top of the original.
16. Select them both, then select **Component > Group**.



In summary, what we did to tint the symbol was to make a flat shape that had the exact same outline as the symbol, and used semi-transparent fills to achieve a tint effect for the underlying symbol.

Using Images

Images can be very useful for displaying important information, such as giving visual representations of real world objects. There are a few ways that images can be brought into a Vision project. The first is by pulling images from The Image Management Tool, where the images are stored in the Gateway. The other way is to grab images using a filepath.

Using the Image Management Tool

Bringing in images using the Image Management Tool is easy.

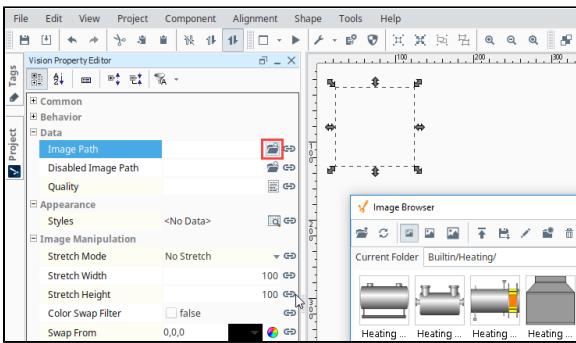
1. Place an Image component on the Window.
2. In the Vision Property Editor, scroll down to Data and click on the **Folder Search** icon next to the Image Path property. This will bring up the [Image Management Tool](#).



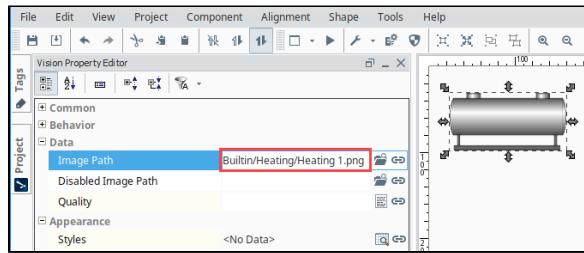
Images (png, jpg, gif)

[Watch the Video](#)

3. In the Image Management Tool, find the image you want, and double click to select it.



The path to the image is now displayed in the components **ImagePath** property. Images can be displayed in the image component, but they can also be used in components like labels and buttons.



Note: You can add any images you want to the [Image Management Tool](#), which are stored on the Gateway.

Instead of using images in the Image Management Tool in Image Path properties of components, you can use the file path to a local image. This is done by prefixing the file path with `file:///`. An example Image Path would look like this:
Using Local Images

```
file:///C:/Users/Public/Pictures/Sample Pictures/Chrysanthemum.jpg
```

It is important to understand that this will only work if the image is accessible from where the client is running. So if you access an image from the Designer on the local machine, clients that launch elsewhere may not have the image stored in the same location. For this reason, we recommend storing the images in a location that everyone can reach, such as a shared drive.

Note: When working with images found online, make sure to follow all applicable copyright laws.

Related Topics ...

- [Symbol Factory](#)
- [Image Management Tool](#)

Adding Icons to Labels and Buttons

[Watch the Video](#)

Comparison Charts

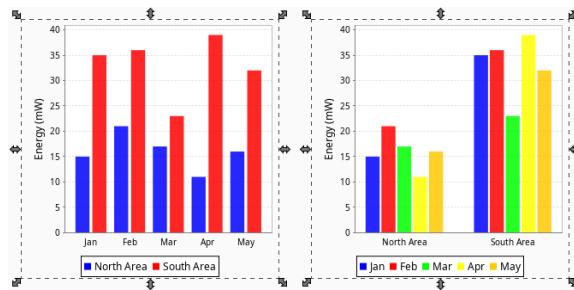
Overview

This page provides an overview of the various comparison charts in Ignition, or charts that allow you to compare different sets of data. Comparison charts differ from trending charts in that they tend to utilize a timestamp to visualize records over a period of time. Several types of comparison charts and how they are used are described on this page.

Bar Chart

The [Bar Chart](#) is an easy-to-use chart that is driven by the **Data** property, and expects a specific format. The first column in the Data property defines the names of the categories, and each additional column defines the groupings for each item in the series (depending on the Extract Order).

Note that additional datasets may not be added to the Bar Chart, so all values must be aggregated into the Data property via SQL Binding, or scripting. If multiple datasets are desired, then the [Classic Chart](#) configured with a bar renderer should be used instead.



Using the Bar Chart

Typically, data is pulled into the chart from a database using either a [Named Query](#) or [SQL Query](#) binding on the **Data** property. This data is typically category based, which typically means there is no timestamp. Generally, if values are split up by time, it is into large chunks of time, like the different months as shown in the example above.

Initial Dataset

When a Bar Chart is first created, the component will contain a dataset that looks like the following:

On this page ...

- [Overview](#)
- [Bar Chart](#)
 - [Using the Bar Chart](#)
 - [Initial Dataset](#)
 - [Extract Order](#)
- [Chart](#)
 - [Using the Chart](#)
 - [Initial Dataset](#)
- [Radar Chart](#)
 - [Using the Radar Chart](#)
 - [Initial Dataset](#)
 - [Min and Max](#)
- [Pie Chart](#)
 - [Using the Pie Chart](#)
 - [Initial Dataset](#)
 - [Extract Order](#)
- [Box and Whisker Chart](#)
 - [Box Anatomy](#)
 - [Using the Box and Whisker Chart](#)
- [Initial Dataset](#)

Dataset Viewer X

| Label | North Area | South Area | |
|-------|------------|------------|--|
| Jan | 15 | 35 | |
| Feb | 21 | 36 | |
| Mar | 17 | 23 | |
| Apr | 11 | 39 | |
| May | 16 | 32 | |

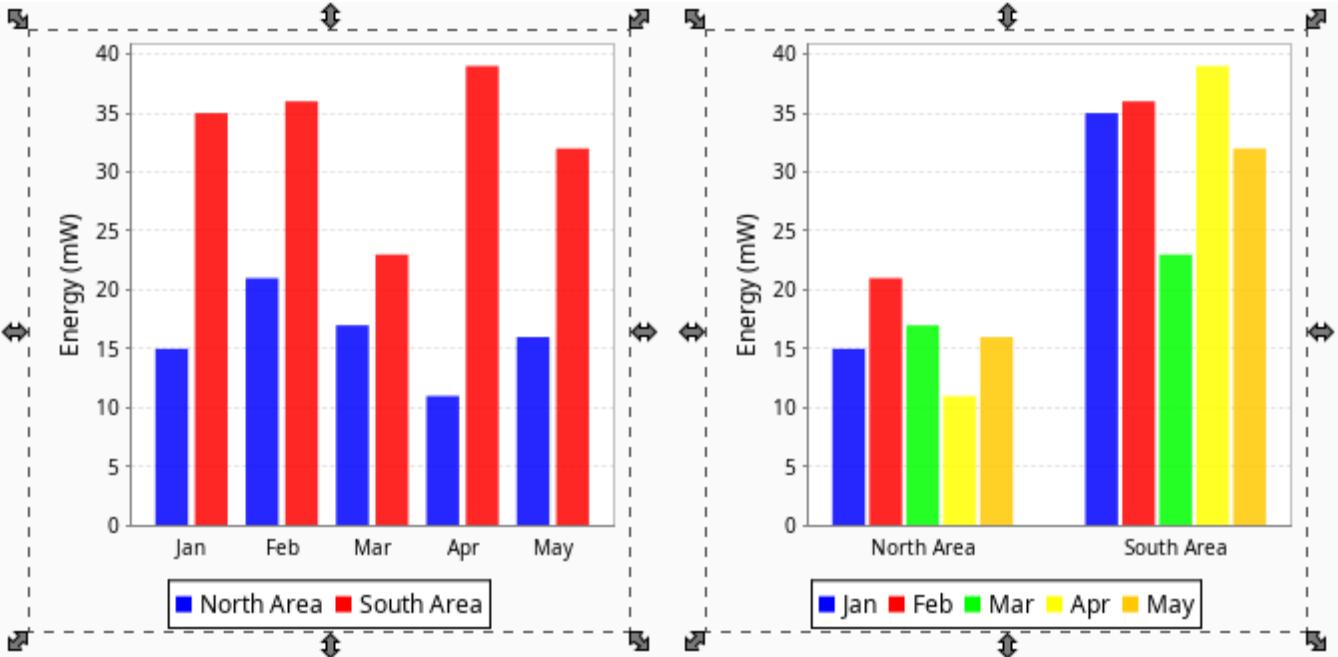
Column Name: ---- Column Type: ----

OK Cancel

```
"#NAMES"
"Label", "North Area", "South Area"
"#TYPES"
"str", "I", "I"
"#ROWS", "5"
"Jan", "15", "35"
"Feb", "21", "36"
"Mar", "17", "23"
"Apr", "11", "39"
"May", "16", "32"
```

Extract Order

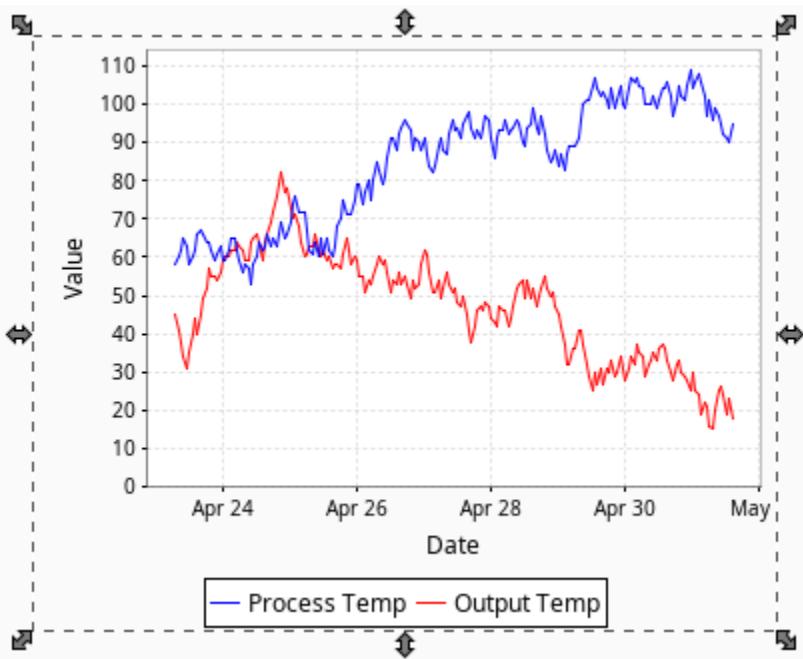
The **Extract Order** property on the chart determines how data series are defined. By default, the property is set to **By Row**, which means each row is a **series**, and each column (except the first) is a **category**. Based on the initial dataset, datapoints are grouped by area (column) and then grouped by each month (row). If we set the **Extract Order** property to **By Column**, then we see that each data point is grouped by the month (row), and then grouped by each area (column). Note that the underlying data has not changed, but rather how it is rendered as shown in the images below.



Chart

The [Chart](#) component, also known as the Classic Chart can be used to create many different types of charts by rendering the data in different ways. This means that depending on the type of data you have, you may use the Chart component in very different ways.

By default the chart is setup to be used as a [time series chart](#), with default data that shows this behavior. However with the right data, the chart can also display an XY coordinate plot or a categorical plot.



Using the Chart

The default settings on the Chart allow it to be used as a time series chart. Simply alter the dataset in the Data property with new time series data to display it in the chart. You can also alter the renderer in the Dataset Properties to any of the XY renderers to change the way the data is displayed.

To use the chart as an XY coordinate plot, data should be loaded with a two column dataset, where one column is the Y coordinate, and another is the corresponding X coordinate. The chart will also need to be setup with a new X axis, since the default axis is a date axis.

To use the Chart as a categorical plot, a few things need to change from the defaults. The Data property will need to be loaded with some categorical data. Categorical data will have one column of the dataset be categories of information in the form of a string. The chart will also need to be setup with

a new categorical X axis, as well as a category renderer in the Chart Customizer. Lastly, the Chart Type will need to be a Category Chart. When the chart is a Category Chart type, the Extract Order property can be changed to alter how the data is pulled out and displayed in the chart. It works very similarly to the Bar Charts Extract Order seen above.

Initial Dataset

Each new Chart randomly generates a new dataset. This Data property will use the default timeseries behavior of the chart, with a t_stamp column for the domain, and two other columns (Process Temp and Output Temp) as values at the specified times.

```
"#NAMES"
"t_stamp", "Process Temp", "Output Temp"
"#TYPES"
"date", "I", "I"
"#ROWS", "200"
"2018-04-30 00:07:15", "64", "35"
"2018-04-30 01:07:15", "60", "35"
"2018-04-30 02:07:15", "56", "36"
"2018-04-30 03:07:15", "52", "31"
"2018-04-30 04:07:15", "53", "26"
"2018-04-30 05:07:15", "57", "28"
"2018-04-30 06:07:15", "60", "27"
"2018-04-30 07:07:15", "57", "26"
"2018-04-30 08:07:15", "59", "31"
"2018-04-30 09:07:15", "57", "36"
"2018-04-30 10:07:15", "55", "39"
"2018-04-30 11:07:15", "52", "41"
"2018-04-30 12:07:15", "56", "40"
"2018-04-30 13:07:15", "51", "41"
"2018-04-30 14:07:15", "52", "36"
"2018-04-30 15:07:15", "53", "32"
"2018-04-30 16:07:15", "57", "30"
"2018-04-30 17:07:15", "52", "32"
"2018-04-30 18:07:15", "57", "32"
"2018-04-30 19:07:15", "55", "29"
"2018-04-30 20:07:15", "53", "30"
"2018-04-30 21:07:15", "54", "31"
"2018-04-30 22:07:15", "50", "29"
"2018-04-30 23:07:15", "54", "25"
"2018-05-01 00:07:15", "49", "21"
"2018-05-01 01:07:15", "53", "21"
"2018-05-01 02:07:15", "50", "16"
"2018-05-01 03:07:15", "51", "19"
"2018-05-01 04:07:15", "49", "23"
"2018-05-01 05:07:15", "48", "25"
"2018-05-01 06:07:15", "51", "20"
"2018-05-01 07:07:15", "55", "18"
"2018-05-01 08:07:15", "50", "22"
"2018-05-01 09:07:15", "49", "26"
"2018-05-01 10:07:15", "53", "22"
"2018-05-01 11:07:15", "50", "27"
"2018-05-01 12:07:15", "46", "26"
"2018-05-01 13:07:15", "48", "27"
"2018-05-01 14:07:15", "52", "26"
"2018-05-01 15:07:15", "51", "24"
"2018-05-01 16:07:15", "55", "24"
"2018-05-01 17:07:15", "58", "23"
"2018-05-01 18:07:15", "61", "22"
"2018-05-01 19:07:15", "60", "27"
"2018-05-01 20:07:15", "59", "32"
"2018-05-01 21:07:15", "60", "33"
"2018-05-01 22:07:15", "56", "38"
"2018-05-01 23:07:15", "51", "35"
"2018-05-02 00:07:15", "47", "34"
"2018-05-02 01:07:15", "45", "32"
"2018-05-02 02:07:15", "46", "27"
"2018-05-02 03:07:15", "43", "31"
"2018-05-02 04:07:15", "39", "33"
"2018-05-02 05:07:15", "39", "30"
"2018-05-02 06:07:15", "40", "26"
"2018-05-02 07:07:15", "41", "27"
```

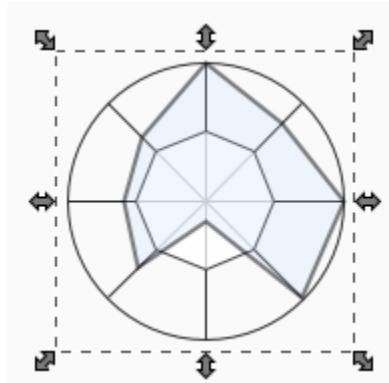
"2018-05-02 08:07:15", "46", "29"
"2018-05-02 09:07:15", "43", "25"
"2018-05-02 10:07:15", "47", "24"
"2018-05-02 11:07:15", "49", "19"
"2018-05-02 12:07:15", "45", "19"
"2018-05-02 13:07:15", "42", "20"
"2018-05-02 14:07:15", "43", "20"
"2018-05-02 15:07:15", "43", "23"
"2018-05-02 16:07:15", "39", "25"
"2018-05-02 17:07:15", "37", "22"
"2018-05-02 18:07:15", "33", "22"
"2018-05-02 19:07:15", "31", "21"
"2018-05-02 20:07:15", "35", "22"
"2018-05-02 21:07:15", "34", "21"
"2018-05-02 22:07:15", "30", "22"
"2018-05-02 23:07:15", "34", "27"
"2018-05-03 00:07:15", "35", "27"
"2018-05-03 01:07:15", "36", "32"
"2018-05-03 02:07:15", "39", "32"
"2018-05-03 03:07:15", "39", "33"
"2018-05-03 04:07:15", "41", "38"
"2018-05-03 05:07:15", "40", "35"
"2018-05-03 06:07:15", "36", "39"
"2018-05-03 07:07:15", "38", "41"
"2018-05-03 08:07:15", "33", "41"
"2018-05-03 09:07:15", "30", "38"
"2018-05-03 10:07:15", "30", "42"
"2018-05-03 11:07:15", "33", "38"
"2018-05-03 12:07:15", "37", "42"
"2018-05-03 13:07:15", "33", "37"
"2018-05-03 14:07:15", "29", "38"
"2018-05-03 15:07:15", "24", "37"
"2018-05-03 16:07:15", "24", "42"
"2018-05-03 17:07:15", "21", "45"
"2018-05-03 18:07:15", "24", "44"
"2018-05-03 19:07:15", "28", "49"
"2018-05-03 20:07:15", "24", "45"
"2018-05-03 21:07:15", "24", "49"
"2018-05-03 22:07:15", "19", "51"
"2018-05-03 23:07:15", "24", "48"
"2018-05-04 00:07:15", "19", "45"
"2018-05-04 01:07:15", "16", "44"
"2018-05-04 02:07:15", "20", "40"
"2018-05-04 03:07:15", "25", "38"
"2018-05-04 04:07:15", "29", "40"
"2018-05-04 05:07:15", "27", "36"
"2018-05-04 06:07:15", "24", "36"
"2018-05-04 07:07:15", "29", "41"
"2018-05-04 08:07:15", "34", "45"
"2018-05-04 09:07:15", "37", "47"
"2018-05-04 10:07:15", "40", "48"
"2018-05-04 11:07:15", "42", "52"
"2018-05-04 12:07:15", "45", "57"
"2018-05-04 13:07:15", "46", "58"
"2018-05-04 14:07:15", "51", "59"
"2018-05-04 15:07:15", "46", "56"
"2018-05-04 16:07:15", "46", "59"
"2018-05-04 17:07:15", "47", "56"
"2018-05-04 18:07:15", "43", "56"
"2018-05-04 19:07:15", "46", "53"
"2018-05-04 20:07:15", "49", "55"
"2018-05-04 21:07:15", "51", "51"
"2018-05-04 22:07:15", "46", "51"
"2018-05-04 23:07:15", "50", "50"
"2018-05-05 00:07:15", "52", "50"
"2018-05-05 01:07:15", "51", "51"
"2018-05-05 02:07:15", "49", "51"
"2018-05-05 03:07:15", "46", "55"
"2018-05-05 04:07:15", "51", "54"
"2018-05-05 05:07:15", "56", "52"
"2018-05-05 06:07:15", "61", "54"

"2018-05-05 07:07:15", "62", "52"
"2018-05-05 08:07:15", "57", "47"
"2018-05-05 09:07:15", "54", "47"
"2018-05-05 10:07:15", "59", "46"
"2018-05-05 11:07:15", "56", "44"
"2018-05-05 12:07:15", "58", "46"
"2018-05-05 13:07:15", "62", "44"
"2018-05-05 14:07:15", "64", "41"
"2018-05-05 15:07:15", "62", "45"
"2018-05-05 16:07:15", "66", "42"
"2018-05-05 17:07:15", "61", "37"
"2018-05-05 18:07:15", "63", "38"
"2018-05-05 19:07:15", "61", "38"
"2018-05-05 20:07:15", "64", "40"
"2018-05-05 21:07:15", "64", "44"
"2018-05-05 22:07:15", "60", "40"
"2018-05-05 23:07:15", "64", "44"
"2018-05-06 00:07:15", "63", "45"
"2018-05-06 01:07:15", "61", "47"
"2018-05-06 02:07:15", "61", "52"
"2018-05-06 03:07:15", "61", "48"
"2018-05-06 04:07:15", "61", "47"
"2018-05-06 05:07:15", "60", "46"
"2018-05-06 06:07:15", "58", "44"
"2018-05-06 07:07:15", "56", "43"
"2018-05-06 08:07:15", "61", "45"
"2018-05-06 09:07:15", "66", "49"
"2018-05-06 10:07:15", "68", "51"
"2018-05-06 11:07:15", "63", "54"
"2018-05-06 12:07:15", "66", "58"
"2018-05-06 13:07:15", "69", "63"
"2018-05-06 14:07:15", "69", "63"
"2018-05-06 15:07:15", "67", "58"
"2018-05-06 16:07:15", "71", "54"
"2018-05-06 17:07:15", "74", "50"
"2018-05-06 18:07:15", "79", "49"
"2018-05-06 19:07:15", "75", "51"
"2018-05-06 20:07:15", "80", "49"
"2018-05-06 21:07:15", "79", "50"
"2018-05-06 22:07:15", "82", "50"
"2018-05-06 23:07:15", "80", "53"
"2018-05-07 00:07:15", "85", "54"
"2018-05-07 01:07:15", "87", "49"
"2018-05-07 02:07:15", "87", "51"
"2018-05-07 03:07:15", "84", "56"
"2018-05-07 04:07:15", "82", "60"
"2018-05-07 05:07:15", "81", "57"
"2018-05-07 06:07:15", "83", "55"
"2018-05-07 07:07:15", "83", "55"
"2018-05-07 08:07:15", "81", "52"
"2018-05-07 09:07:15", "77", "49"
"2018-05-07 10:07:15", "75", "46"
"2018-05-07 11:07:15", "79", "45"
"2018-05-07 12:07:15", "82", "47"
"2018-05-07 13:07:15", "81", "48"
"2018-05-07 14:07:15", "82", "53"
"2018-05-07 15:07:15", "81", "48"
"2018-05-07 16:07:15", "81", "43"
"2018-05-07 17:07:15", "85", "40"
"2018-05-07 18:07:15", "90", "37"
"2018-05-07 19:07:15", "94", "34"
"2018-05-07 20:07:15", "90", "38"
"2018-05-07 21:07:15", "89", "40"
"2018-05-07 22:07:15", "85", "45"
"2018-05-07 23:07:15", "81", "48"
"2018-05-08 00:07:15", "83", "43"
"2018-05-08 01:07:15", "78", "42"
"2018-05-08 02:07:15", "73", "40"
"2018-05-08 03:07:15", "72", "44"
"2018-05-08 04:07:15", "71", "42"
"2018-05-08 05:07:15", "75", "44"

```
"2018-05-08 06:07:15", "71", "46"
"2018-05-08 07:07:15", "67", "50"
```

Radar Chart

Radar Charts, also known as web charts, spider charts, and spider plots, are useful for displaying values that are out of spec, and several of them at once. Each value is plotted on a separate axis with the middle of the axis representing the ideal value. The chart draws a line between the different values, which create a shape that changes as those values change. Inside the chart, there is a polygon that represents what the chart would look like if all of its values were in their ideal range. A good use of radar charts is to display realtime information in such a way that outliers can be quickly identified. This can be an efficient way to convey if a process is running on-spec or off-spec at a glance. So the Radar Chart lets you quickly see where the values are in comparison to where they should ideally be.



Using the Radar Chart

The Radar Chart can be used to show realtime values by dragging and dropping Tags from the **Tag Browser** on to the chart. Doing so will create a **Ce II Update** binding on the Data property that is tied to the Value, EngLow, and EngHigh properties on the Tag. Adding additional Tags will add additional spokes to the chart. Alternatively, a **Named Query** or a **SQL Query** binding on the data property can be used to display historical values, or aggregate previous historical values.

Initial Dataset

Each new Radar Chart randomly generates a new dataset. The **Data** property on the Radar Chart must have at least a **Value**, **Min**, and **Max** column. Any additional columns are ignored.

To render properly, the dataset must have at least three rows. Datasets with only one or two rows will be drawn as a vertical line.

Dataset Viewer

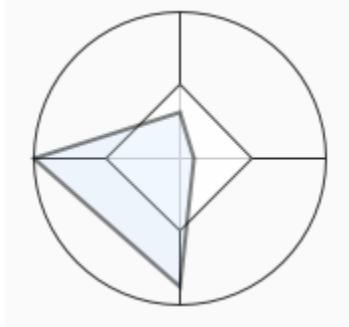
| Value | Min | Max |
|--------|-----|--------|
| 89.599 | 21 | 89.599 |
| 72.625 | 5 | 91 |
| 98.097 | 25 | 98.097 |
| 86.972 | 19 | 86.972 |
| 33.674 | 23 | 96 |
| 63.141 | 14 | 86 |
| 57.073 | 20 | 83 |
| 58.821 | 14 | 83 |

OK
Cancel

Min and Max

The **Min** and **Max** column, aside from determining the limits on the chart, are also used to determine the desired value. The **Desired** value is drawn as the midpoint between the **Min** and **Max** for a single row in the Dataset. Each row of the dataset has a **Min** and **Max** column. The values in these columns are used to determine the scale of the spoke for that variable with the midpoint representing the desired value.

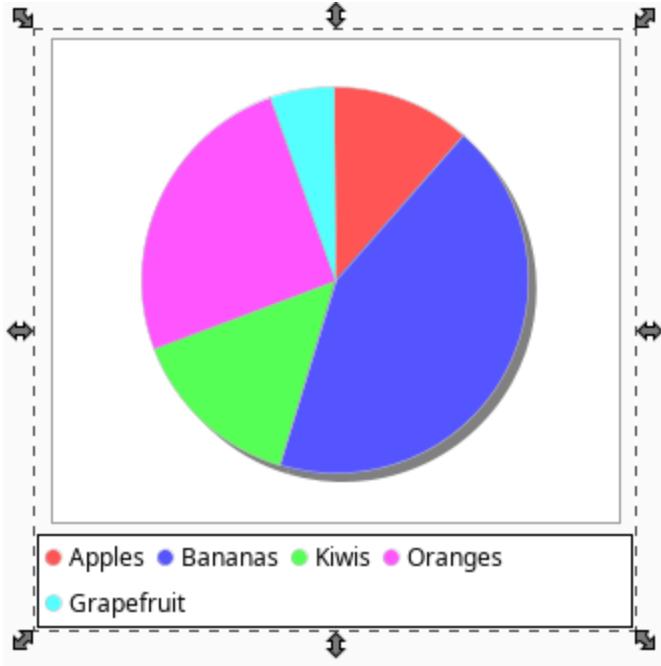
Below we see the white polygon in the center of the chart. This represents the midpoint between **Min** and **Max**



```
"#NAMES"
"Value", "Min", "Max"
"#TYPES"
"D", "D", "D"
"#ROWS", "8"
"41.51196715968135", "18.0", "86.0"
"72.21343683086239", "2.0", "88.0"
"98.91484924220774", "16.0", "98.91484924220774"
"23.189112936965692", "1.0", "78.0"
"33.45468212322838", "23.0", "82.0"
"77.17126241429432", "7.0", "100.0"
"53.529302336166836", "25.0", "79.0"
"62.058120439146435", "6.0", "94.0"
```

Pie Chart

A [Pie Chart](#) displays values from several categories, each category is a separate "wedge" of the chart. The total is the sum of all wedges. The key to the Pie Chart component is the Data property, which contains the items that will be displayed as pie wedges.



Using the Pie Chart

Typically, data is pulled into the chart from a database using either a **Named Query** or **SQL Query** binding on the **Data** property. The data typically consists of a list of name-value pairs of things that are related.

Initial Dataset

The Pie Chart component contains an initial dataset with two columns, **Label** and **Value**. As the name implies, the **Label** column determines the text associated with each wedge of the pie, while the value is the weight of the wedge.

Dataset Viewer

| Label | Value |
|------------|-------|
| Apples | 15 |
| Bananas | 56 |
| Kiwis | 19 |
| Oranges | 33 |
| Grapefruit | 7 |

Column Name: ---- Column Type: ----

OK Cancel

```
"#NAMES"
"Label", "Value"
"#TYPES"
"str", "I"
```

```

"#ROWS", "5"
"Apples", "15"
"Bananas", "56"
"Kiwis", "19"
"Oranges", "33"
"Grapefruit", "7"

```

Extract Order

When **Extract Order** is set to **By Row**, then the data must be formatted differently. This order expects each column to be a wedge. Note that only the first row is utilized when extracting by row: subsequent rows are ignored.

```

"#NAMES"
"Grapefruit", "Apples", "Bananas", "Kiwis", "Oranges"
"#TYPES"
"I", "I", "I", "I", "I"
"#ROWS", "1"
"7", "15", "56", "19", "33"

```

Box and Whisker Chart

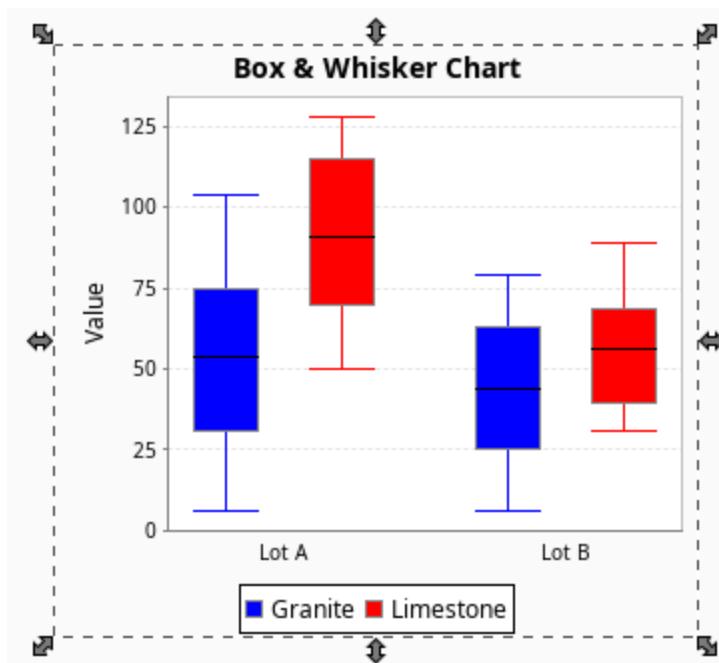
A [Box and Whisker Chart](#) displays pertinent statistical information about sets of data. Each 'Box and Whisker' item on the chart should represent a large amount of data: The high, low, median, and where the middle 50% of the data falls. The dataset that is required for this chart type will be all of your raw data, and it will calculate the box and whiskers for you.

Box Anatomy

The upper and lower bounds of each box (the colored in parts) represent the 1st and 3rd quartiles (quarters of a dataset range). This means the space filled in by the box is 50% of your raw data.

The horizontal line inside of the box represents the median (middle) value.

The lines that stick out above and below the box (whiskers), represent the minimum and maximum values from the raw data.



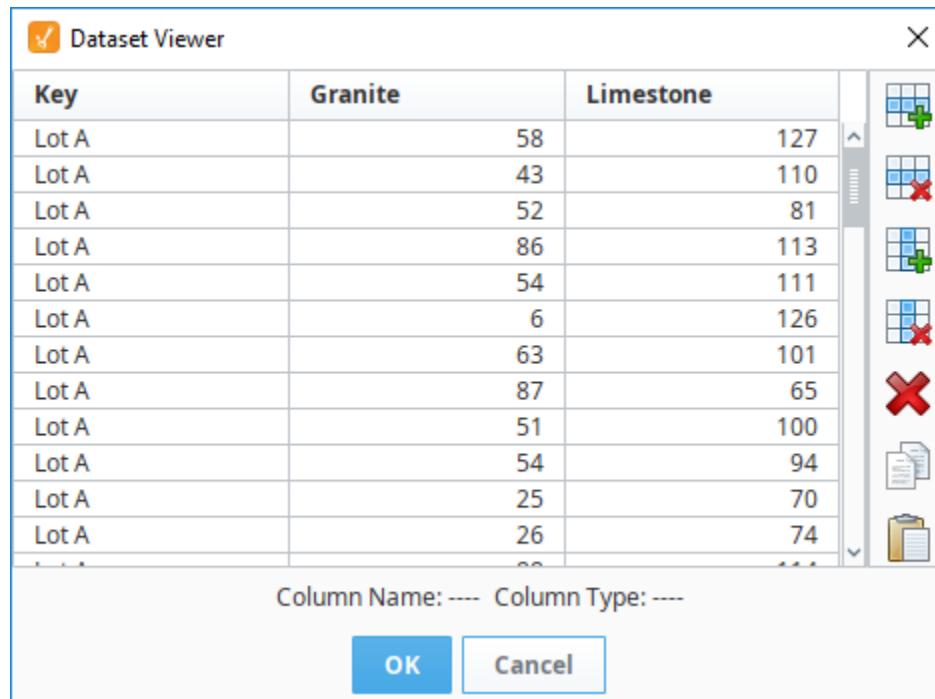
Using the Box and Whisker Chart

Typically, data is pulled into the chart from a database using either a **Named Query** or **SQL Query** binding on the **Data** property. The data typically comes in categories separated by an optional key column, with each category containing multiple values.

Initial Dataset

The first column in the Box and Whisker Chart's dataset is the **Key** column. The Key column determines which series the data pertains to (domain labels). Values in the **Key** column are case sensitive.

The second and additional columns denote categories (legend labels). The initial dataset contains two categories: **Granite** and **Limestone**. Additional columns in the dataset would add additional boxes to the chart.



| Key | Granite | Limestone |
|-------|---------|-----------|
| Lot A | 58 | 127 |
| Lot A | 43 | 110 |
| Lot A | 52 | 81 |
| Lot A | 86 | 113 |
| Lot A | 54 | 111 |
| Lot A | 6 | 126 |
| Lot A | 63 | 101 |
| Lot A | 87 | 65 |
| Lot A | 51 | 100 |
| Lot A | 54 | 94 |
| Lot A | 25 | 70 |
| Lot A | 26 | 74 |
| ... | 22 | 22 |

Column Name: ---- Column Type: ----

OK Cancel

```
"#NAMES"
"Key","Granite","Limestone"
"#TYPES"
"str","I","I"
"#ROWS", "200"
"Lot A","28","108"
"Lot A","46","81"
"Lot A","103","57"
"Lot A","16","93"
"Lot A","41","91"
"Lot A","55","68"
"Lot A","23","93"
"Lot A","49","97"
"Lot A","36","69"
"Lot A","47","106"
"Lot A","75","86"
"Lot A","14","115"
"Lot A","42","70"
"Lot A","100","129"
"Lot A","16","118"
"Lot A","62","125"
"Lot A","14","51"
"Lot A","73","64"
"Lot A","35","55"
"Lot A","96","113"
"Lot A","50","93"
"Lot A","97","72"
"Lot A","7","80"
"Lot A","86","62"
"Lot A","87","78"
"Lot A","80","51"
"Lot A","100","94"
"Lot A","79","124"
```

"Lot A", "39", "107"
"Lot A", "16", "119"
"Lot A", "20", "60"
"Lot A", "50", "124"
"Lot A", "37", "50"
"Lot A", "36", "98"
"Lot A", "46", "77"
"Lot A", "33", "106"
"Lot A", "49", "75"
"Lot A", "84", "60"
"Lot A", "17", "94"
"Lot A", "44", "93"
"Lot A", "72", "105"
"Lot A", "35", "106"
"Lot A", "20", "119"
"Lot A", "90", "51"
"Lot A", "37", "88"
"Lot A", "75", "103"
"Lot A", "13", "104"
"Lot A", "47", "55"
"Lot A", "65", "126"
"Lot A", "32", "90"
"Lot A", "85", "126"
"Lot A", "95", "77"
"Lot A", "74", "123"
"Lot A", "104", "68"
"Lot A", "90", "109"
"Lot A", "63", "66"
"Lot A", "60", "90"
"Lot A", "28", "65"
"Lot A", "64", "69"
"Lot A", "55", "62"
"Lot A", "98", "64"
"Lot A", "69", "100"
"Lot A", "35", "110"
"Lot A", "31", "115"
"Lot A", "51", "106"
"Lot A", "16", "76"
"Lot A", "91", "93"
"Lot A", "90", "77"
"Lot A", "93", "64"
"Lot A", "98", "84"
"Lot A", "61", "95"
"Lot A", "65", "97"
"Lot A", "67", "54"
"Lot A", "80", "92"
"Lot A", "104", "123"
"Lot A", "104", "112"
"Lot A", "20", "71"
"Lot A", "95", "99"
"Lot A", "37", "98"
"Lot A", "91", "51"
"Lot A", "101", "106"
"Lot A", "68", "94"
"Lot A", "9", "96"
"Lot A", "14", "77"
"Lot A", "46", "95"
"Lot A", "45", "95"
"Lot A", "79", "90"
"Lot A", "92", "110"
"Lot A", "29", "80"
"Lot A", "42", "80"
"Lot A", "15", "126"
"Lot A", "68", "77"
"Lot A", "69", "98"
"Lot A", "52", "119"
"Lot A", "11", "72"
"Lot A", "14", "122"
"Lot A", "36", "115"
"Lot A", "41", "66"
"Lot A", "98", "73"

"Lot A", "46", "116"
"Lot B", "49", "75"
"Lot B", "33", "46"
"Lot B", "53", "32"
"Lot B", "51", "58"
"Lot B", "34", "81"
"Lot B", "44", "73"
"Lot B", "71", "43"
"Lot B", "64", "37"
"Lot B", "58", "77"
"Lot B", "35", "37"
"Lot B", "76", "88"
"Lot B", "11", "42"
"Lot B", "11", "64"
"Lot B", "28", "85"
"Lot B", "26", "58"
"Lot B", "78", "43"
"Lot B", "43", "69"
"Lot B", "66", "32"
"Lot B", "7", "42"
"Lot B", "17", "71"
"Lot B", "59", "68"
"Lot B", "7", "31"
"Lot B", "53", "48"
"Lot B", "20", "52"
"Lot B", "71", "58"
"Lot B", "57", "85"
"Lot B", "14", "61"
"Lot B", "34", "47"
"Lot B", "59", "74"
"Lot B", "78", "58"
"Lot B", "64", "81"
"Lot B", "19", "31"
"Lot B", "43", "48"
"Lot B", "58", "38"
"Lot B", "22", "48"
"Lot B", "20", "83"
"Lot B", "36", "61"
"Lot B", "40", "69"
"Lot B", "64", "50"
"Lot B", "67", "70"
"Lot B", "46", "36"
"Lot B", "9", "51"
"Lot B", "10", "41"
"Lot B", "66", "35"
"Lot B", "46", "44"
"Lot B", "10", "62"
"Lot B", "13", "35"
"Lot B", "74", "49"
"Lot B", "69", "64"
"Lot B", "15", "68"
"Lot B", "56", "38"
"Lot B", "35", "69"
"Lot B", "61", "37"
"Lot B", "25", "80"
"Lot B", "38", "89"
"Lot B", "79", "56"
"Lot B", "6", "64"
"Lot B", "49", "58"
"Lot B", "5", "54"
"Lot B", "6", "35"
"Lot B", "38", "75"
"Lot B", "6", "77"
"Lot B", "39", "36"
"Lot B", "27", "63"
"Lot B", "72", "78"
"Lot B", "55", "38"
"Lot B", "9", "36"
"Lot B", "40", "65"
"Lot B", "57", "76"
"Lot B", "65", "55"

```
"Lot B", "74", "81"
"Lot B", "47", "85"
"Lot B", "66", "84"
"Lot B", "10", "38"
"Lot B", "23", "53"
"Lot B", "79", "80"
"Lot B", "27", "58"
"Lot B", "71", "58"
"Lot B", "27", "32"
"Lot B", "73", "43"
"Lot B", "24", "57"
"Lot B", "27", "59"
"Lot B", "56", "30"
"Lot B", "32", "55"
"Lot B", "7", "40"
"Lot B", "20", "63"
"Lot B", "68", "74"
"Lot B", "64", "57"
"Lot B", "57", "31"
"Lot B", "54", "61"
"Lot B", "33", "35"
"Lot B", "61", "73"
"Lot B", "36", "61"
"Lot B", "26", "34"
"Lot B", "9", "59"
"Lot B", "47", "60"
"Lot B", "61", "86"
"Lot B", "45", "88"
"Lot B", "5", "87"
"Lot B", "6", "36"
```

Related Topics ...

- [Bar Chart](#)
- [Classic Chart](#)
- [Pie Chart](#)
- [Radar Chart](#)
- [Box and Whisker Chart](#)

HTML in Vision

HTML stands for HyperText Markup Language. It is commonly used to style text within web pages. The features that HTML brings to style web pages can be applied to many components within Ignition to style the text within components.

Using HTML in Components

Many Vision components display a text string. By default, a component's text is displayed in a single font and color and will not wrap when its content exceeds the space the component has made available to the text. However, you can use HTML if you want to mix fonts or colors within the text or if you want formatting such as multiple lines. HTML formatting can be used in Vision components such as buttons, labels, and tables. It can be used in common properties such as the mouse over text property.

To specify that a component's text has HTML formatting, just put the `<html>` element at the beginning of the text, then use any valid HTML element in the remainder.



Closing the HTML element is optional. In other words, there is no need to place a `</html>` at the end of your stylized text. Also, the HTML elements are not case sensitive.

Common HTML Elements

HTML Tags are the special characters that instruct text to become stylized differently than other text within the same text. The following table describes the most common HTML elements that you can use within Ignition.

| HTML Element | Name | Description |
|---|----------------|---|
| <code><html>...</html></code> | HTML | Initiates an html formatting. In most cases closing the html with <code></html></code> is optional. |
| <code>...</code> | Bold | Applies a bold style to the contents of these elements. |
| <code><u>...</u></code> | Underline | Underlines the text contained within the elements. |
| <code><s>...</s></code> | Strikethrough | Draws a line through the text contained within the elements. |
| <code> </code> | Break | Applies a line break at this specific location. |
| <code>...</code> | Ordered List | Places the text into an ordered list. Text inside list items are ordered by number. |
| <code>...</code> | Unordered List | Places the text into an unordered list. Text inside list items are ordered by bullets. |
| <code>...</code> | List Item | Used to represent a list item. Should be contained in an order list (<code></code>) or unordered list (<code></code>). |
| <code><center></code> | Center | Centers the contents of the text. Used directly after the HTML Tag (that is, <code><html><center>...</center></html></code>) |
| <code>...</code> | Font | Colors the contents red. Works with standard color names, hex numbers, or RGB numbers. |

Applying HTML to Components

In Vision, you can add HTML to the text property of any component such as, a label, button, or table. These examples aren't unique to their specific components, but can be used on any component that has a Text or Mouseover Text property.



A good rule of thumb for what can be html formatted is text on components that is used for display, not for input. So while the Label components have a Text property that accepts html formatting, the Text Field component's Text property does not accept html formatting, as a user may type into the component.

On this page ...

- [Using HTML in Components](#)
- [Common HTML Elements](#)
- [Applying HTML to Components](#)



INDUCTIVE
UNIVERSIT

HTML in Ignition

[Watch the Video](#)



INDUCTIVE
UNIVERSIT

Multi-Line Labels and Buttons

For example, individual words or phrases within the text can be made bold:

```
<html>This is a <b>bold</b> word
```

[Watch the Video](#)

You can also create a list, such as instructions in the Mouseover Text on a component:

HTML in mouseover property

```
<html>
These are the instructions:
<ol>
<li>Stop the process.</li>
<li>Check on this.</li>
<li>Remove that.</li>
</ol>
```

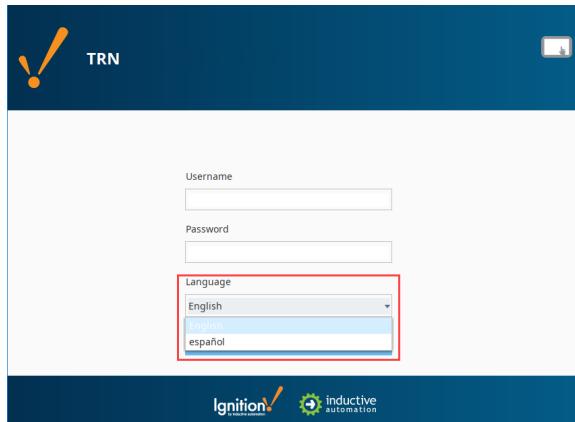
Localization in Vision

Localization in the Vision module utilizes the terms in the [platform's Translation system](#). Once terms have been defined, translations can be enabled by either component or scripting.

Selecting a Language on Client Startup

Once you create a second language, the Client Login Screen will automatically display a Language Selector where you can select your preferred language. There is Project Property setting that allows you to Show or Hide the Language Selector at login. By default, it is set to Automatic so you will see the Language Selector at login when two or more languages are created unless you choose to hide it.

If a user that has a preferred language selected in their [user profile settings](#), Ignition will login to the Client with their preferred language automatically.



On this page ...

- [Selecting a Language on Client Startup](#)
- [Using the Language Selector Component](#)



Switching the Current Language

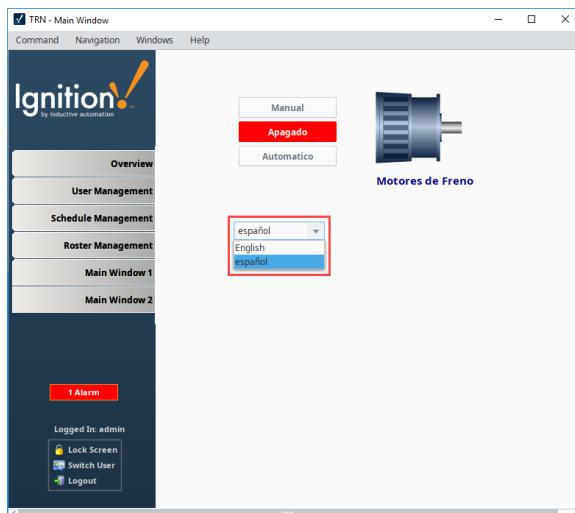
[Watch the Video](#)

Using the Language Selector Component

A single Language Selector on a window has the potential to trigger translations on all windows. There is no binding involved in selecting a language because they are compared directly against the Translation Manager database, so the component only needs to be placed onto a window after a second language has been defined.

The component also offers an easy way to switch languages without forcing the user to log out first. This way a single Language Selector component can exist on a navigation window and provide language translations for all components on all windows.

More information on the [Language Selector Component](#) can be found in the Appendix.



Related Topics ...

- [Localization and Languages](#)

Binding Types in Vision

Binding is perhaps the most important concept to understand when designing a project using the Vision module. It is primarily through property bindings that you bring windows to life and have them display useful things. A binding simply links one component's property to something else.

When you initially place a component on a screen, it doesn't really do anything. Changing its properties in the Designer will make it look or act different, but it has no connection to the real world and this is what bindings adds.

Binding, as its name suggests, lets you bind a property to something else, such as

- A Tag
- The results of a SQL query executed against a remote database
- Another component's property
- An expression involving any of these things

For example, bind the Value property of an LED Display to an OPC Tag, and voilà - the value property will always be the value of that Tag - creating a dynamic display. Bindings can also work the other way, using a [bidirectional binding](#). Bind the value of a numeric text box to a Tag, and that Tag will be written to when someone edits the value in the text box.

The power of bindings comes from the variety of different binding types that exist, and the fact that you can bind nearly any property of a component to anything else. Want its foreground to turn red when an alarm is above a certain severity? Bind its **LED Lit** property color to a Tag's **Alarms**.

HighestActivePriority property. Want it to only appear if a supervisor is on shift? Bind its visible property to the result of a SQL query that joins a personnel table with a shift table. The possibilities are nearly endless.

Property Binding Types

A property can have one of many different types of bindings. Instead of setting a label statically, the text might change based on a PLC value or on-screen selection. There are many ways to bind your components to show values from PLCs, databases, other components, or user input. You can even bind some or all of the properties on each component. You can bind component values using:

- **Property** simply binds one property to another. When that property changes, the new value is pushed into the property that the binding is setup on.
- **Tag** binds a property directly to a Tag property (typically the value) which sets up a Tag subscription for that Tag, and every time the chosen Tag property changes, the binding is evaluated, and pushes the new value into the bound property.
- **Indirect Tag** is similar to the standard Tag binding except that you can introduce any number of indirect parameters to build a tag path dynamically in the runtime.
- **Tag History** is used for Dataset type properties. It runs a query against the Tag Historian.
- **Expression** uses the simple [expression language](#) to calculate a value which can involve lots of dynamic data.
- **Named Query** executes a Named Query that had been previously created.
- **SQL Query** is a polling binding type that runs a SQL Query against any of the database connections configured in your Gateway.
- **Database Browse** is equivalent to the SQL Query binding except that it helps write the queries for you.
- **Cell Update** enables you to easily make one or more cells inside a dataset dynamic. This is useful for components that store configuration information inside datasets like the Easy Chart.
- **Function** is a generic binding type that lets you bind a dataset property to the results of a function. It allows any of the function's parameters to be calculated dynamically via Tag and property bindings.
- **Style Customizer** is not one of the standard bindings, but changes properties to create cohesive styles based on different states.

On this page ...

- [Property Binding Types](#)
- [Setting Up Bindings](#)
- [Event-Based Bindings vs. Polling Bindings](#)
 - [Polling Options](#)
- [Copying Bindings](#)



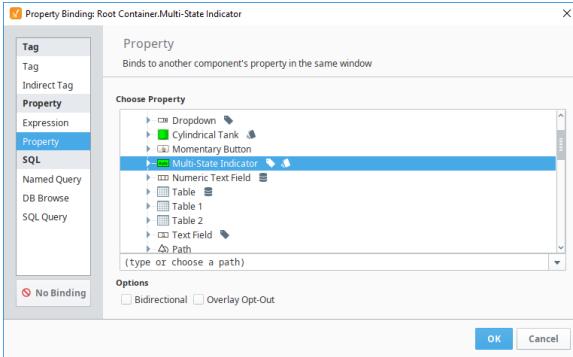
Property Binding

[Watch the Video](#)



Property Binding - Bidirectional

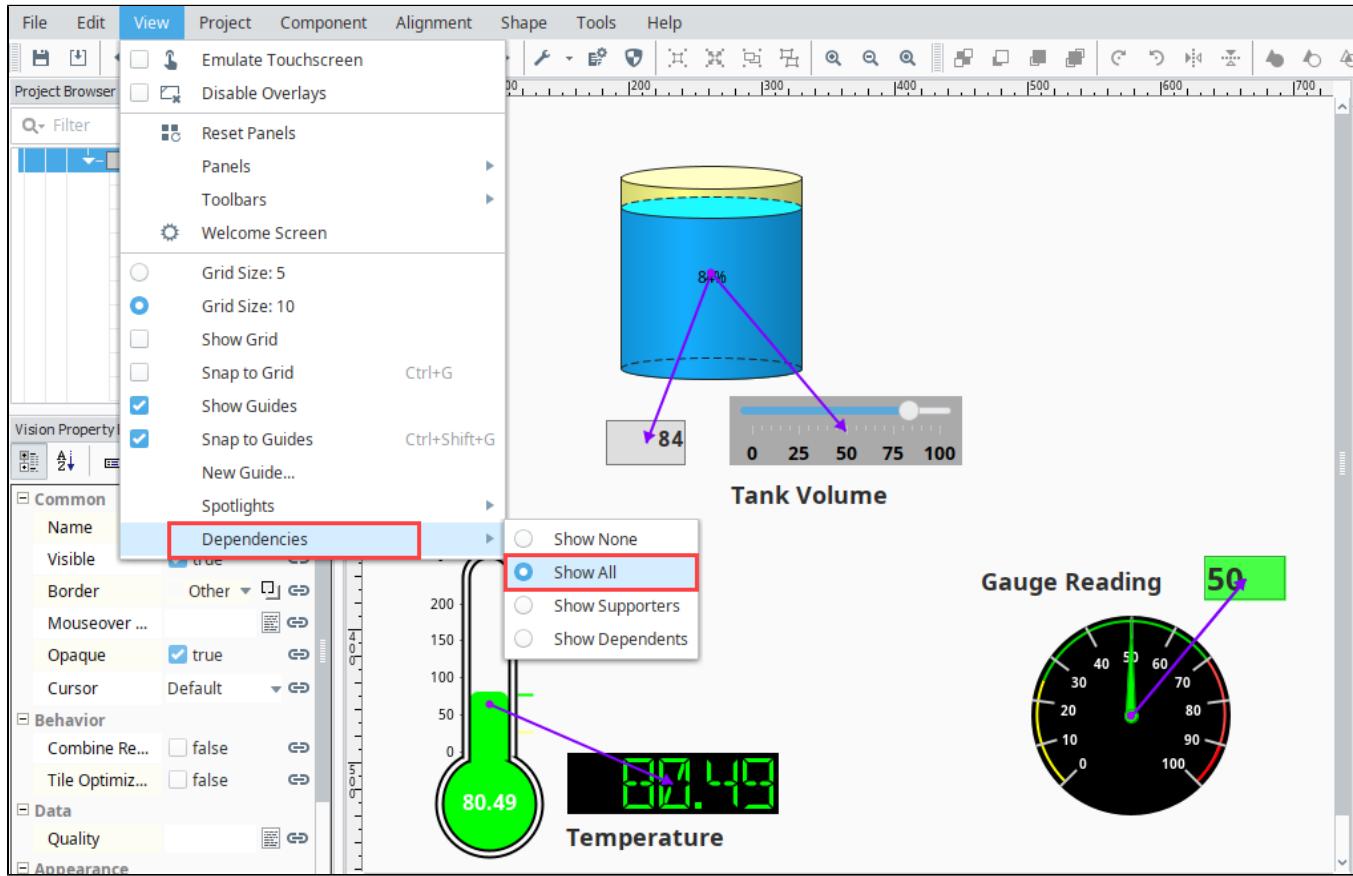
[Watch the Video](#)



Setting Up Bindings

Every component that you put on the screen has various properties that change the component's appearance and behavior. To make components do something useful, like display dynamic information or control a device register, you configure bindings on the component. It's the bindings that brings your components to life and have them do useful things. Components can be configured to do just about anything using bindings. To set up a binding on a property, simply click the Binding  icon to the right of the property in the Property Editor.

In this image, bindings were set to make these random components do something. You can quickly view dependencies to determine what is linked to what by going to **View > Dependencies > Show All**. As shown below, a line is drawn from the Tank to the Slider letting you know the Tank is bound to the Slider.



Event-Based Bindings vs. Polling Bindings

While there are quite a few different binding types, they fall into two broad categories: event-based and polling. Some complex bindings can span both categories.

Event-based bindings are evaluated when the object they are bound to changes. For example, when you bind a property to a Tag, that binding listens to the Tag, and every time the Tag changes, it assigns the Tag's new value into the property that it is on. If you bind the value of a Cylindrical Tank to the value of a Slider, every time the slider changes, it fires a `propertyChangeEvent`. The binding is listening for this event, and when it is fired, the binding updates the tank's value. The following bindings are event-based:

- Tag and Indirect Tag bindings
- Property bindings
- Some Expression bindings
- Cell Update bindings

Polling bindings are evaluated when a window first opens, on a timer, or when they change. For example, if you bind the data property of a Table to the results of a SQL query, that query will run on a timer, updating the Table every time it executes. The following bindings are based on polling:

- Bindings that query a database, including Named Query bindings, DB Browse bindings, SQL Query bindings, and Tag History bindings
- Some Expression bindings, like `runScript()` or `now()`
- Function bindings

Many bindings can combine elements of a polling binding and event-based binding. An expression binding may combine lots of other bindings to calculate a final result. A query binding will often itself be dynamic, altering the query based on other bindings.

For example, you might have a dropdown on a window that lets the operator choose a type of product that is produced. Then you can use a query binding like the following to calculate the defect rate for the given product:

SQL - Using a Component Property Reference

```
SELECT
    SUM(defective) / COUNT(*) AS DefectRate
FROM
    production_table
WHERE
    productCode = '{Root Container.ProductPicker.SelectedValue}'
```

The **blue** code is a property binding inside of the query binding. Every time this (event-based) binding fires, the query will run again, but will also run on a set timer based on its polling schedule. Using bindings like this, you can create highly dynamic and interactive screens with no scripting whatsoever.

Polling Options

The following are the options you can choose from for bindings that poll:

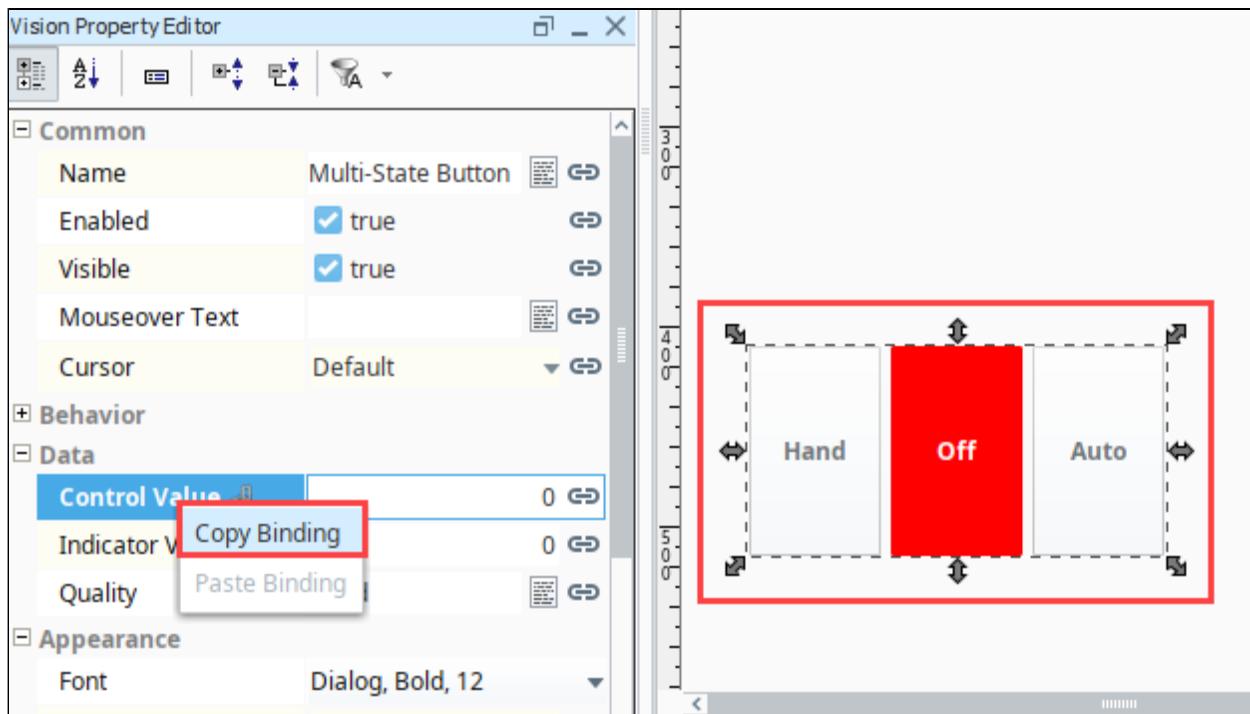
- **Polling Off**
The query will run once when the window is opened, and again whenever a reference inside of the the binding changes.
- **Relative Rate**
The binding will poll at the project's Base Polling Rate, which is **5 seconds by default**, plus or minus the given Polling Rate.
- **Absolute Rate**
Using this option, you can specify an absolute rate for the binding to execute at, instead of one that is based off the relative rate.

Regardless of which option is selected, polling bindings always fire when the window the component is on opens. This allows the component an opportunity to fetch an initial value.

Additionally, all three types will **always** update if the binding contains a reference to something else, such as a Tag or property value (noted with the brace-notation "`{}`"), and the value of that reference changes. Typically this is seen in SQL query bindings: polling can be turned off, and the query can reference a component value in a WHERE clause. When the referenced property value changes, the query will execute and retrieve new results.

Copying Bindings

When you copy a component, all bindings, scripts, etc. are copied along with it, but you can also copy a property binding from one property to another. Bindings can be copied from one property to another by right clicking on a property with a binding on it and selecting **Copy Binding**. Then, on another property, right click and select **Paste Binding** to paste the binding onto the property. This can be on the same component or a completely different component. The only prerequisite is that both property bindings must use a compatible property type. (For example, a binding that resolves to a string will not work on an integer property.)



In This Section ...

Property Bindings in Vision

Property to Property Binding

A property binding is a simple type of binding. It binds one component's property to another. When that property changes, the new value is pushed into the property that the binding is set up on.



Why aren't all properties listed?

You may notice that the list of properties available to bind to is smaller than the list of all properties. While nearly all properties can be bound, only some properties can be bound to. Only properties that fire a **propertyChangeEvent** may be bound to.

On this page ...

- [Property to Property Binding](#)
- [Bidirectional Property Bindings](#)

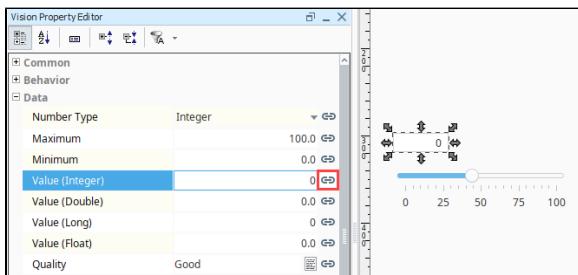


Property Binding

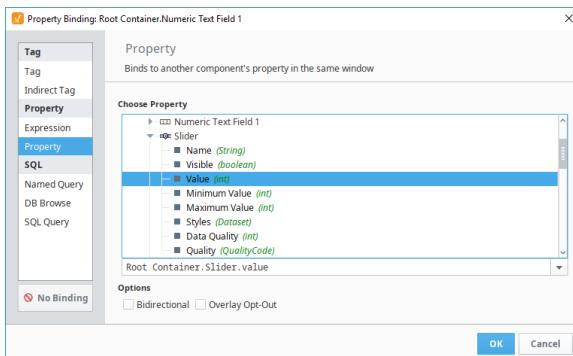
[Watch the Video](#)

In the following example, we'll bind the value of a Numeric Text Field to a Slider component.

1. Place a Numeric Text Field component and a Slider component on window.
2. Select the **Numeric Text Field**.
3. Click the Binding icon next to the Value (Integer) property.

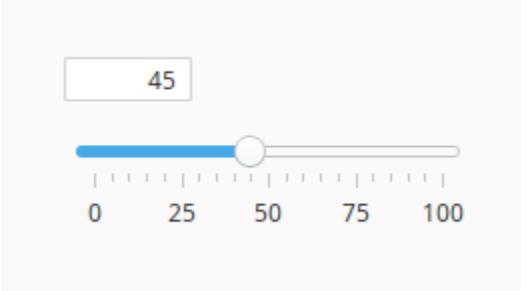


4. Select the **Property Binding Type**. Choose the **Slider's Value** property.



5. Click **OK**. Put the Designer in **Preview Mode** .

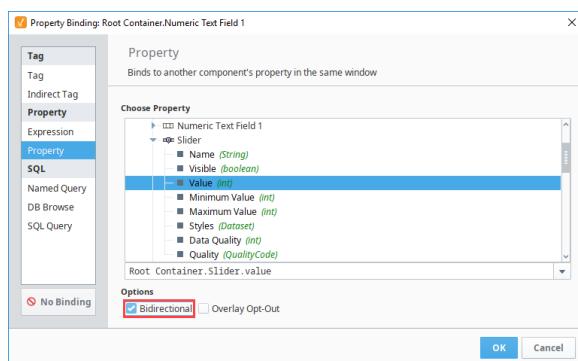
6. Move the slider. You'll see that the value from the Slider component appears in the Numeric Text field.



This can be useful to provide visual feedback to what a user is doing. The operator would input something, and they would see another component adjust to match the setting they just changed. Notice though, how if I were to change the value of the Numeric Text Field, the Slider will not update. Bindings are one direction only by default.

Bidirectional Property Bindings

Property Bindings have the ability to become Bidirectional, meaning instead of having the binding go one way only, it will work both ways, even with just the one binding. Take the previous example with the Numeric Text Field and Slider again. When changing the value of the Slider, the Numeric Text Field would update, but updating the value of the Numeric Text Field would not update the Slider. If we reopen the binding on the Value (Integer) property of the Numeric Text Field, we can see in the bottom left corner a checkbox for **Bidirectional**.



**INDUCTIVE
UNIVERSIT**

**Property Binding -
Bidirectional**

[Watch the Video](#)

Check the **Bidirectional** option and then save the binding. It will now be a bidirectional binding.

Tag Bindings in Vision

Binding Properties to Tags

A Tag binding is a very straight-forward binding type. It simply binds a property directly to a Tag. This sets up a subscription for that Tag, and every time the chosen Tag changes, the binding is evaluated, pushing the new value into the bound property. If you choose a Tag in the tree, and not a specific property of that Tag, the Value property is assumed.

Drag and Drop

There are several ways to drag and drop Tags onto a component allowing you to create screens very quickly. You can quickly bind a Tag by dragging it from the Tag Browser into a component or into the component property.

Binding to a Component

Ignition automatically creates the Tag bindings to several of the component properties when you choose to bind a Tag to a component by dragging and dropping. This is true for both creating a component by dragging and dropping a Tag onto empty space on a window at the same time Ignition prompts you for what type of component you want to create, and by dragging and dropping a Tag directly onto a component that already exists on a window. In both cases, Ignition automatically creates the Tag bindings on the component.

In addition, some of the bindings will be bidirectional or they may be expression bindings. How Ignition handles the binding depends on the disparity between the data types of the Tag and the target component. For example, binding a numerical Tag to a label component will result in an expression binding that formats the number to a string.

Binding to a Component Property

Ignition automatically creates a Tag binding to the property that you dropped the Tag into in the Property Editor, resulting in the property binding to the value of the Tag. In addition, it is possible to bind the Tag attributes to the component's property. For example, the Tag's Engineering High Limit attribute could be bound to the capacity property of a cylindrical tank component.

On this page ...

- [Binding Properties to Tags](#)
- [Drag and Drop](#)
 - [Binding to a Component](#)
 - [Binding to a Component Property](#)
- [Bidirectional Tag Bindings](#)
- [Bindings to Tag Properties](#)



INDUCTIVE
UNIVERSITY

Tag Binding

[Watch the Video](#)



INDUCTIVE
UNIVERSITY

Tag Binding – Drag and Drop

[Watch the Video](#)

Bidirectional Tag Bindings

Tag bindings can be made bidirectional simply by checking the **Bidirectional** checkbox at the bottom of the **Property Binding** window. A Tag can be set as a bidirectional binding, if it has a read/write permission and if the user has the security permission to write to the Tag. The Fallback Delay is the amount of time that the value will remain at the written value, waiting for a Tag change to come in. If no Tag change comes in within the allotted time (specified in seconds), the property will fall-back to the value as it was before the write. This is needed, because sometimes even if a write succeeds, another write or ladder logic in a PLC might have written something different, even the old value, in which case no Tag change event will be generated. As a rule of thumb, the fallback delay should be twice the Tag's scan class rate.



INDUCTIVE
UNIVERSITY

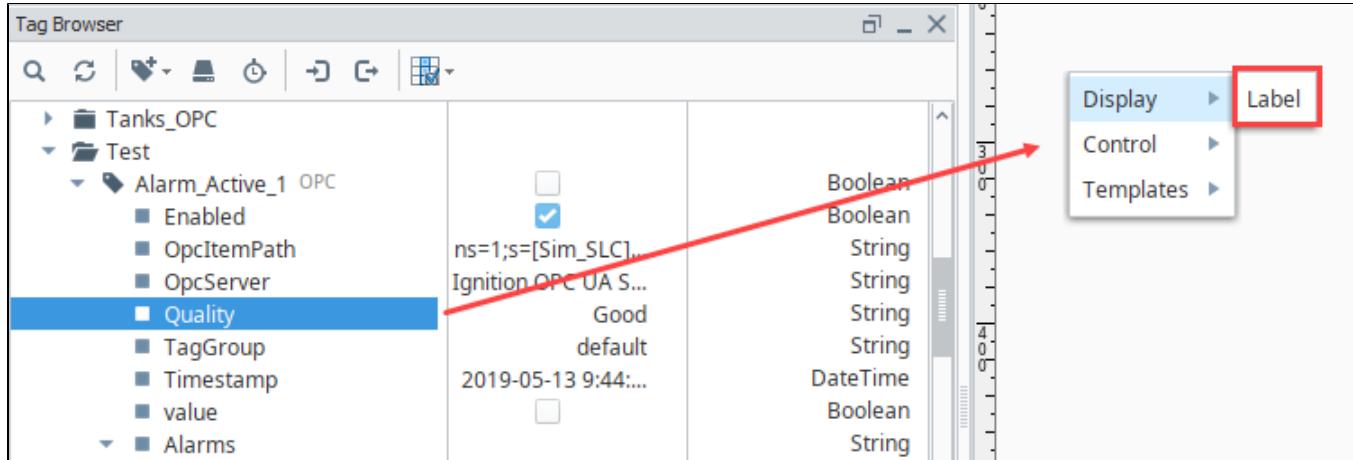
Tag Binding - Bidirectional

[Watch the Video](#)

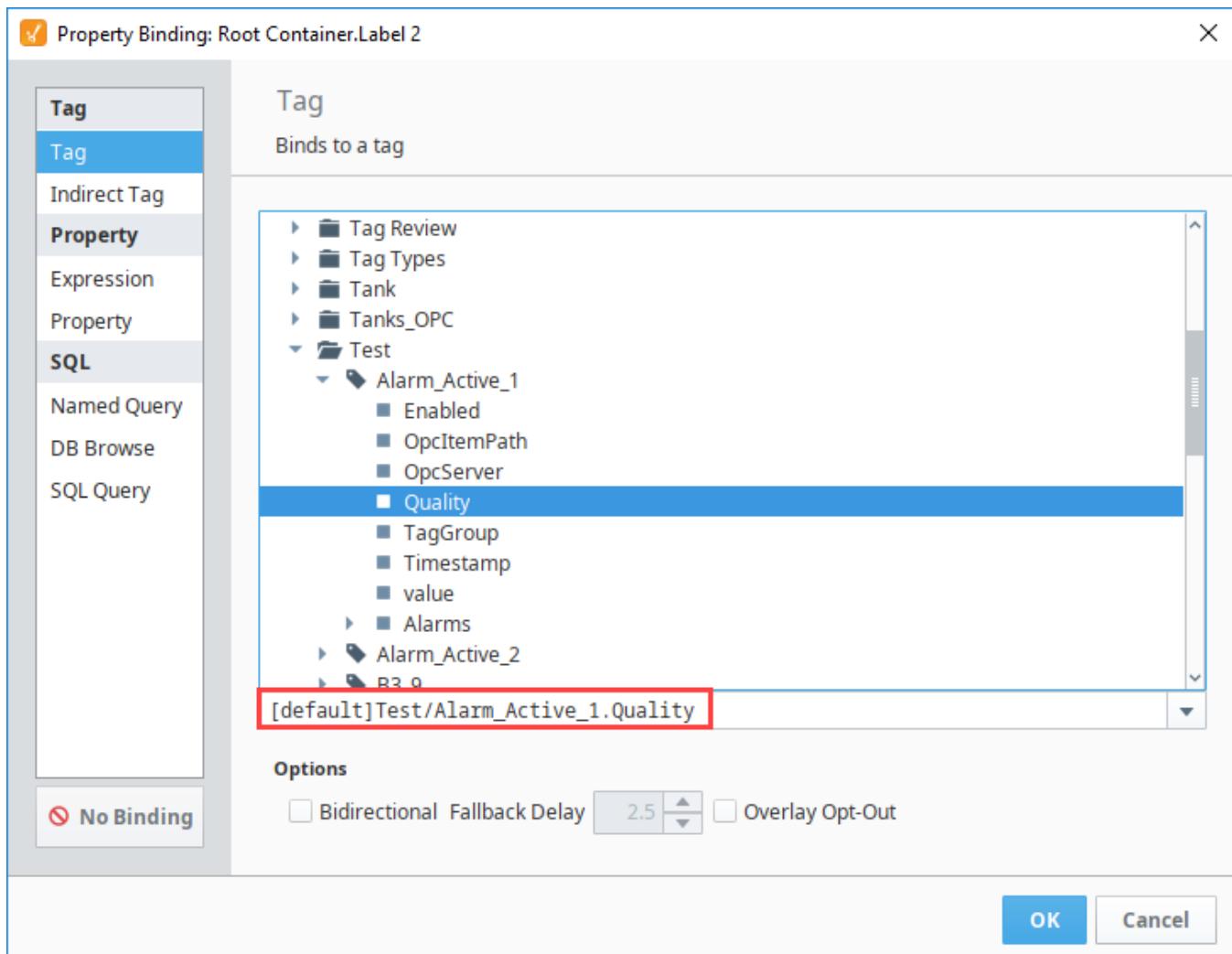
Bindings to Tag Properties

Aside from binding a property to a Tag's value, you can also bind to properties on a Tag, such as **Tooltip**, **Quality**, or **AlarmActiveAckCount**. This is useful when you don't need the value of the Tag, but rather the state, or some other configuration on the Tag. Here we see a boolean Memory Tag. It has a property indicating the Tag's quality. We can easily display that quality property on a component.

The simplest approach involves a **Tag Binding**. This can be achieved by dragging-and-dropping the Tag Property onto a Window, component, or component property. This example uses a Label component.



This is similar to creating a standard Tag Binding, except we're using a property on the Tag instead of the Tag's value. The resulting Tag Binding would look like the image below. Note that the property name has been appended to the path.



The Label component now displays the current value for the Tag's Quality property.

Tag Browser

| ▶ | Tanks_OP | | |
|---|--------------------|-------------------------------------|----------|
| ▼ | Test | | |
| ▼ | Alarm_Active_1 OPC | <input type="checkbox"/> | Boolean |
| ■ | Enabled | <input checked="" type="checkbox"/> | Boolean |
| ■ | OpcItemPath | ns=1;s=[Sim_SLC]... | String |
| ■ | OpcServer | Ignition OPC UA S... | String |
| ■ | Quality | Good | String |
| ■ | TagGroup | | String |
| ■ | Timestamp | 2019-05-13 9:44:... | DateTime |
| ■ | value | <input type="checkbox"/> | Boolean |
| ▼ | Alarms | | String |

Indirect Tag Bindings in Vision

Binding Properties to a Dynamic Set of Tags

An Indirect Tag binding is very much like a standard Tag binding, except that you may introduce any number of indirection parameters to build a Tag path dynamically in the runtime. These parameters are numbered starting at one, and denoted by braces, for example, {1}. The binding will be linked to the Tag represented by the Tag path after the indirection parameters have been replaced by the literal values they are bound to. An indirection parameter may represent a property on any component in the same window.

For example, instead of binding straight to a Tag's path, like

```
[TagProvider]MyPlant/EastArea/Valves/Valve4/FlowRate
```

or

```
[TagProvider]MyPlant/WestArea/Valves/Valve2/FlowRate
```

You can use other properties to make that path indirect. Suppose the "area" and "valve" number that we were looking at was passed into our window via parameter passing. Then we might use those parameters in the Tag path, like this:

```
[TagProvider]MyPlant/{1}/Valves/Valve{2}/FlowRate  
{1}=Root Container.AreaName  
{2}=Root Container.ValveNumber
```

Now our binding will change which Tag it is pointing to based on the values of those Root Container properties.

Creating an Indirect Tag Binding

When setting up an Indirect Tag Binding, there are a few tools in the binding window that help make it easier.

First there is the Indirect Tag Path. This field is where the Tag Path with parameters needs to be entered. Indirect Tag Bindings use numbered parameters at places in the Tag Path where indirection is going to occur. To the right of the Indirect Tag Path field are the Tag and Property reference helper buttons. The Tag button will enter the full Tag Path of the selected Tag into the Indirect Tag Path, while the Property button will add a new parameter reference to the Indirect Tag Path, and bind it to the selected property. The last area is the list of references, where each row in the list corresponds to a {1} parameter reference, and each row can be bound to property on the window. To bind a parameter reference to a property, simply select its corresponding row, and use the property selector to the right of the References list to select a property from the window.



Putting some thought into your Tag structure will make using the Tags indirectly much easier!

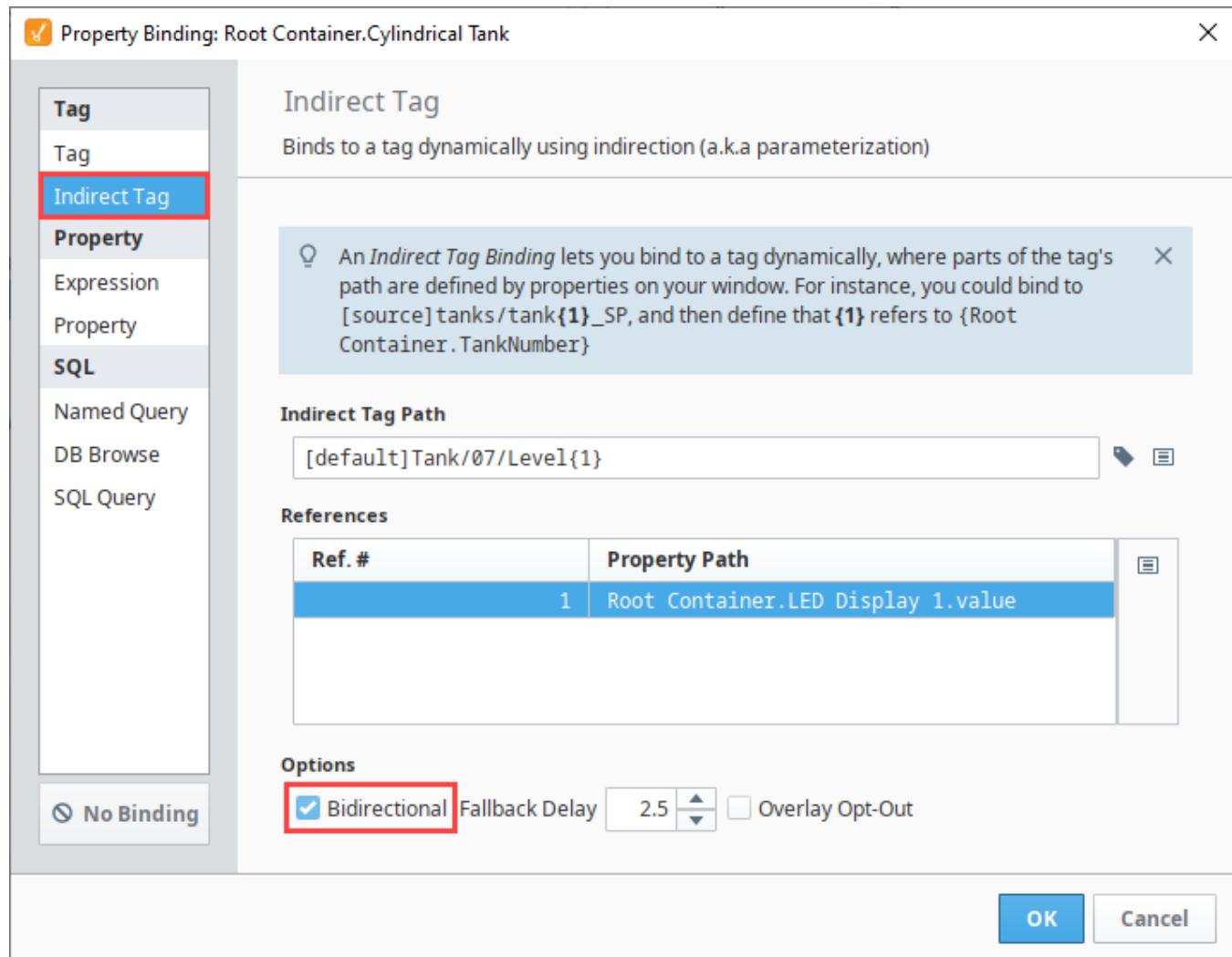
Bidirectional Indirect Tag Binding



Indirect Tag Binding

[Watch the Video](#)

Indirect Tag Bindings can also be made Bidirectional by clicking the **Bidirectional** checkbox at the bottom of the binding window. This will allow any input from a user on that property to be written back to the Tag. To work properly, the Tag needs to have the proper security to accept writes.



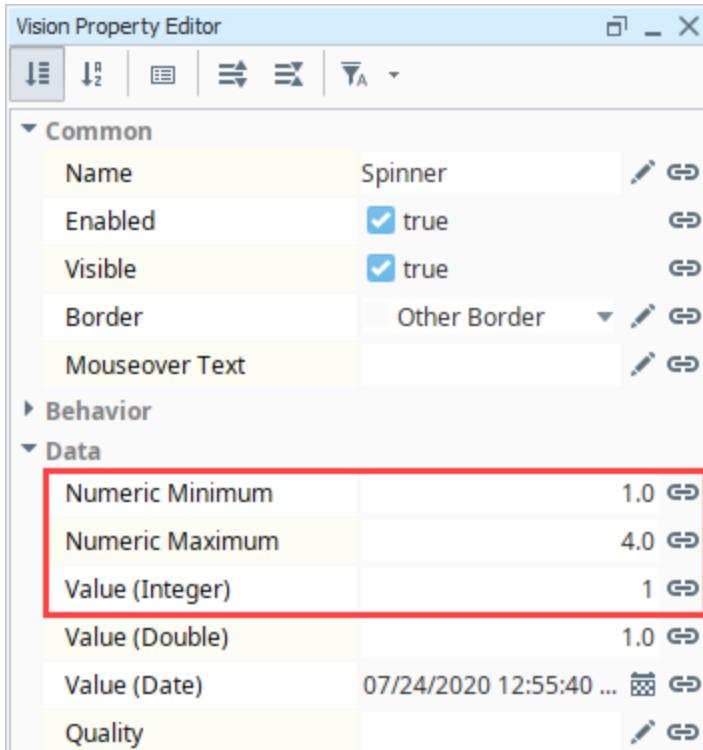
Indirect Tag Binding Example

In this example, we have some different motors, where each motor is a folder of Tags. Each motor has an amps Tag that is within the folder, so that our Tag paths look like the following:

```
Motor 1/Amps  
Motor 2/Amps  
Motor 3/Amps  
Motor 4/Amps
```

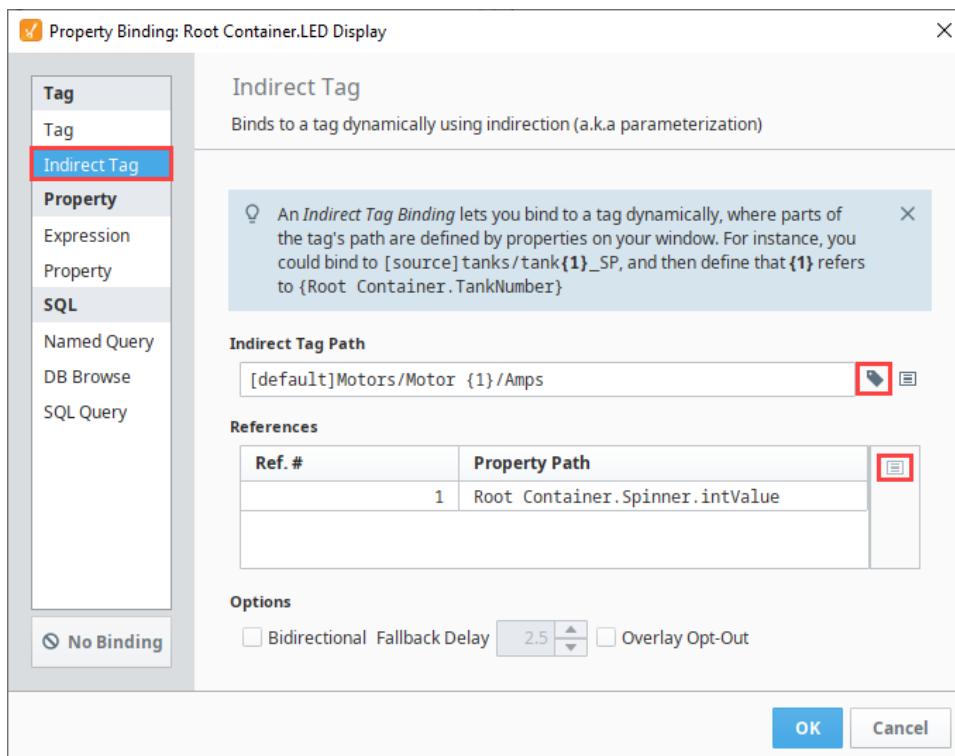
Instead of creating four different displays for these four different Tags, we can create a single display and make it indirect. We need two things for this example: a component to display the value in, and a component which allows the user to select which motor they are looking at.

1. Drag an **LED Display** component onto the window.
2. Then drag a **Spinner** component onto the window. This we will use to enable the user to select which motor they are looking at.
3. There are four motor Tags, so change the **Numeric Maximum** property of the Spinner to 4, and the **Numeric Minimum** property to 1. You may also need to change the **Value (Integer)** to 1.



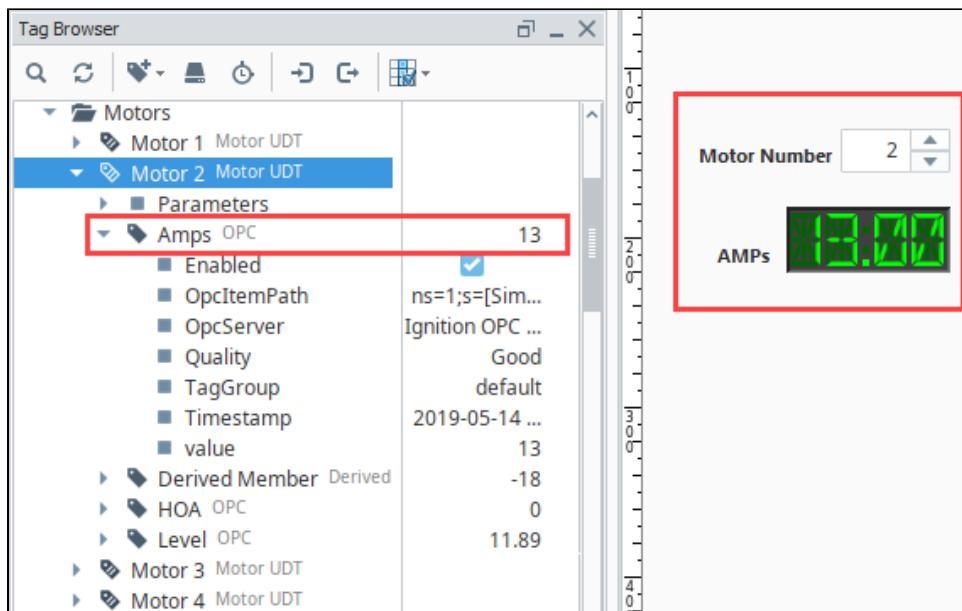
4. Select the **LED Display** component. Click on the Binding icon next to the **Value** property of the LED Display.
5. Select the **Indirect Tag** type.

- a. Click the Tag icon and select the **Motor 1/Amps** Tag.
- b. Delete the '1' in the Tag Path, and replace it with {1}.
- c. In the References section, select the row, and click the Insert Property Value icon. Select the **Value (Integer)** property of the Spinner.
- d. Click **OK** to save the binding.



6. To test it out, put the Designer into **Preview mode** . Notice how the value represented in the LED Display depends on what value is in the Spinner. Because the Spinner has the maximum value set to 4, users won't be able to set a motor number that does not exist. Additionally,

adding new motors simply means adjusting the maximum value on the Spinner.



Tag History Bindings in Vision

Binding Properties to the Tag Historian

The Tag Historian binding type, which is only available for Dataset type properties, runs a query against the Tag Historian.

Selected Historical Tags

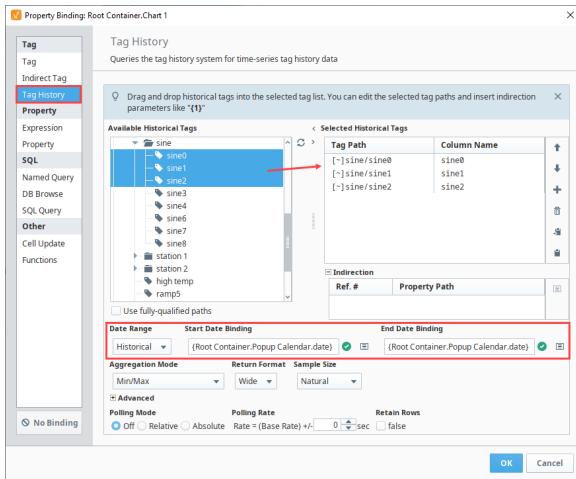
For this type of query, you must select at least one Tag path from the **Available Historical Tags** to query. The Dataset returned by the query will have a timestamp column, and then a column for each path that you select here.

Date Range

Choose either a Historical or Realtime query. Historical queries use a date range that must be bound in from other components on the screen, typically a **Date Range** or a pair of **Popup Calendars**. Realtime queries always pull up a range that ends with the current time, so all they need is a length.

This example uses a Historical query and two Popup Calendars for the start and end dates. The history is presented in the Table below.

1. In the Designer, drag two **Popup Calendar** components and a **Table** component from the Component Palette into your workspace.
2. Select the **Table** and right click on the Binding  icon for the **Data** property.
3. Drag a sine0, sine1, and sine2 under the Tag Path column under the **Selected Historical Tags** area.
4. Under Date Range, select **Historical**.
5. Under **Start Date Binding**, click on the Property  icon and under one of the Popup Calendars, select **Date**.
6. Under the End Date Binding, click on the Property  icon and under the second Popup Calender, select **Date**.
7. Click **OK**.



8. Now you can see the history of the three Sine tags along with a timestamp. You can scroll through the information in the table to see the history that was logged. To change the date range, click on dropdown buttons to bring up the popup calendars to change the date range. The tag history binding type allows you to bring back this history.

On this page ...

- [Binding Properties to the Tag Historian](#)
 - [Selected Historical Tags](#)
 - [Date Range](#)
- [Sample Size and Aggregation Mode](#)
 - [Aggregation Mode](#)
 - [Sample Size](#)
 - [Return Format](#)
 - [Advanced Options](#)
- [Indirect Tag History Binding](#)



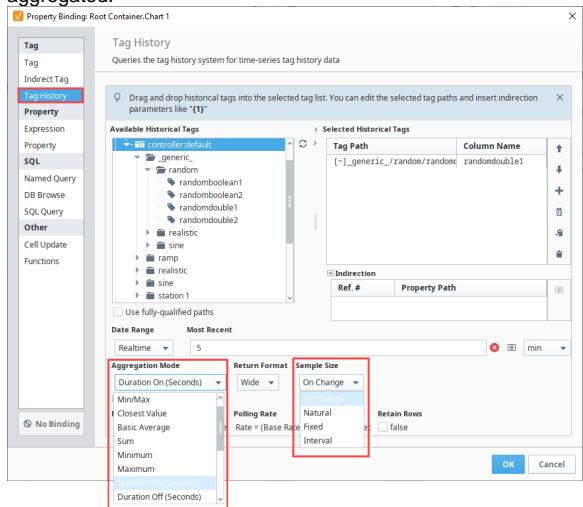
Tag Historian Binding

[Watch the Video](#)

| Start Date | | End Date | |
|---|--------|---------------------|-------|
| 06/25/2020 12:00 AM | | 06/26/2020 12:00 AM | |
| t_stamp sine0 sine1 sine2 | | | |
| Jun 25, 2020 5:56 AM | -22.13 | 169.82 | 83.14 |
| Jun 25, 2020 5:56 AM | -22.13 | 169.82 | 75.51 |
| Jun 25, 2020 5:56 AM | -30.98 | 169.82 | 75.51 |
| Jun 25, 2020 5:56 AM | -30.98 | 192.43 | 75.51 |
| Jun 25, 2020 5:56 AM | -30.98 | 192.43 | 66.68 |
| Jun 25, 2020 5:56 AM | -38.47 | 192.43 | 66.68 |
| Jun 25, 2020 5:56 AM | -38.47 | 214.46 | 66.68 |

Sample Size and Aggregation Mode

In places where the Tag History system can be queried, a **Sample Size and Aggregation Mode** can be selected that will determine how the results will be queried out and how the raw values will be aggregated.



Tag History Aggregates

[Watch the Video](#)



Table – Fixed Sample Size

[Watch the Video](#)

Aggregation Mode

The **Aggregation Mode** dictates what happens when multiple raw values are encountered for a given sample window (the size of which is determined by the number of requested rows, or the interval size).

| Aggregation Mode | Description |
|-----------------------|--|
| Time-weighted Average | The values are averaged together, weighted for the amount of time they cover in the interval. |
| Min/Max | The minimum and maximum values will be returned for the window. In other words, two rows will be returned. If only one value is seen in the interval, only one row will be returned. |
| Closest Value | The value closest to the ending time of the interval will be returned. |
| Basic Average | The values are summed together and divided by the number of values. |
| Sum | The values in the interval are summed together. |
| Maximum | The maximum value in the interval. |
| Minimum | The minimum value in the interval. |
| Duration On | Returns the number of seconds that the value was recorded as non-zero. |
| Duration Off | Returns the number of seconds that the value recorded as zero. |

| | |
|--------------------|---|
| Count On | Returns the number of times the Tag's value went from a zero value to non-zero. |
| Count Off | Returns the number of times the Tag's value changed from a non-zero value to zero. |
| Count | Returns the number of times a value was recorded |
| Percent Good | Time-weighted percentage of good values over the date range. |
| Percent Bad | Time-weighted percentage of bad values over the date range. |
| Range | Returns the range between the highest and lowest value for the period. |
| Standard Deviation | Standard Deviation - Returns the standard deviation of values, or how much spread is present in the data; low standard deviation shows the values are close to the mean, and high standard deviation shows that the data points are spread out over a large range of values. Only good quality values are used when calculating |
| Variance | Returns the variance of values. Similar in concept to standard deviation. Only good quality values are used when calculating. |

Sample Size

The sample size determines how many data points will be returned from the query.

On Change

An On Change query will return points as they were logged, and can be thought of as a "raw" query mode. This means that the results may not be evenly spaced. Also, it is important to note that every changed value will result in a row, and therefore if you are querying multiple tags and once, you may end up with more rows than you anticipated. For example, if Tag A and Tag B both change, you would end up with [[A₀, B₀],[A₁, B₀],[A₁, B₁]].

If you want to essentially retrieve raw values, while coalescing them down into fewer rows, try using the Interval sample mode, with an interval set to your largest acceptable time between rows, and select "prevent interpolation" from the advanced settings.

Natural

A Natural query will look up the logging rate for the queried tags (when possible), and return results spaced apart at that rate. This means that the return size will vary with the date range.

Fixed

You can use the **Sample Size** and **Aggregation Mode** on the **Tag History** binding type to fix the number of records that are retrieved. The Fixed sample size will cause the binding to retrieve all records from the date range, and aggregate them evenly between a fixed number of points. This will ensure that the number of rows will remain the same without regard to the size of the dataset. In windows where users are able to select a large range of data, Fixed is recommended as it will prevent the property from loading an excessive number of records.

In cases where the number of points can not evenly represent the data from the date range, an extra point will be added, making the final size of the dataset the fixed value + 1.

Selecting the **Min/Max** aggregation mode returns two rows of data for every row requested. Each pair represents a minimum and a maximum result from the underlying data. Therefore, a table with a fixed length, would return double the requested amount with Min/Max aggregation mode selected. With Min/Max aggregation mode selected, and with a fixed row length of one, the data set returns the oldest tag value of the time range

The following image shows a Tag History Binding pulling data from the last one day. The **Sample Size** is configured to **Fixed** with a value of **100**, and the **Aggregation Mode** is set to **Basic Average**. This means that the binding will query for data from the last one day, regardless of how many records there are, and create 100 time-slices that are evenly dispersed between the start and end periods of that range. Then, a basic average of the

tag values are calculated for each time-slice. The resulting values are then returned to the property.

The screenshot shows the 'Available Historical Tags' and 'Selected Historical Tags' panes. In the 'Available Historical Tags' pane, under the 'sine' folder, 'sine0' is selected and highlighted in blue. Other tags listed include sine1, sine2, sine3, sine4, sine6, sine7, sine8, station 1, station 2, and high temp. A checkbox 'Use fully-qualified paths' is unchecked. Below this is a 'Date Range' section set to 'Most Recent' with a 'Realtime' dropdown set to 60. Under 'Aggregation Mode', 'Basic Average' is selected. Under 'Return Format', 'Wide' is selected. Under 'Sample Size', 'Fixed' is selected with a value of 100. A red box highlights the 'Aggregation Mode' and 'Sample Size' sections. At the bottom right of the dialog is an 'Indirection' section with a table:

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

Note the **Insert Property** icon next to **Sample Size**. This allows a property binding to determine the number of data points, so you could change the size to increase or reduce the amount data points on the chart from the client.

Interval

Whereas the **Basic Average** sample size will calculate time slices based on the date range, the **Interval** sample size allows you to determine the size of the time slices. This sample size will divide the date range by the interval size to determine the size of each slice. Because of this, it is recommended to use an interval that is evenly divisible by the date range. However, in the event that the date range is dynamic or user driven, interpolation will handle any partially built slices. Even though the binding may attempt to evenly distribute the slices, there may be an extra row that represents the current values as they are building an interval.

The image below shows a **Realtime** range of **60 minutes**. The **Aggregation Mode** is set to **Time-weighted Average**, and the **Sample Size** is set to **Interval** for **5 minutes**. This means that the binding will query for data ranging from 60 minutes ago to now (or whenever the binding last executed, in the case that polling has been turned off). That 60 minute window will be divided as evenly as possible into 5 minute time-slices, so there should be around 12 time-slices. Each time slice will aggregate its value based on the time-weighted average of all values within that slice.

The example uses a Realtime range, but a Historical range could easily be used instead.

The screenshot shows the Tag Selection dialog. On the left, the 'Available Historical Tags' pane lists tags under categories like 'realistic' and 'sine'. A tag named 'sine0' is selected and highlighted in blue. On the right, the 'Selected Historical Tags' pane shows a table with one row: 'Tag Path' [~]sine/sine0 and 'Column Name' sine0. Below this is an 'Indirection' section with a table for 'Ref. #' and 'Property Path'. At the bottom, there are sections for 'Date Range' (set to 'Most Recent'), 'Aggregation Mode' (set to 'Time-weighted Average'), 'Return Format' (set to 'Wide'), and 'Sample Size' (set to 'Interval' with value '5').

Note the **Insert Property** icon next to **Sample Size**. This allows a property binding to determine the number of data points, so you could change the size to increase or reduce the amount data points on the chart from the client.

Return Format

Return format dictates how the requested data will be returned. The options are "wide" (default), in which each Tag has its own column, and "tall", in which the Tags are returned vertically in a "path, value, quality, timestamp" schema.

Advanced Options

These options affect the query results in more subtle ways.

- **Ignore Bad Quality** - Only data with "good" quality will be loaded from the data source.
- **Prevent Interpolation** - Requests that values not be interpolated, if the row would normally require it. Also instructs the system to not write result rows that would only contain interpolated values. In other words, if the raw data does not provide any new values for a certain window, that window will not be included in the result dataset.
- **Avoid Scan Class Validation** - "Scan class validation" is the mechanism by which the system determines when the Gateway was not running, and returns bad quality data for these periods of time. By enabling this option, the scan class records will not be consulted, which can improve performance, and will not write bad quality rows as a result of this check.

The screenshot shows the 'Advanced' options panel. It contains three checkboxes: 'Ignore Bad Quality', 'Prevent Interpolation', and 'Avoid Scanclass Validation'. The first two are checked, while the third is unchecked.



Tags Historian information is often easiest to work with in the [Easy Chart](#) component, which handles all of these options automatically.

Indirect Tag History Binding

The Tag History Binding can be made indirect but using indirection parameters in the Tag Paths of the Selected Historical Tags. This works similarly to the [Indirect Tag Binding](#), which uses Indirection References within the Tag Paths to substitute something into the path. Valid Indirection references consist of a reference number within curly braces. Simply type the indirection parameters (for example, {1}) into a selected Tag path by double-clicking in the list of selected paths. All valid parameters will appear in the lower indirection table.

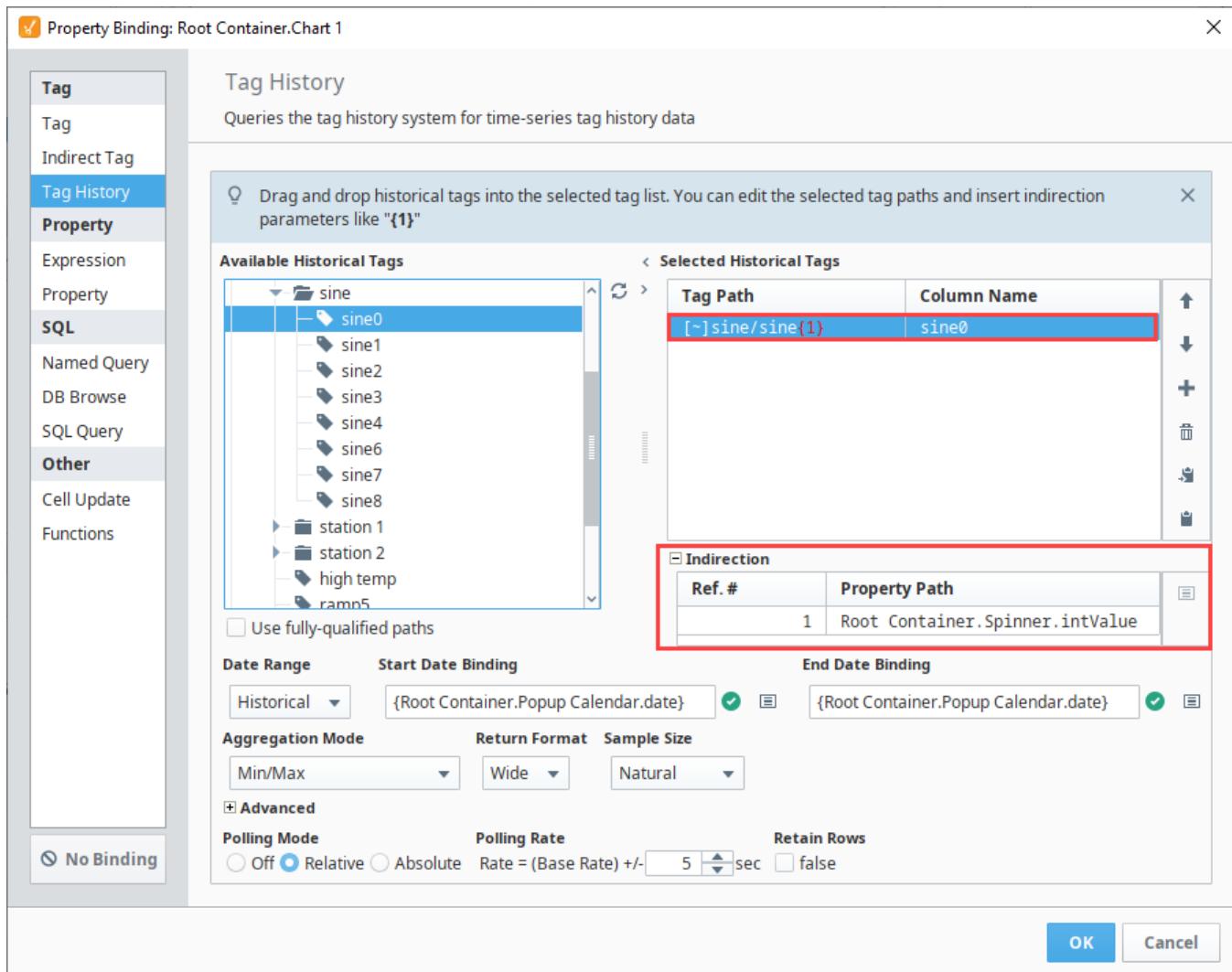
In this example, the Tag Path points to the Spinner component for the indirection parameters.



INDUCTIVE
UNIVERSITY

Indirect Tag History Binding

[Watch the Video](#)



Expression Binding in Vision

Binding Properties to the Outcome of an Expression

An expression binding is one of the most powerful kinds of property bindings. It uses a simple [expression language](#) to calculate a value. This expression can involve lots of dynamic data, such as other properties, Tag values, results of Python scripts, queries, and so on. Any time information needs to be massaged, manipulated, extracted, combined, split, and so on, expressions can get the job done.

Event Based and Polling

Expression bindings fall into the unique category of having the possibility of using both Events and Polling to update. How an expression updates depends on what is being done in the expression. Expression bindings will always update immediately when the window they are in is opened. When they update again depends on if they are driven by events or polling. Typically, expressions are driven by events. If the expression was adding multiple values together, then when one of those values changed the expression would update, regardless of whether those values came from other properties or Tags. However, the expression function has some unique functions that can update at a set rate such as the [no w\(\)](#) function. When these functions are used within the expression, the expression binding will update based on the specified polling rate.

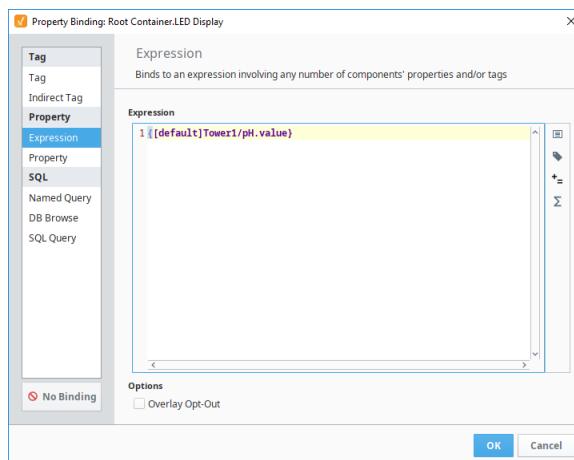
Using Expression Bindings

The expression language has lots of tools available that help calculate a specific value such as [built-in expression functions](#), [multiple operators](#), and the ability to reference Tags. While all of these can be manually typed into the expression, the expression binding window makes it easy to reference these options.

Helper Icons

To the right of the expression binding window, there are four icons that can be used to reference specific objects or functions easily.

-  **Properties** - Places a property reference into the expression at the cursor, pulling in that property's value into the expression at the time of evaluation.
-  **Tags** - Places a Tag reference into the expression at the cursor, pulling in that Tag's value into the expression at the time of evaluation.
-  **Operators** - Places the operator into the expression at the cursor. Mostly used as a reference to what operators are available for use.
-  **Functions** - Places the function into the expression at the cursor. Can be used as a reference for what functions are available, as well as the parameters the function is expecting.



On this page ...

- [Binding Properties to the Outcome of an Expression](#)
 - [Event Based and Polling](#)
- [Using Expression Bindings](#)
 - [Helper Icons](#)
 - [Expression Binding Examples](#)



Expression Binding

[Watch the Video](#)

Expression Binding Examples

Example 1

You have a button that starts a batch, but you only want to let it be pressed after the operator entered a scale weight. An expression binding can be set up on the enabled property of the button:

```
{Root Container.EntryArea.WeightBox.doubleValue} > 0.0
```

Example 2

You want to display a process's current state, translating a code from the PLC to a human-readable string. The examples below will yield the same results, but in different ways.

This first example uses nested "if" statements to produce the string. Notice that the false return for the first "if" statement is another "if" function, and the same with the second "if" function. Since the "if" function can only do simple if/then/else logic, this method allows us to do an if/then/else if/else.

```
if ({CurrentProcessState} = 0, "Not Running",
if ({CurrentProcessState} = 1, "Warmup phase - please wait",
if ({CurrentProcessState} = 2, "Running", "UNKNOWN STATE")))
```

This example will yield the same result as the previous example, but works differently. Instead of using multiple functions, this example uses a single switch function to decide which string to use.

```
switch ({CurrentProcessState},
0,1,2,
"Not Running",
"Warmup phase - please wait",
"Running",
"UNKNOWN STATE")
```

For more examples, see [Expression Overview and Syntax](#).

Named Query Bindings

Binding Properties to a Named Query

The [Named Query](#) binding is where you can configure a property to call a Named Query that you had previously created in the project. Using Named Query binding instead of a [SQL Query](#) or [DB Browse binding](#) helps to make your project more secure due to the built-in [Security Zone](#) and [User Role restrictions](#).

Polling Mode

Each Named Query binding type will use polling to determine when to update the results and run the query again. The Polling Mode dictates how often the query will execute, and works in a similar fashion to [polling on other bindings](#).

Creating Named Queries

In order to use a Named Query, first one has to be created. You might already have some that another developer created but if not, you will have to make one by going to the [Named Query](#) section in the Designer or converting a SQL query.

On this page ...

- [Binding Properties to a Named Query](#)
- [Polling Mode](#)
- [Creating Named Queries](#)
- [Using Named Queries on Dataset Properties](#)
- [Using Named Queries on Scalar Properties](#)



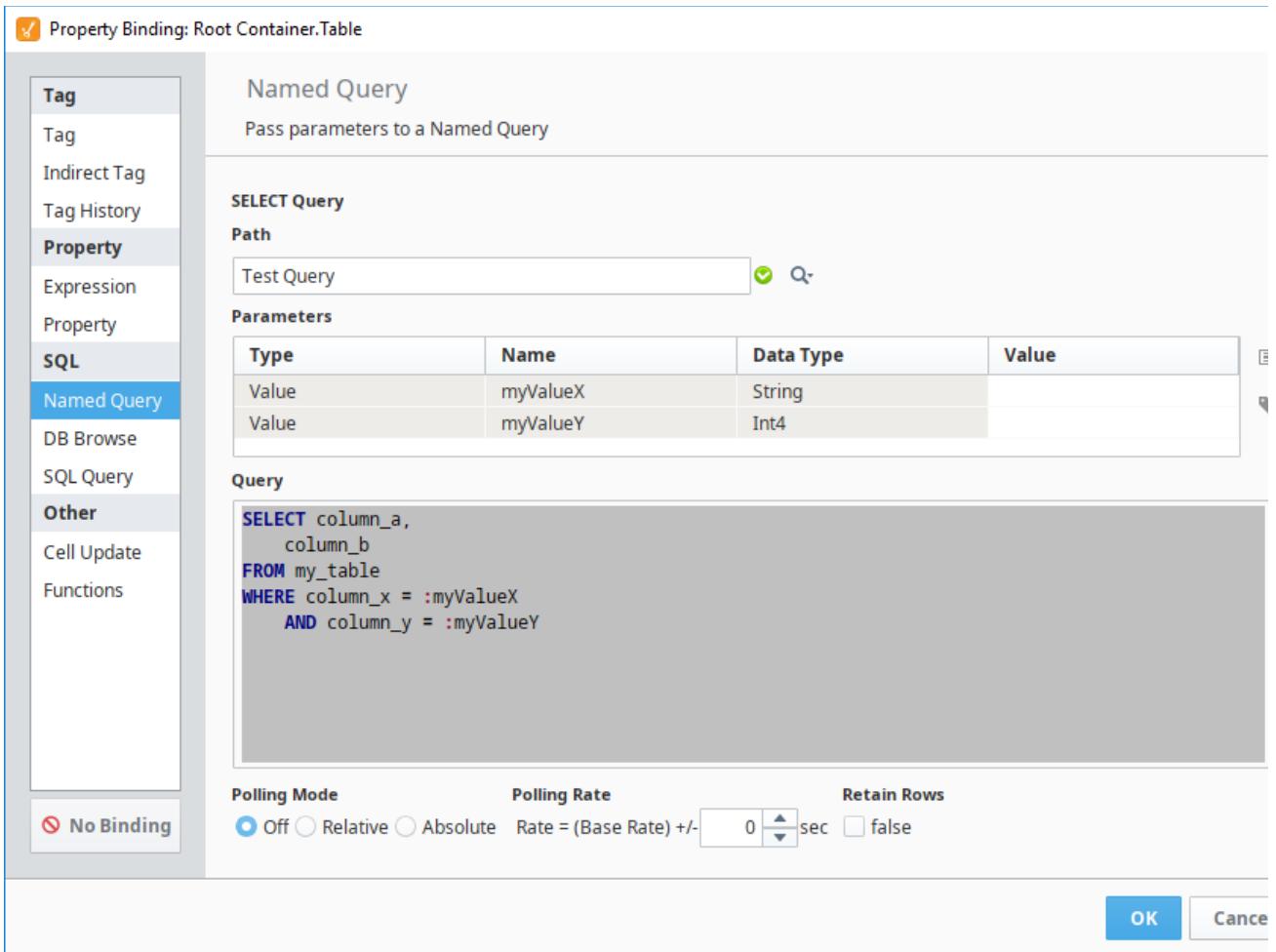
Named Query Binding

[Watch the Video](#)

Using Named Queries on Dataset Properties

The majority of your Named Query bindings will most likely be on a dataset type property. When placed on a Dataset type property, only a single Named Query needs to be specified. An explanation of the various fields on the binding are detailed below:

- **Path:** Here you can enter in the path to the Named Query. Click on the Search icon to view a list of available Named Queries.
- **Parameters:** Here you can see a table of all defined [Named Query parameters](#). You can pass in property or Tag values to the parameters by first highlighting the parameter and then selecting either the **Insert Property** icon or the **Tag** icon.
- **Query:** The Query section shows what Named Query looks like. Note that you can't modify the query on this page.
- **Polling Mode:** Here you can set the Polling Mode of the Named Query binding based on the [Polling Rate](#).
- **Retain Rows:** If true, any rows that you have returned within the Designer will be saved along with the window. This may slow window load times.



Using Named Queries on Scalar Properties

When placed on a non-dataset type property (such as a String or Integer), then the Named Query binding allows for a second Named Query to be specified in the case that the user can update the value on the property. This provides an opportunity to both return and update values in the database from the same component.

The configuration is very similar to a Named Query binding on a dataset property. You need to specify a Named Query path, set up your Parameters, and choose a **Polling Mode**. You can finish setup at this point, leaving the update query disabled so that the property will simply pull the value from the database.

However, if you want the binding to be bidirectional, you need to specify an UPDATE query. This works similar to a SELECT query, in that you need to select the path to the Named Query and set up any Parameters. However, it is important to make sure that the Named Query chosen for the UPDATE query is in fact setup as an UPDATE query by setting the [Query Type property on the Authoring section](#) of the Named Query.

Property Binding: Root Container.Numeric Text Field

Named Query

Pass parameters to a Named Query

SELECT Query

Path

Contacts/Find Contact

Parameters

| Type | Name | Data Type | Value |
|-------|-------------|-----------|------------------------|
| Value | Badgenumber | Int4 | {Root Container.Nu...} |

Query

```
SELECT name FROM contacts WHERE badgeid = :Badgenumber
```

Polling Mode

Off Relative Absolute Rate = (Base Rate) +/- 0 sec

UPDATE Query Enabled

Path

Contacts/Update Contact

Parameters

| Type | Name | Data Type | Value |
|-------|-------------|-----------|--------------------------|
| Value | Badgenumber | Int4 | {Root Container.Nu...} |
| Value | NewName | String | {Root Container.Text...} |

Query

```
UPDATE contacts FROM Set Name :NewName WHERE badgeid = :Badgenumber
```

OK **Cancel**

Related Topics ...

- Cell Update Bindings
- Named Queries
- Named Query Parameters

DB Browse Bindings

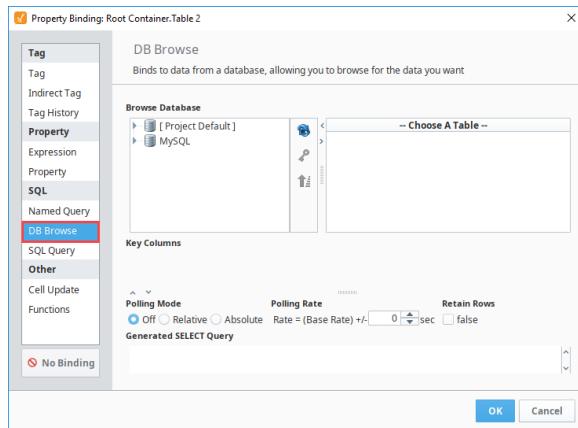
Binding Properties to Database Tables

The DB Browse binding is technically equivalent to the [SQL Query binding](#), except that it helps write the queries for you. Using the Database Browse binding type, you can pick the table from a list of tables in each database that you want to pull content from. If you have a fixed range of data you need to return, simply select it in the table, and watch the query get generated.

In the Browse Database tree, you can choose which columns in your table should act as your keys (these columns get put in the WHERE clause based on your selection) and which columns should be used to sort the data (these columns are put in the ORDER BY clause).



This binding type also serves as a convenient jumping-off point for the more flexible SQL Query binding. Construct the basic outline of your query in the DB Browse section, and then select the SQL Query radio button to convert to the new binding type. Your query will be retained and can then be modified manually.



On this page ...

- [Binding Properties to Database Tables](#)
- [Configuring the Binding](#)
 - [Key Column](#)
 - [Sort Order](#)
- [Dynamic Filters](#)
- [Scalar Query Update](#)



DB Browse Binding

[Watch the Video](#)

Configuring the Binding

After selecting a table in the Browse Database tree, you can customize which columns the query is selecting by selecting one or more columns under the table to select just the highlighted columns, or selecting the table to use the * symbol to select all columns.

Key Column

The DB Browse binding has the ability to designate key columns within the query. A key column is used within the select query's where clause, and can be given a value. A column is denoted as a key column when it has a key symbol next to it.

Clicking the Key icon to the right of the Browse Database tree will designate a column as a key column. Alternately, if the highlighted column is already a key column, then clicking the Key icon will remove that column as a key column.

Property Binding: Root Container.One-Shot Button

DB Browse
Binds to data from a database, allowing you to browse for the data you want

Browse Database

| | tank_overview... | Tank_Number | Lot_ID | Notes |
|---|------------------|------------------|-----------------|-------|
| 1 | 1 | 496 | 500 | |
| 2 | 2 | The Lot is th... | Note thistwi... | |
| 3 | 3 | Lot 3 | 3 notes, 3 n... | |
| 4 | 4 | 534 | 540 | |
| 5 | 5 | 550 | 550 | |
| 6 | 6 | 554 | 560 | |
| 7 | 7 | 570 | 570 | |

Key Columns
`tank_overview_ndx = 1`

Polling Mode **Polling Rate**
 Off Relative Absolute Rate = (Base Rate) +/- 0 sec

Generated SELECT Query
`SELECT tank_overview_ndx FROM Tank_Overview WHERE tank_overview_ndx = 1`

Enable Database Writeback
`UPDATE Tank_Overview SET tank_overview_ndx = {this} WHERE tank_overview_ndx = 1`

No Binding

OK **Cancel**

Sort Order

In the DB Browse binding, you can also sort data in ascending or descending order. Select the column that you want to sort by and click the Sort icon. Multiple columns can be used for sorting.

Dynamic Filters

DB Browse bindings also give the ability to bind a property to a key column to allow for dynamic filtering of the returned data. Simply click the binding icon next to the key column field. This allows you to give the operators some control over the data they are seeing.

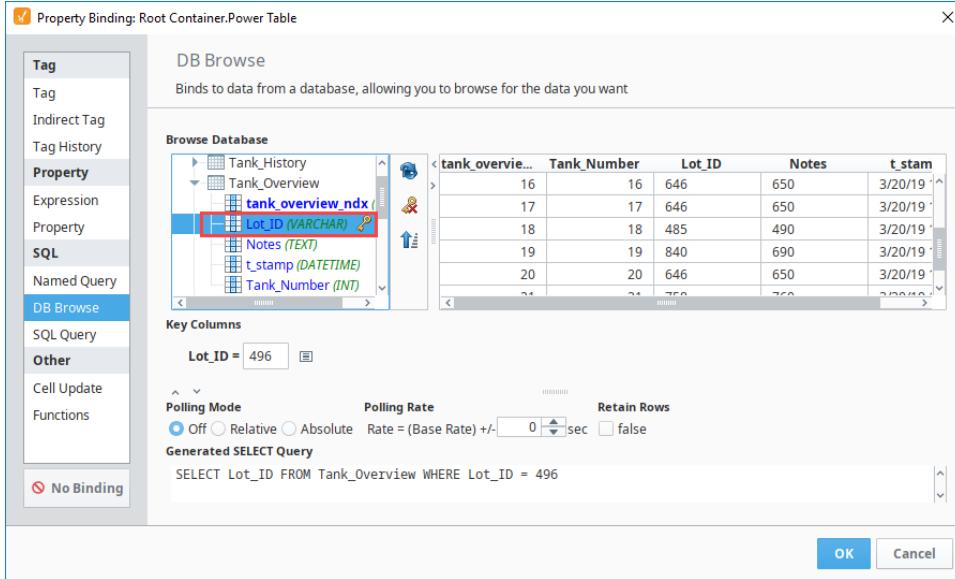


**INDUCTIVE
UNIVERSITY**

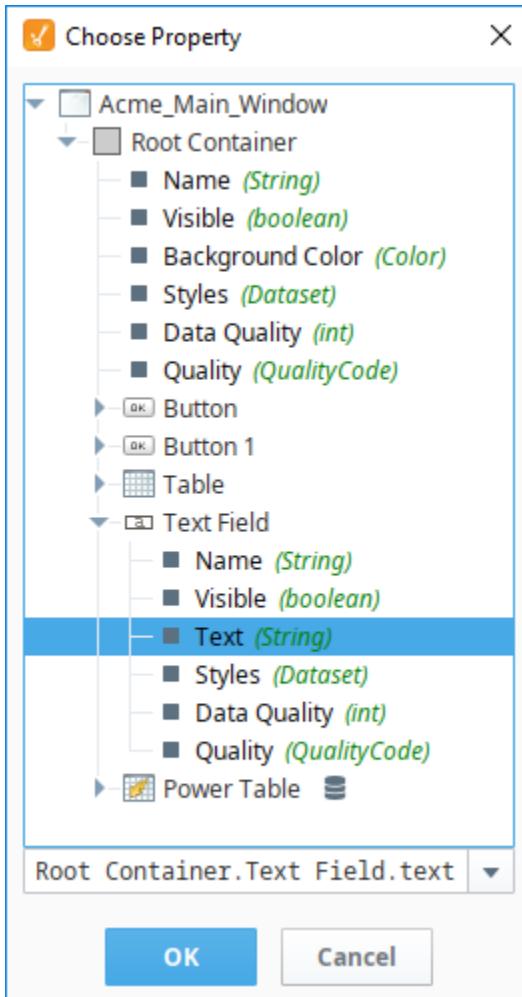
**DB Browse Binding
- Dynamic Filters**

[Watch the Video](#)

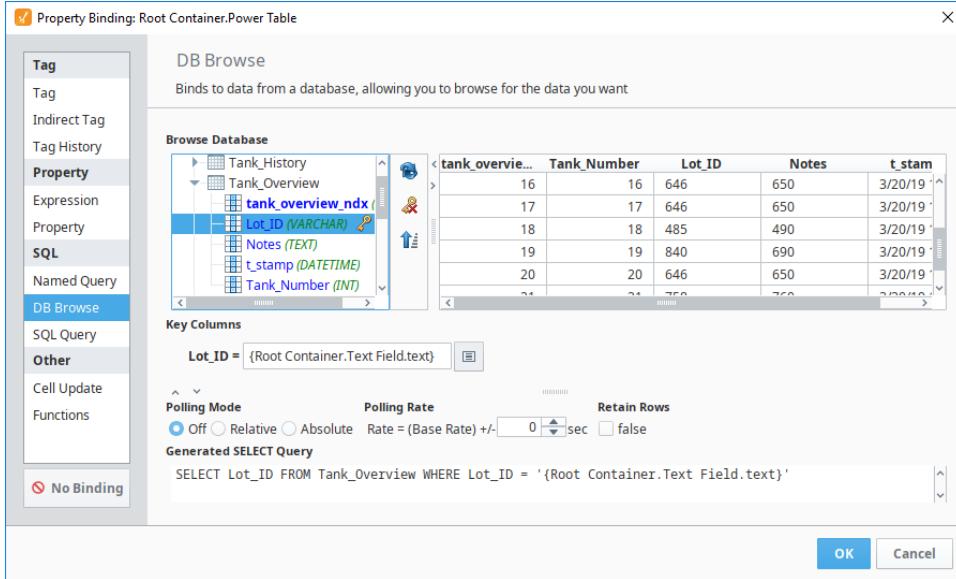
1. In the Designer, drag a Table component and a Text Field component on a window.
2. With the Table component selected, click the Binding icon next to the Data property.
3. Choose DB Browse under Binding Types > Database.
4. Let's pull all the data from this Table except for the id, and filter on state. Remove the Key icon from the id column and place it on the state column.



5. Select the **Tank_Number**, **Lot_ID**, **Notes**, and **t_stamp** columns. You can do this with Control+Click, or by clicking and dragging in the results table in the upper right.
6. Instead of statically typing in a value like we did in the above example, let's make it dynamic using the Text Field. Click the Insert Property Value  icon next to the value in the **Key Columns** section, and select the Text property of the Text Field.



7. Notice there is now a property reference in the **Key Column** as well as the **Generated SELECT Query**.



8. Click **OK** to confirm the binding.
9. Put the Designer into Preview mode.
10. Enter the Lot_ID that you want to view. You'll see the Table update to display just the data for that Lot ID.

296

| Tank_Number | Lot_ID | Notes | t_stamp |
|-------------|--------|-------|---------------------|
| 23 | 296 | 300 | 2019-03-20 12:36:00 |
| 24 | 296 | 300 | 2019-03-20 12:36:00 |

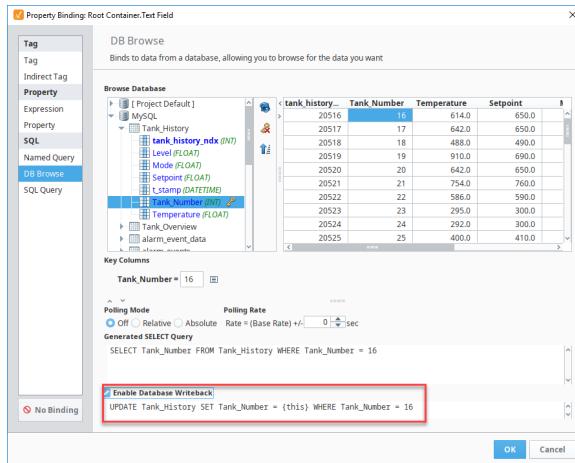
646

| Tank_Number | Lot_ID | Notes | t_stamp |
|-------------|--------|-------|---------------------|
| 16 | 646 | 650 | 2019-03-20 12:36:00 |
| 17 | 646 | 650 | 2019-03-20 12:36:00 |
| 20 | 646 | 650 | 2019-03-20 12:36:00 |

Scalar Query Update

Similar to the SQL Query Binding, the DB Browse Binding has the ability to become bidirectional by doing a database write back when the property being bound is a non-dataset type. In this case, the select query should be configured to only return a single row from a single column.

For example, this option can provide a single value to the Text property of a Text Field component. If you check the **Enable Database Writeback** checkbox, then any user input will write back to the database. This will automatically generate an update query that will push the input value into the database from the location where the original value was retrieved.



SQL Query Binding - Scalar Query and Update

[Watch the Video](#)

SQL Query Bindings in Vision

Binding Properties to a SQL Query

The SQL Query binding is a [polling binding type](#) that will run a SQL Query against any of the database connections configured in the Gateway. It is very similar to the [DB Browse binding](#) type in that both query a database to return data. The difference is the SQL Query Binding can manually be modified. This is useful for complex queries where you will use the more advanced functions of the SQL language that can not be accomplished with the DB Browse binding.



Pro Tip!

The query that gets generated by the DB Browse will transfer over to the SQL Query binding when you switch the binding type. It may be useful to build the basic query structure with DB Browse first, then switch to SQL Query binding to modify the query to fit your needs.

Dataset Binding

The majority of SQL Query bindings will return a dataset. These will return many rows with multiple columns. For example, showing all customer details from a certain account, or all downtime events in the facility. This type of SQL binding is used on properties of type **dataset** like the Data property on a [Table component](#).

On this page ...

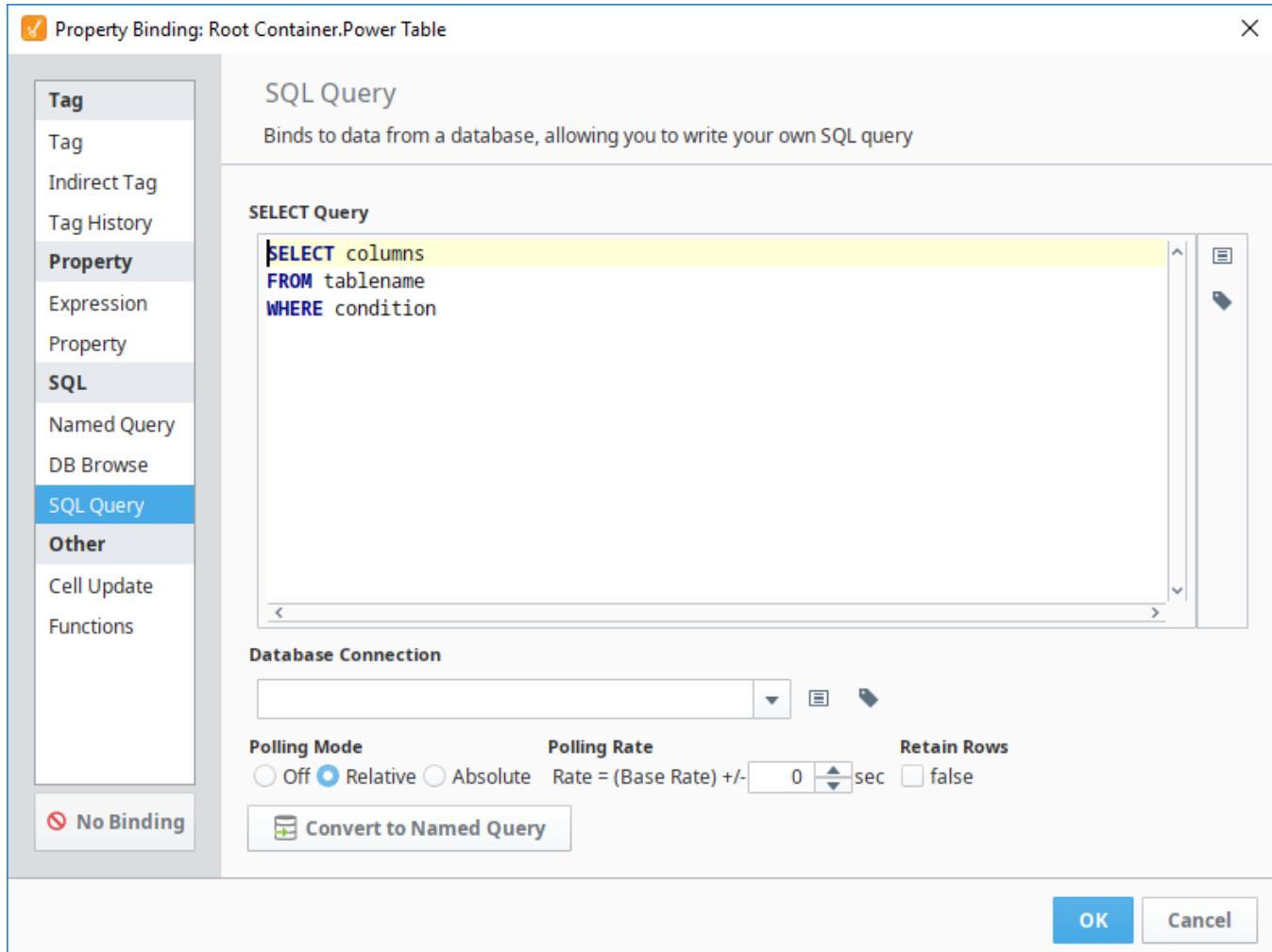
- [Binding Properties to a SQL Query](#)
- [Dataset Binding](#)
- [Dynamic Filters](#)
 - [Example](#)
- [Scalar Query Update](#)
- [Scalar Query Fallback](#)
- [Stored Procedures](#)
- [Named Query Conversions](#)



INDUCTIVE
UNIVERSITY

SQL Query Binding

[Watch the Video](#)



Dynamic Filters

Using the curly brace {} notation, you can include the values of component properties (within the same window) and Tag values inside your query. This is a very common technique to make your query dynamic. The values of the property or Tag represented are simply substituted into the query where the braces are.

Because the substitution is direct, you'll often need to add quotes to literal strings and dates to make your query valid. If you're getting errors running your query complaining about syntax, it is important to realize that these errors are coming from the database, not from Ignition. Try copying and pasting your query into the Query Browser and replacing the braces with literal values.

 INDUCTIVE
UNIVERSITY

**SQL Query Binding
- Dynamic Filters**

[Watch the Video](#)

Example

A common requirement is to have a query filter its results for a date range. You can use the [Date Range](#) component or a pair of [Popup Calendar](#) components to let the user choose a range of dates. Then you can use these dates in your query like this:

SQL - SQL Query Binding with Parameter References

```
SELECT
    t_stamp, flow_rate, amps
FROM
    valve_history
```

```

WHERE
    t_stamp >= '{Root Container.DateRange.startDate}' AND
    t_stamp <= '{Root Container.DateRange.endDate}'

```

Notice the single quotes around the braces. This is because when the query is run, the dates will be replaced with their literal evaluations. For example, the actual query sent to the database might look like this:

SQL - SQL Query Binding with the Values Replaced

```

SELECT
    t_stamp, flow_rate, amps
FROM
    valve_history
WHERE
    t_stamp >= '2010-03-20 08:00:00' AND
    t_stamp <= '2010-03-20 13:00:00'

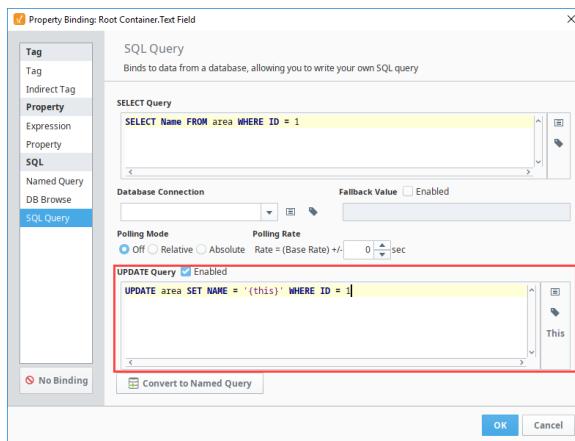
```



It is important to use single quotes and not double quotes (`t_stamp = "2010-03-20 08:00:00"`) because these mean something different in certain databases like Microsoft SQL Server.

Scalar Query Update

You can bind a non dataset type property to a SQL query to allow a singular value to be returned from the database with a scalar query. Now instead of returning multiple rows and columns, the query returns a single value from the first row of the first column. These types of SQL Query bindings can also be used to update the database on input components like a Text Field. Essentially, we mimic the bidirectionality of Tag and property bindings by adding in an update query to run whenever a value gets entered into the property with the binding. In our update query, we use the special parameter `{this}` to denote the new value from the bound property. If `{this}` is a string, it needs single quotes around it.



Take a Text Field with a simple query on it.

SQL - Simple Select Query

```
SELECT Name FROM area WHERE ID = 1
```

This will return a single value that can populate our text field. We then enable the Update Query at the bottom of the Property Binding window, and add in the update query.

SQL - Using Ignition's 'this' Keyword

```
UPDATE area SET Name = '{this}' WHERE ID = 1
```



SQL Query Binding - Scalar Query and Update

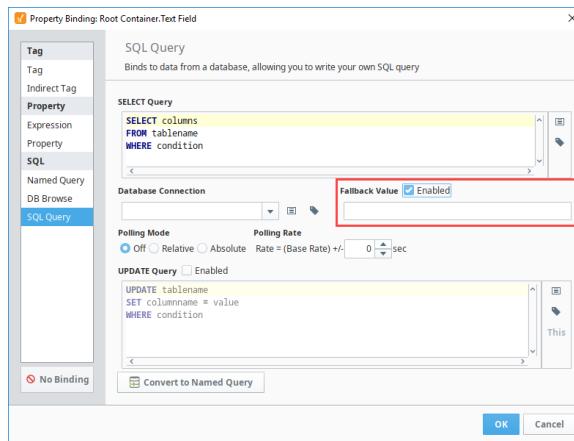
[Watch the Video](#)

After confirming the binding, we can see that our text field contains the value from the database and will update the database cell if we enter in a new value into the text field. This is a good way to alter very specific cells in a database table.

Scalar Query Fallback

If the property that is being bound is a scalar datatype (that is, not a Dataset), the value in the first column in the first row of the query results is used. If no rows were returned, the binding will cause an error unless the Use Fallback Value option is selected. The value entered in the fallback value text box will be used when the query returns no rows.

When binding a Dataset to a SQL Query, no fallback value is needed, because a Dataset will contain zero rows.



SQL Query Binding - Scalar Query and Fallback

[Watch the Video](#)

Stored Procedures

While queries can manually be written on a SQL Query binding, [SQL Stored Procedures](#) may also be called from a SQL Query Binding.

Note: The exact syntax is highly dependent on the type of database you are using.

For example, calling a Stored Procedure from MySQL would involve using the CALL command, while SQL Server utilizes the EXEC command.

SQL - MySQL Stored Procedure Call

```
CALL retrieve_daily_total
```

SQL - SQL Server Stored Procedure Call

```
EXEC retrieve_daily_total
```

Named Query Conversions

You can convert the SQL Query created here to a Named Query. For more information, see [Named Query Conversions](#).

Cell Update Bindings

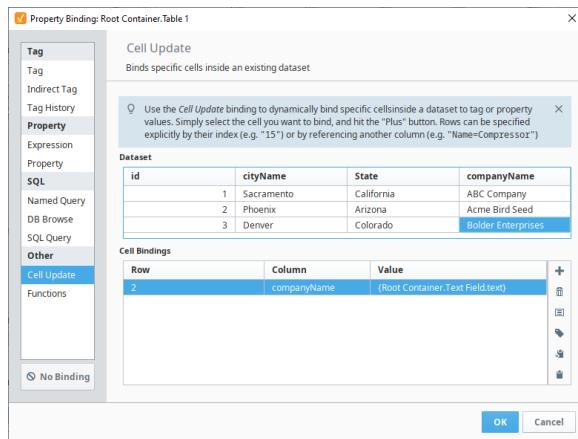
Binding a Dataset Property to Realtime Values

The Cell Update binding enables you to easily make one or more cells inside a dataset dynamic. This is particularly useful for components such as the Table to easily display realtime Tag information, or in the Linear Scale which stores configuration information in a dataset. The Cell Update binding allows you to bind a cell inside of a dataset to a Tag or to a property.

Cell Update Bindings work really well with the Easy Chart component, allowing you to indirectly show Tag history on the same Easy Chart. See [Indirect Easy Chart](#) for more details.

Note: The Cell Update binding type will only appear when setting up a binding on a **dataset** property, otherwise, it won't be present.

The cell update binding has a few tools that help you make certain cells dynamic.



The top table called **Dataset** displays the original dataset, while the bottom **Cell Bindings** table displays each of the cell updates that will be occurring. Each Cell Update binding can have multiple cell updates happening on a single dataset. To the right of the Cell Bindings table are the buttons that are used to create the Cell Bindings:

- **Add Row** - Adds a new empty row to the Cell Bindings table, which can be customized with a new Cell Binding. If a cell is selected in the Dataset table, it will instead add a row with the Row and Column values already filled in with the appropriate values.
- **Delete Row** - Removes one of the rows in the Cell Bindings table.
- **Insert Property** - Inserts a property reference into the Value cell of the selected row.
- **Insert Tag** - Inserts a Tag reference into the Value cell of the selected row.
- **Copy** the current selection to the clipboard.
- **Pastes** the contents of the clipboard into the current context.

Adding Values to the Cell Bindings Rows

Each row in the Cell Bindings table needs three values to work properly: a **Row identifier** which is used to figure out which row the cell binding is on, a **Column identifier** to determine the column of the cell, and a **Value** which will replace the original value of the cell. These three values can pinpoint a specific cell, and replace its value with the dynamic Tag or property value specified. The examples in this section use the dataset in the image above.

Caution: No two rows should specify the same cell, as this will throw an error.

Row Column

The Cell Bindings **Row** column can be filled one of two ways.

On this page ...

- [Binding a Dataset Property to Realtime Values](#)
- [Adding Values to the Cell Bindings Rows](#)
 - [Row Column](#)
 - [Column Column](#)
 - [Value Column](#)
- [Realtime Tag Values in a Table Example](#)
- [Adding a Realtime Indicator to the Linear Scale Example](#)



Cell Update Binding

[Watch the Video](#)

Row Index

The easiest and most often used way is by specifying the row index of the cell that you want to target for an update. It is important to remember that the row index is zero based, so the index of the first row is always 0. There can be multiple rows, each with the same row index, as long as the Column is different. The order of the rows in this case does not matter, as in the image below, where it specifies a cell with a row index of 0, then 5, then 0 again.

| | | |
|---|----------------|--|
| 0 | Boolean Column | |
| 5 | String Column | |
| 0 | Int Column | |

Column Value

The other way to identify the row that the intended cell belongs to, is to use the value of a different column. This is done using the syntax:

```
columnName=value
```

Where the columnName is the name of the column and the value is the value that needs to match. The columnName and value are both case sensitive, so you will need to use care when filling in these values.

| | | |
|-----------------------|----------------|--|
| 0 | Boolean Column | |
| 5 | String Column | |
| String Column = Test1 | Boolean Column | |

There are three things that make using the Column Value unique. The first is that there is the possibility for duplicates to happen. Take the image above, with both a Row index of 0 and a Column Value of 'String Column=Test 1'. Looking at the original dataset, both of those point to the same row, and with both of them pointing to the Boolean Column, they both refer to the same cell in the dataset. This instance of duplicates will not throw an error, but instead will work. In this case, the updates happen from top to bottom, so the 'String Column=Test 1' would be what writes to the cell last and what ultimately gets displayed.

The second is that there is the potential for multiple possible matches to that evaluation. For example, if I had this in the Row value:

```
Boolean Column=True
```

This could potentially be true for multiple rows of my dataset, in which case, the binding will apply to all of them. This allows you to change multiple cell values that should all be the same.

The final thing that makes the Column Value unique is that it itself is ultimately dynamic. The 'Boolean Column=True' Column Value matches the rows at index 1, 3, 4, 7, and 9 in the original dataset. However, if any of those were to change to False through another cell update, then that row would no longer be updated as part of this cell update. Conversely, if any of the currently False values were to change to a True, then those rows would fall under this cell update and the appropriate cell will be updated.

Column Column

The Cell Bindings column named **Column**, or **Column column**, expects the name of a column that will match in the "Dataset" table above. These values are case sensitive, so care should be taken when entering them manually.

Value Column

The **Value column** is what will get pushed into the cell that is being updated. There are three possible types of values that can be placed in here.

Static Value

A static value can be written in here. This will overwrite the existing value of the cell with whatever static value is in the cell update. This does defeat the purpose of using the cell update though, since the static value could just be placed directly in the original dataset.

Tag or Property Reference

A Tag or Property reference can be used for the cell updates value, updating the value of the cell whenever the value of the Tag or Property changes. The Tag or Property Selectors to the right of the Cell Bindings table can be used to add in the reference.

Note:

When adding a Tag or Property reference, be sure that the cell is only selected and that the cursor is not placed in the Value cell. The cursor in the cell is used for typing in a static value, and trying to enter in a Tag or Property reference will not work.

This is used to enter in static text:

0

Int Column

This accepts Tag and Property references:

0

Int Column

Both Static Values and a Reference

The Value column can also accept a combination of a reference and static values. This allows you to build unique strings or numeric values within the Value cell. The syntax used is:

```
numbers1234{tagOrPropertyReference}orcharacters
```

The reference value will be concatenated into the Value at the location specified. For example, if my Value was

```
{MemoryTags/IntegerTag}000
```

and the IntegerTag had a value of 5, the updated cell value would be 5000. This method of combining both a reference and a static value is great for updating Tag paths, such as in an Easy Chart's Tag Pens dataset.

```
[~]Motors/Motor {Root Container.MotorNumber}/Amps
```

Here, the MotorNumber property on the root container is replacing the motor number inside of the Tag path. See [Indirect Easy Chart](#) for more details.

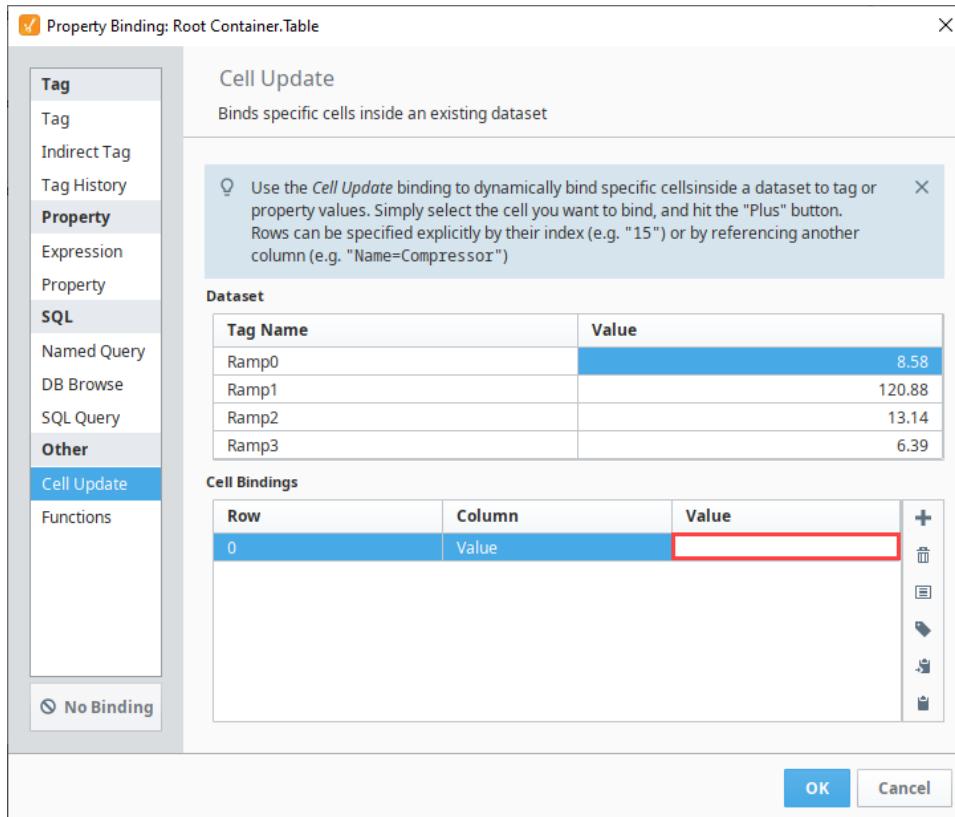
Realtime Tag Values in a Table Example

The Cell Update binding allows you to place the value of a Tag into a dataset easily. With a dataset property like the one on the [Table](#) component, getting updating values into it requires either a SQL query, or some constantly running script. With the Cell Update binding, that isn't necessary. The simple static table, as shown below, contains four rows for four different tags, and each having a value. As an alternative, you could use four numeric text fields and labels, but the Table component looks much nicer compared to all of those labels and numeric text fields.

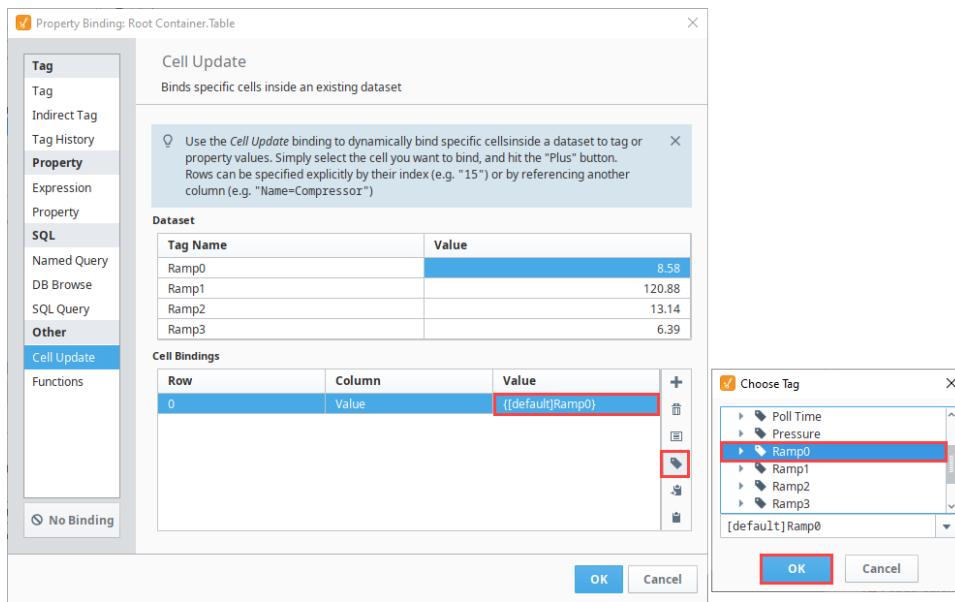
| Tag Name | Value |
|----------|--------|
| Ramp0 | 8.58 |
| Ramp1 | 120.88 |
| Ramp2 | 13.14 |
| Ramp3 | 6.39 |

Let's use the table above table to create a Cell Update binding to get the pressure readings into the table.

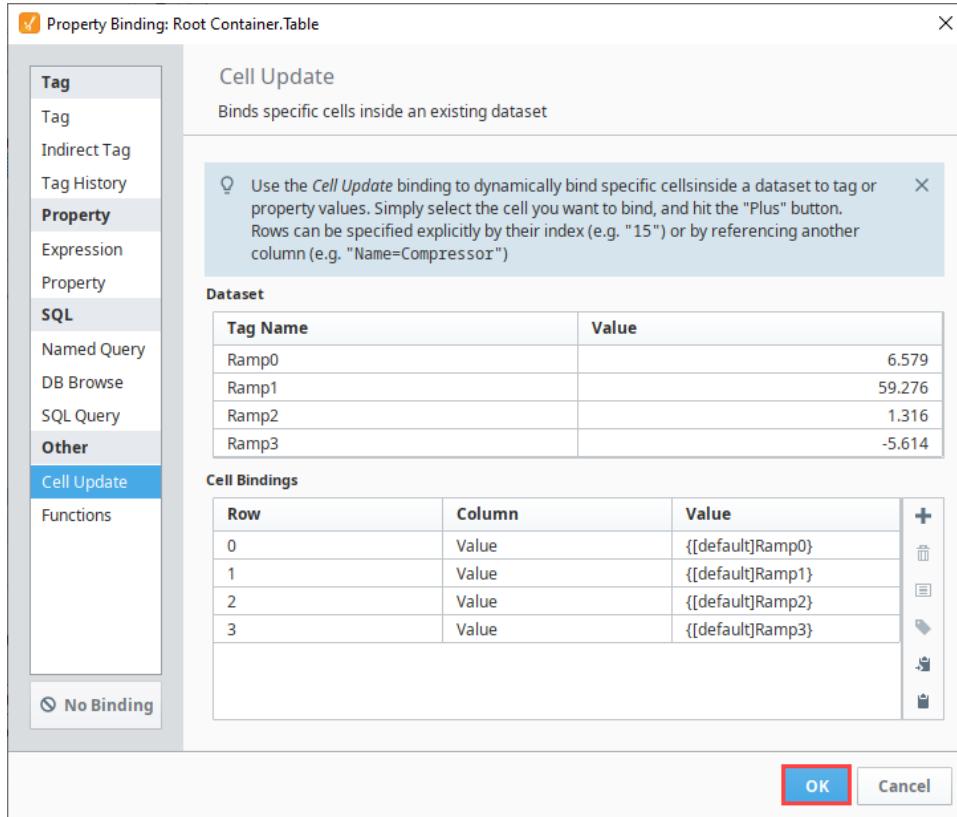
1. Open the property **binding**  icon on the data property.
2. Click the **Cell Update** binding type, and you'll see your dataset along with a **Cell Bindings** area.
3. Select the first value, and click the **Add**  icon in the Cell Bindings area to add a row. You'll notice it added a row number, column name, and blank value cell because we selected a cell from above. What you place in the blank **Value** cell determines what the value will be in the table.



4. Select the first **Row** and first **Value** cell in the Cell Binding area. Use the **Tag** icon to the right to select the Tag reference (i.e., Ramp0). You can also manually type in your Tag reference, or place in a property reference instead.
5. Click **OK**.



6. Now, you have the value of that Tag to the Value column in row 0. Repeat Step 3 for the other three Tags so that the Cell Binding area looks like the image below, then click **OK**.



7. You can see that the Table values are now updating with the value of their respective Tag.

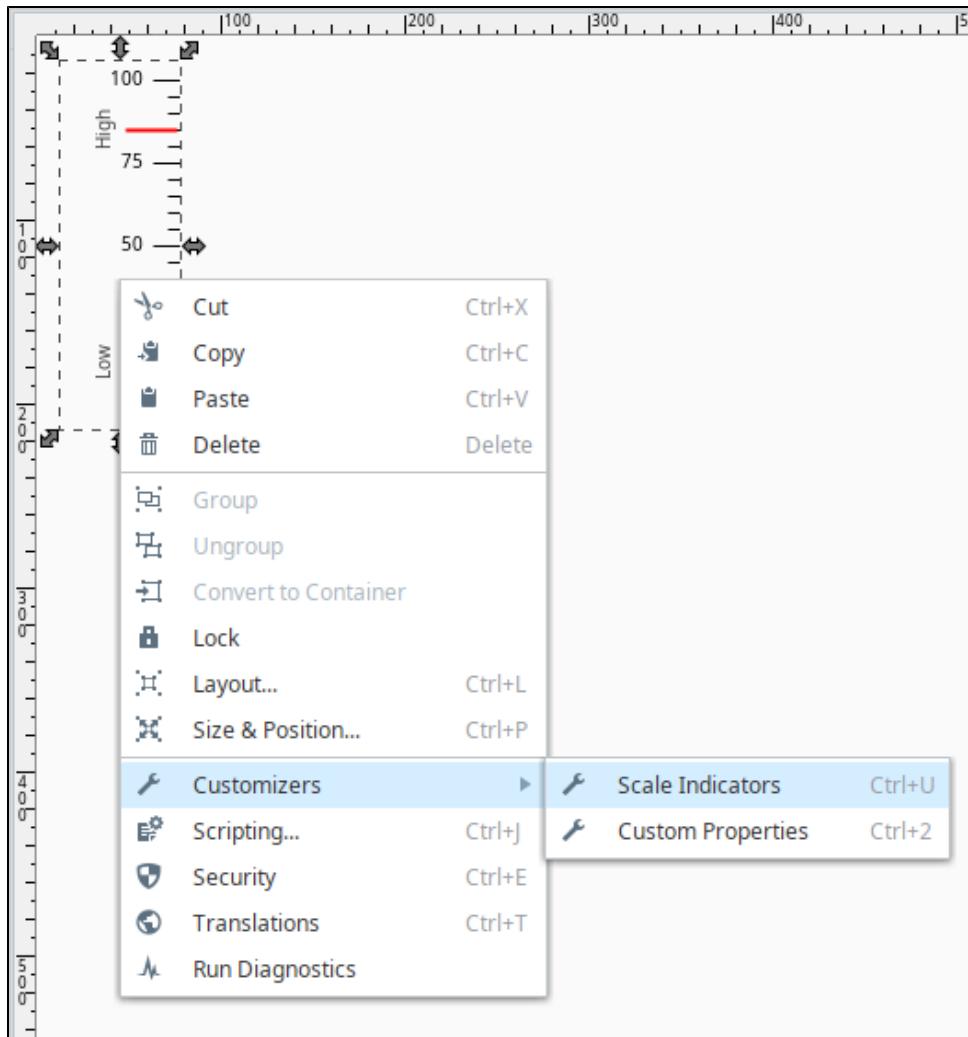
| Tag Name | Value |
|----------|-------|
| Ramp0 | 2.59 |
| Ramp1 | 23.44 |
| Ramp2 | 1.52 |
| Ramp3 | 5.06 |

Adding a Realtime Indicator to the Linear Scale Example

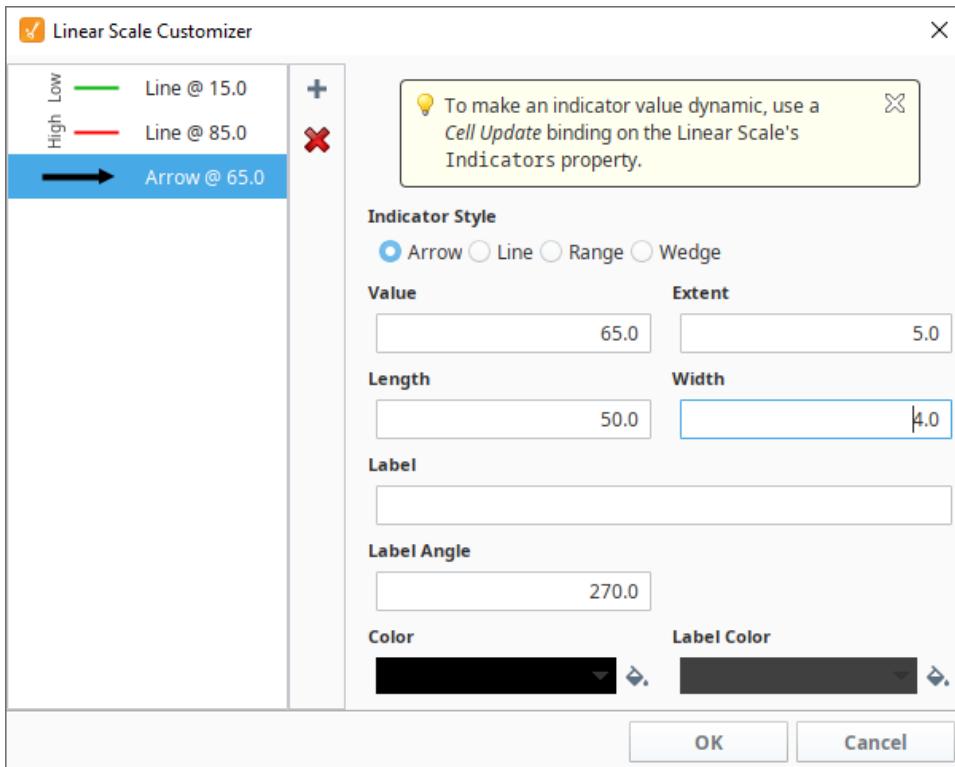
The **Linear Scale** is a component that has a special Scale Indicators customizer that allows you to configure setpoints. You can setup the value, color, style, and more for each of the indicators. During runtime, these values are normally static, as there is no way to setup a binding in the customizer. However, the customizer merely configures a dataset that the component uses to create the indicators. If you look at the Indicators dataset property of the Linear Scale, it has all of the properties that are configurable in the customizer. Changing them from the customizer will alter the dataset, and changing the dataset values will alter what you see in the customizer. Knowing this, we can setup a Cell Update binding to manipulate the values during runtime.

Let's add another indicator to the Linear Scale component and configure it.

1. Select the Linear Scale and right click on the **Customizers > Scale Indicators**.

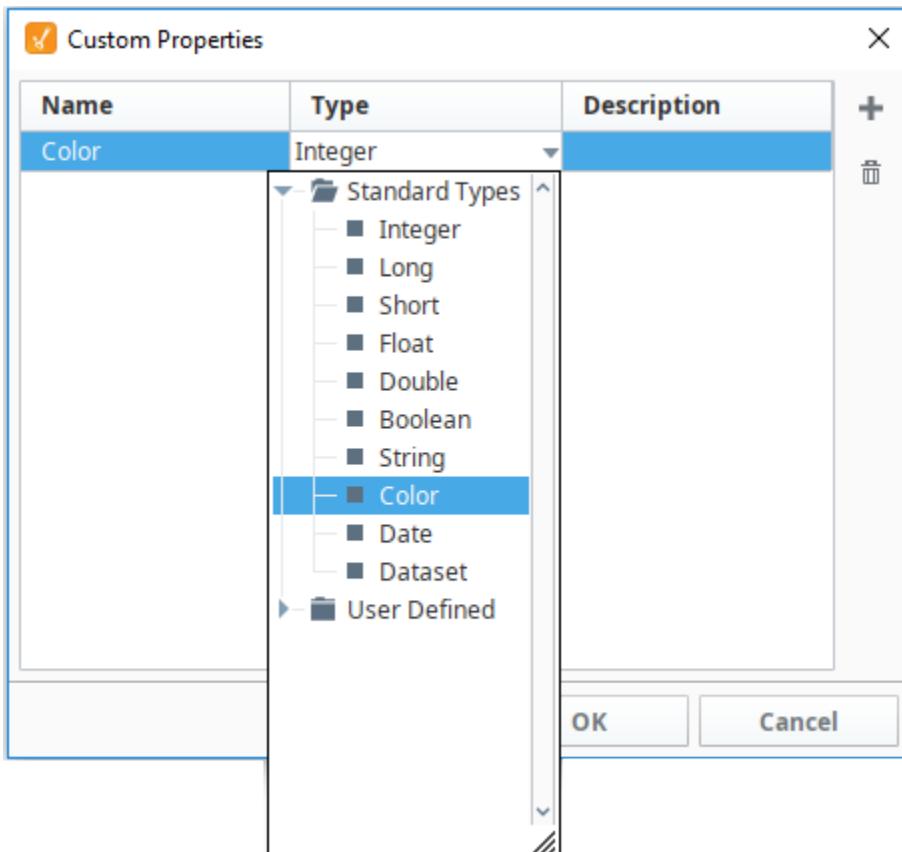


2. Click on the **Add**  icon to add a new indicator.
3. Click the **Arrow** radio button and set the following properties for the Arrow.
 - a. **Value:** 65.0
 - b. **Length:** 50.0
 - c. **Width:** 4.0

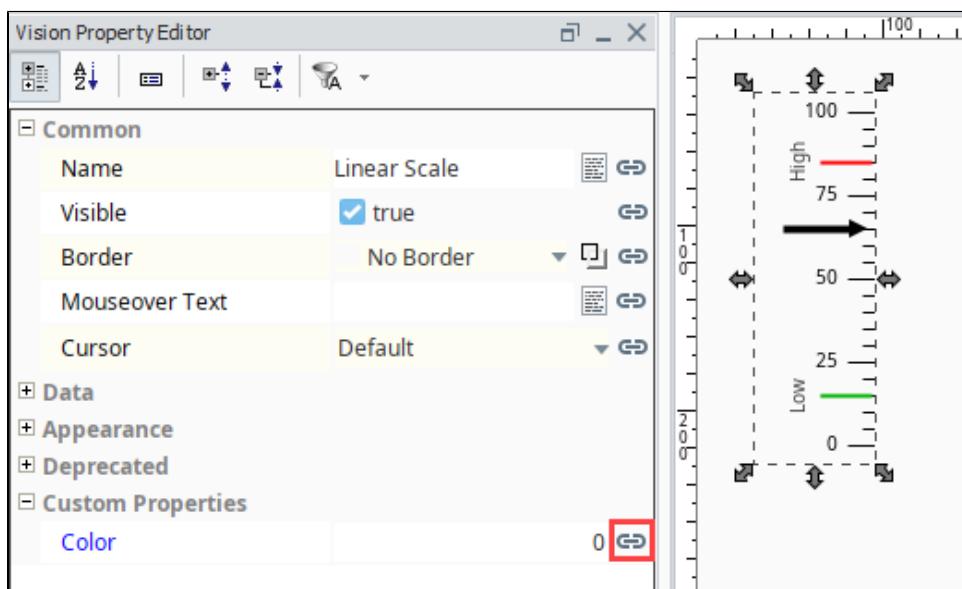


4. Click **OK**. This new indicator will be whatever the current value is.
5. In addition to changing the value of the realtime Indicator, you can also change its color based on its value. If the value is above the high indicator (i.e., 85), the Arrow will change to red, and if the value is below the green indicator (i.e., 15), it will change to green. To accomplish this, setup a [custom property](#) on the Linear Scale of type color:
 - a. With the Linear Scale selected, right click on the **Customizers > Custom Properties**.
 - b. Click the **Add** icon to add a property.
 - c. Enter "Color" as the Name of the property.
 - d. In the Type column, choose Color from the dropdown list.

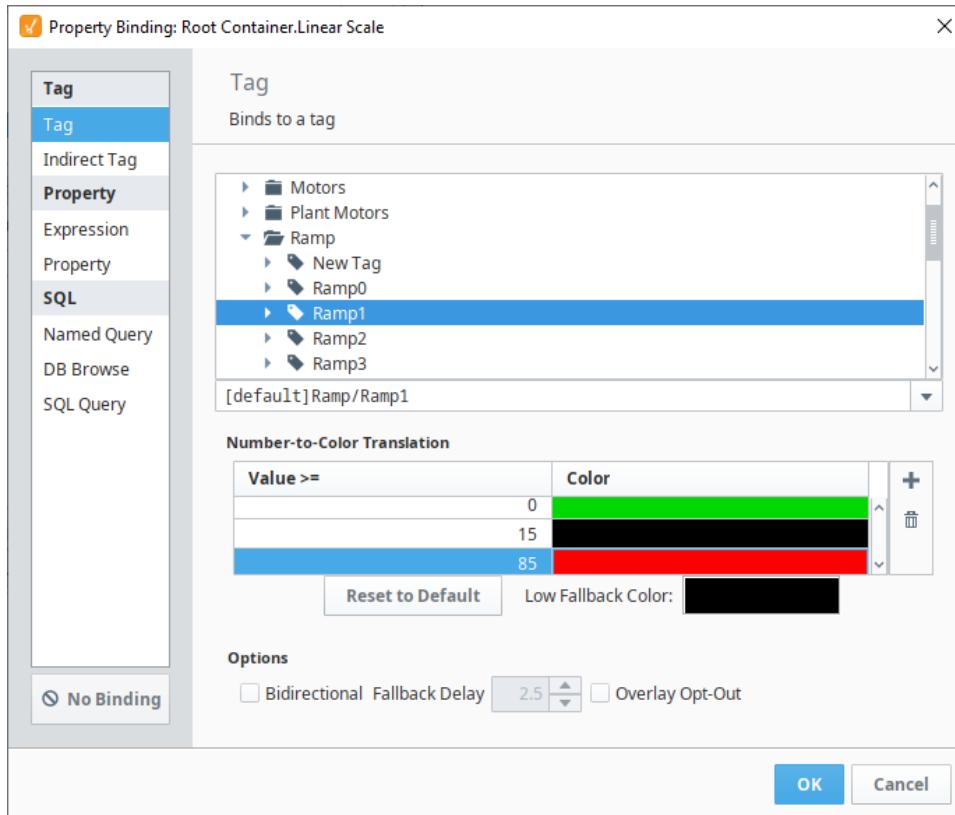
e. Click **OK** to save the custom property.



6. Next, we'll setup a **Number-to-Color** Tag binding on the custom **Color** property. Select the Linear Scale component, then click the **binding** icon next to the **Color** property.

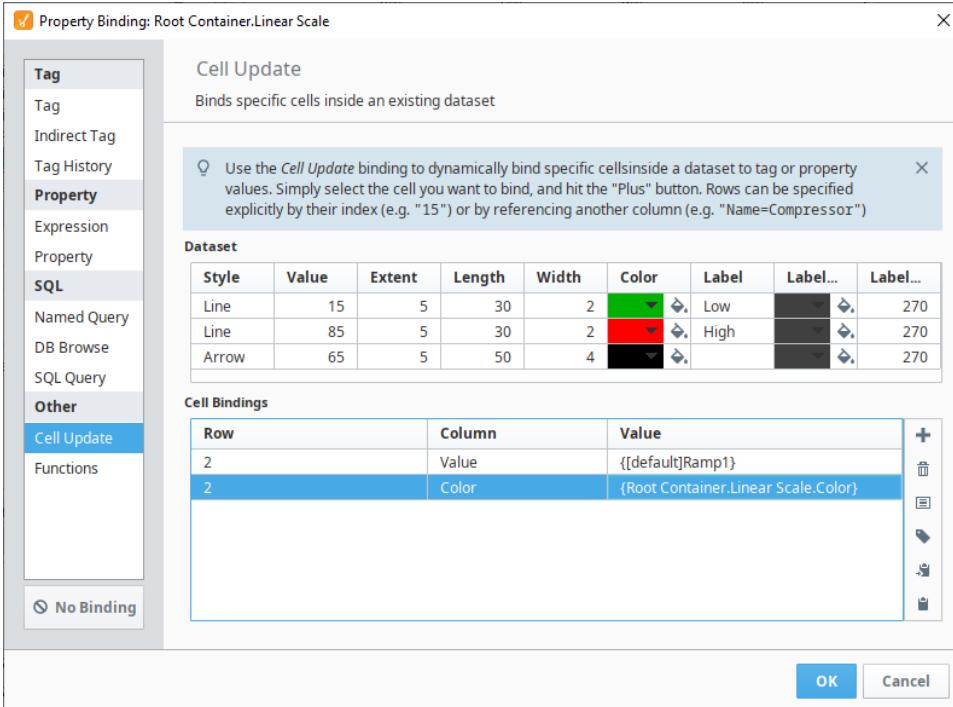


7. The values shown in the image below are based on the static setpoints that were set in the Linear Scale customizer.

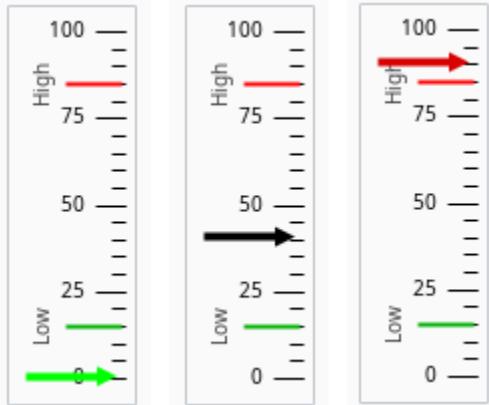


8. Finally, setup a Cell Update binding on the **Indicators** dataset property of the Linear Scale. Notice below that both of the cell bindings are for the same row, just different columns. This is fine, as long as there aren't duplicate bindings for the same cell.
9. Bind the Value cell to the same Tag used in the custom property Color binding to get the value:
 - a. Click the **Add**  icon.
 - b. Click the **Tag**  icon.
 - c. Select the **Ramp1** Tag,
 - d. Click **OK**.
10. Bind the Color cell to the Color custom property we just created.
 - a. Click the **Add**  icon.
 - b. Click the **Property**  icon and select the **Color** property.
 - c. Click **OK**.

11. Click **OK**.



12. Now, the Linear Scale has an Indicator that moves to show a realtime value and changes color whenever it goes outside the setpoints. The image below shows the indicator below the setpoint, within range, and then exceeding the setpoint.



Related Topics ...

- [Function Bindings](#)
- [Property Bindings in Vision](#)
- [Tag Bindings in Vision](#)
- [Indirect Tag Bindings in Vision](#)

Function Bindings

Binding Properties to Prebuilt Functions

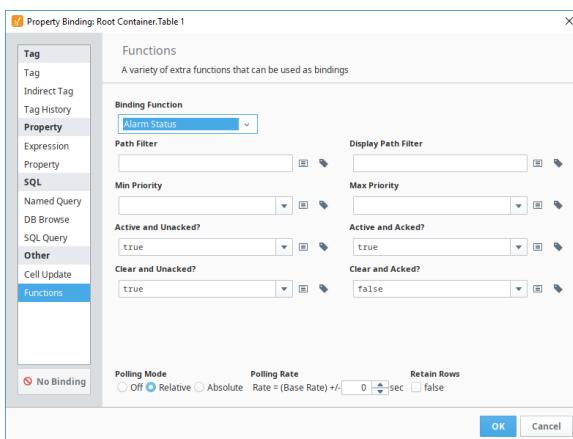
The Function Binding is a generic binding type that lets you bind a dataset property to the results of a function. It allows any of the function's parameters to be calculated dynamically via Tag and property bindings. The function that you choose determines the parameters that are available. The most common functions are the Alarm Status, Alarm Journal, and Audit Log functions, but there may be more depending on the modules you have installed.

Using Function Bindings to Customize

While there may already be an Alarm Status Table component, it may not be able to customize to exactly fit your needs. The Table component has many more customization options. When you use a Function Binding on the alarm status, you can pull that same data into a Table component, and then customize the table exactly how you need.

The following example shows the default settings for the Function binding. Notice how each field has Tag and property binding buttons next to them. This allows you to make this entire function dynamic by binding either Tag or property values to the different values of the function.

1. In the Designer, drag a **Table** component into your workspace.
2. Select the binding  icon for the **Data** property.
3. This opens the property binding window. Select the **Functions** binding.
4. Select a **Binding Function** from the dropdown. In this example, we used the **Alarm Status** binding function and the default settings for alarm state.



5. Now our Table component is pulling in similar information as the Alarm Status Table, but we have the freedom to customize the table exactly how we need using any of its available scripting or extension functions, or its customizer. We can also add our own buttons to acknowledge or shelf alarms using our own scripting functions. This may be more work in setting it up than the generic Alarm Status Table component, but it allows for full control over what is being displayed and what can be done with that information.

| EventId | Source | DisplayPath | EventTime ▲ | State | Priority |
|----------------------------|--------------------------|-------------|----------------------|-------|----------|
| 8065def6-de16-4cc7-b79... | prov:default:tagSpe... | | Jun 22, 2020 9:08 AM | 2 | 4 |
| 300db8a-cd32-4290-bb5... | prov:default:tag:An... | | Jun 22, 2020 9:08 AM | 2 | 4 |
| f8a9ffaa-39b4-a33e-a3cf... | prov:default:tag:Writ... | | Jun 22, 2020 9:08 AM | 2 | 4 |
| fea7dd77-c859-4db7-9e0... | prov:default:tag:Sine... | | Jun 29, 2020 2:45 PM | 0 | 4 |
| 9bbea9-96b9-4a33-89e... | prov:default:tag:Sine... | | Jun 29, 2020 2:53 PM | 0 | 4 |
| 5d417457-30e5-4eb4-96f... | prov:default:tag:Sine... | | Jun 29, 2020 3:02 PM | 0 | 4 |
| 85bf1b0a-9afe-404f-89cf... | prov:default:tag:Sine... | | Jun 29, 2020 3:10 PM | 0 | 4 |

| Active Time | Display Path | Current State | Priority | Event.Id | Label |
|------------------|--------------------------|---------------------|----------|---------------|---------------|
| 6/22/20, 9:08 AM | Speed/High Speed | Active, Unacknow... | Critical | 8065def6-d... | High Speed |
| 6/22/20, 9:08 AM | Tank Level 2/Low SP2 | Active, Unacknow... | Critical | 300db8a-c... | Low SP2 |
| 6/22/20, 9:08 AM | Writeable/WriteableIn... | Active, Unacknow... | Critical | f8a9ffaa-3... | Low Tank L... |
| 6/29/20, 3:20 PM | Sine/Sine0/High level | Active, Unacknow... | Critical | 313a384c-b... | High level |
| 6/29/20, 3:22 PM | Sine/Sine2/Low Level | Active, Unacknow... | Critical | d6d93dc5-... | Low Level |
| 6/22/20, 9:08 AM | Tank 100 | Active, Unacknow... | High | b17eadb-... | Low SP |
| 6/22/20, 9:08 AM | Tank 100 | Active, Unacknow... | High | f56369cc-b... | High SP |
| 6/22/20, 9:08 AM | Turbine Number 200 lo... | Active, Unacknow... | High | e89562d9-... | High Wind... |
| 6/22/20, 9:08 AM | Turbine Number 100 lo... | Active, Unacknow... | High | 46ef1ef0-4... | High Wind... |
| 6/22/20, 9:08 AM | Turbine Number 150 lo... | Active, Unacknow... | High | 6f69459e-c... | High Wind... |
| 6/22/20, 9:08 AM | Turbine Number 300 lo... | Active, Unacknow... | High | a2fd48e9-9... | High Wind... |
| 6/22/20, 9:08 AM | High Temp/High Temp | Active, Unacknow... | Medium | 39448cd3-3... | High Temp |

On this page ...

- [Binding Properties to Prebuilt Functions](#)
- [Using Function Bindings to Customize](#)



Function Binding

[Watch the Video](#)

Color Animation in Vision

Using Color on Components

Using color on components is an important part of creating effective HMLs. While static colors can help identify specific features on the screen, dynamic colors can help draw the users attention to certain areas. Making color type properties such as Fill Paint dynamic works a little bit differently than other properties with simple types. There typically aren't Tags of type color, so the way we set up bindings on these types of properties works a little bit differently, and we have a few options available to us.

Using Expression Bindings

You can use the expression language to calculate a color using the `color()` function. If you have a color that depends on multiple properties, then using an express is recommended to evaluate correctly. This first example returns a static color using the Fill Color property.

Expression

```
// binding on the Fill Color property  
color(255,0,0) // static red color
```

This example takes a Tag value and translates it to a color that ranges from white to blue as the Tag value increases.

Expression

```
// binding on the Fill Color property  
color(255,255,255-({tag value}/100*255)) // fades from white to blue when Tag value goes from 0 to 100 %
```

If you have multiple properties or Tags, you can use the logic Expression Functions to select between a few colors.

Custom Properties

```
if({Tag1}>50,  
    if({Tag2},  
        3, // if tag1>50 and tag2 is true  
        1), // if tag1>50 and tag2 is false  
    if({Tag3},  
        2, // if tag1<=50 and tag3 is true  
        0)) // if tag1<=50 and tag3 is false
```

This example takes one integer value and selects from several options.

Expression Referencing a Tag

```
// binding on the fill color property  
switch({HOA tag},  
    0,1,2, // off, on, hand  
    color(255,0,0), color(0,255,0), color(255,255,0), // red, green, yellow  
    color(0,0,0)) // black (fallback color)
```

The Number to Color Translator

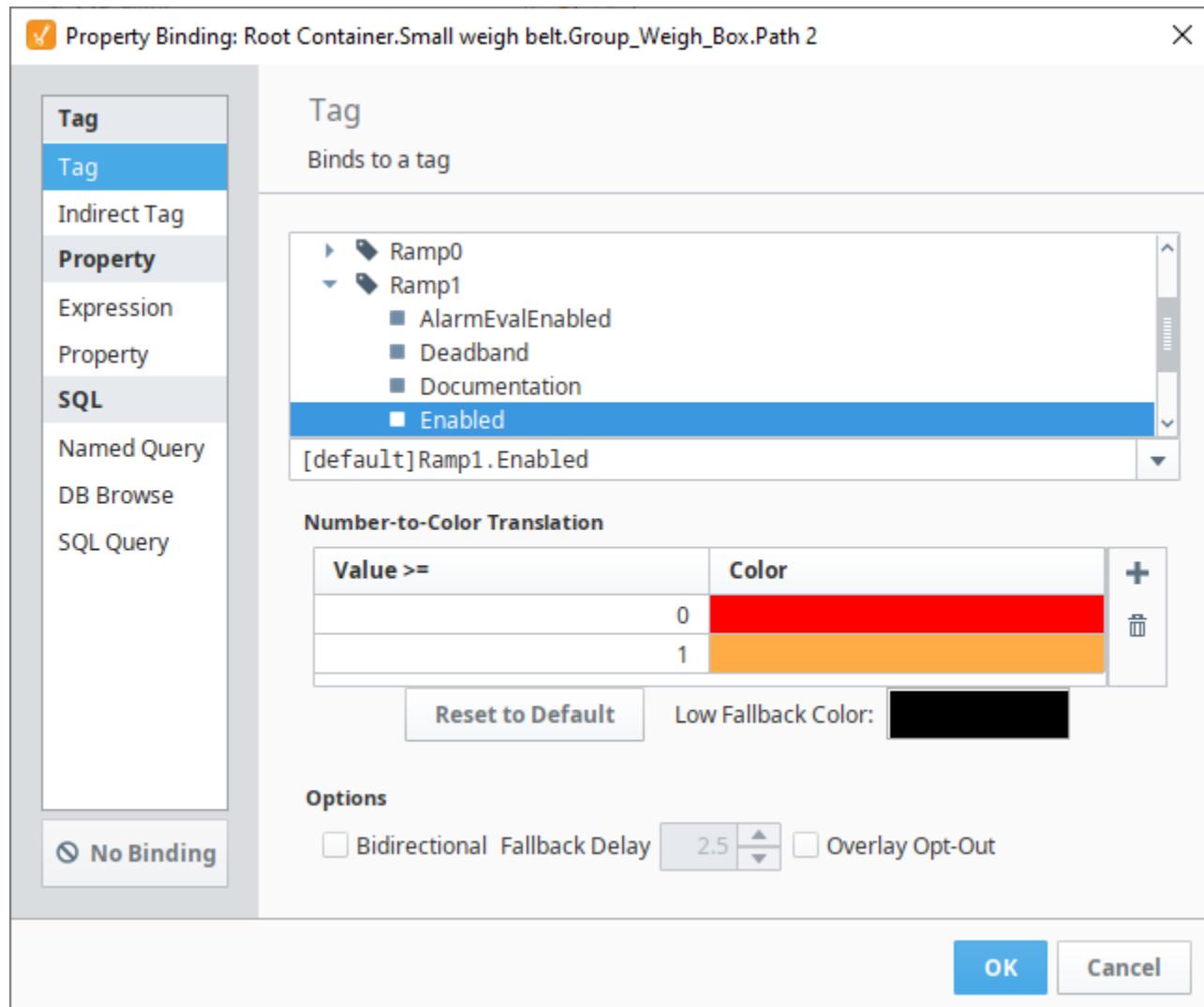
The Number-to-Color Translation, commonly known as Color Mapping is where you map a value to a color within a binding. When selecting a binding type where producing a color won't be possible, the Number-to-Color Translator will appear at the bottom of the binding window. This includes [Property Bindings](#), [Tag Bindings](#), and [Indirect Tag Bindings](#). The way the Number-to-Color Translator works is that for every number range there is a set color. The binding then translates the numeric value into a color based on the mapping table. You can choose a different color for each value, and even

make it blink between two different colors. If you need to add or remove values, use the **Add New Translation**  icon or **Delete Selected**

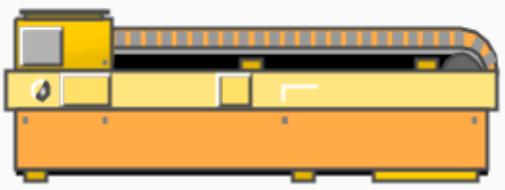
On this page ...

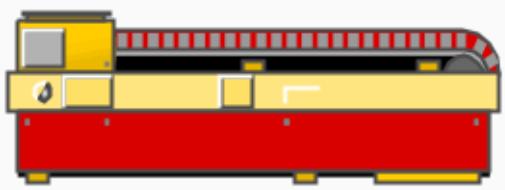
- [Using Color on Components](#)
- [Using Expression Bindings](#)
- [The Number to Color Translator](#)
- [Style Customizer](#)
 - [Style Customizer Window](#)
 - [Value Conflict](#)
 - [Style Customizer Example](#)

Translation  icon on the right side of the Number-to-Color Translation table. There is a **Low Fallback Color** option so when a value falls below your lowest value, a default color can be set.



In this example, the fill color two parts of the conveyor symbol has been bound to the Ramp1 tag, enabled value. When the Ramp is enabled (value = 1), the symbol displays parts in the normal, yellow color. When the Ramp is disabled (value = 0), the fill color on those two parts is red, indicating the conveyor belt is not running. For more information on how this was applied to a symbol, see [Images and SVGs in Vision](#).

| | | | |
|---|-------------------------------------|----------|--|
|  Ramp | | | |
| ▶  Ramp0 OPC | 636.96 | Double |  |
| ▶  Ramp1 OPC | 27.72 | Double | |
| ■ Enabled | <input checked="" type="checkbox"/> | Boolean | |
| ■ OpcItemPath | ns=1;s=... | String | |
| ■ OpcServer | Ignition ... | String | |
| ■ Quality | Good | String | |
| ■ TagGroup | default | String | |
| ■ Timestamp | 2019-0... | DateTime | |

| | | | |
|---|--------------------------|----------|--|
|  Ramp | | | |
| ▶  Ramp0 OPC | 983.79 | Double |  |
| ▶  Ramp1 OPC | disabled | Double | |
| ■ Enabled | <input type="checkbox"/> | Boolean | |
| ■ OpcItemPath | ns=1;s=... | String | |
| ■ OpcServer | Ignition ... | String | |
| ■ Quality | Bad_Dis... | String | |
| ■ TagGroup | default | String | |
| ■ Timestamp | 2019-0... | DateTime | |
| ■ value | disabled | Double | |

Style Customizer

Many Vision components support the **Style Customizer**, which lets you define a set of visual styles that change based on a single **Driving Property**. Typically, you'll have a property on your component that you want to use as a driving property (like a discrete state), which then drives multiple visual properties, like the font, border, and foreground color, to change to a specific style that was setup per state beforehand. Style Customizer lets you define these relationships all at once, and lets you preview them too! Without styles, you would have to go to every property and bind them all individually.



Component Styles

[Watch the Video](#)

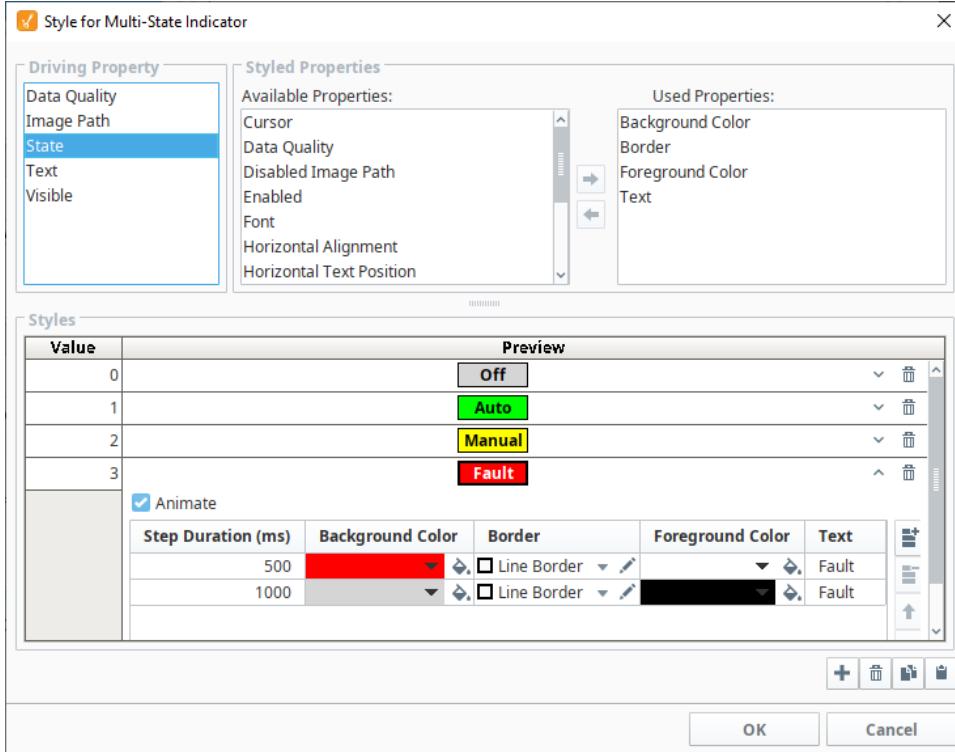
Style Customizer Window

The Style Customizer window has multiple parts to it.

- **Driving Property** - The value of the selected property will be used to determine the style used. Only certain properties on the component can be used as driving properties, but the most common are discrete state properties. Custom Properties can also be used here.
- **Styled Properties** - Here you can select which properties will be used in the styles. Any properties that are in the left panel are available to be used in the styles, while properties in the right panel are already being used in the style. Properties can be moved between the panels by selecting it and clicking the appropriate arrow button.
- **Styles** - The list of styles that will be available for this component. Each style has a Value property on the left. When the value of the Driving Property is greater than or equal to the value of a style, that style will be applied to the component. Each style gives a preview of what it looks like, and can be expanded to by clicking the expand icon  to edit the properties within that style.

You will notice in the image below that the properties being used in the Styled Properties are the Background Color, Border, Foreground Color, and Text, which corresponds to the properties we have available within each style in the Styles area. Each style can also be animated by clicking the animation checkbox. This allows you to add different steps to the style, where each step of the style can have its own unique style. Each step also gains a Step Duration (ms) property that is used to determine how long the step is active for, as shown in the fourth

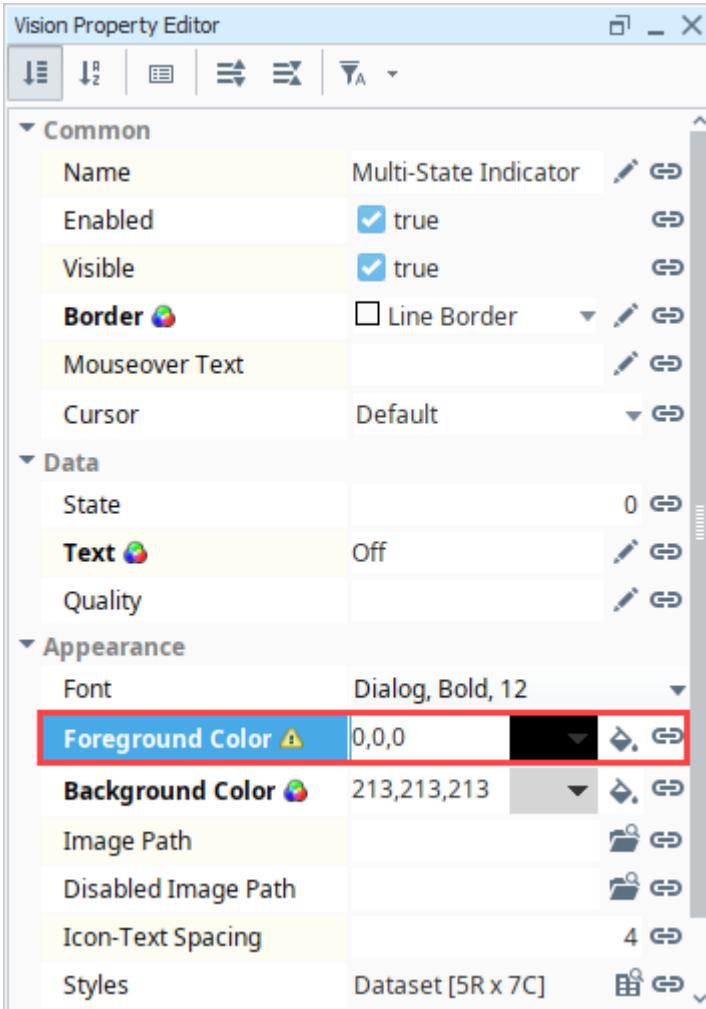
row. This is typically used to create a flashing effect, where the component will flash between two different colors such as red and gray.



Value Conflict

You can bind a property that is already being used by a style, but a warning icon will appear on the property in the Property Editor. This means there is a conflict between the binding on the property, and the style on the component. As a general practice, only the style or binding should write to the

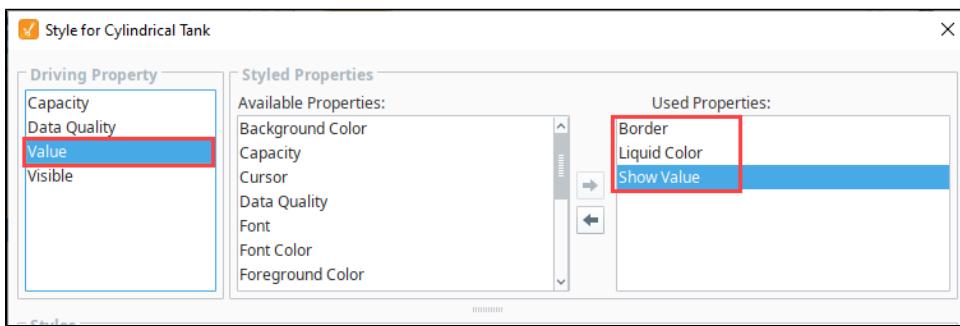
property, not both.



Style Customizer Example

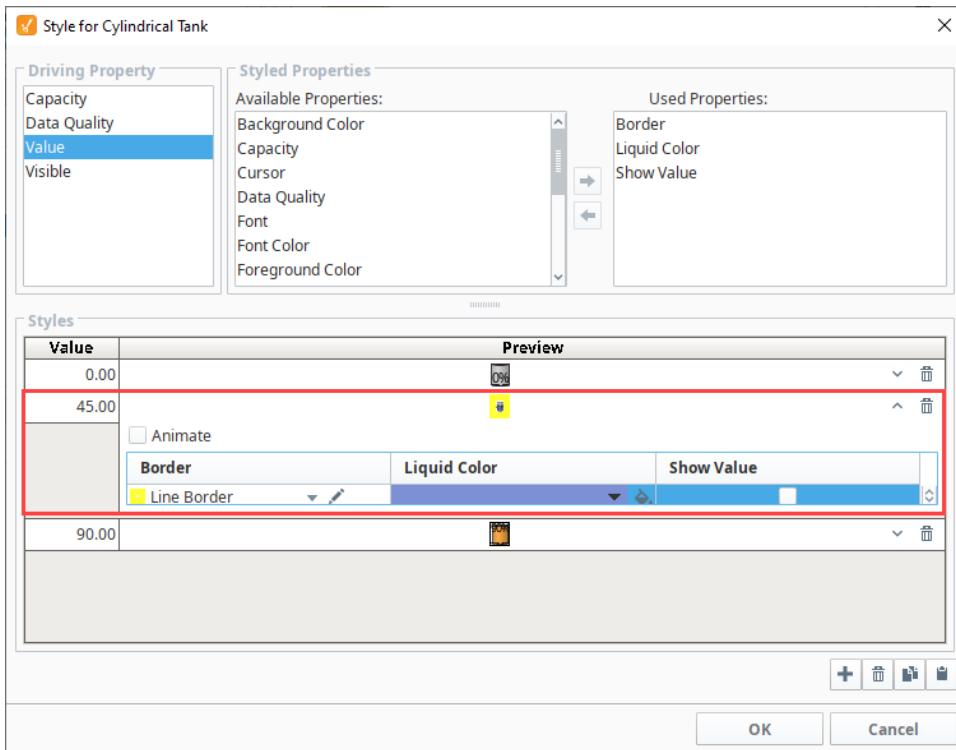
The best example of the Style Customizer in action is the [Multi-State Indicator](#), as this component uses the style customizer to work properly and switch between different states, so it can be used as an example already built in. However, the many other components can use the Style Customizer, so this example sets up styles for a Cylindrical Tank.

1. Add a Cylindrical Tank component to the window, and add a Tag to the **Value** property. (This example uses a Rampt1 tag).
2. Right click on the Cylindrical Tank and go to **Customizers > Style Customizer**.
3. Select a **Driving Property**. Here, the Value is a good choice as we can change the tank to flash when the contents get too high.
4. In the **Styled Properties** select the Border, Liquid Color, and Show Value.



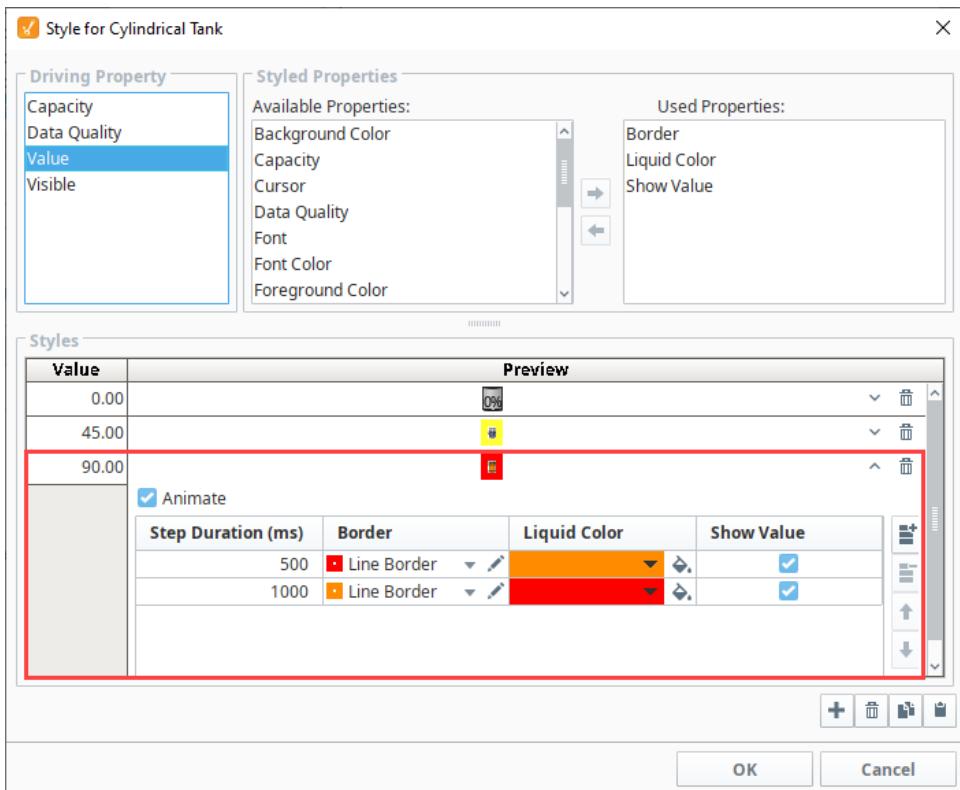
5. Next, we need to set up the different styles. Click the **Add** icon three times to add three styles.
6. Leave the first style set to Value 0.00, and don't change any of the other settings.
7. Set the second style to Value 45.

- Click the **Expand** icon.
- In the Border Chooser, select the **Line Border** style, set the line width to 5px, and the line color to yellow. This way, it is obvious the tank is filling up.
- Set the **Liquid Color** to blue or keep the default color.

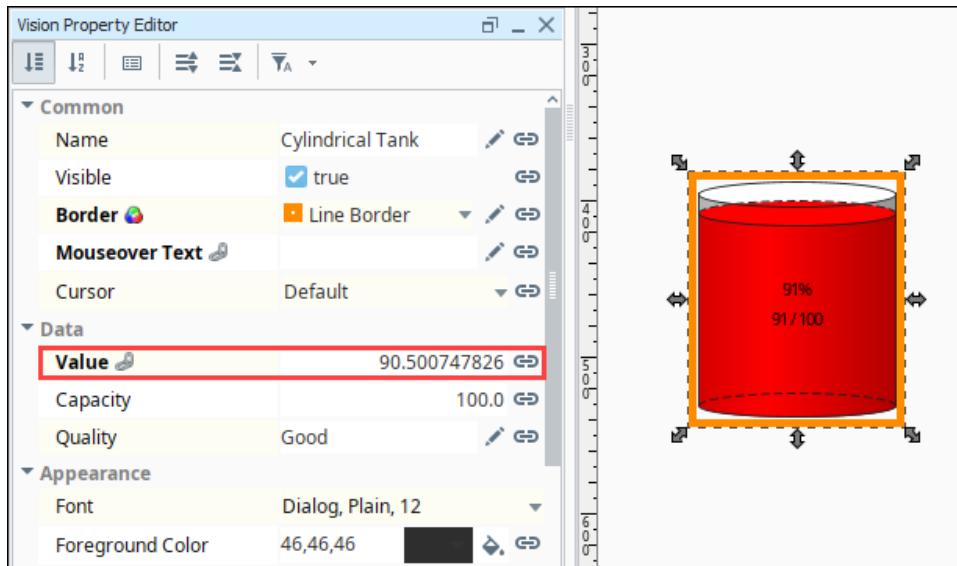


- For the third style, we're going to animate it and create two steps to alert the user that the tank is almost full. Set the third style to **Value 90**, then click the **Expand** icon. Set the following options for the first step:
 - Animate checkbox** - checked
 - Step Duration** 500 ms.
 - Border** - Line Border Orange and set the Line Width to 5px, and the line color to Orange.
 - Liquid Color** - Orange
 - Show Value** checkbox - checked

- Click the **Add** icon to add another step. Set the following options for the second step:
 - Step Duration** 1000
 - Border** - Line Border Red and set the Line Width to 5px, and the line color to Red.
 - Liquid Color** - Red
 - Show Value** checkbox - checked



10. Click **OK** to save the style.
11. In the Vision Property Editor, when the **Value** changes to $>= 90$, the Tank component and border will change to reflect the style settings. The tank colors and border will flash between red and orange.

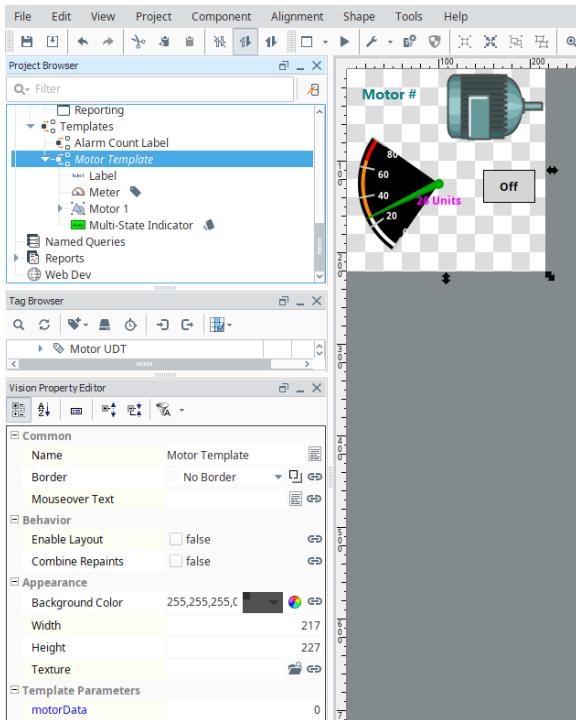


Vision Templates

Templates are a simple but a very powerful feature in Ignition that you can use with the Vision windows. The power comes from the ability to modify only the template in one location while affecting all of the instances used throughout the project. HMI and SCADA systems typically have a lot of repetitions in their screens. You might use a group of the same components over and over within your project. The data driving each set of graphics is different, but the graphics themselves are copies of each other. You can make a single template and use instances of the template over and over again.

When using templates, you define the graphical display in one place, called the master template. You then use this master template many times in your project on multiple windows, thus making a number of template instances. Any changes made to the master template are then reflected in all of the template instances. Using templates early in your project development for any repeating displays can save a significant amount of time later on.

Without templates, the only way to do this is to copy-and-paste the components then re-bind them each time you want another. This is simple, and it works, but it can cause major headaches and time consuming corrections later on because if you ever want to make a change to how they are represented, you're stuck making the change to each copy of the group.



Template Properties

Template Properties (called Template Parameters) allow each template instance to reference different data. Because the primary use of templates are the ease of maintaining repeated user interface elements, correct use of Template Parameters is very important. This is very similar to the concept of [Parameterized Popup Windows](#). In that case, any **Custom** property on the **Root Container** of the window is used as a parameter, and is passed into the window when it is opened. With Templates, you have a property in the root of the master template that is exposed when you drop a Template Instance on a window.

Template Parameters and Internal Properties

When you open the **Custom Properties** window (right-click the checkered-box of the template and select **Customizers > Custom Properties**), you'll notice it is different than the Custom Properties of all other components. There are two kinds of custom properties here, as follows:

- **Template Parameters**

These parameters appear on each **template instance**, allowing each instance to be configured differently. Commonly, this is some sort of indirection. For example, if you have a template representing motors, you might have **MotorNumber** as a parameter property. Then you can use that property as an indirection variable in other bindings within the template. Parameter properties are not bindable from **within** the template master design. When you use the template to create a template instance, the property becomes bindable. This ensures that the property only has a single binding configured for it.

- **Internal Properties**

These properties cannot be used as parameters in your instances. They show up when designing the template master, but it does not show

On this page ...

- [Template Properties](#)
 - [Template Parameters and Internal Properties](#)
- [Indirection and UDT Tags](#)
 - [Standard Indirection](#)
 - [UDT Parameter](#)
- [Changing Template Path](#)
- [The Drop Target Parameter](#)
- [Resizing Templates](#)
- [Nested Templates](#)
- [Accessing Components Inside a Template Instance](#)



Template Overview

[Watch the Video](#)

up on the template instances. Internal properties are bindable from within the template master design. These properties are intended to be used for the internal workings of the template.

Indirection and UDT Tags

There are two primary ways to achieve indirection when using templates. Let's continue to use the example of a motor. Your system has many motors in it, and your template is used to display the status of the motors and control the motor's running mode. The goal is to be able to drop instances of the template onto your windows, and configure them in a single step to point to the correct motor's Tags.

Standard Indirection

If the Tags representing the datapoints of each motor are arranged in an orderly way in folders or with a consistent naming convention, you can use [standard indirection](#) to configure your template. You can add a parameter such as MotorNum to the template. Then you configure the contents of the template using indirect Tag binding, where the value of MotorNum is used for the indirection.

UDT Parameter

If your motors are represented by a [User Defined Type \(UDT\)](#), you can save some effort and [use a property of that type](#) directly. Make your indirection property the same type as your custom data type. Then inside your template, you can use simple property bindings to create a link to the members of the UDT. When you create a template instance, you can simply bind that property directly to the correct Motor Tag, and all of the sub-Tags of motor are correctly mapped through the property bindings.

Changing Template Path

An instance of a template on a window has a property called **Template Path**. You can change this property on a window dynamically, and it can be bound to anything that produces a valid template path. For example, if there are two tank templates in a folder called Tanks, one template is called Tank A and the other is called Tank B. Each tank has a different look, but they have the same Custom properties. Their respective template paths are Tanks/Tank A and Tanks/Tank B. The template rendered on a window can swap between Tank A and Tank B by binding the instance's Template Path property to any string reference that says Tanks/Tank A or Tanks/Tank B.



**INDUCTIVE
UNIVERSITY**

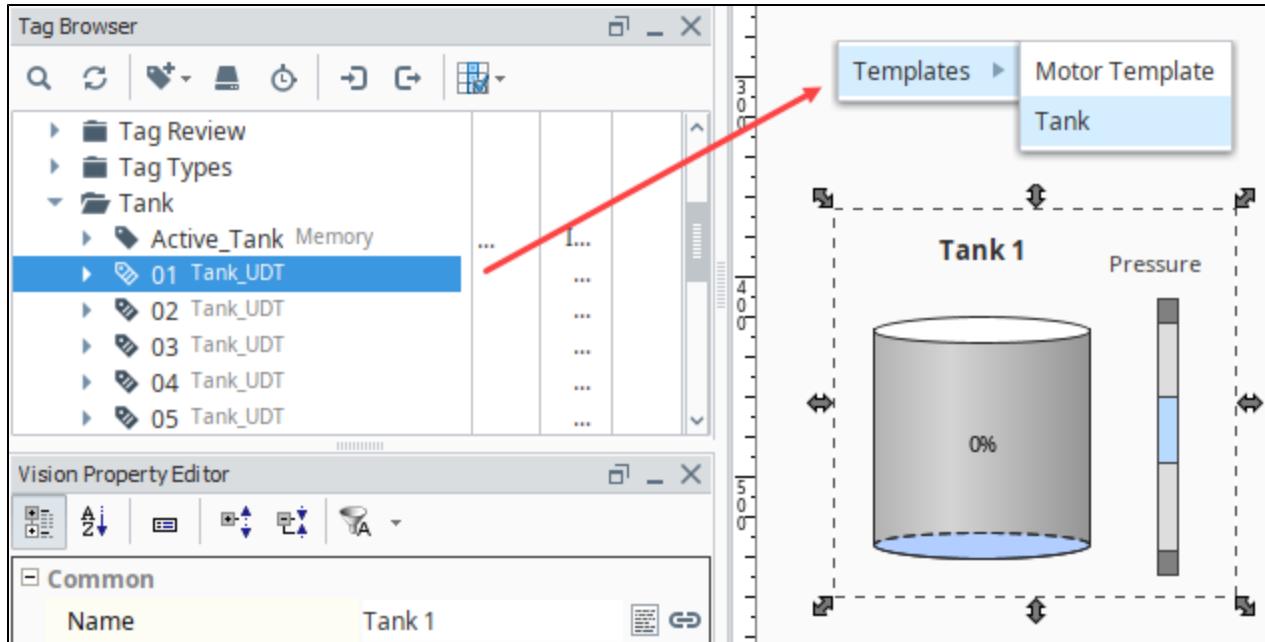
**Changing Template
Path**

[Watch the Video](#)

The Drop Target Parameter

When you specify parameters in the Custom Properties window (right-click the checkered-box of the template and select **Customizers > Custom Properties**), you can set one of the parameters as the Drop Target. This allows you to drop a Tag of that type onto your template instances or onto a window to facilitate even quicker binding. For example, let's say that you have a parameter that is an integer and you've made it the drop target. If you drop an integer Tag onto a window, your template appears in the menu dropdown list of components which is displayed. Choosing your template creates a template instance and binds that parameter to the Tag.

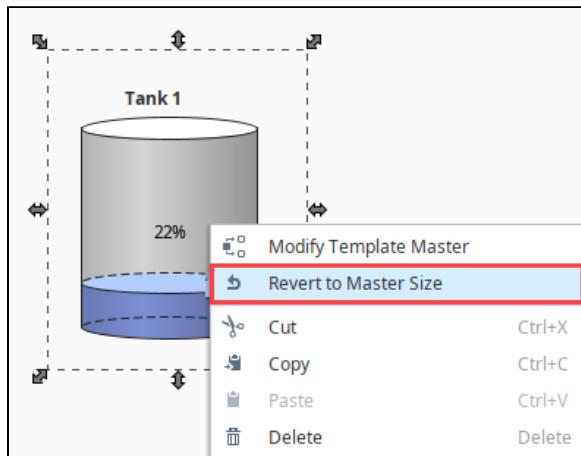
This also works for UDT Tags. Let's say you have a custom data type called Motor and a template with a Motor-typed parameter set as the drop target. If you drop a motor Tag onto a window, it creates an instance of your template automatically. If you have more than one template configured with Motor drop targets, you have to choose which template to use.



Resizing Templates

You can configure the layout of each template so it resizes properly.

By default, when you drag a template into the window from the Project Browser, the size of the instance is exactly the same size as the master template. You can make the size larger or smaller. To go back to the same size as the master template, right-click on the instance and choose **Revert to Master size**.

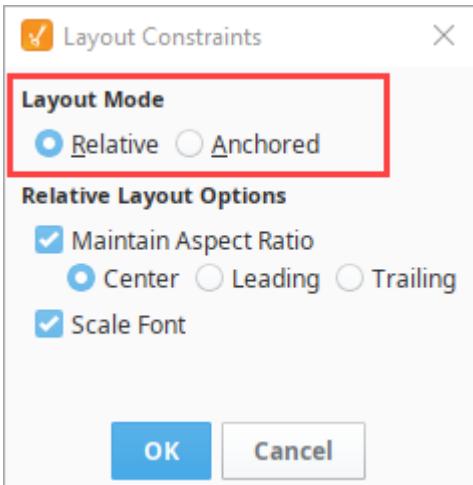


Template - Resizing and Enable Layout

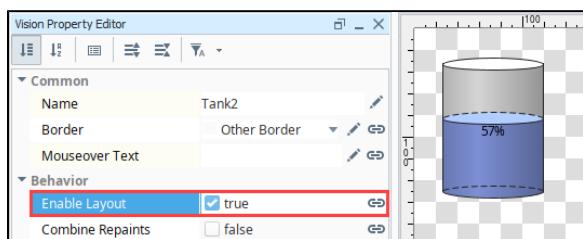
[Watch the Video](#)

For every component or template instance you add to the window, you can pick a Layout option as to how it is going to resize in the Client. Right-click on the instance, select **Layout** from the menu, the Layout Constraints window displays showing all the default settings.

To learn more about layout, see the [Component Layout page](#).



If your template instance resizes in the client, then it will stretch all of the components inside it in the same way a [Component Group](#) works. That is: it will ignore any layout settings and stretch without maintaining aspect ratio. If you want the template instances to respect your layout settings, set the **Enable Layout** property to true in the template definition.



Nested Templates

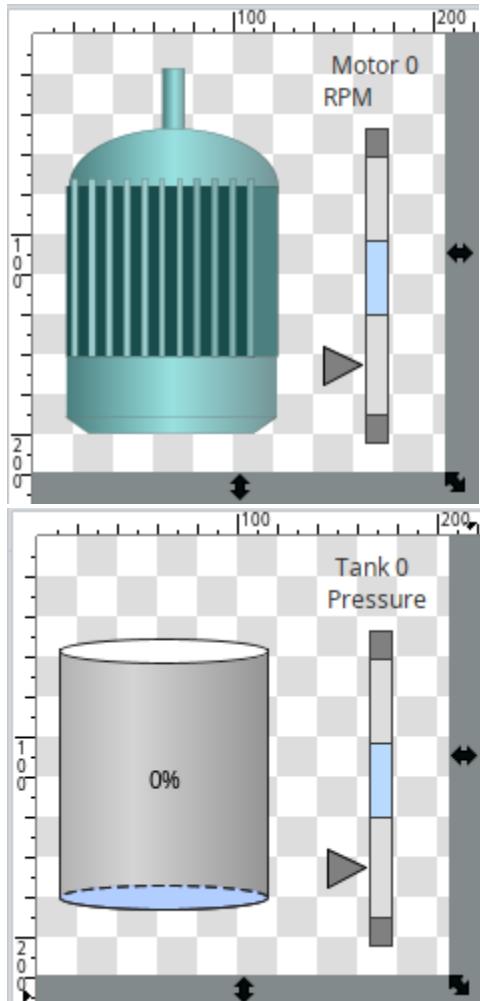
You can embed templates inside of other templates. The nested template behaves like a component. This can be useful if the project can be broken down into many similar, small parts. Instead of building a template for a tank with a gauge, a motor with a gauge, and a compressor with a gauge, it might instead be better to first build a simple gauge template that can then be added to each of three templates so that it already is setup correctly.

Simply drag the already made template into the new template, just like you would onto a window. I can easily use an indicator template that I made to display values from a motor template, or values from a tank template, as long as I setup the indicator template with the proper indirection. This way, I only have to setup the indicator once and write in a few parameters, instead of having to customize the indicator the same way for every template that it gets added to.



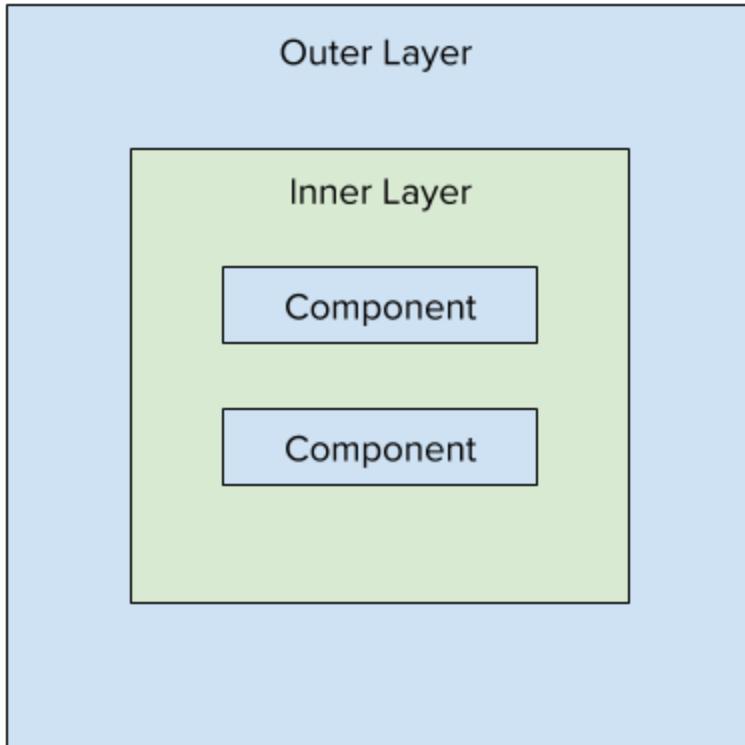
Embedding Vision Templates

[Watch the Video](#)



Accessing Components Inside a Template Instance

When working with a Template Instance, the components inside that instance are normally hidden, and otherwise inaccessible. However, you can access these components via scripting. It helps to think of Template Instances as multi-layer containers. In most cases, users interact with only the Outer Layer, which contains the Template parameters, and the other default Template Instance properties. A Python script can access the Inner Layer, which then provides access to the components within.



A script can traverse to the Inner Layer from the Outer Layer with a `getComponent` call.

Pseudocode - Accessing the Inner Layer

```

myTemplate = event.source.parent.getComponent('MyTemplate')

# The '0' in the first getComponent call effectively refers to an index value of a component, which happens
# to be the Inner Layer.
myTemplate.getComponent(0)
  
```

From the Inner Layer, a script can then call `getComponent` again to access any components within. Assuming a Template with a Label component named "Label", we could access the Text property with the following:

Pseudocode - Accessing a Component From the Outer Layer

```

myTemplate = event.source.parent.getComponent('MyTemplate')

print myTemplate.getComponent(0).getComponent('Label').text
  
```

[In This Section ...](#)

Creating a Template

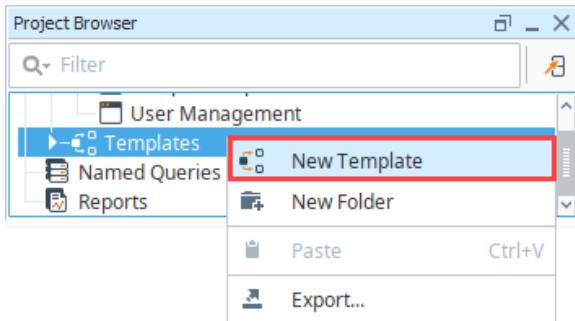
Templates are a simple but very powerful feature that you can use with your Vision windows. Templates are a built once, in one place, which is called the master template. You then create template instances throughout your project. Each of these instances will have the same components and properties as the master template, and will automatically update as changes are made to the master template.

Creating a template is easy. To start, right click on the Templates section of the Project Browser and select New Template.

Basic Template

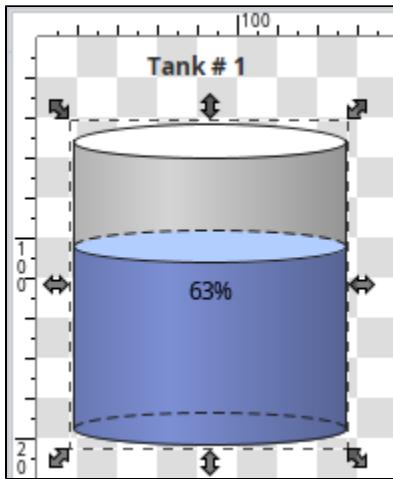
Here we create a basic template. After the template is configured, you can create multiple instances of it throughout your project.

1. In the Project Browser, right-click **Templates** and select **New Template**.



A checkered box is displayed in the design space where you design your template. The checkered box means that the template is completely transparent. You can set a background if you want.

2. Right-click on **New Template** and click **Rename** to change its name to something else, for example "Tank".
3. Drag a Cylindrical Tank and a Label component onto the screen. Resize the components to fill the area of the template.
4. Bind the Value of the tank to a Tag, and set up the label to be a name for the tank.



5. Now that we made a template, use that template on a window. Navigate to a window, and then click and drag our tank component onto the window. The template can be dragged onto the window multiple times to create multiple instances of the template, or even added to other windows.

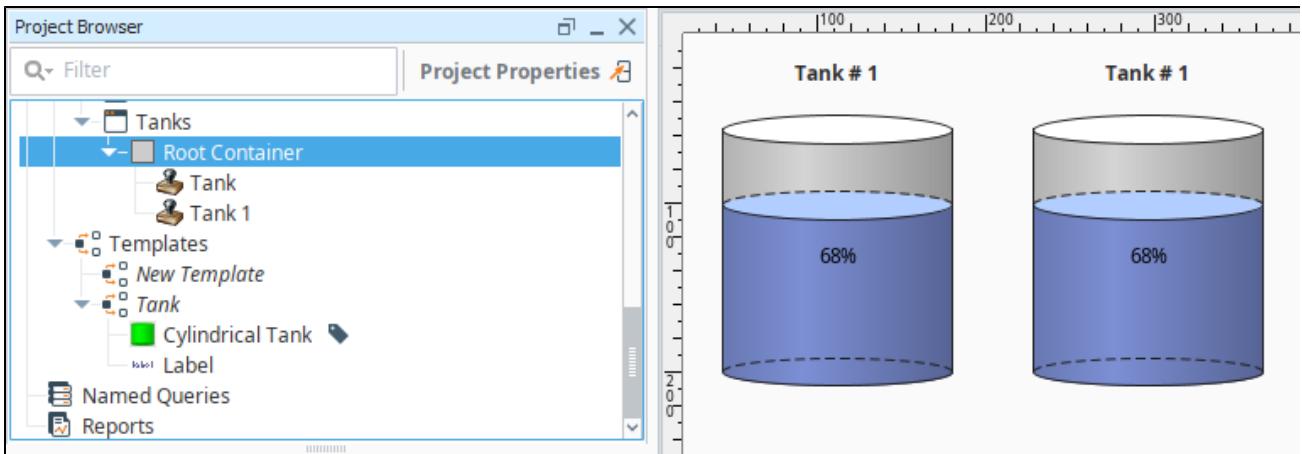
On this page ...

- Basic Template
- Dynamic Templates
- Edit a Template
 - Example - Send a Template to a Different Project
- Template Custom Properties
- Creating the Template Instances



Creating a Vision Template

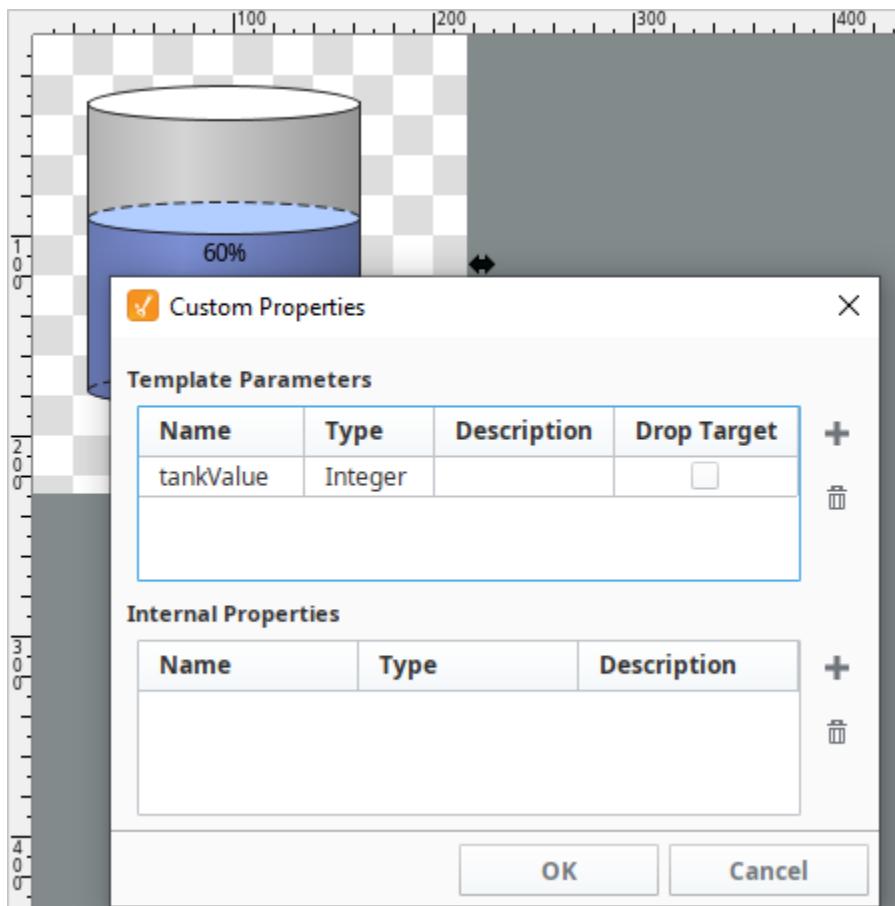
[Watch the Video](#)



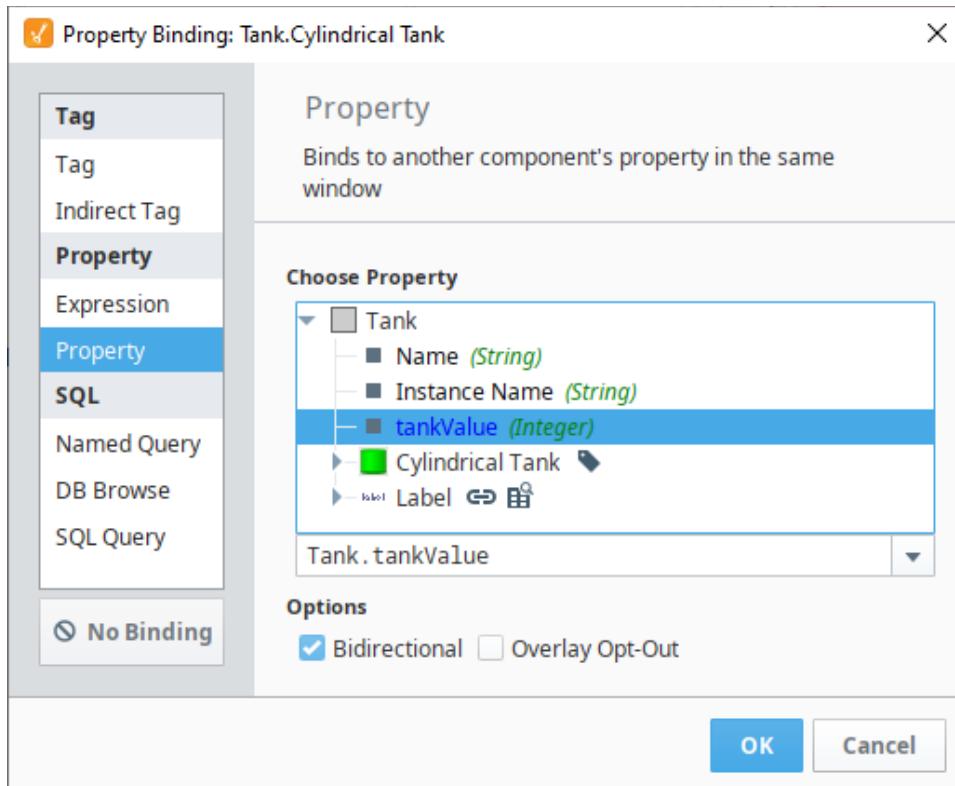
Dynamic Templates

While the basic template is a good example of what a template is, the real power of the template is its ability to be dynamic. The template can create parameters that can accept values from the instance, and use them throughout the template. This way, instead of always displaying Tank 1 values, we can pass in a tank value for the cylindrical tank component to use.

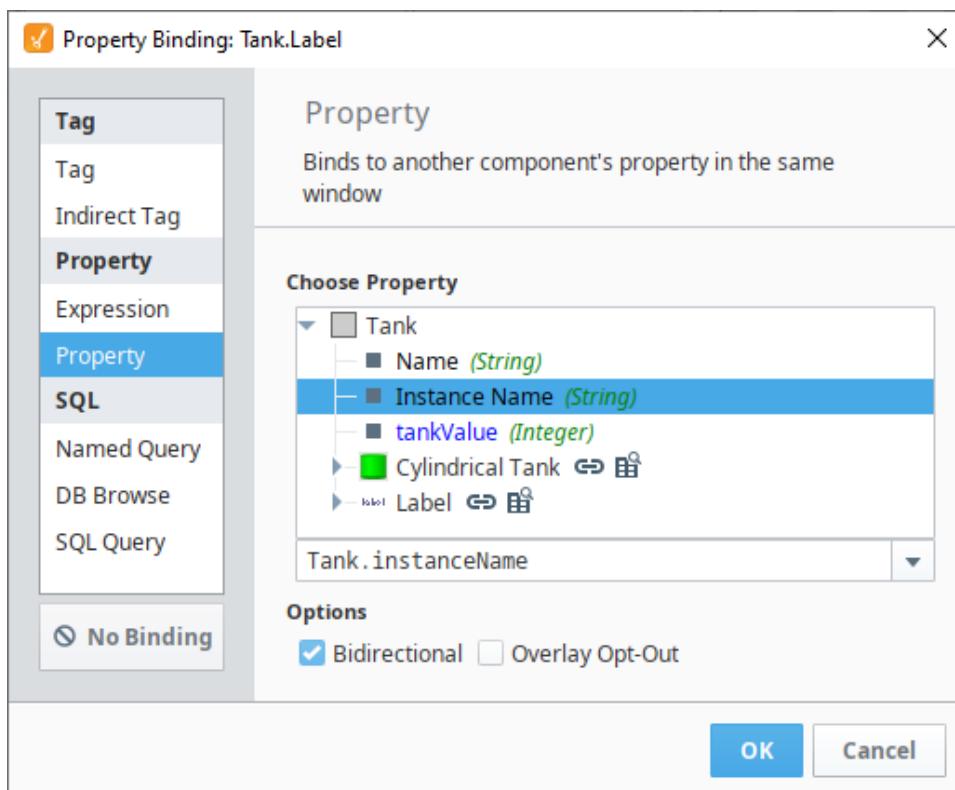
1. In the Tank template, right click on the gray background and go to **Customizers > Custom Properties**.
2. Click the plus button to add a Template Parameter. Give it a name like **tankValue**, and click **OK**. There should now be a new property on the Tank Template object called **tankValue**.



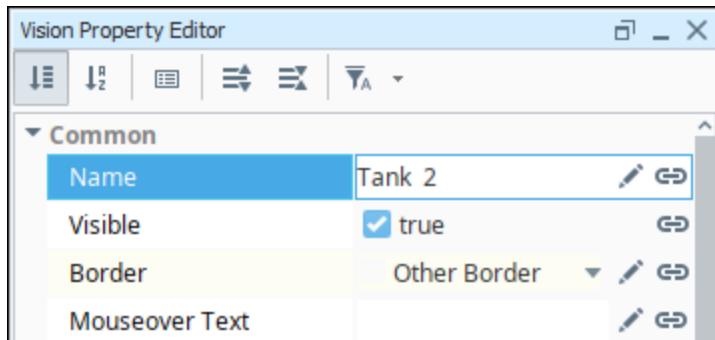
3. Now bind the **Value** property of the cylindrical tank to the **tankValue** property that we just made. Note that it will be 0 right now, but we will pass in a value later.



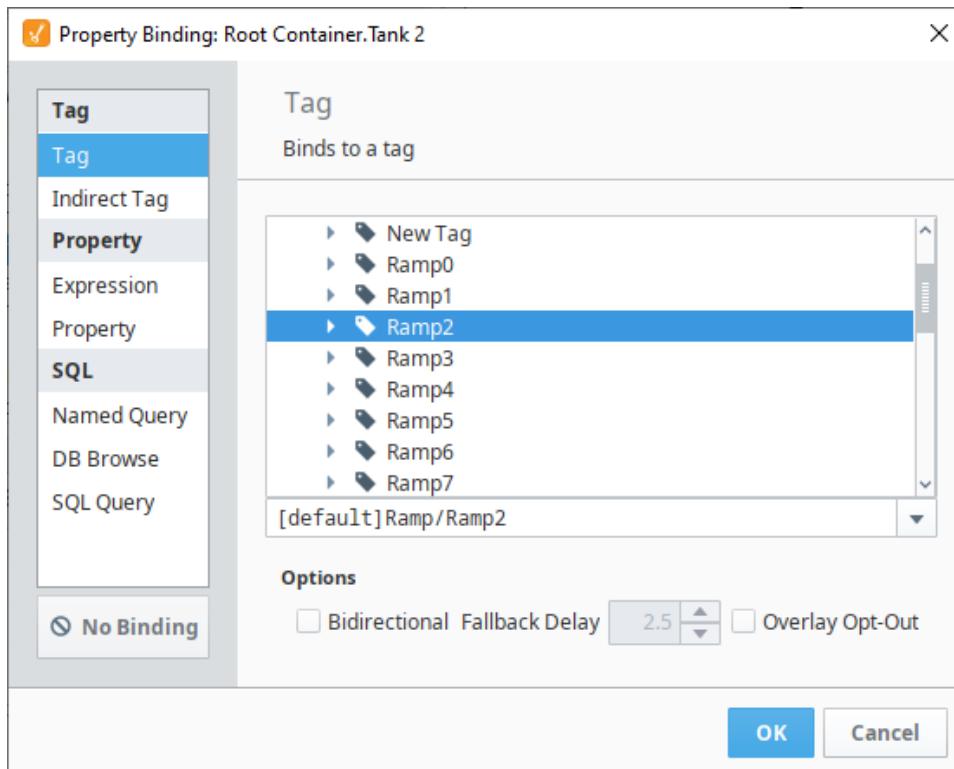
4. We also want to change the label, since this template won't always be pointing at Tank 1. So we can bind the **Text** property of the label to **Tank Template's Instance Name**. This way, whatever we name the template instance is what the label will show.



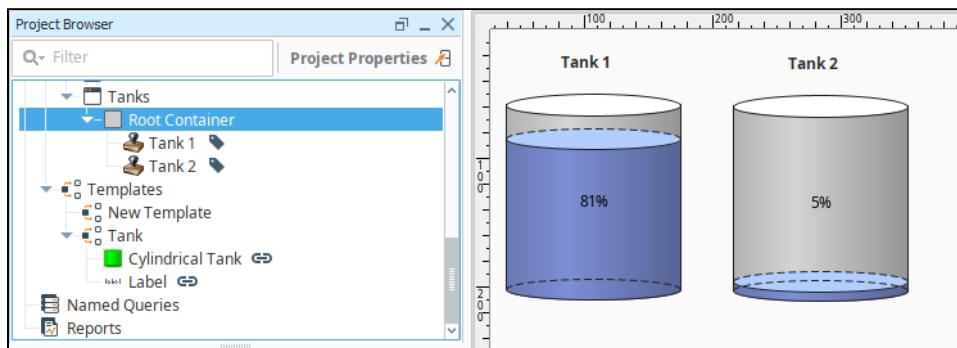
5. Back on our window, we want to change our two instances so that the first one is called Tank 1, and the second is called Tank 2. Notice how the labels change when we change the name of the template. Select your second Tank, and change the **Name** property to Tank 2.



6. We also want to bind the **tankValue** property of each instance to separate Tags. In this example, you can do this simply by selecting Tank 2 and clicking on the binding icon for the **tankValue** property and selecting another Tag.



7. We now have two instances of the same template, but they are displaying different information because we are passing different values into each.



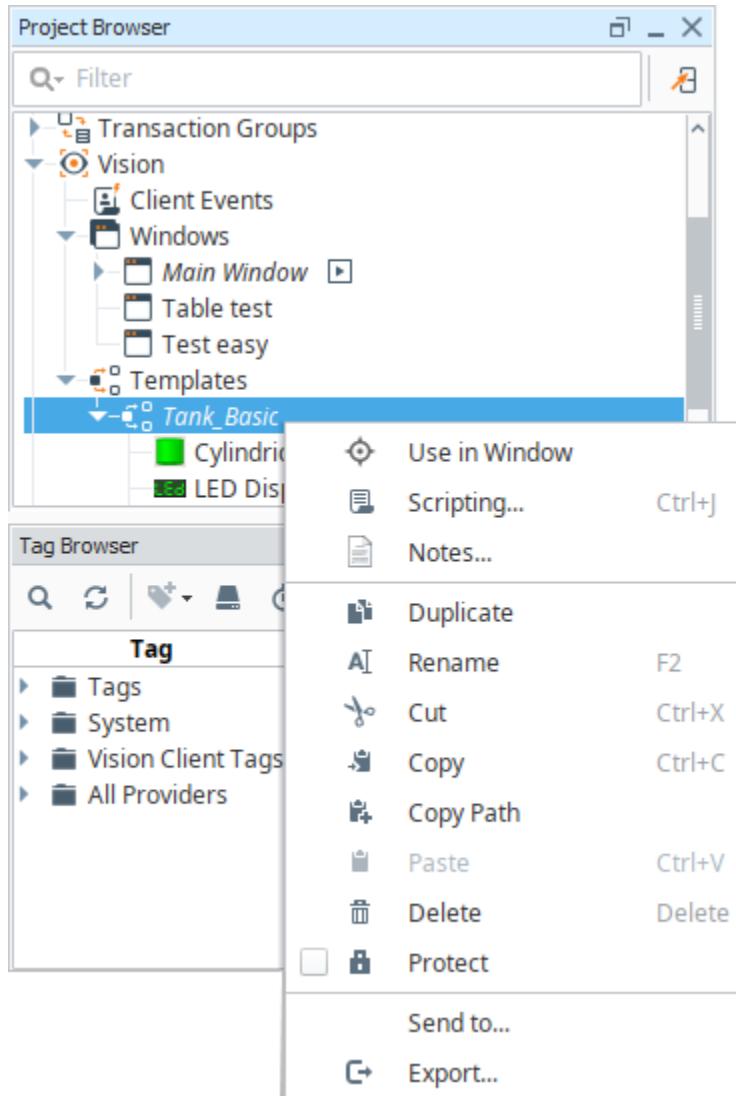
Edit a Template

You can open a template for editing by double-clicking on it in the Project Browser, or by double-clicking any instance of that template within that window. You design your template the same way you design windows: by adding components to it, and configuring those components using property bindings and scripting.

There are a few differences between templates and windows from an editing perspective. Templates, unlike windows, have a transparent background by default. This can be changed simply by editing the background color of the template. Templates also do not have the concept of the "Root Container" - the template itself acts like a container.

Once you change the master template, all the instances of that template are updated.

Templates are a project resource. As such they can be copied, duplicated, protected and more. There are several actions available with a right click menu.



| Action | Description |
|---------------|--|
| Use in Window | Places the selected template on the current window. |
| Scripting | Opens the Component Scripting Window where you can set scripting on this template. For more information, see Script Builders in Vision . |
| Notes | Opens a popup window where you can make notes about the template. |
| Duplicate | Duplicates the template in the Templates folder. |
| Rename | Renames the template. |
| Cut | Cuts the template, but leaves it on the clipboard. |

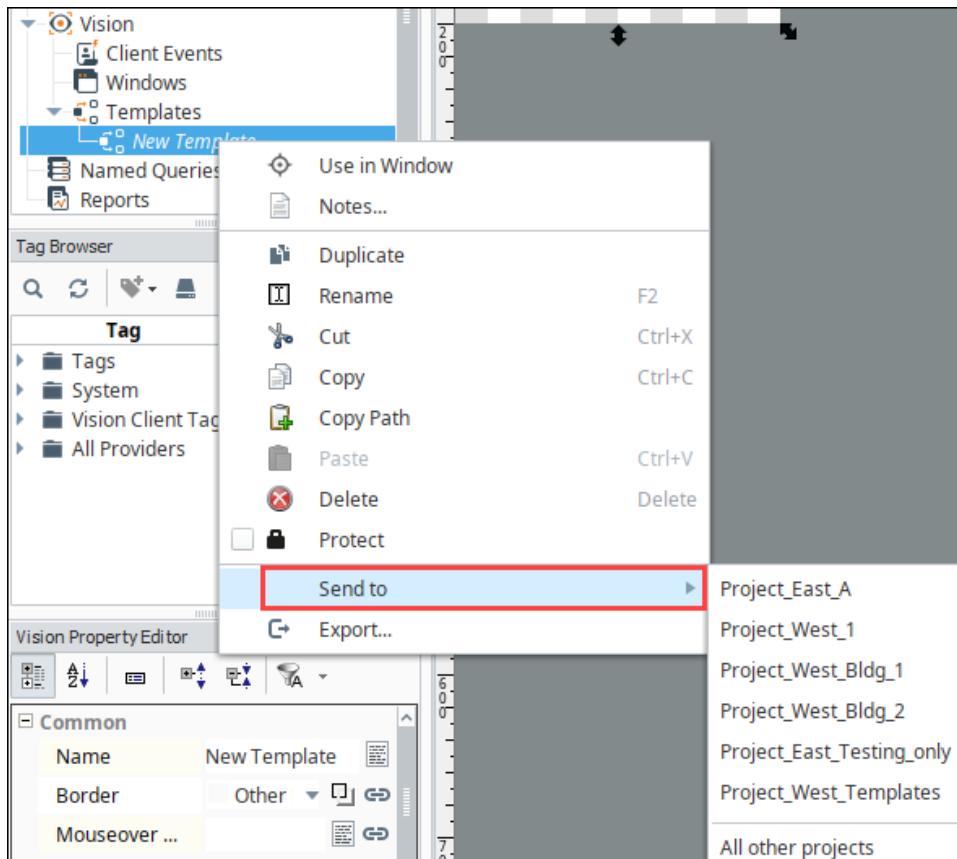
| | |
|-----------|--|
| Copy | Makes a copy of the template on the clipboard. |
| Copy Path | Copies the path to the template and places it on the clipboard. |
| Paste | Pastes a template that's currently on the clipboard. |
| Delete | Deletes the selected template. |
| Protect | Once a project resource protected, it cannot be changed except by someone that has the permission to unprotect it, and modify it. For more information, see Project Security in the Designer . |
| Send To | Sends this template to another project on this Gateway. |
| Export | Opens export Project Resources window. where you can export this template and other resources. For more information, see Project Export and Import . |

Example - Send a Template to a Different Project

You can share Templates with other projects.

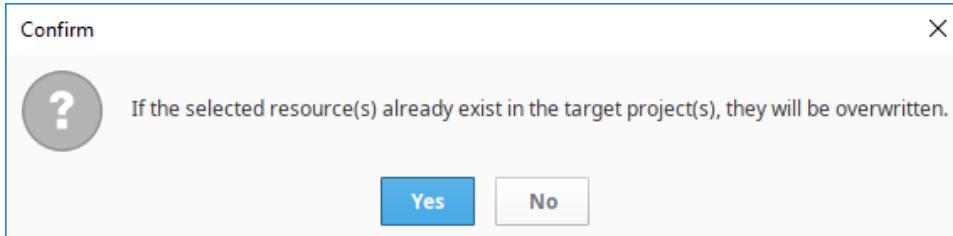
Caution: If a Template by the same name already exists in the target project, it will be overwritten.

1. To send a Template to a different project, right click on the template name and choose **Send to**.



2. A list of existing projects is displayed. Choose a specific project, or select **All other projects** if you want to send the template to all projects.

3. A confirmation message is displayed. Click Yes to confirm.



Template Custom Properties

Templates have the capability to incorporate custom properties. In this way, they are the same as any other Ignition components. The main difference between the custom properties for an Ignition component and a template is that the template has internal properties and template parameters.

- **Internal Properties** - Internal properties help facilitate the bindings within a template in the same way that a window's root container will help facilitate bindings between components that make up the template. When a template is deployed onto a window the internal custom properties are not exposed to the world outside the template.
- **Template Parameters** - The template parameters are the template's custom properties that are exposed to the outside world. In other words, when a template is deployed onto the root container of a window, the template parameters are available for binding with the objects on that window or to Tags.

Creating the Template Instances

Once you've made your template, you can use it on any of the windows in your project by doing any one of the following steps:

- You can drag the template from the Project Browser into an open window just like you can drag components into the window for display.
- You can right-click on the template in the Project Browser and choose **Use In Window**, which will let you place the template inside a window with another click.
- You can drag a Tag from the Tag Browser to a window and from the pop-up menu, which is displayed, you can choose a template. This only works if the template has a configured Template Parameter that been enabled as a [drop target](#).

The template instance can then be moved and resized like any other component.

Related Topics ...

- [User Defined Types - UDTs](#)

Template Indirection

Indirection in Templates

You can create templates that point indirectly to a set of Tags based on a simple parameter. This is very helpful when you have a large number of UDTs with the same type of Tags that only differ in one parameter. For example, lets say you have 100 Tank UDTs that all have the same kind and number of Tags. The only thing that is different is the Tank number.

If each Tag inside the UDT has a Tag path that looks like this:

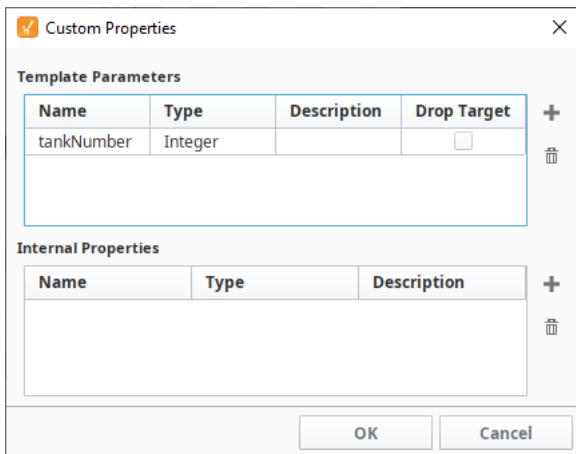
```
Tanks/Tank 1/Volume
```

You can create a Template Parameter on the Custom Properties that can help make the template indirect. There are two main ways of doing this. The first method is to pass in an indirection value so that [Indirect Tag bindings](#) can be set up within the template. In this case, we would want to pass in a tank number, which can be substituted in for the tank number in the Tag Path. The second method is to pass in a reference to the entire UDT, essentially turning that UDT reference into a property that we can use within the template.

Indirect Binding

This example demonstrates, how to set up a parameter to be used as an Indirect Tag binding.

1. Create a new template, and call it Tank. Right click on the background of the template, and select **Customizers > Custom Properties**, and add a Template Parameter called **tankNumber** of type Integer.



2. Next, add some components to our template, being sure to utilize the Template Parameter **tank Number** that we created earlier to make them indirect.

- a. Add a **Label** component at the top with a simple expression binding to clearly display what tank is being shown in the template using the **tankNumber** property.



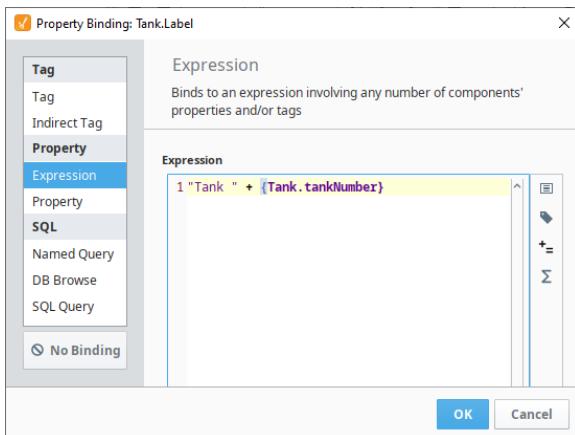
On this page ...

- [Indirection in Templates](#)
- [Indirect Binding](#)
- [UDTs in Templates](#)

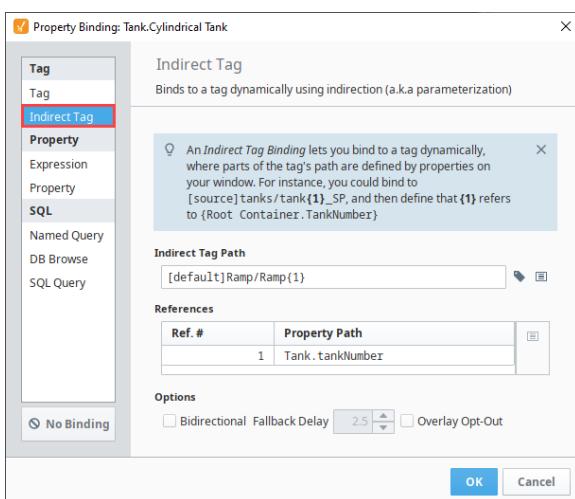


Template - Indirect Binding

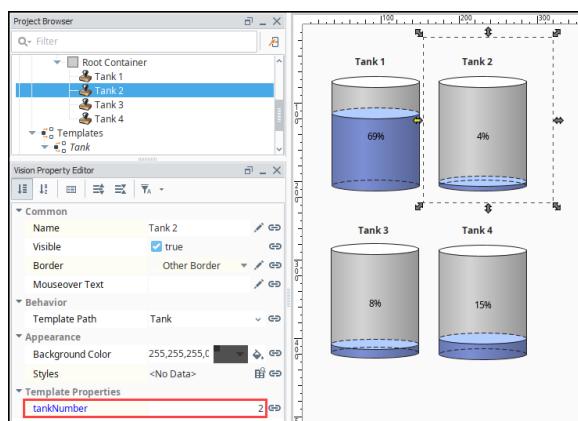
[Watch the Video](#)



- b. Add a **Cylindrical Tank** component that has an Indirect Tag binding to the Volume Tag using the tankNumber property for indirection.



3. The only thing left to do is drag a few instances of the Template onto a window. Then enter a different value into our **tankNumber** for each template and all of the templates will now display different Tag values from one another. In addition, it is very easy to bind the tankNumber property to something, allowing you to easily change what tank the template is displaying in the client during runtime.



UDTs in Templates

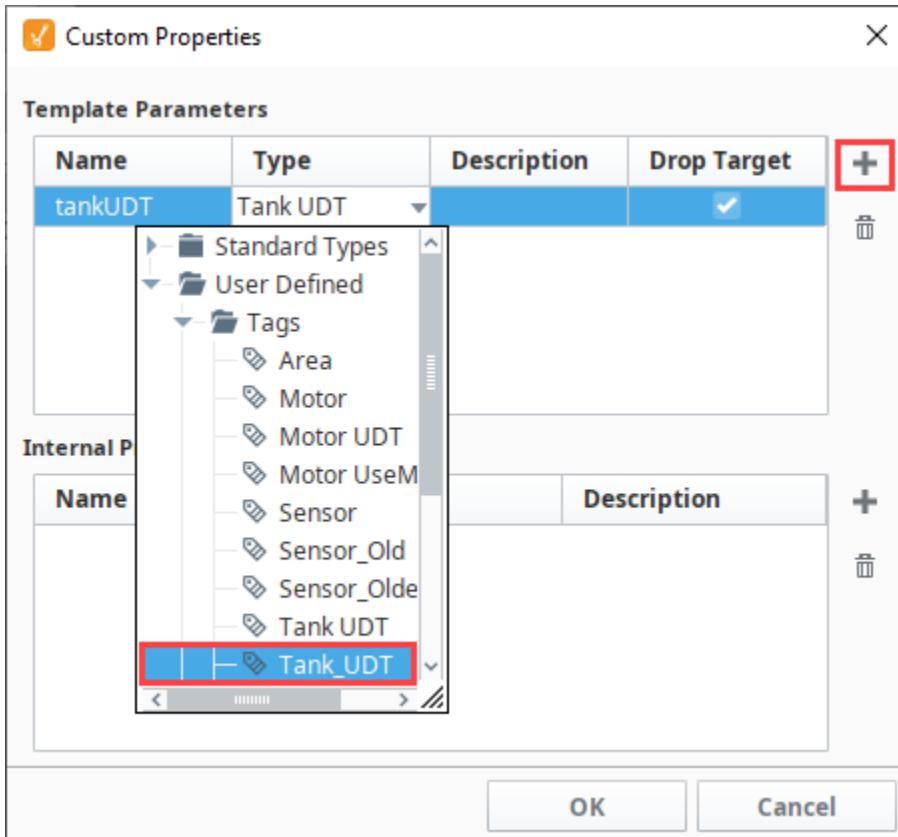
When adding custom parameters to a template definition, the type of the property can be set to a [User Defined Type \(UDT\)](#). This creates a complex property with several child properties, where each sub property represents a tag in the UDT. These properties can be bound to within the template. An instance of a UDT can be passed into a Template Instance just like any other Tag would.

When using a UDT as a parameter in a template, be mindful that the [Template Canvas](#) and [Template Repeaters](#) components can not make use of UDT parameter types on embedded templates: when using either component, [Indirect Binding](#) on standard data types is the preferred approach.

Template – UDT Parameter

[Watch the Video](#)

1. [Create a new template](#), and call it **Tank2**.
2. Right-click on the background of the template, and select **Customizers > Custom Property**.
3. Add a Template Parameter called **tankUDT**, but this time, we want the type to be a UDT (named "Tank_UDT") that we have already created.



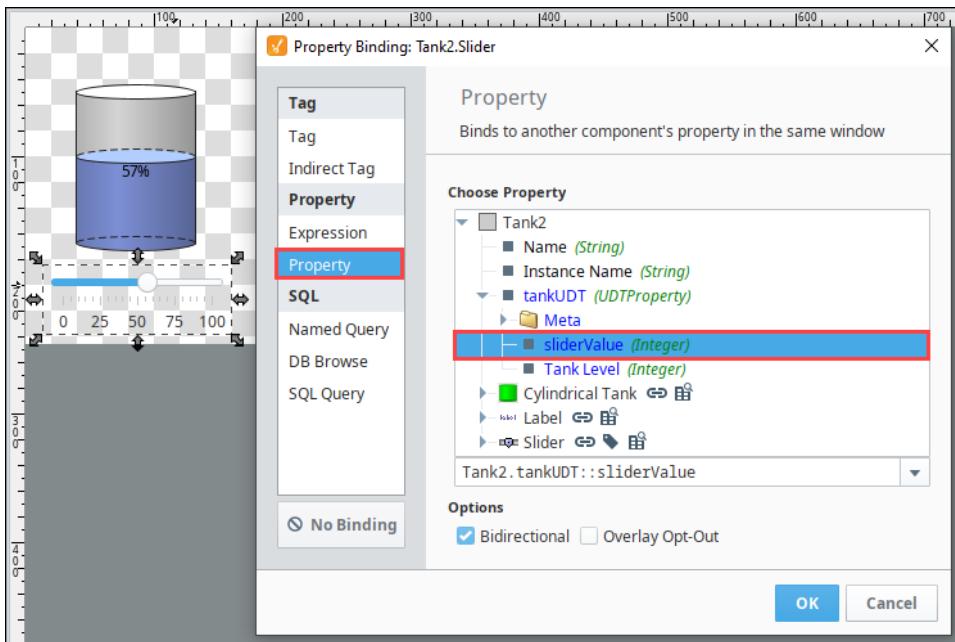
4. Next, add some components to our template: Label, Cylindrical Tank, and Slider. Be sure to utilize the Template Parameter **tankUDT**.

- a. Add a **Label** component at the top of the template to show the name of the tank. Using a property binding, bind the label to the Tag **Name** property of the UDT property **tankUDT** in the Meta folder. This will pull the name of the UDT instance in as the title of the template. Don't worry if it makes your label blank, as there is nothing in that custom property yet.

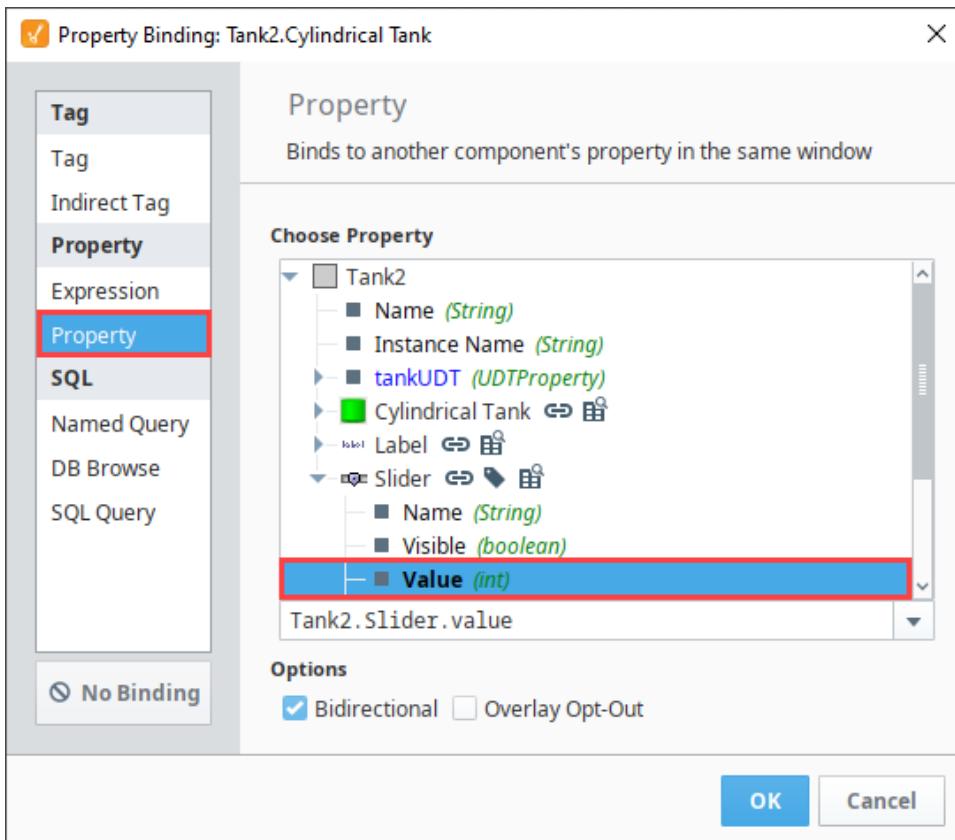
Expression - Indirect Label

```
{Tank.tankUDT::Meta.TagName}
```

- b. Add a **Slider** component and bind it to the **sliderValue** property within the **tankUDT** property.

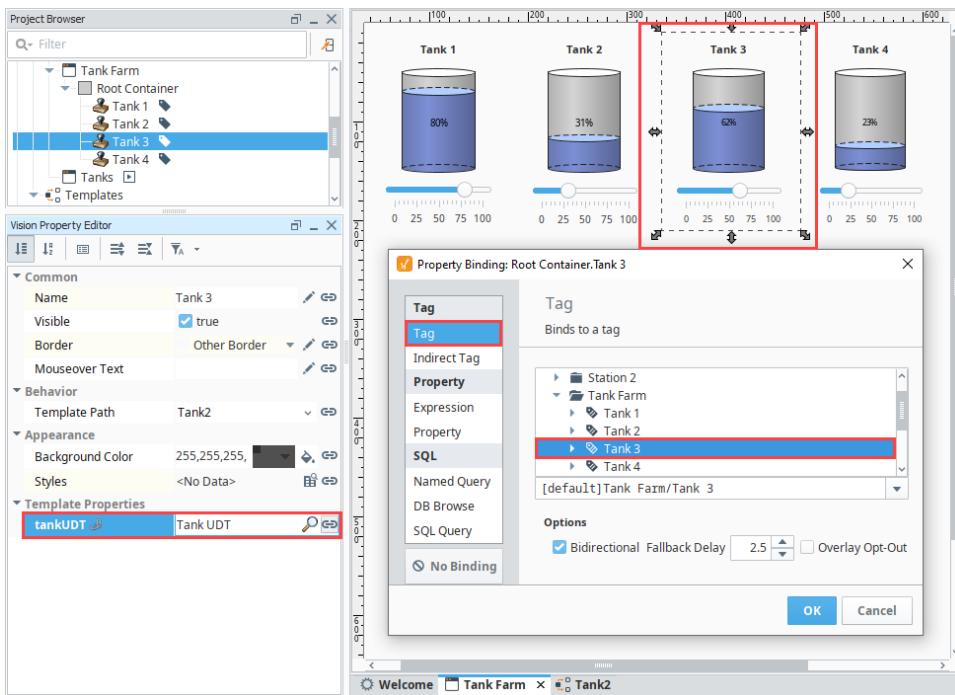


- c. Add a **Cylindrical Tank** component and bind it to the Slider component's **Value** property.



5. Next, add an instance of the **Tank2** template onto a window. In this example, we added three instances.
 6. Bind the **tankUDT** property to each of the UDT instances. In the image below, we selected the **Tank 3 instance** and bound the **tankUDT** template property to the **Tank 3 Tag**. The Tag bindings to the UDTs can even be made indirect to allow the passed UDT to be changed at runtime.

You can see how easy it is to create these tanks using UDTs.



Related Topics ...

- [Using the Template Repeater](#)
- [Using the Template Canvas](#)

Using the Template Repeater

The [Template Repeater](#) component lets you easily create multiple instances of a master template for display on the HMI. Each instance shown in the Template Repeater has the same look, feel, and functionality of the master template. The instances can be arranged vertically, horizontally, or in a "flow" layout, which can either be top-to-bottom or left-to-right. If there are too many instances to fit, a scrollbar is added to the display. The Template Repeater also gives you the ability to pass parameters to each instance of the template, making the templates dynamic.

The Template Repeater can create multiple template instances in two different ways, which will also affect how it passes parameters to those instances. The first is Count mode, which will allow you to specify how many times the Template Repeater will repeat a template. It will then use the index number of each template as a parameter that it will pass into the template. The second method is Dataset mode, where each row of a dataset will be a new template instance and each column will be a parameter that will be passed into the template. This is useful if you have multiple parameters that need to be passed into a template. See the [Vision - Template Repeater](#) page for a more detailed description of this property and how it works. We will go over both methods of using the repeater below.

Creating a Template

Before we use the Template Repeater, we need to first have a template that we need multiple copies of. We used Ramp Tags in the Generic Simulator device built into Ignition. The template will have a label at the top with the ramp name, and it will display the ramp number value. To do this, our template needs to have two parameters. One called RampNumber, which will be used to display the ramp number and also used for in an indirect Tag binding. The second parameter called RampName, which is passed in a string name that is the name of the ramp that I made up. The steps for making this template are listed below, or you can skip ahead to the next section if you are familiar enough with making templates.

On this page ...

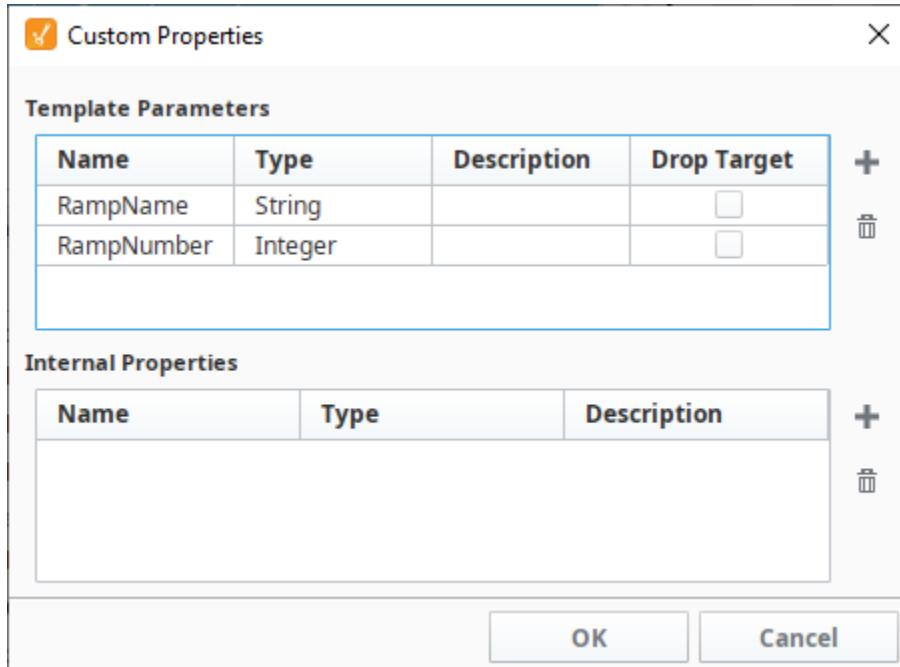
- [Creating a Template](#)
- [Using the Template Repeater with Count Mode](#)
- [Using the Template Repeater with Dataset Mode](#)



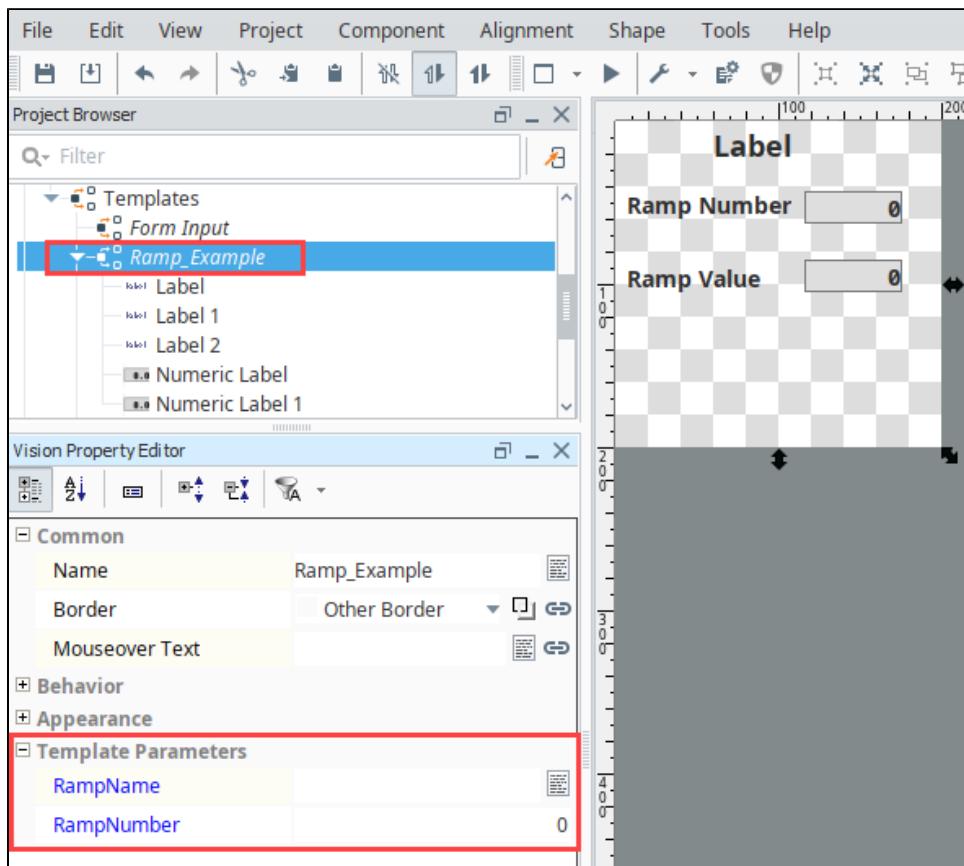
Template Repeater

[Watch the Video](#)

1. In the Templates section of our Project Browser, create a new template. I named mine, Ramp_Example. See the [Templates](#) section for a more detailed description on what templates are and how to use them.
2. We are going to want to pass in a value to the template, so we need to create a Template Parameter.
 - a. Right click on the Root Container, select **Customizers > Custom Properties** and add a Template Parameter called **RampNumber**, and make it an **Integer** type.
 - b. Add a second Template Parameter called **RampName**, and make it a **String** type.

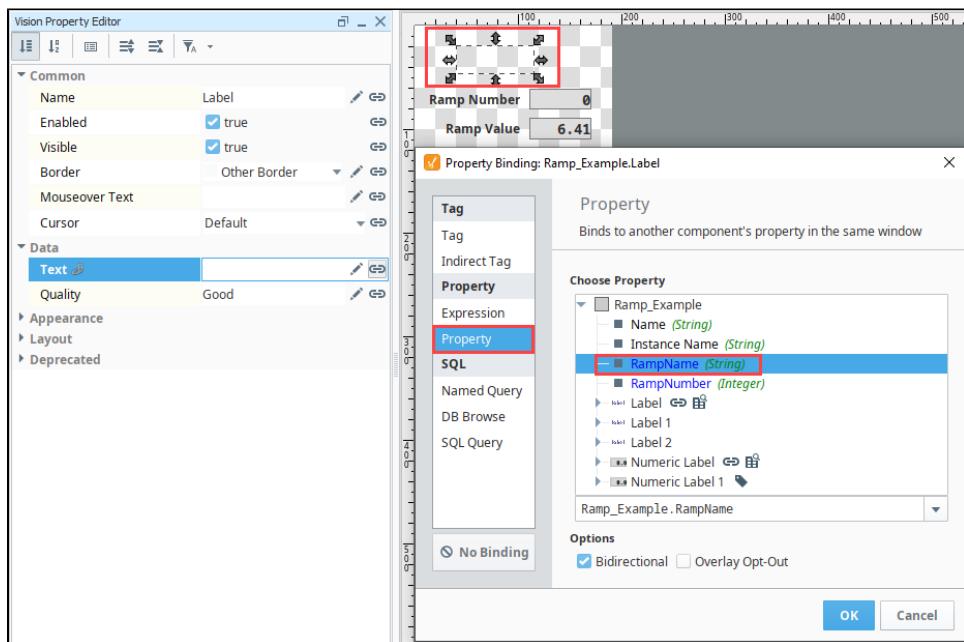


3. Next, add some components to our Template.
 - a. Add two Numeric Labels and one Label. I also added two extra label components, and manually typed in some static text into their **Text** property. In one label, we entered "Ramp Number" and in the other, "Ramp Value".
 - b. Resize the components so that they are easy enough to read. Place the blank Label component at the top of the Template, and place each Label that we wrote text into next to a Numeric Label.

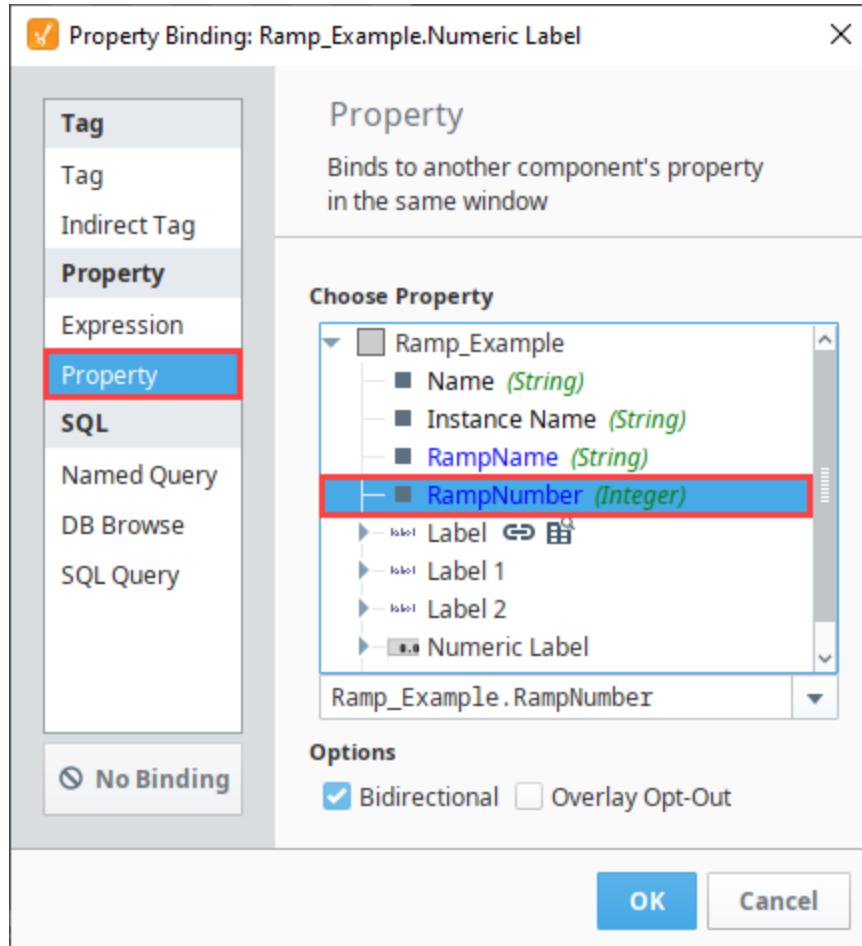


4. Next, setup some bindings on our Template.

- Using a [property binding](#), bind the **Text** property of the blank Label component at the top of the Template to the **RampName** Template Parameter.

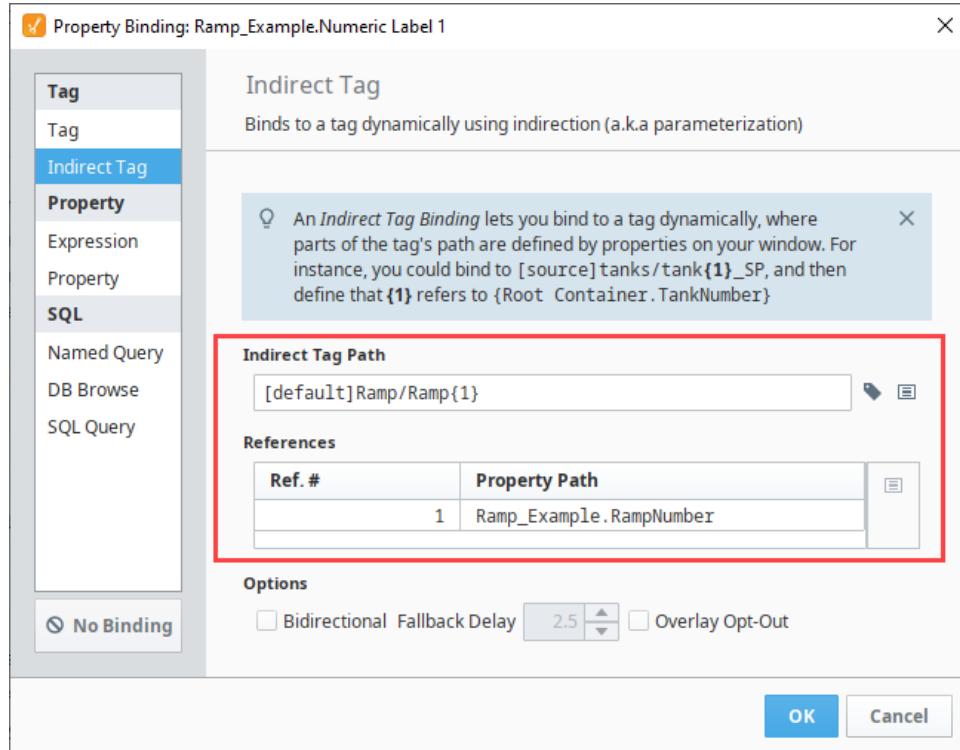


- Using a property binding, bind the **Value** property of the Numeric Label next to the "Ramp Number" label to the **RampNumber** Template Parameter.

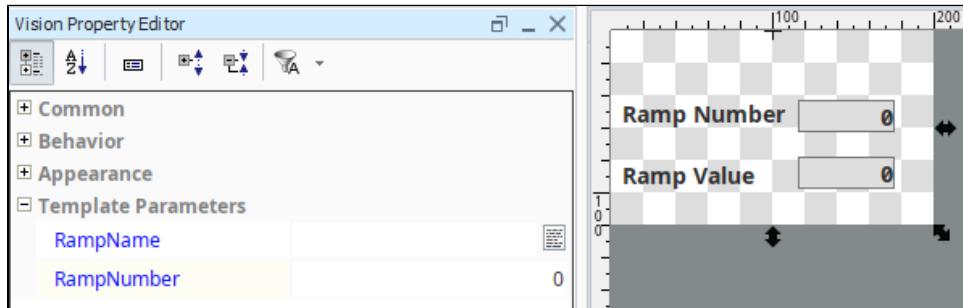


- c. Using an [indirect Tag binding](#), bind the **Value** property of the Numeric Label next to the "Ramp Value" label. We bound it to a Ramp Tag that is in our Tag Browser, using the **RampNumber** Template Parameter as the indirect reference for the number in the name of the Tag. It should look similar to the image below.

Note: Remember to remove the number portion of the tag name and replace it with {1}. For example, [default]Ramp/Ramp0 becomes [default]Ramp/Ramp{1}.



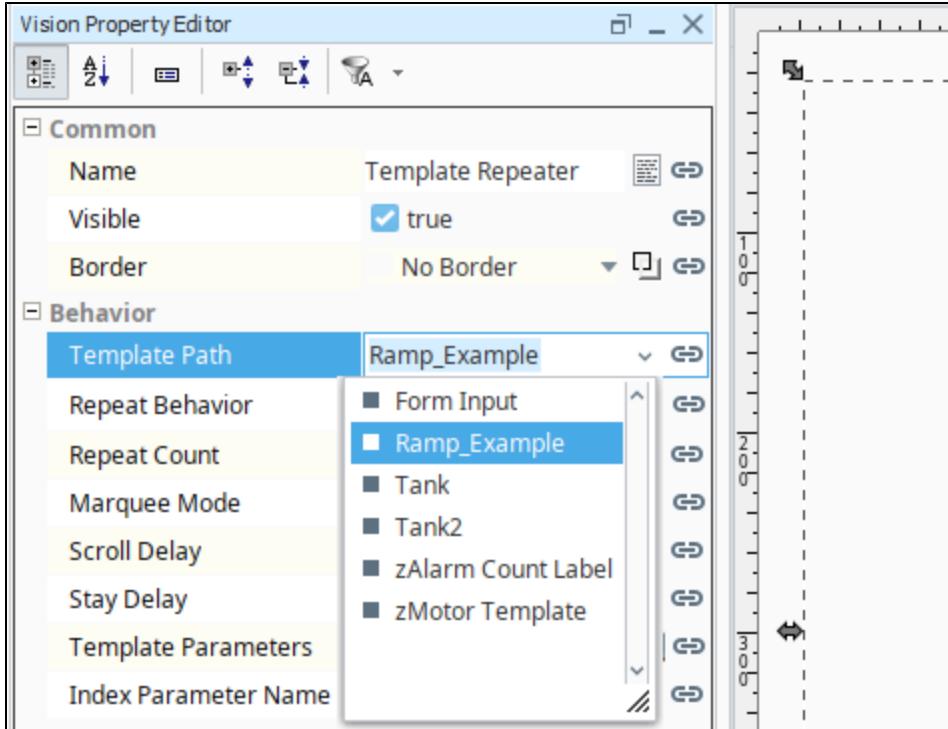
5. The template is now finished and ready to use. See the image below for an idea of what mine looks like. Yours should look similar. Notice the two Template Parameters, as well as the placement of the components. The label at the top that can't be seen because there is no text in it.



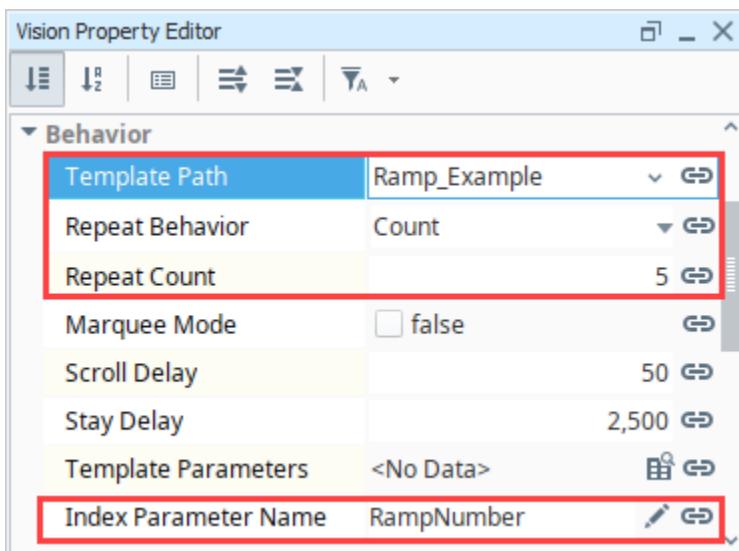
Using the Template Repeater with Count Mode

Now let's take a look at how to use the Template we just made in a Template Repeater.

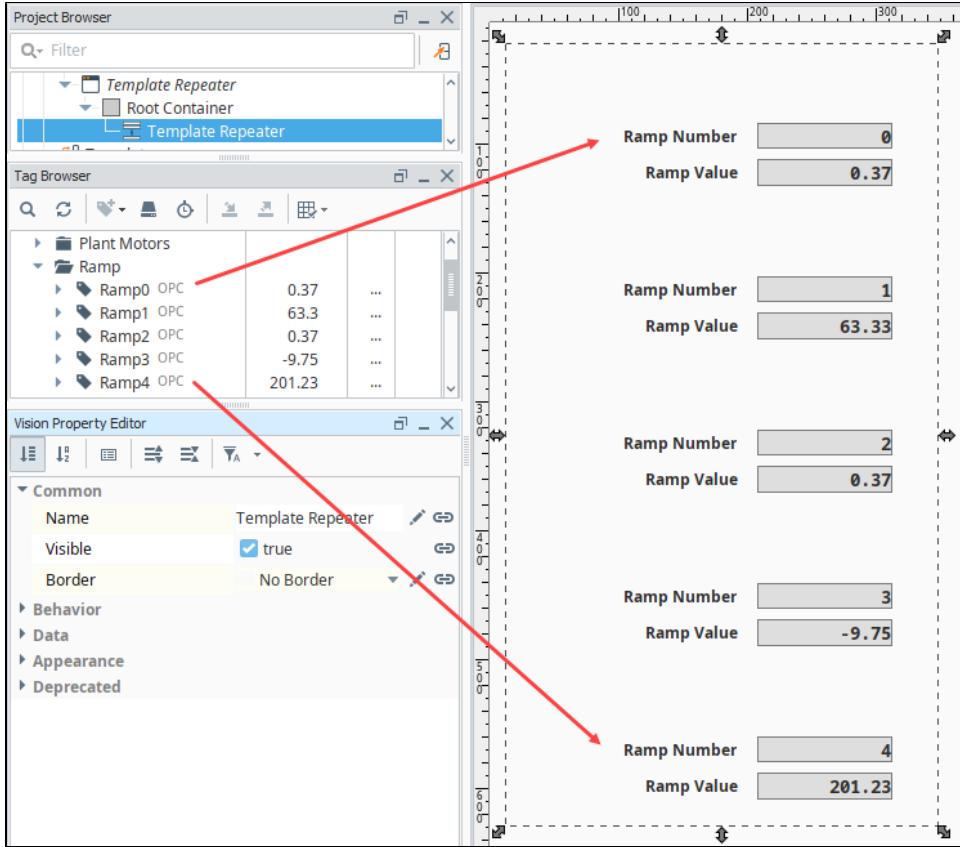
1. On a new window, drag a **Template Repeater** component. Resize it so that it is taller than it is long.
2. With the Template Repeater selected, find the **Template Path** property, and find the name of the template that we just created in the drop down menu.



3. With the Template Repeater still selected, find the **Index Parameter Name** property. Enter in '**RampNumber**', which was the name of one of our Template Parameters from our Template.
4. Ensure that the **Repeat Behavior** property of the Template Repeater is set to Count.
5. In the **Repeat Count** property of the Template Repeater, set the value to '5'.



6. You should see your Template repeated five times within the Template Repeater. Our Ramp Number value in the Template corresponds to the index value of the Template, 0 through 4 because it is 0 based. You will see that the value in our Ramp Value component corresponds to the value of one of our Ramp Tags.

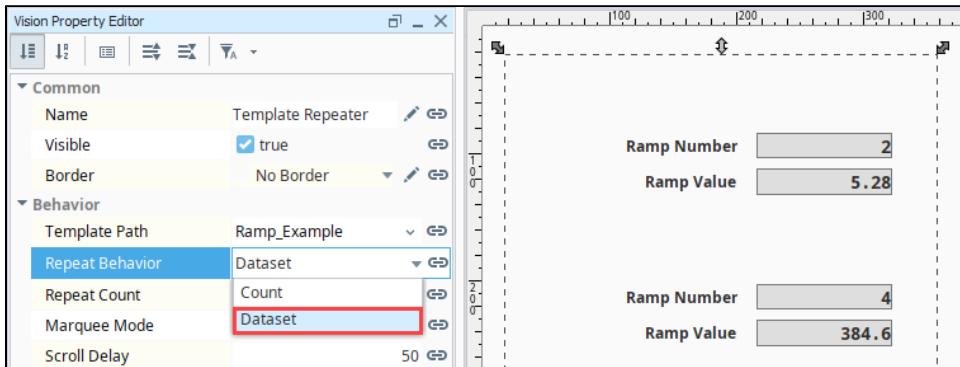


Your Repeater is complete. You can modify the Repeat Count to change how many times the Template gets repeated.

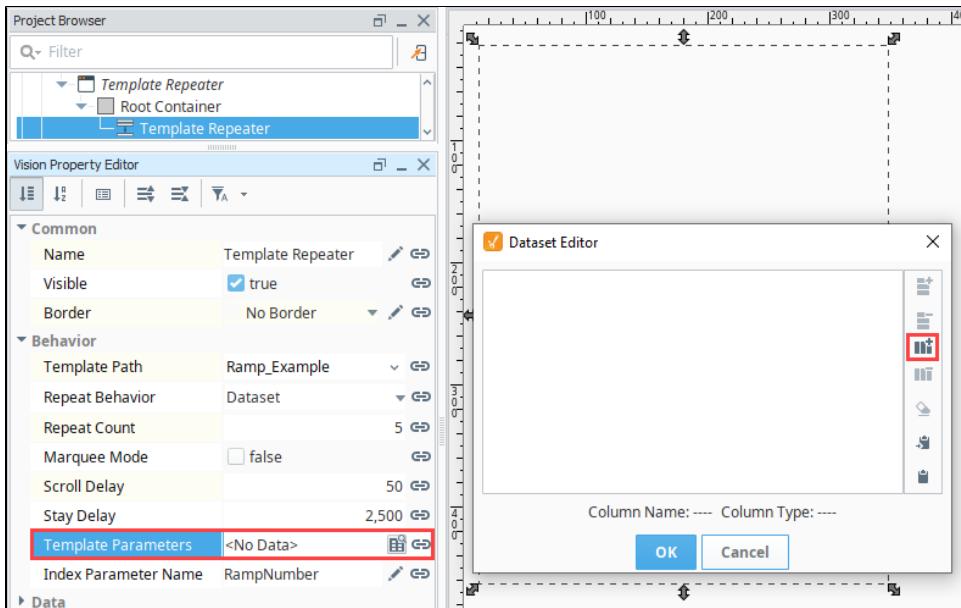
Using the Template Repeater with Dataset Mode

In our previous example, although we were able to successfully pass in the index parameter into our Template, we were not passing anything into the **RampName** Template Parameter and because of this, the blank label at the top of our Template remained blank. This is because **Count** mode only allows a single index parameter to be passed in. However, since we want to pass in two parameters, we can instead use **Dataset** mode. In addition, because we are going to be defining a dataset to pass in parameters, we don't have to use a 0 indexed parameter, and can instead use whatever values we want. We will use the same Template and Template Repeater from before.

1. On the Template Repeater, set the Repeat Behavior to **Dataset**. All of the Templates that were previously in will temporarily disappear.



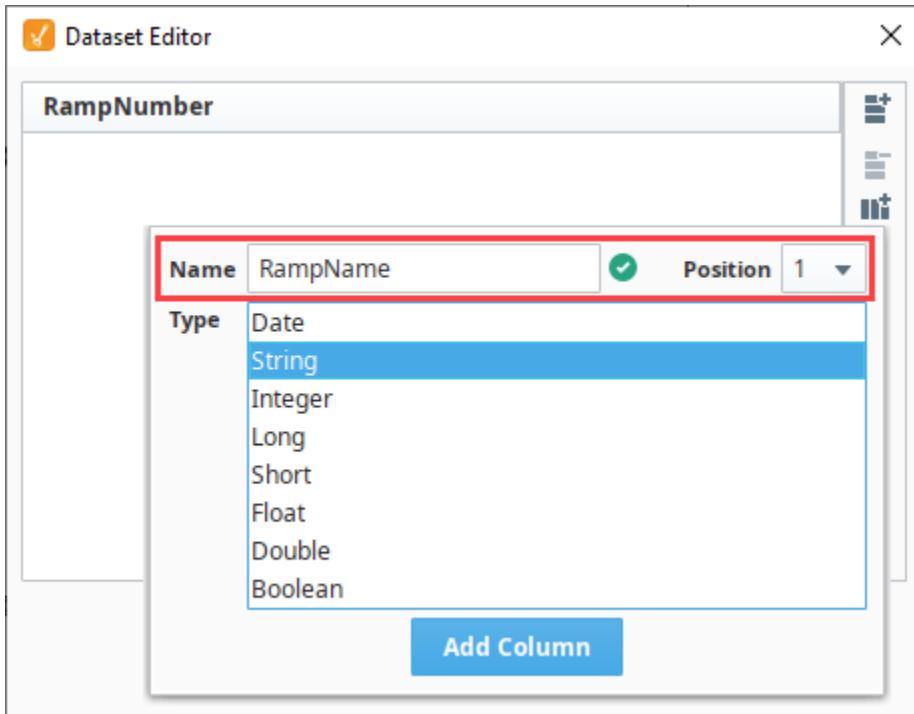
2. Click the **Dataset Viewer** icon on the **Template Parameters** property of the Template Repeater. You should see a new popup that allows you to build a dataset.



3. First, we'll add two columns to our dataset.

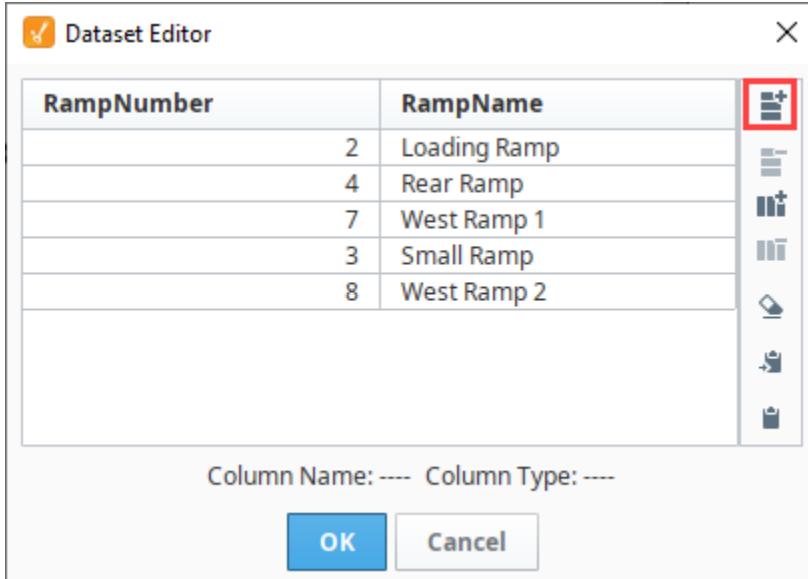
- Click the Add Column icon. Set the Name as 'RampNumber' in Position '0', and enter 'Integer' as the Type. Click Add Column.
- Click the Add Column icon. Set the Name as 'RampName' in Position '1', and enter 'String' as the Type. Click Add Column.

Note: Make sure the name of each column exactly matches the names of the Template Parameters (RampNumber and RampName). They are case sensitive.

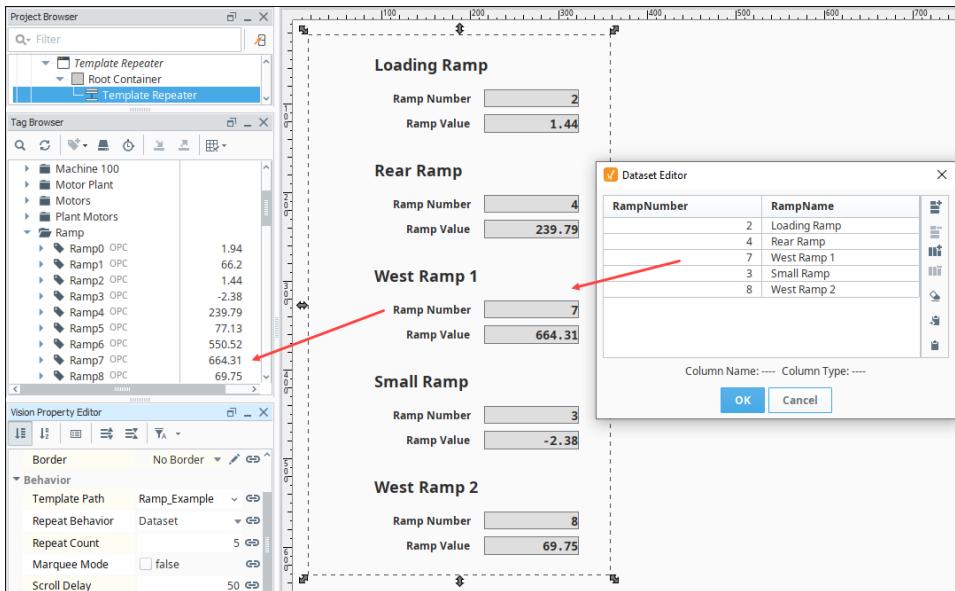


- Next let's add some rows by clicking the Add Row icon. Click it five times to add five rows.
- Add in some values for the RampNumber column. Since we are defining all of the values, we don't have to start with a RampNumber of '0'. In fact, we don't even need to have them be in sequence! We do need to make sure that the numbers each correspond to one of our Ramps, so enter in a value '0' through '9' for the RampNumber column for each row. In this example, we used 2, 5, 7, 3, 8 in that order.

6. Next, add some values for the **RampName**. These can be whatever you want, since we are going to be giving unique names to each of our Ramps. When finished, the dataset should look something like the image below. Click the **OK** button to save your dataset.



7. Notice how we now have our five templates back, and they are using the parameters that we are passing in from the dataset we just created. The Templates are receiving the parameters in the order that you made them in the dataset. We can also see that each Template instance corresponds to a specific ramp.



Dataset mode is great when you need to pass in multiple parameters, or if you have a single parameter that is not zero based. In addition to manually specifying values like we did in this example, you can instead setup a binding on the **Template Parameters** property to something like a database table. This allows you to easily modify the templates that are being displayed in the window simply by modifying the database table.

Related Topics ...

- [Using the Template Canvas](#)

Using the Template Canvas

The Template Canvas component works much like the Template Repeater in that it can easily create multiple copies of a master template. What makes the Template Canvas unique is that it can display instances of multiple master templates, and set their layout in any way you want. The Template Canvas has a customizer that can help put the templates together within it, but the customizer is just driven by a Templates dataset property on the Template Canvas. The Template Canvas can be made dynamic by setting up a binding on the Templates property, such as a query that pulls in an entirely new dataset of information, or even a cell update binding, which updates individual cells of the dataset. With that, you can load new templates into the canvas at runtime, or even move the templates around.

The Template Canvas has two layout systems:

- **Absolute Positioning**, where each template instance has an absolute position within the Template Canvas
- **Layout Positioning**, which uses the MiG Layout system to place the instances within a grid like system.

Creating a Template

Before we get started with the Template Canvas, we first need to have a template. Here we are going to make a simple Form Input template, that consists of a Label and a Text Field. The template will need two parameters. A **Label_Text** parameter which is what will get displayed in the label, and a **TextField_Text** which we can use to pass user input outside of the template by making it bidirectional. This template is a great way of quickly making user input forms, by using a template for each piece of info that needs to be collected. The steps for making the template are listed below, or you can skip ahead to using the Template Repeater in the next section.

On this page ...

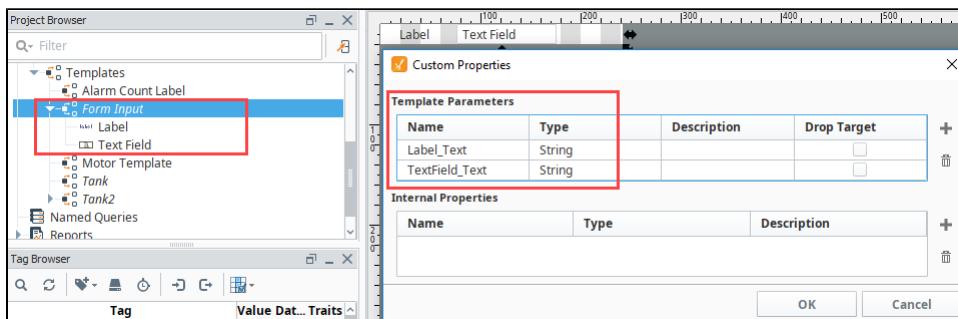
- [Creating a Template](#)
- [Absolute Positioning](#)
- [Layout Positioning \(MiG Layout\)](#)
- [Read User Input](#)



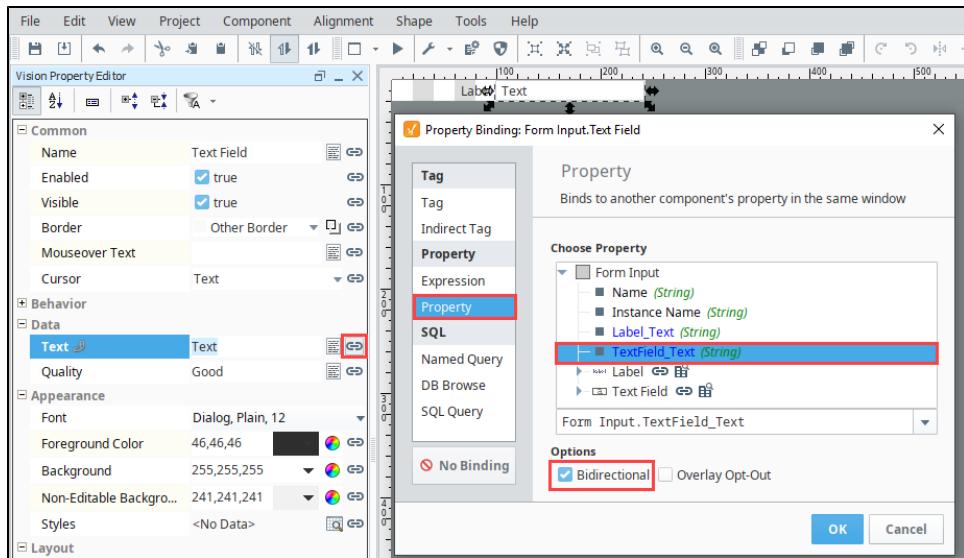
Template Canvas

[Watch the Video](#)

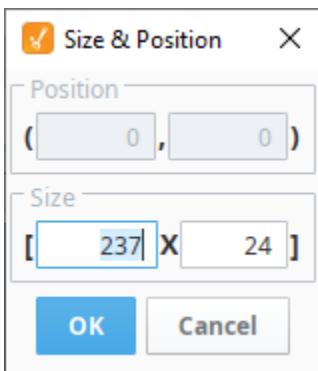
1. In the Templates section of our Project Browser, [create a new template](#) called **Form Input**. We are going to want to pass in values to and from the template, so we need to create Template Parameters.
2. Right click in the Form Template workspace, and select **Customizers > Custom Properties**. Add a Template Parameter called **Label_Text**, and make it a **String** type.
3. Add a second Template Parameter called **TextField_Text**, and make it a **String** type, and click **OK** to save your template parameters.
4. Next, drag a **Text Field** component and a **Label** component onto the Template.
5. On the **Label** component, set the **Horizontal Alignment** property to **Trailing**.



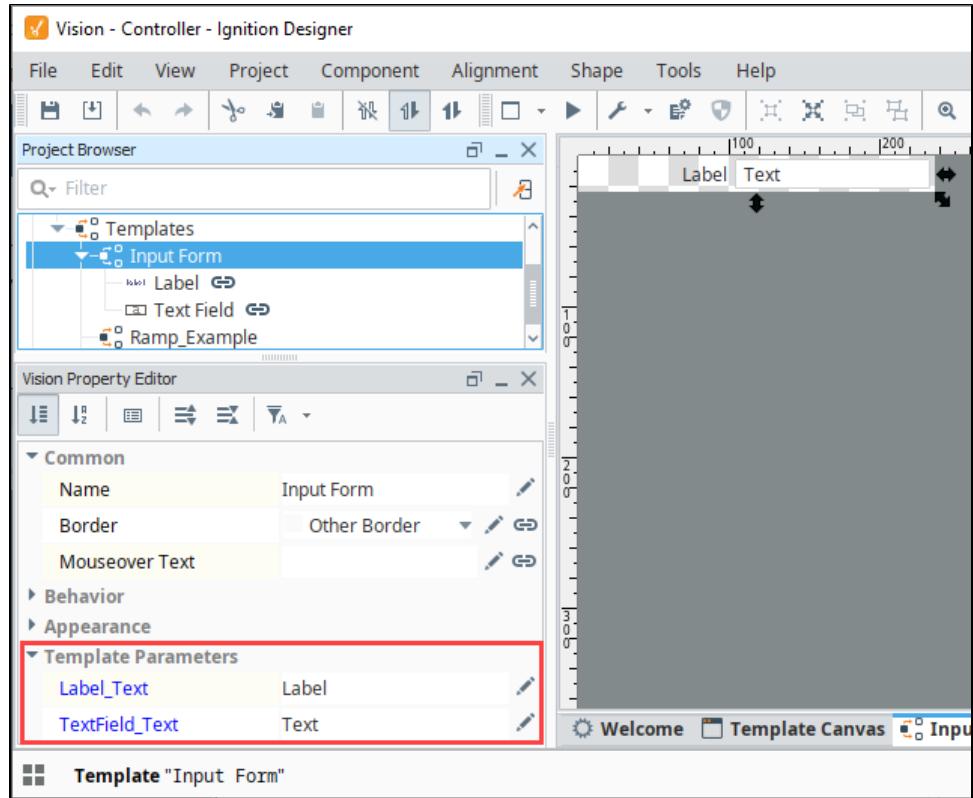
6. Finally, we want to setup some bindings on our Template.
 - a. Using a [property binding](#), bind the **Text** property of the **Label** component to the **Label_Text** Template Parameter.
 - b. Using a property binding, bind the **Text** property of the **Text Field** component to the **TextField_Text** Template Parameter, and make it **Bidirectional**.



- Set the Size and Position for the components by right clicking on each component inside to 20 pixels. Next, set the height for the Form Input template as shown in the image below. Make sure to drag your two components to the top of the template before adjusting the size.



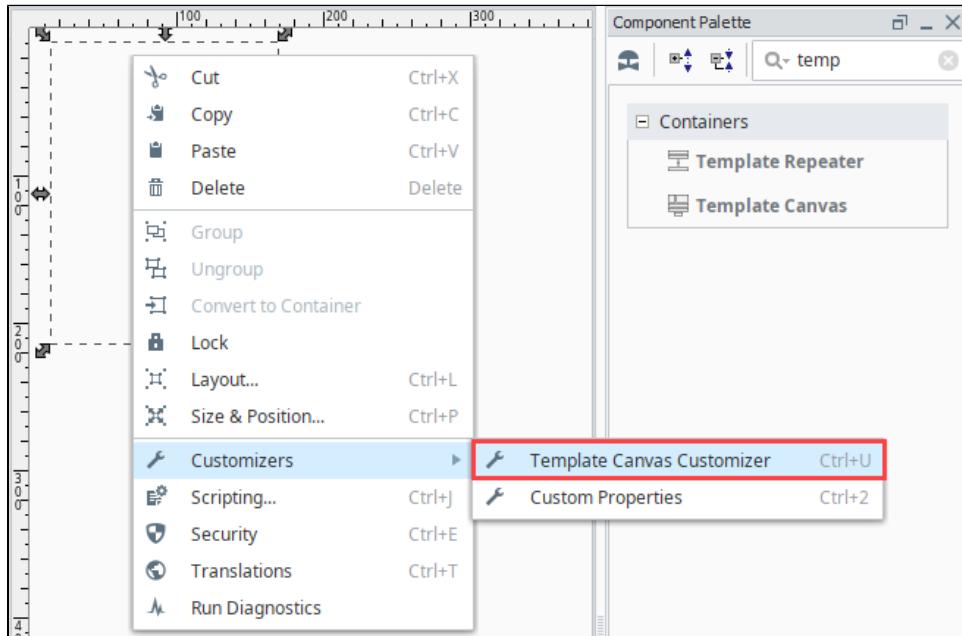
- The template is now finished and ready to use. See the image below. Yours should look similar. Take notice of the two Template Parameters, as well as the placement of the components.
- Once you have the properties added and components bound, type a value into each Template Parameter so we can easily tell that our bindings are working.
 - Label_Text: Label
 - TextField_Text: Text



Absolute Positioning

First, we can set up our Template Canvas using Absolute Positioning, which is a little simpler to understand as each template has a width and height as well as an x and y position.

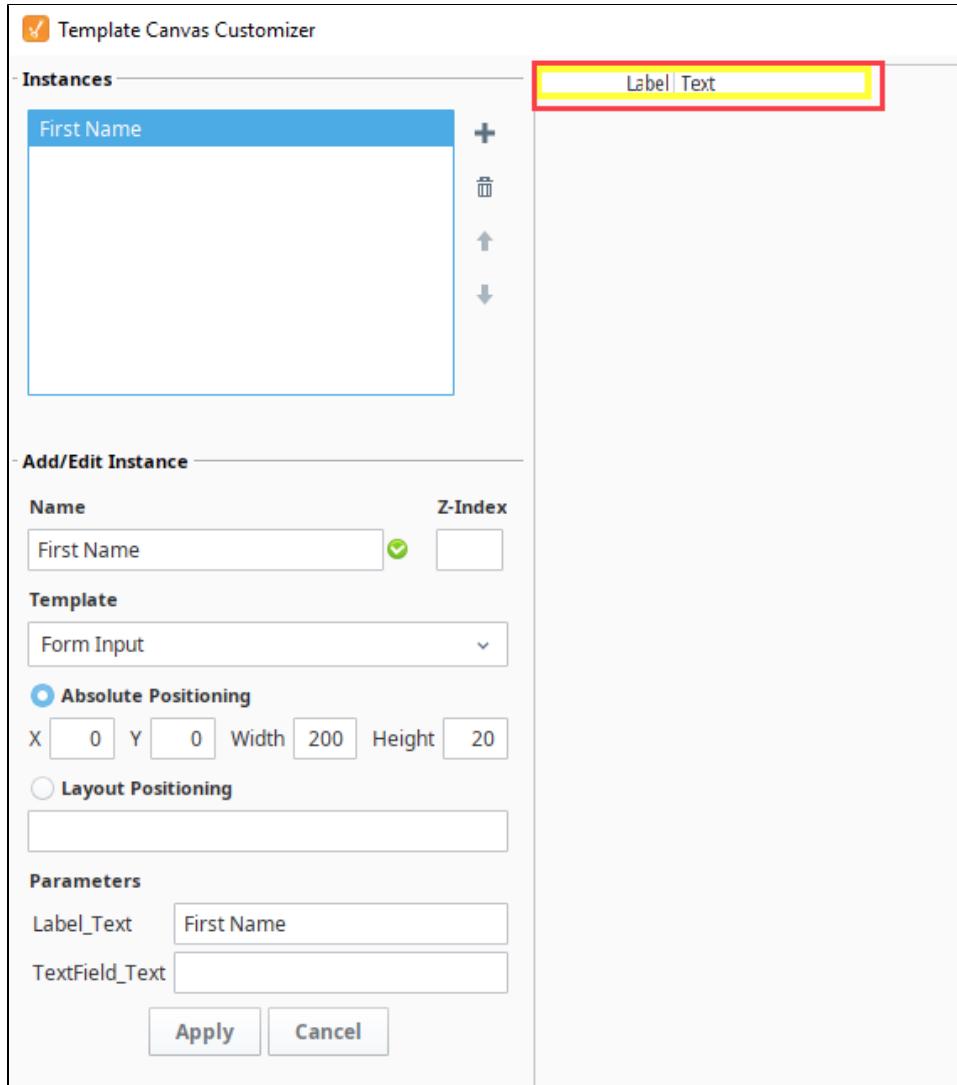
1. Drag a **Template Canvas** component to a window and open the **Template Canvas Customizer**. The customizer provides an easy interface to setting up our templates.



2. Enter in the following values for the first instance:
 - a. **Name:** First Name
 - b. **Template:** Form Input

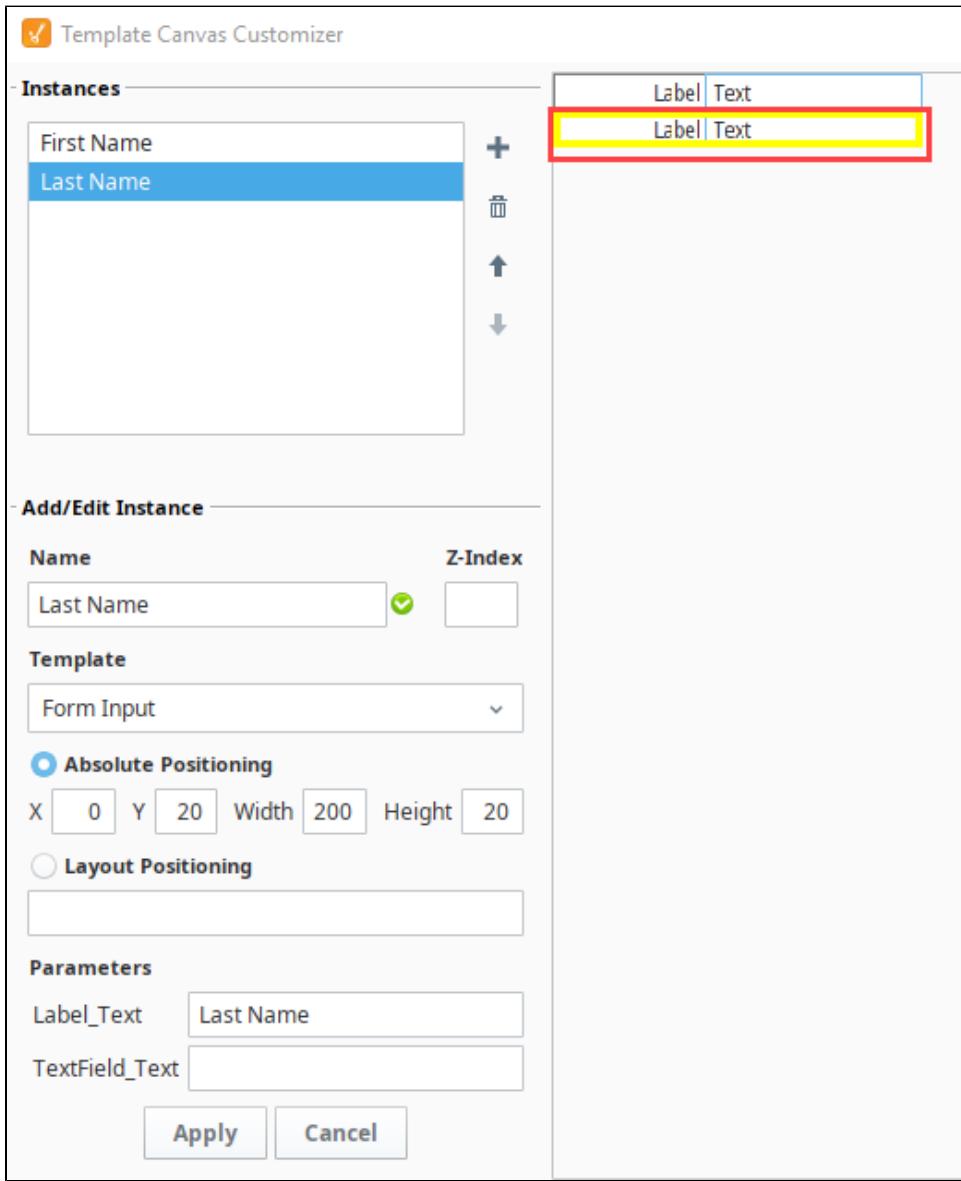
- c. **Absolute Positioning:** 0, 0, 200, 20
 - d. **Parameters:** Label_Text = First Name (leave TextField_Text blank)
3. Click the **Add** button to add the instance. The instance will then be visible in the preview section of the window. Values for the components are still using the default values for the Label_Text and TextField_Text. This is intentional. The new values will appear once you hit **OK** and close the canvas customizer.

Labels have Absolute Positioning properties: X, Y, width, and height labels. A Z-index field indicates where a given template instance should appear in the z-order within the template canvas (top or bottom).

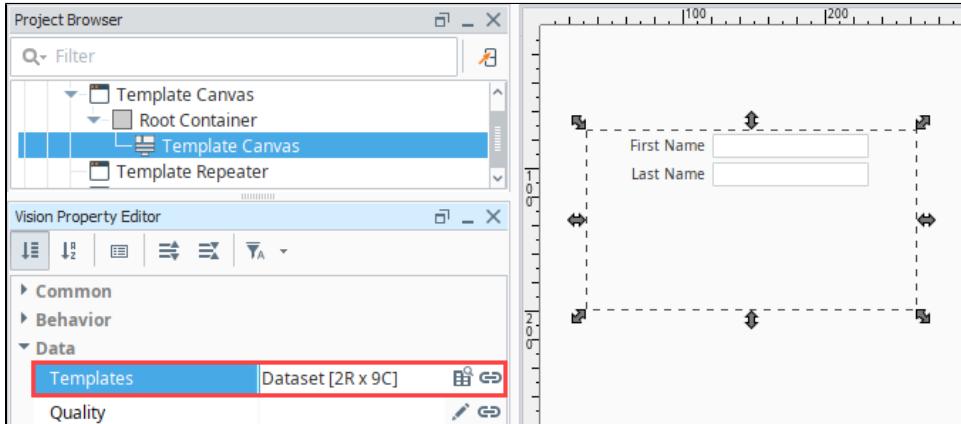


Note: Take notice of the yellow outline around the instance in the image, and how First Name is highlighted at the top of the customizer. This means that the instance is selected, and the customizer is in edit mode. This allows you to make changes to the selected instance. To exit edit mode and add a new instance, click on the **Cancel** button in the lower **left** of the window. Clicking on the **Cancel** button in the lower **right** will cancel out of the customizer.

4. Let's add another instance. Click on the **Add** icon in the **Instance area** to clear the Add/Edit Instance fields from the prior entry, and enter the following:
 - a. **Name:** Last Name
 - b. **Template:** Form Input
 - c. **Absolute Positioning:** 0, 20, 200, 20
 - d. **Parameters:** Label_Text = Last Name (leave TextField_Text blank)
5. Once entered, click the **Add** icon again. You'll see that you have two instances visible in the preview section of the window. Once both instances are configured, click the **OK** button.

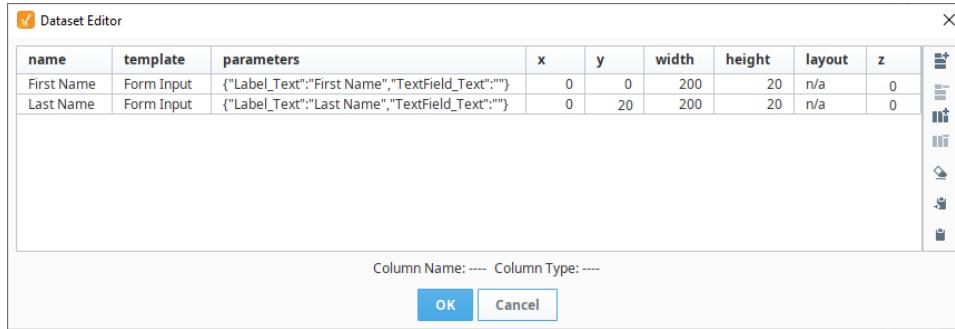


6. Now you will see both instances appear in the Template Canvas.



7. If a new instance needs to be added, it can be added through the Template Canvas Customizer. However, the Template Canvas also has a 'Templates' property. This property stores all of the data that was entered into the customizer into a dataset, so new instances can be configured directly on the Templates property. View the dataset by clicking the Dataset Viewer button next to the Templates property. Furthermore, template instance configurations could be stored in a database table, and the Template Canvas could fetch the data with a SQL

Query binding on the Templates property.



Layout Positioning (MiG Layout)

Instead of having to manually enter a size and position for each instance, we can make use of Layout Positioning to have the Template Canvas determine the best position for each instance, while also making suggestions as to where each instance is placed in relation to another. The layout positioning uses a grid-methodology to instance placement. Each instance, unless otherwise specified, is considered a single "cell" in the grid.

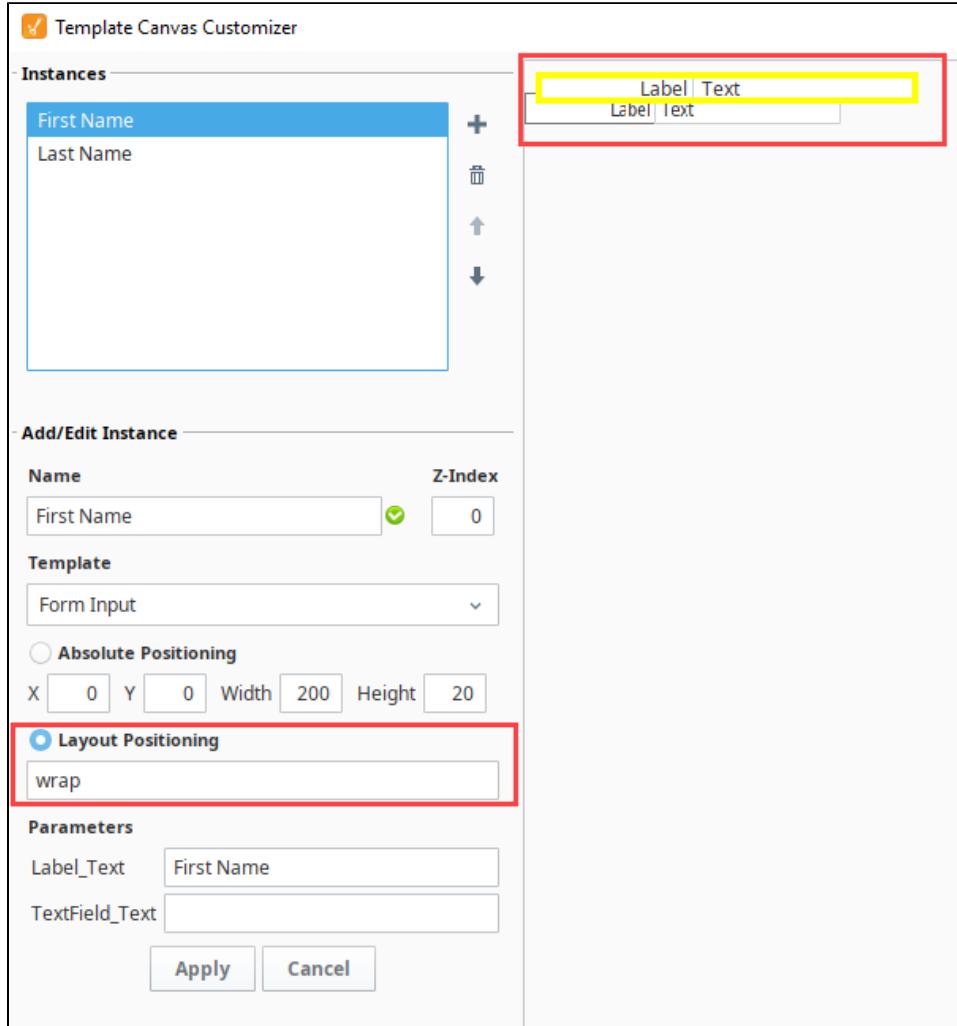
Caution: Don't Mix Absolute and Layout Positioning

We do not recommend using both Absolute and Layout Positioning for instances on the same Template Canvas. Select either Absolute or Layout Positioning for your instances. Layout Positioning will determine the best position for each instance in your canvas, where Absolute Positioning allows you to manually specify the width, height, x and y positions for each instance.

Continuing from the example above, the First Name and Last Name instances are using Absolute Positioning. Let's tell the First Name instance to use Layout Positioning and enter the 'wrap' parameter. This means the next cell in the grid should be placed on the next row.

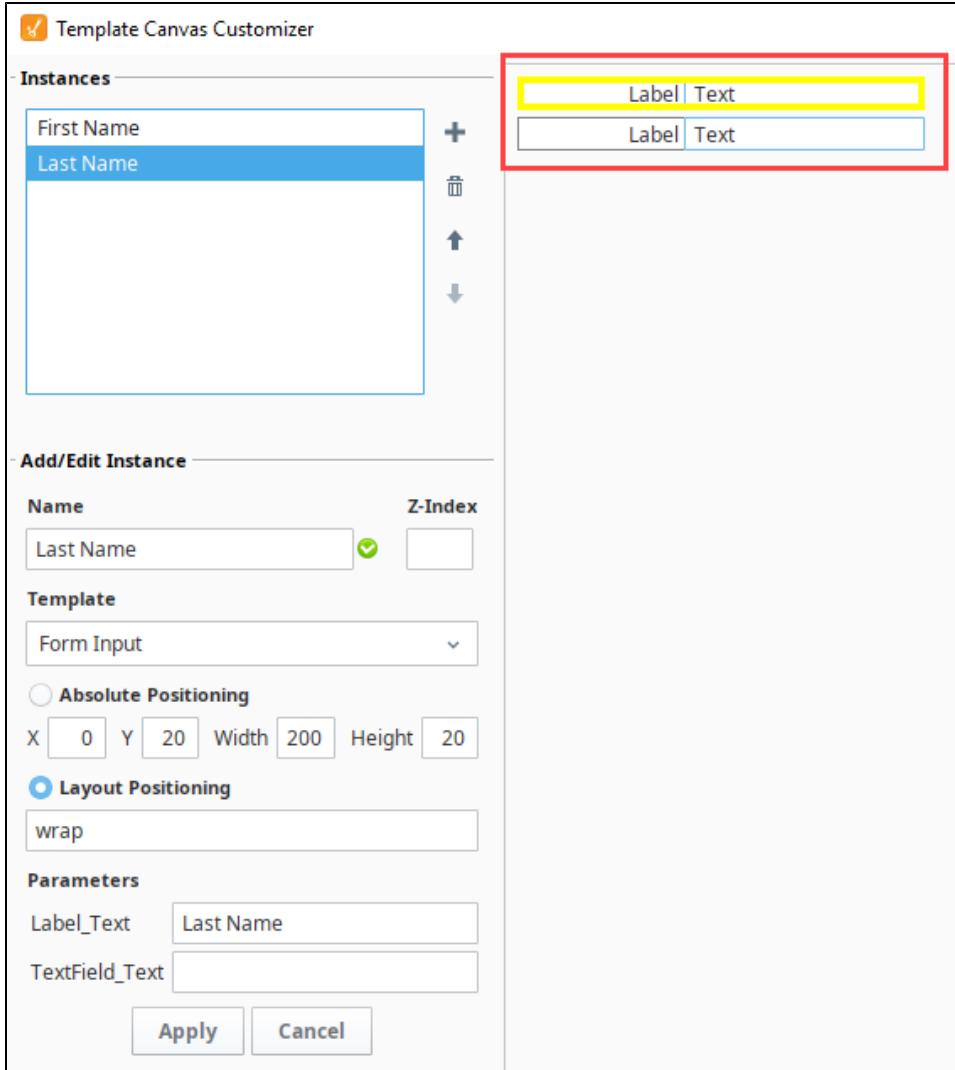
1. With the Template Canvas component selected, open the the **Template Canvas Customizer** again, and make the following modification to the **First Name** instance.
 - a. Click the radio button for the **Layout Positioning** property and enter 'wrap' the field.

2. Click **Apply**, and the First Name instance will appear to overlap with the Last Name instance. This is because the grid only accounts for instances using the Layout Positioning.



3. Next, we can configure the Last Name with Layout Positioning as well. Make the following changes to the Last Name.
a. Click **Layout Positioning** radio button and enter 'wrap' in the field below.

4. Once the changes have been applied, click **OK**. You'll notice in the preview section of the window that both instances are now in line.



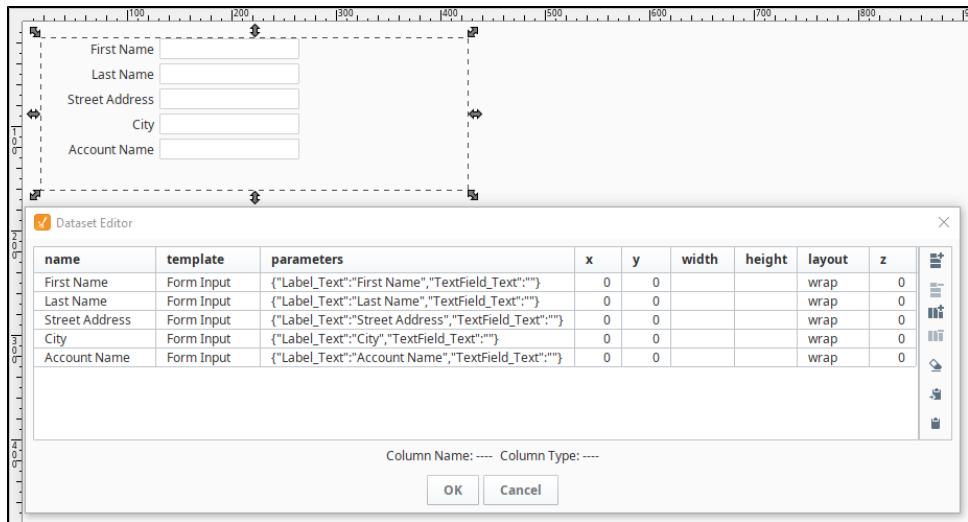
5. Open the **Templates** property by checking the **Dataset View** button. Notice that the x, y, width, and height columns are no longer used, but the **layout** columns for First Name and Last Name now have a value of '**wrap**'.

The screenshot shows the 'Dataset Editor' with a table of template instances:

| name | template | parameters | x | y | width | height | layout | z |
|------------|------------|--|---|---|-------|--------|--------|---|
| First Name | Form Input | {"Label_Text": "First Name", "TextField_Text": ""} | 0 | 0 | | | wrap | 0 |
| Last Name | Form Input | {"Label_Text": "Last Name", "TextField_Text": ""} | 0 | 0 | | | wrap | 0 |

Buttons for 'OK' and 'Cancel' are at the bottom. A sidebar on the right contains icons for various dataset operations.

6. Like the previous example, new rows can be added directly to this dataset. Furthermore, the '**wrap**' value means the next template instance will begin on a new line. Add three new instances for Street Address, City and Account Name. Use either the Template Canvas Customizer or simply add two new rows in the dataset viewer with the values shown in the image below.



Read User Input

The last step is to read the user input. Put the Designer into **Preview Mode** and add some values for each text field component. Once finished, switch the Designer back to **Design Mode**, and add a Button component to the window (not the template canvas)

Add a script to the Button component using the Code Snippet below - **Read User Input Example**. Place the code on the **actionPerformed** event of your component by double clicking on the Button component and opening the **Script Editor** tab.

```
# Reference the template canvas component, and call the getAllTemplates() method.
# This will return a list of every instance in the canvas
templateList = event.source.parent.getComponent('Template Canvas').getAllTemplates()

# Initialize a list. User input from each text field will be stored in this variable
userInput = []

# Iterate through each template instance inside the canvas
for template in templateList:

    # add the user inputted value to the userInput list. The values are originally returned in Unicode.
    # the Python str() function is casting the Unicode values as string values.
    userInput.append(str(template.TextField_Text))

# Show the values in a messageBox. This could be replaced with an INSERT query, or some other action.
# str() is used again to case the list as a string. This only required to work with the messageBox function
# since the function requires a string argument be passed in
system.gui.messageBox(str(userInput))
```

When running the script, each value should appear in the message box. If you're not getting a value in the message box, make sure the Text Field property is bound to the `TextField_Text` template parameter as mentioned in the [Creating a Template](#) section, Step 6b.

This example can easily be expanded to do something more meaningful with the input, like store to a database table.

| | |
|----------------|----------|
| First Name | Tim |
| Last Name | Smith |
| Street Address | Dover St |
| City | Utah |
| Account | 123456 |

Information

X



['Tim', 'Smith', 'Dover St', 'Utah', '123456']

OK

Security in Vision

Security in Vision is managed through one of two authentication strategies, either the [Classic Authentication Strategy](#) or [Identity Providers \(IdP\)](#). Classic Authentication Strategy involves a concept known as a User Source, which is a configuration that contains multiple roles and users. IdPs allow users to authenticate against a trusted third party. Refer to [Security](#) for more information on these authentication strategies.

Once you have an authentication strategy setup, there are additional ways to control security for the following:

- Vision Clients
- Client Login Security
- Component and Window Security

Permissions can be set at the [Project level](#) in the Designer. This restricts actions such as publishing, viewing, saving, deleting, and editing of project resources to users who have sufficient security levels to do so.

Client Permissions

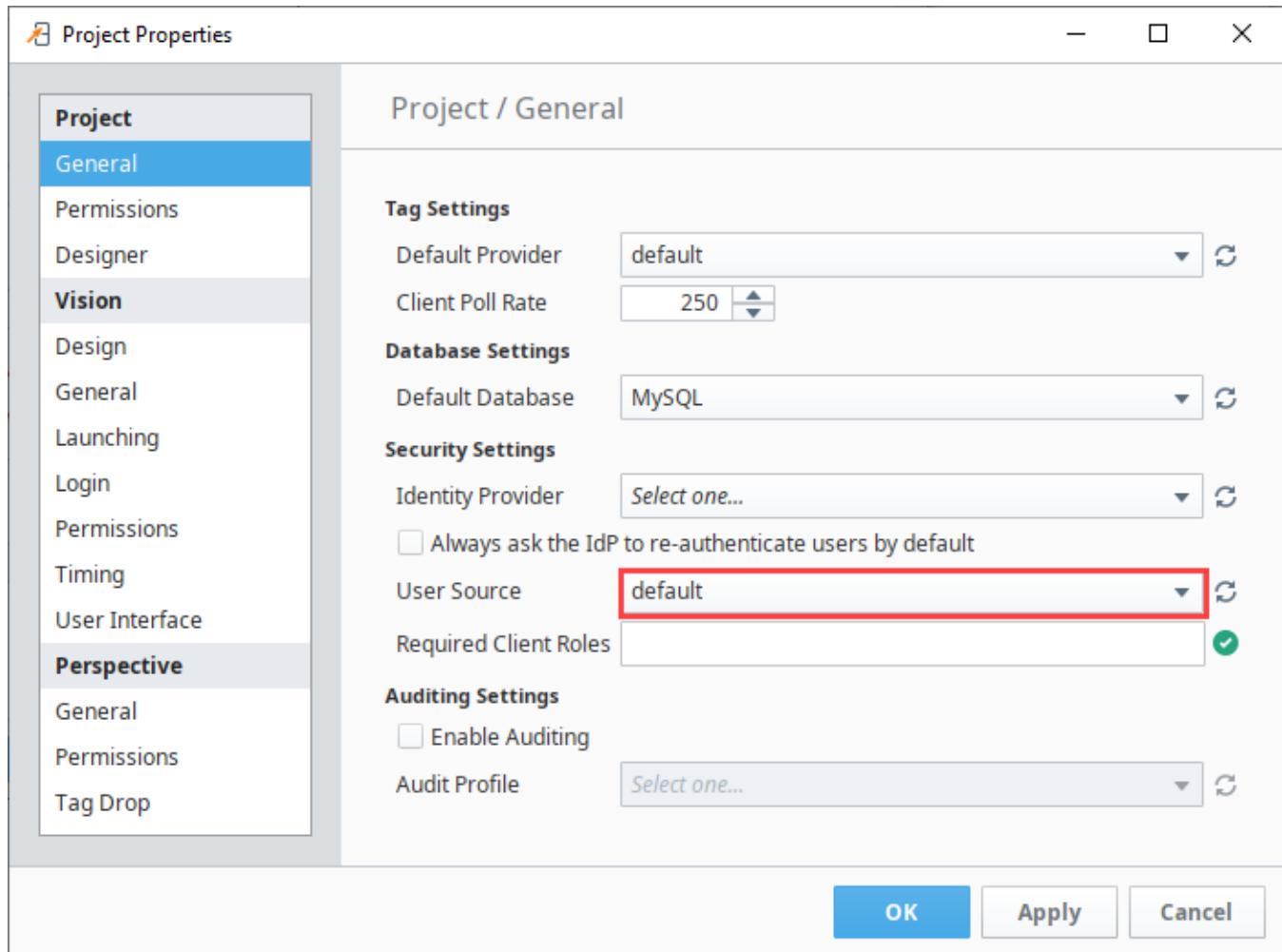
Every project's clients are governed by a set of permissions to control what is allowed to originate from the client. For example: access to construct queries against the database, or the ability to edit Users and Roles in your authentication profile. To maintain a secure system, these are all set to disabled by default, but you can enable them for everyone, for specific users, or even for specific users that are logged into certain zones. See descriptions of these categories and how to change them on the [Project Permissions](#) page.

Client Login Security

On this page ...

- [Client Permissions](#)
- [Client Login Security](#)
- [Role-Driven Client Security](#)
- [Incorporating Scripting into Security](#)

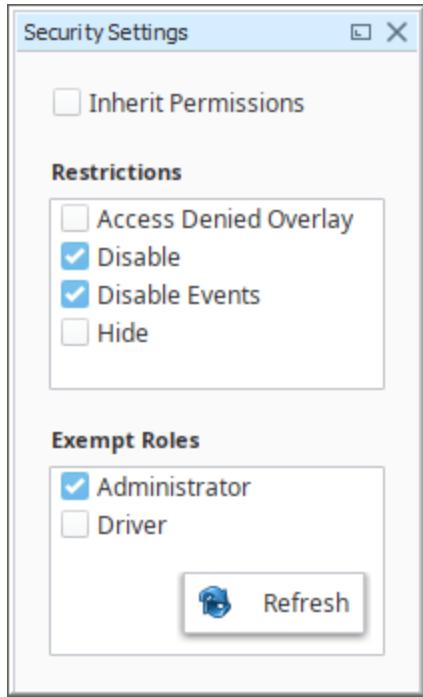
Projects are assigned a User Source to authenticate against. All roles that have been defined in the User Source can be used to prevent users from logging in to the project.



Role-Driven Client Security

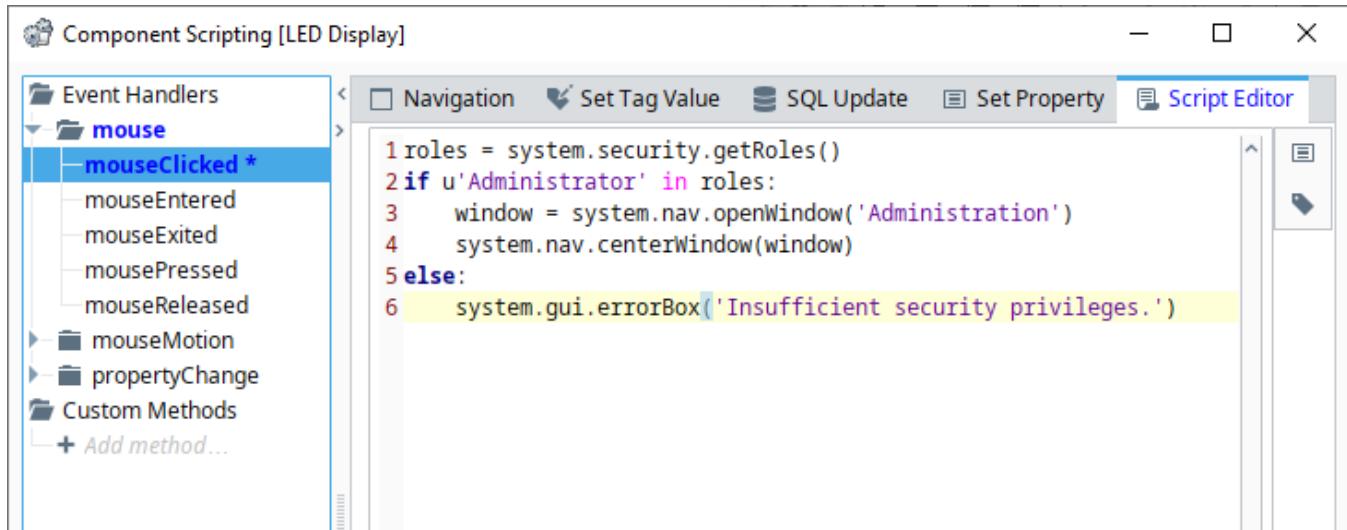
On the simplest level, security settings can be applied to [individual windows or components](#). Users with different roles can all view the same project from the client, but the functionality and readability can change based on the roles assigned to each user. Generally, higher level access provides full functionality to all contents of a project, and lower level access is restricted to generalized read-only privileges.

Below we see the Security Settings panel in action. This panel is the interface that applies Ignition's built-in security settings. Security settings can be applied to a single component, multiple components simultaneously, or even a whole window. Users who should be allowed full access can be selected, and restrictions can be applied for users that should not have full access.



Incorporating Scripting into Security

The component-based security settings are fairly simplistic: the user either has the required roles, or a restriction is applied. In situations where consideration for access should go beyond a simple role check, [security-based scripting](#) can provide a larger degree of granularity. Information about the logged-in user, such as user-name or roles, can be detected by scripting, allowing for the creation of a robust security system.



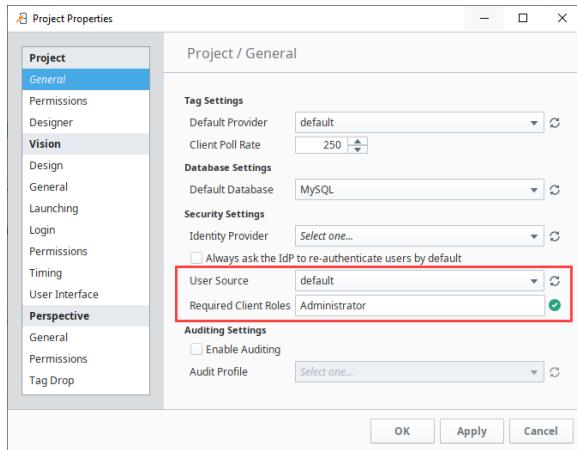
[In This Section ...](#)

Login Security

In Gateways with multiple projects, sometimes client access should be limited on a per-user-basis. The most common use-case is when users from one area of a facility should be able to launch clients that are specific to their area, but should not be able to launch clients from other areas. While there are several different ways to approach this, the easiest is to require different roles to log into each project.

Every project in Ignition is going to have a user source associated with it. For Vision you can change the **User Source** (Authentication profile) for a project as well as assign Required Client Roles within the **Project Properties**.

1. In the Designer, click on **Project** tab on the menu bar at the top of the Designer, then select **Project Properties**.



2. Use the **User Source** dropdown list to choose a User Source for this project. The selected User Source will only allow users from that User Source to access the project.
3. In the **Required Client Roles** field, enter the roles you want to grant access to this project. The Required Client Roles field will show all matching roles as you start typing. Multiple roles may be specified by separating them with a comma.
4. Click **OK** to save the changes.

Note: Changing your security settings will not log users out of an open client, but it will stop them from logging in again.

On this page ...

- [Login Issues](#)



Project Security

[Watch the Video](#)

Login Issues

If a user is experiencing some issues logging into a project, there are a few things that may be causing the problem.

1. First, make sure the credentials they are using are valid. This is easily done by going to the Gateway Webpage and [testing the login](#).
2. If the user's credentials were successfully tested against the user source, you can then verify if that user source is the one being used by the project that the user is trying to log in to. (See Above.)
3. If the user's credentials failed, they may have forgotten their password, and it may [need to be reset](#).
4. Alternately, the user source may not be reachable. Depending on the type of user source, this can happen when there are network problems. For example, a database user source will no longer work if the database connection is faulted.

Related Topics ...

- [Security](#)
- [Project Security in the Designer](#)

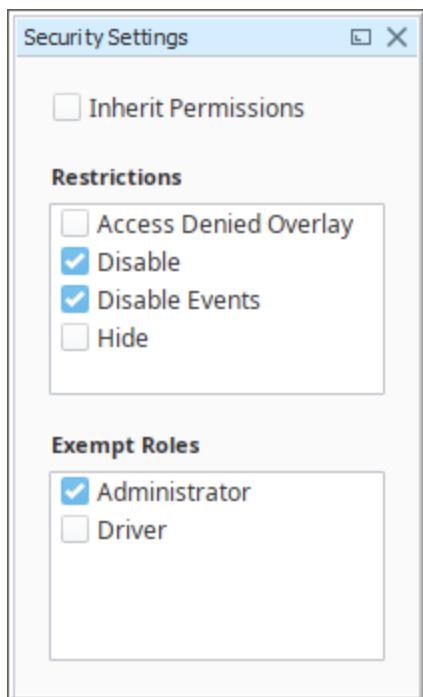
Component and Window Security

Role-based security inside of Ignition works on multiple levels: component, group, container, and window levels. Each of these levels also have special categories of security that help with tuning security to various design considerations.

Each window and component can define its own security settings. These settings determine who can see and/or use the component. It's good business practice to have a well thought out security policy for your project.

Changing Security Settings on a Component

By default, each component inherits the security that is on its parent which initially gives anyone with access to the project the ability to use the components. This can be changed on a per component basis by right clicking on the component and selecting **Security**. This brings up the Security Settings panel.



On this page ...

- [Changing Security Settings on a Component](#)
- [Exempt Roles](#)



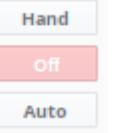
INDUCTIVE
UNIVERSITY

Component and Window Security

[Watch the Video](#)

The **Inherit Permissions** checkbox signifies that the component is inheriting the security settings of the parent.

In the **Restrictions** section on the right, choose the restrictions that will be placed on the component if the user does not have the selected role. Multiple restrictions can be selected to combine their effects.

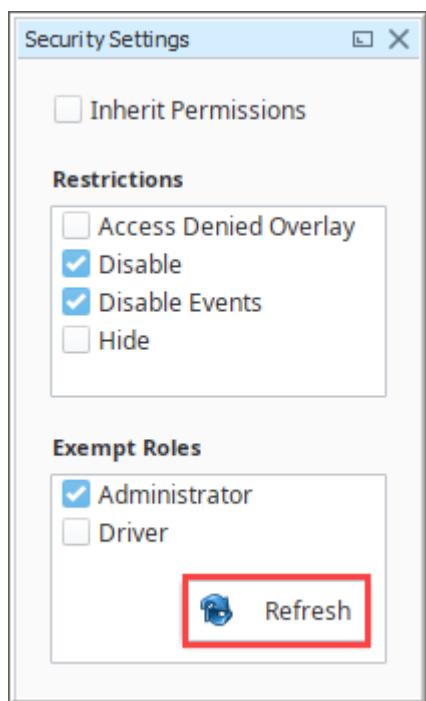
| Restriction | Used On | Description | Image |
|-----------------------|----------------------------|---|---|
| Access Denied Overlay | Components | Shows an overlay on top of the component when the user doesn't have security clearance. | Access Denied |
| Disable | Components | Sets the Enabled property to false on the component when the window opens up. | Disabled  |
| Disable Events | Components, Root Container | Prevents event scripts from running when the user doesn't have security clearance. | N/A |
| Hide | Components, Root Container | Sets the Visible property to false on the component when the window opens up. | |

| | | | |
|-------------|-------------------|--|--------|
| | | | Hidden |
| Do Not Open | The Window Object | Only used on the window object itself, will prevent the window from opening if the user doesn't have one of the specified roles. | N/A |

Exempt Roles

Unchecking the **Inherit Permissions** checkbox enables the role checkboxes under Exempt Roles. Each role that is selected will have access to the component. So if the Administrator role was checked, then all users with the Administrator role will be able to use the component, while users without the role would have Restrictions to the component. A user only needs to have one of the selected roles to be able to use the component, not all of them.

If the roles you created do not appear, it is probably because the Designer was open before those roles were created. To update the list of roles, right-click in the **Exempt Roles** section and select **Refresh**.



Note: If the roles you created do not appear, it is probably because the Designer was open before those roles were created. To update the list of roles, right-click in the **Exempt Roles** section and select **Refresh**.

Security in Scripting

While the Vision system has many options for securing individual components and clients, it is possible to have requirements that go further than the options that are available. With scripting, you can create any type of security that you may need. This is mainly done through the use of the `system.security.getRoles` function, as well as the other `system.security` functions. The `getRoles` function gets a list of the users roles, which you can check for specific roles within your script. You can then write code for what would happen if the user has the role and if they don't have the role.

Additionally, while the typical security set up only requires the user to have one of the required roles, in scripting you can ensure that the user has any combination of roles.

Securing Event Handlers

Security can be added to any of the event handlers in Ignition. This works on both components as well as on event handlers within the Scripting window. While the typical security settings for a window only give you the option of not opening it if the user does not meet the required roles, you can instead do something else like opening a different window.

On this page ...

- Securing Event Handlers
 - Script Builder Security Example
 - Security in Client Event Scripts
 - Setting Client to Read Only



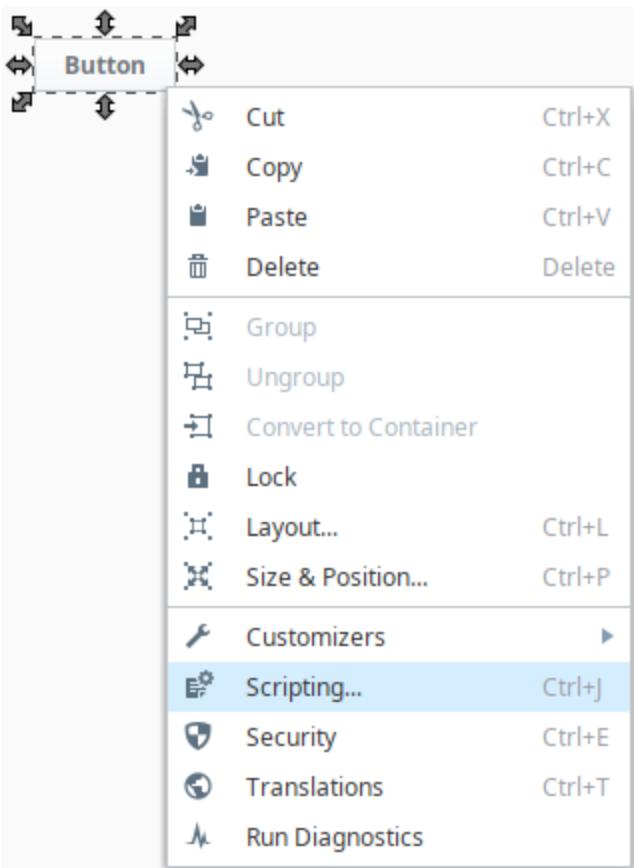
Securing Event Handlers

[Watch the Video](#)

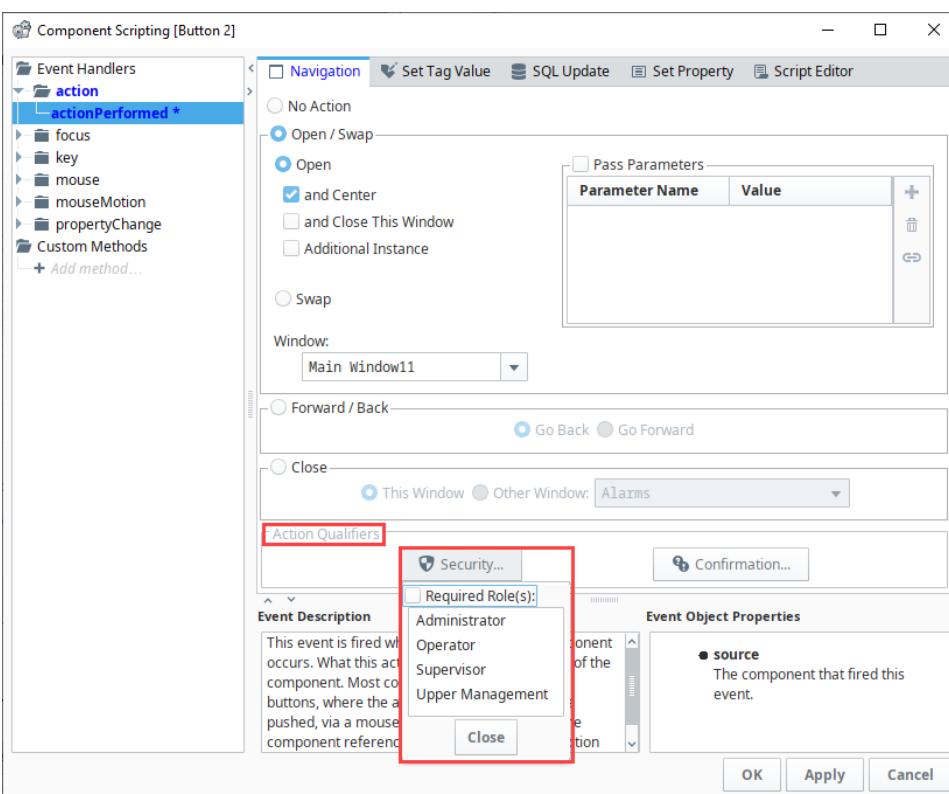
Script Builder Security Example

Each one of the `script builders` also has the ability to add security to them. The following is an example of setting up security on a Button component.

1. Drag a **Button** component onto a window.
2. Right click on the component, and choose **Scripting**.



3. Under the Navigation tab, click the **Open/Swap** radio button.
4. From the Window dropdown list, choose a window for this navigation.
5. Click the **Security** button in the **Action Qualifiers** section. Click the **Required Roles** checkbox.
6. Select the roles that are required for this navigation. Click **Close**.



7. Next, click on the **Script Editor** tab. You'll see the script that is generated by the options you chose in the Navigation tab.

```

1 # This script was generated automatically by the navigation
2 # script builder. You may modify this script, but if you do,
3 # you will not be able to use the navigation builder to update
4 # this script without overwriting your changes.
5
6 if u'QA' in system.security.getRoles():
7     window = system.nav.openWindow('Main Window11')
8     system.nav.centerWindow(window)
9 else:
10    system.gui.errorBox('Insufficient security privileges.')

```

8. Click **OK** to save the scripting. Now you added security to the Button component.

Security in Client Event Scripts

The **Client Event Scripts** can also be used to set up security within the project. While any of the Client Events can be used for different security purposes, the most common is the **Startup Script**. This allows you to create a customized, secure environment right as the Client is started. A Script here can do certain things based on the roles of the user such as open certain windows, write to client Tags to enable or disable certain things within a project, or even retarget to an entirely new project.

Setting Client to Read Only

There are times when it is best to open a Client in a Read-Only mode to eliminate the possibility that a Client will affect a device or database. The Client event startup script that sets the Client mode to Read-Only is an easy way to accomplish this. Similar to the buttons in the Designer, this function can be used to set Disconnected, Read-Only, and Read/Write modes in any script in Ignition that runs in a Client. This function can be called in any Client scoped script, but is most commonly used in the Startup script.

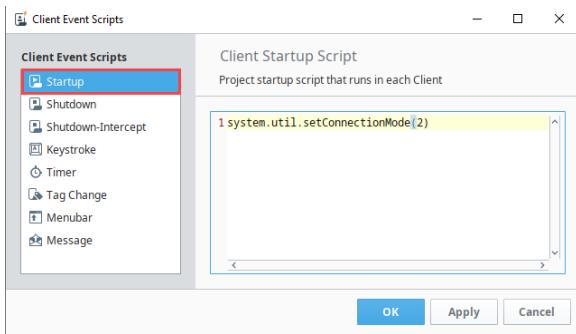
This example creates a Client event script that sets the Connection Mode to Read-Only.

1. From the Designer, go to **Project Browser > Client Event Scripts**. The Client Event Scripts window is displayed.
2. In the **Startup** script area, enter the following: `system.util.setConnectionMode(2)` where 2 means Read-Only.



Setting Client Read-Only

[Watch the Video](#)



3. Click **OK**. The startup script will run the next time a user logs into the Client, resulting in the Client being Read-Only.

Scripting in Vision

A lot of the scripting that happens in Vision is either located on components and windows, or it manipulates component and window values. Many times, you can easily add the path to another component or property from your script. It is important to understand how the component hierarchy of a window works as well as how to properly access components and properties in a script anywhere in a project

Component Hierarchy

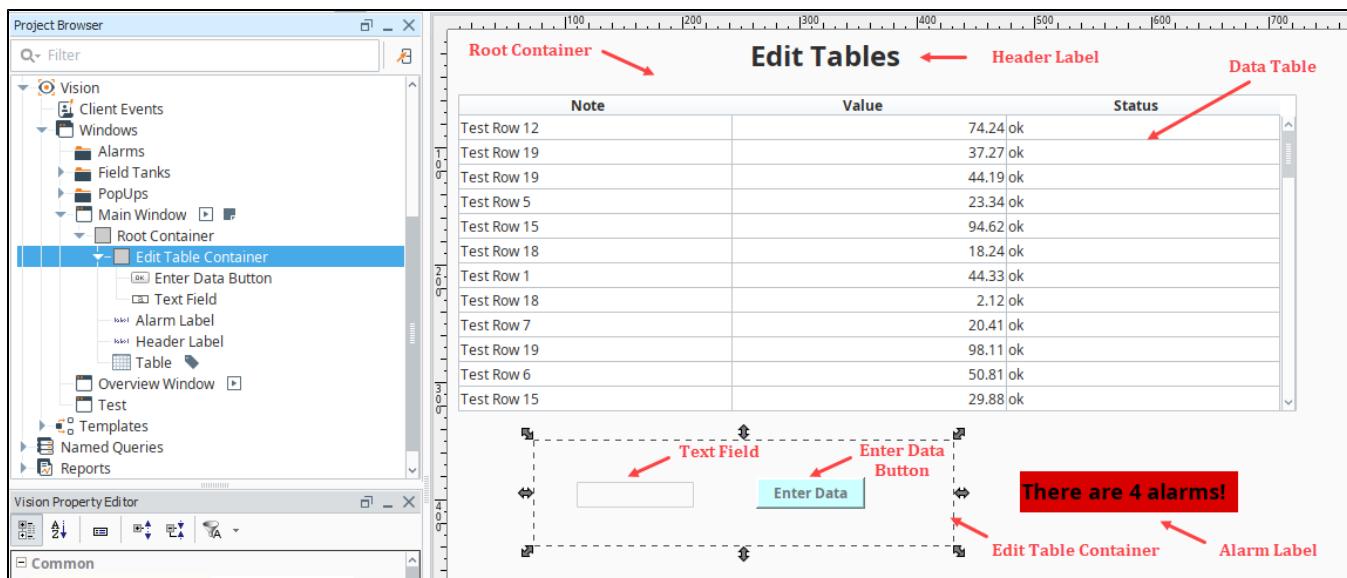
Every window in a project has a hierarchy to it, with components and containers arranged in a tree structure, with a single parent up at the top, and many children down at the bottom. While small windows with only a few components can have a simple hierarchy, windows with many containers can get much more complex. Since the components are arranged like a tree, this means that we can only move up and down through the tree structure and not sideways. To get from one component to a sibling component, we must first navigate up towards the common parent, and then back down to the desired child. Let's take a look at an example of how this works. Here we have a simple window, with just a few components:

On this page ...

- Component Hierarchy
 - Accessing Component Properties
- Accessing a Component
 - From an Event Handler
 - From an Extension Function
 - From a Client Event Script
 - From a Project Script
- Accessing Components on Other Windows

- Root Container
 - Edit Table Container
 - New Value Text Field
 - Enter Data Button
 - Alarm Label
 - Data Table
 - Header Label

We can flip the tree around to get a better understanding of exactly how the tree works:



Here we can get a better idea of why we can only move up and down through the tree structure, and how getting to a component can be very different depending on where you start. Let's say we want to go to the New Value Text Field so that we can grab its value.

| Start From | Path To New Value Text Field |
|----------------------|--|
| Edit Table Container | 1. Down to New Value Text Field |
| Enter Data Button | 1. Up to Edit Table Container 2. Down to New Value Text Field |
| Data Table | 1. Up to Root Container |

- | | |
|--|--|
| | <ol style="list-style-type: none"> 2. Down to Edit Table Container 3. Down to New Value Text Field |
|--|--|

To move up or down within the hierarchy, there are two special commands we can use on component objects: **parent** and **getComponent()**. The **parent** property allows us to grab a reference to whatever is directly above the component in the hierarchy, which in most cases is the root container. We could then access any component on the root container by using **getComponent("Component Name")** and then placing the name of the component we want to access within the parenthesis.

Pseudocode - Component Hierarchy

```
# This pseudo code shows how to grab the parent of a component
component.parent

# This pseudo code shows how to grab a child of a component
component.getComponent("Text Field")
```

Both **parent** and **getComponent()** can be used as many times as necessary to reach the desired component, drilling up or down through layers of containers or grouped components. Once you have a component reference, you can then access any one of that component's attributes by using the name of the property, just like when accessing a property on the source component.

Exception to the pattern of using .parent

There is one exception to the pattern of using **.parent** to go up the hierarchy and using **.getComponent(name)** to go down. The parent of a root container is not the window, and a reference to the window does not have a **.getComponent(name)** function. To get a reference to a window, simply use **system.gui.getParentWindow** with any component's **event** object as the parameter. Once you have a reference to a window, you can use its **.rootContainer** property to get to the root of the component hierarchy, and from here you can follow the rules laid out above.

Accessing Component Properties

To access a property within the component, we simply need to use the scripting name of the property. The scripting name can be found in the description of each property, either by enabling the description field or hovering over the property until the mouseover text appears. The scripting names for every property on every component can also be found in the [appendix](#). For a text field, the scripting name of the text property is just **text**, so we would need to call that on the text field which has the **text** property we want to access.

Pseudocode - Component Properties

```
# This pseudo code will access the text property of a component and assign
it to value.
value = component.text
```



Accessing Component Properties

[Watch the Video](#)

Accessing a Component

Now that we have an understanding of how the component hierarchy works, we can apply that knowledge to accessing a component from anywhere within the project. While moving up and down within the hierarchy remains the same, accessing our initial component can differ depending on where we start our script from.

From an Event Handler

Event Handlers get a special **event** object that has special properties depending on the type of event. Regardless of the event, all event objects have a **source** property, which gives the component that fired the event. When accessing a component from an event handler, we can first use **event.source** to get a component on the window to start at. From there, we can use **parent** or **getComponent()** as needed to get to the component we need to access.

Python - Accessing a Component from an Event Handler

```
# This would access the text property of a Text Field component.
print event.source.parent.getComponent('Text Field').text
```

When accessing a component from an event on the window, there will be a different path to the component than normal. If you already have a window object, you can use the function `getComponentForPath()`. This allows you to enter in the path to the component as a string (similar to expression bindings), and will end up looking something like this:

Python - Accessing a Component from an Event on a Window

```
system.gui.getParentWindow(event).getComponentForPath('Root Container.Text Field').text
```

You can also get the Root Container directly using the `getRootContainer()` function. This link of code works the same as the one above:

Python - Accessing a Component from an Event on a Window

```
system.gui.getParentWindow(event).getRootContainer().getComponent('Text Field').text
```

From an Extension Function

[Extension Functions](#) get a special `self` object which is actually a direct reference to the component that the extension function is on. This provides a direct reference point from which to access other components within the component hierarchy.

Python - Accessing a Property of a Component from an Extension Function

```
# This would access the text property of the component running the extension function script.  
print self.text  
  
# This would access the text property of the component named 'Text Field' if it is in the same container.  
print self.parent.getComponent('Text Field').text
```

From a Client Event Script

[Client Event Scripts](#) are special because they don't start with a direct reference to anything on a particular window. So, we have to use another means of finding a starting point on the window. The `system.gui.getWindow()` function allows us to get a reference to a window which we can use to navigate to the root container with `.getRootContainer()`, and then to any component on that window. However, this will only work if the window is currently opened. If the window is closed, it will throw an error, which can be handled with normal exception handling.

Python - Accessing a Component from a Client Event Script

```
# Start the try block in case the window is not open.  
try:  
    # Grab the window reference and assign it to the variable window.  
    window = system.gui.getWindow("Other Window")  
  
    # Use the window reference to get the text property off of a text field.  
    print window.getRootContainer().getComponent("Text Field").text  
  
# Handle the exception by opening an informative error.  
except ValueError:  
    system.gui.errorBox("The window is not open!", "Error")
```

From a Project Script

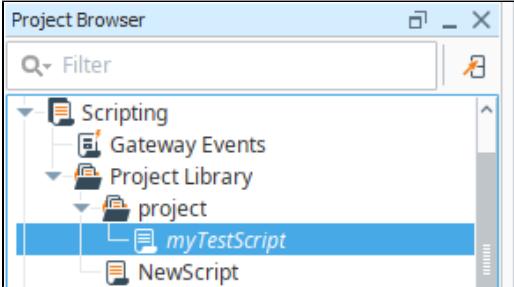
[Project Library](#) are unique in that how they access components can vary depending on where the Script Module is being called from and what is being passed to it. If the script module is being called from an event handler or an extension function, it is possible to pass in the event or self objects and use them within the script module.

Python - Accessing a Component from a Project Script

```
# This code would go in a project script. We are defining our function that takes an event object  
# and uses it to find the value of the text property on the text field in the same container.  
def func(event):  
    print event.source.parent.getComponent('Text Field').text
```

Python - Calling a Function from the actionPerformed Button

```
# On the action performed of a button on our window, we could then use this to call our function.  
myTestScript.func(event)
```



The screenshot shows the Project Browser interface. On the left, there's a tree view under 'Scripting' with 'Gateway Events', 'Project Library', and a 'project' folder containing 'myTestScript' and 'NewScript'. On the right, the code editor displays the following Python script:

```
project.myTestScript  
1 def func(event):  
2     print event.source.parent.getComponent('Text Field').text
```

However, this may not always be the case. In these instances, it is possible to instead use the same method that Client Event Scripts use and grab the window object instead.

Accessing Components on Other Windows

You can also grab properties from components on other open windows from anywhere in the project using the same method used in Client Event Scripts. This allows you to grab properties on a main window from an event handler on a popup window.

Note:

Remember, you can only grab a property from another window if the other window is open.



Finding Components on Other Windows

[Watch the Video](#)

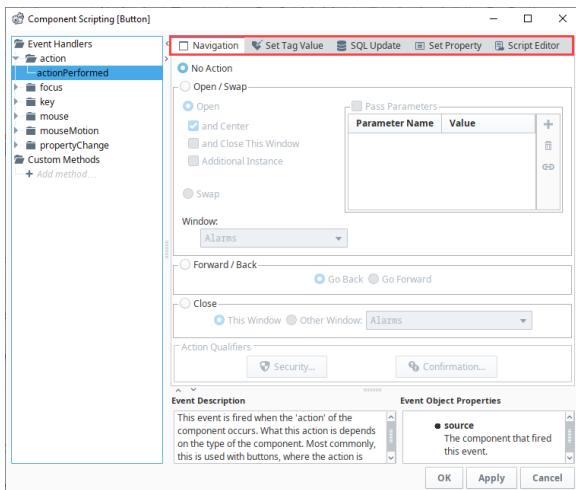
Related Topics ...

- [Project Library](#)

In This Section ...

Script Builders in Vision

When creating an Event Handler on a component, you can use one of the handy **Script Builders** instead of writing your own script. In the Component Scripting window, the Script Builders are accessible as tabs along the top. Each tab represents a different kind of action that users can associate with an event. The last tab, **Script Editor**, lets you write your own event handler. You can also use the **Script Editor** tab to view a script that was generated by one of the builders, which is a good way to get started learning how to write your own event handlers. Each script builder example on this page shows the actual script in the Script Editor, just simply click the link under each example.



On this page ...

- [Navigation Script Builder](#)
 - [Open / Swap](#)
 - [Forward / Back](#)
 - [Closing Windows](#)
- [Set Tag Value Script Builder](#)
- [SQL Update Script Builder](#)
- [Set Property Script Builder](#)
- [Script Editor](#)
 - [Advanced Settings](#)
- [Action Qualifiers](#)
 - [Security Qualifier](#)
 - [Confirmation Qualifier](#)



INDUCTIVE
UNIVERSITY

Vision Event Scripts Overview

[Watch the Video](#)

Note: Only one Script Builder can be used at a time. If you previously picked another action using a different Script Builder, it will get overwritten by enabling another Script Builder. If you need to do more than one action at a time, use the Script Editor to combine scripts, or create your own script.

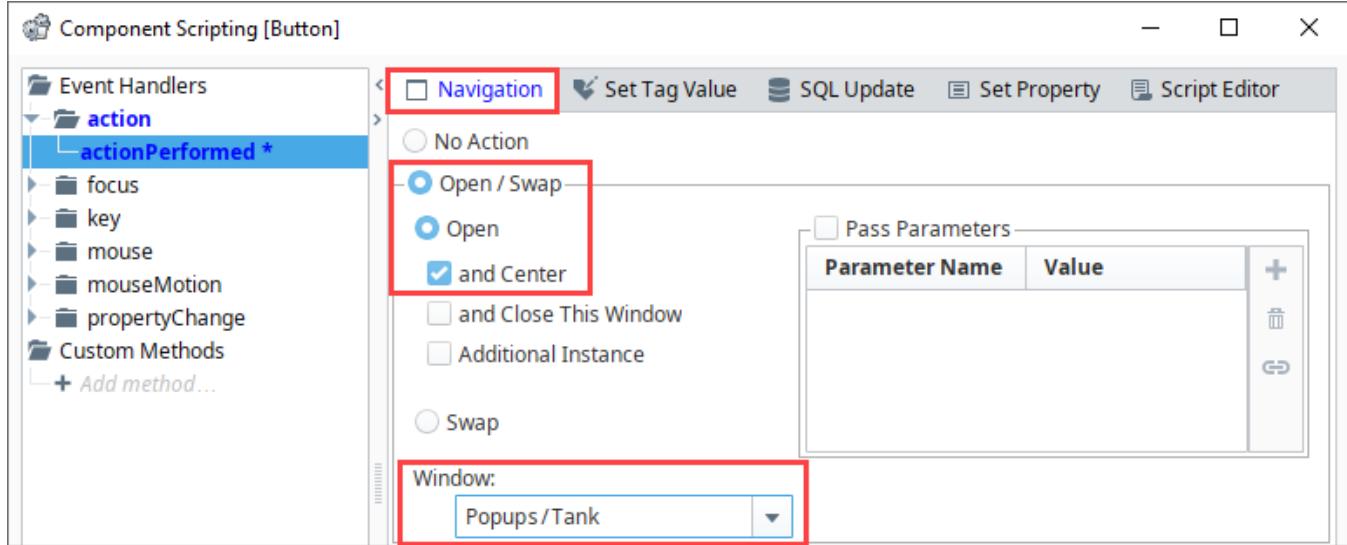
Navigation Script Builder

The **Navigation** Script Builder has various functions that deal with opening and closing windows.

Open / Swap

Opening is a very straight-forward operation, it simply opens the specified window at the same size it was in the Designer. Simply click on the **Open / Swap** button, and select a Window from the dropdown list that you want to open. There are options to center that window within the Client, and to close the window that the event was fired from. The opened window can also be opened as an additional instance, meaning there can be multiple

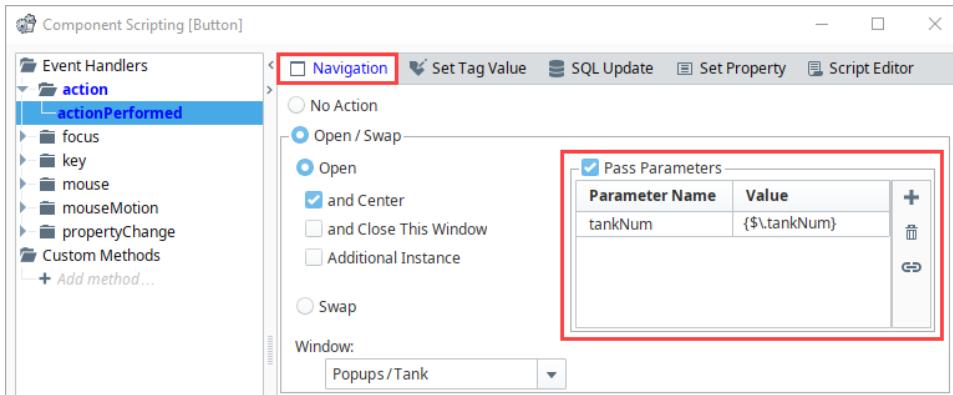
copies of the same window. This is useful when opening dynamic popups, so that a couple of popups can be opened, each with different values.



Swapping is the practice of opening another window in the same size, location, and state as the current window, and closing the current window. This gives the appearance of one window simply swapping into another, seamlessly. The Navigation Builder uses the **swapWindow** version of swapping, but most "by hand" script authors will use the **swapTo** version. This last version relies on the fact that the windows being swapped are both maximized windows. See the [navigation strategy](#) section for more information.

You can also **pass parameters** to the opened or swapped-to window. Check the **Pass Parameters** box, and click the **Add** icon to add a row, where each row is another passed parameter. The names of these parameters must match names of custom properties on the root container of the target window. The values can either be literals or values of other properties from the source window. Use the **Binding icon** to navigate to the properties that you want to pass. It will construct the path to the property on the window. To pass parameters in a navigation window, follow the steps below.

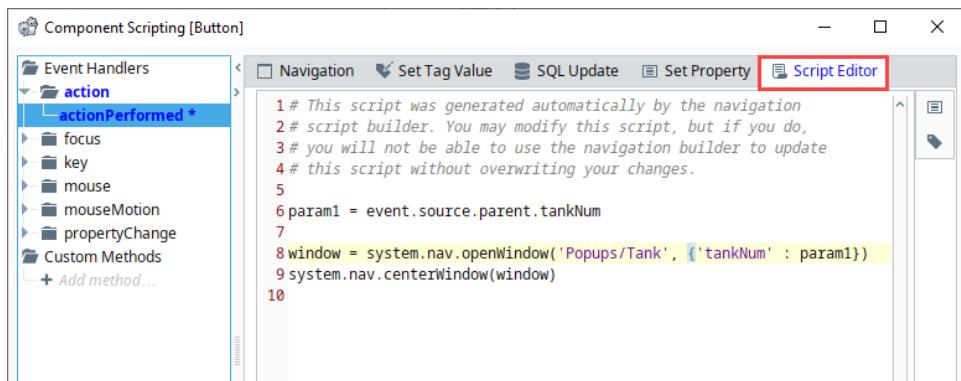
1. Check the **Pass Parameters** box, and click the **Add** icon to add a row.
2. Choose a custom property from the "Parameter Name" dropdown list. This will be filled with custom properties on the root container of the selected window. You can also type a name in directly.
3. Highlight the empty cell in the **Value** column of the parameter table, click the **Binding icon** , select the component property you want to enter the value and press **OK**.
4. Press **OK** to commit the change.



To learn more about passing parameters, refer to the [parameterized windows](#) section for more information.

The following Navigation builder script was generated by the Script Editor. If you compare the settings in the Navigation tab with the documented code in the Script Editor, you'll notice the Tank popup window will be opened and centered in the window, and the value of

the "tankNum" parameter passed.



The screenshot shows the 'Component Scripting [Button]' dialog. The left sidebar lists event handlers: 'Event Handlers', 'action' (selected), and 'Custom Methods'. Under 'action', 'actionPerformed' is selected. The right panel contains a script editor with the following code:

```
1 # This script was generated automatically by the navigation
2 # script builder. You may modify this script, but if you do,
3 # you will not be able to use the navigation builder to update
4 # this script without overwriting your changes.
5
6 param1 = event.source.parent.tankNum
7
8 window = system.nav.openWindow( 'Popups/Tank' , { 'tankNum' : param1 })
9 system.nav.centerWindow(window)
10
```

Forward / Back

The **Forward / Back** actions give you a simple way of implementing *browser-style* forward/back buttons in your client. You must be swapping between windows for this to work, because these functions rely on calls to `system.nav.swapTo` in order to keep track of what the sequence of recent windows has been.

Closing Windows

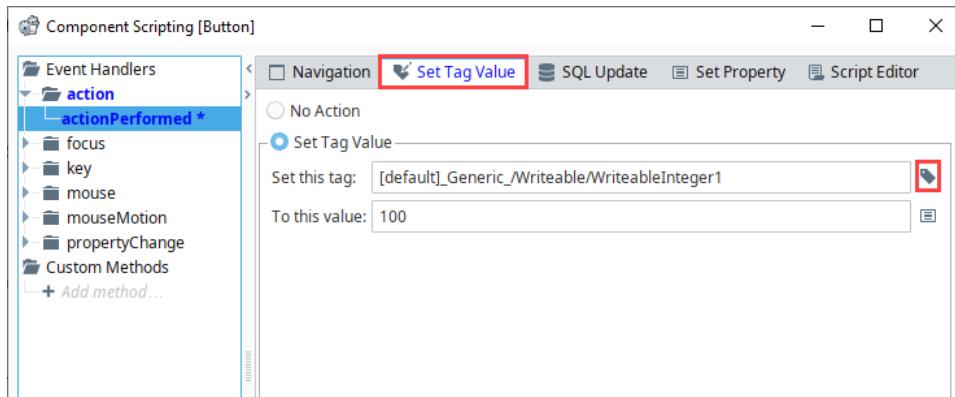
The Closing windows action allow for an easy way to have an event handler close the window that it is a part of, or any other window.

Set Tag Value Script Builder

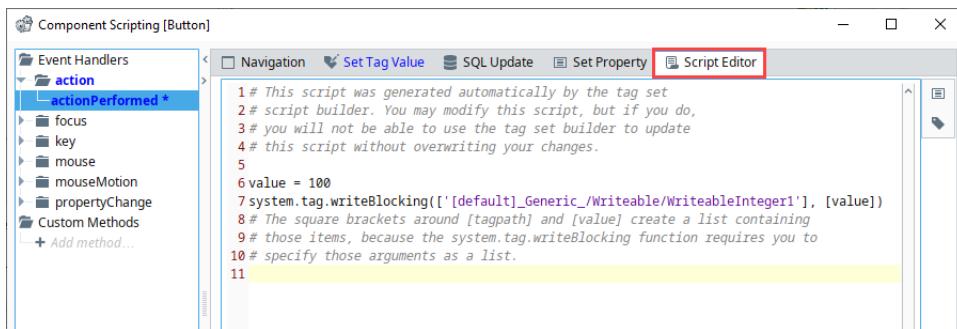
The Set Tag Value Script Builder responds to an event by setting the value of a Tag. You can set the Tag to either a literal value directly typed in, but we recommend using the chain link **Tag** icon to have the event handler use the value of another property from the same window.

Use the steps below to create a Set Tag Value.

1. Under the **Set Tag Value** tab, click the **Set Tag Value** radio button.
2. Click the **Tag** icon and choose a **Tag** from the Tag browser list to write to (i.e., `WriteableInteger1`).
3. In the **To this Value** field, enter a number (i.e., 100) or click the **Property** icon to browse for a component property to use as the set-to value.
4. Press **OK** to commit the change.



The **Set Tag Value** script shown below was generated by the Script Editor. Compare the settings in the Set Tag Value tab with the documented code in the Script Editor and you'll see the Tag is set to "WriteableInteger1" and the Tag value is set to "100".



```

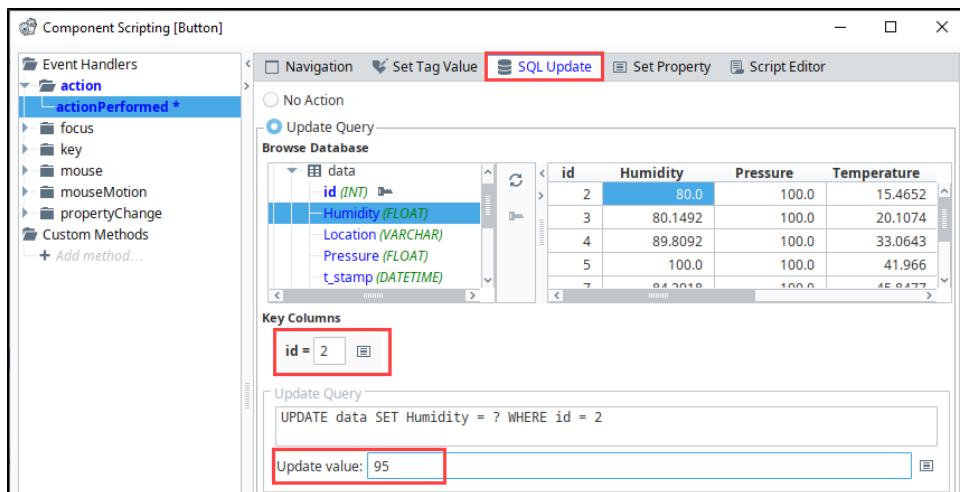
Component Scripting [Button]
Event Handlers
  action
    actionPerformed *
      focus
      key
      mouse
      mouseMotion
      propertyChange
    Custom Methods
      + Add method...
Navigation Set Tag Value SQL Update Set Property Script Editor
1 # This script was generated automatically by the tag set
2 # script builder. You may modify this script, but if you do,
3 # you will not be able to use the tag set builder to update
4 # this script without overwriting your changes.
5
6 value = 100
7 system.tag.writeBlocking(['[default]_Generic_/Writeable/WriteableInteger1'], [value])
8 # The square brackets around [tagpath] and [value] create a list containing
9 # those items, because the system.tag.writeBlocking function requires you to
10# specify those arguments as a list.
11

```

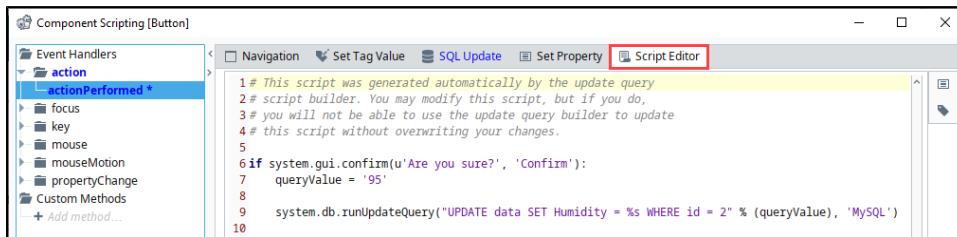
SQL Update Script Builder

The SQL Update Script Builder helps you build an update query using a [database browsing interface](#). Choose a database and table in your target database, and the update query will be built for you. The key query will help identify a specific row to update, and can be made dynamic using Update Query and Update Value text boxes.

- Under the SQL Update tab, click the **Update Query** radio button.
- Select a database and choose a table in your database. Select an event value in the table on the right.
- This example has the **id** as the Key Column, so we entered '**id = 2**'.
- We selected the **Humidity** for 'id 2'.
- To change the value in the database, we entered the new value in the **Updated Value** field to replace the previous value when the action is executed.
- Press **OK** to commit the change.



The following SQL Update script was generated by the Script Editor. If you compare the settings in the SQL Update tab with the documented code in the Script Editor, you'll see your database, table, and column name, including the row **id** of the searched value. You'll also see the new update value that will replace the existing value when the action is executed.



```

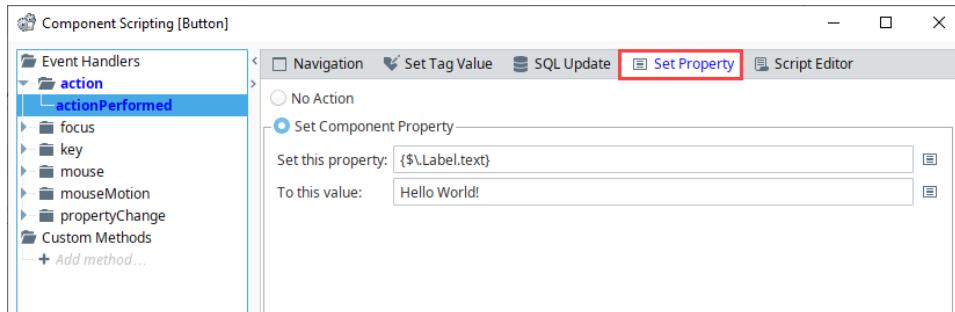
Component Scripting [Button]
Event Handlers
  action
    actionPerformed *
      focus
      key
      mouse
      mouseMotion
      propertyChange
    Custom Methods
      + Add method...
Navigation Set Tag Value SQL Update Set Property Script Editor
1 # This script was generated automatically by the update query
2 # script builder. You may modify this script, but if you do,
3 # you will not be able to use the update query builder to update
4 # this script without overwriting your changes.
5
6 if system.gui.confirm(u'Are you sure?', 'Confirm'):
7   queryValue = '95'
8
9   system.db.runUpdateQuery("UPDATE data SET Humidity = %s WHERE id = 2" % (queryValue), 'MySQL')
10

```

Set Property Script Builder

The Set Property Script Builder will respond to an event by altering a property in the window. You must choose the property to alter and choose the value that you wish to assign to it. The new value can be a literal value or the value of any other property on the window.

1. Drag a Label component to your window to set the property to.
2. Select the **Button** component and open the Scripting window, and select **actionPerformed**.
3. In the Set Property tab click the **Set Component Property** radio button.
4. On the **Set this property** click on the **Property** icon to browse for the Label component's **Text** property, and click **OK**.
5. Type something into the **To this value:** field (i.e., Hello World!), or click the **Property** icon to browse for a component property to use as the set-to value.
6. Press **OK** to commit the change.



This Set Property script was generated by the Script Editor. Compare the settings in the Set Property tab with the documented code in the Script Editor and you'll see the value string "Hello World!" will be written to the Label component when the action executes.



Script Editor

The Script Editor allows you to add more complexity to existing scripts, combine scripts, and even write your own custom scripts. For example, if you need to perform two or more actions at once, (i.e., set a Tag and navigate to another window), you can update the script by combining the two actions in the Script Editor. The Script Editor even gives you the flexibility to create your own code for any action you want to perform on an event handler.

As seen elsewhere on this page, the other builders ultimately generate a script that can be modified from the from the Script Editor Builder.

Advanced Settings

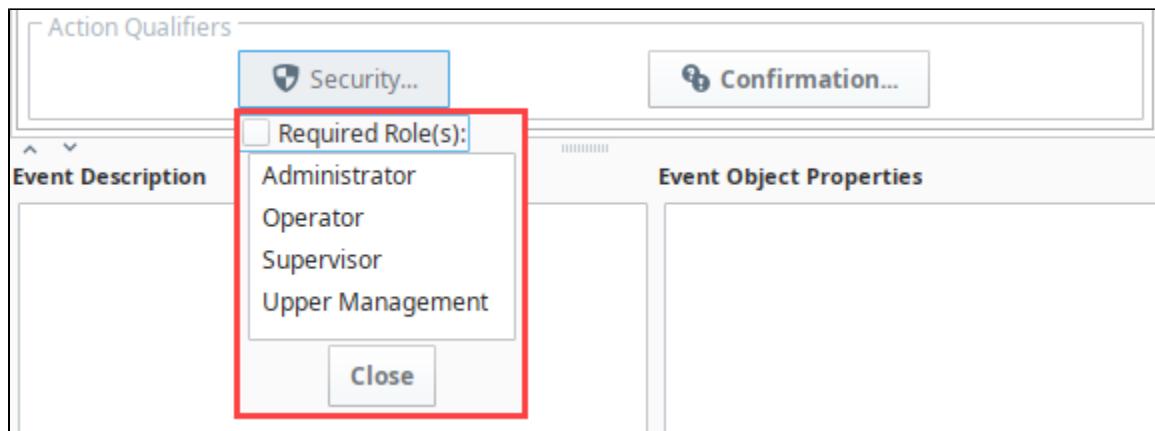
- **Scoping Dropdown** - Allows you to specify the scoping of the script. The scoping of event handlers in older versions of Ignition work differently than modern versions. This setting was added as a way to provide backwards compatibility when upgrading. All new scripts should ideally leave the scope set to Standard Scoping, as there is no reason for new scripts to use the Legacy Scoping option.
- **Invoke Later** - Provides an opportunity to allow the script to run after other events have finished processing. Most scripts will leave this setting disabled, however it can be useful in some scenarios:
 - [Controlling the focus order in a window](#), since requesting focus from a component in the middle of an event being processed can cause undesirable results.
 - When writing a script on a window's [visionWindowOpened](#) event, you may wish to have your script run after processing.

Action Qualifiers

All of the Script Builders allow you to put Security and/or Confirmation qualifiers onto an event handler. These Action Qualifiers are optional.

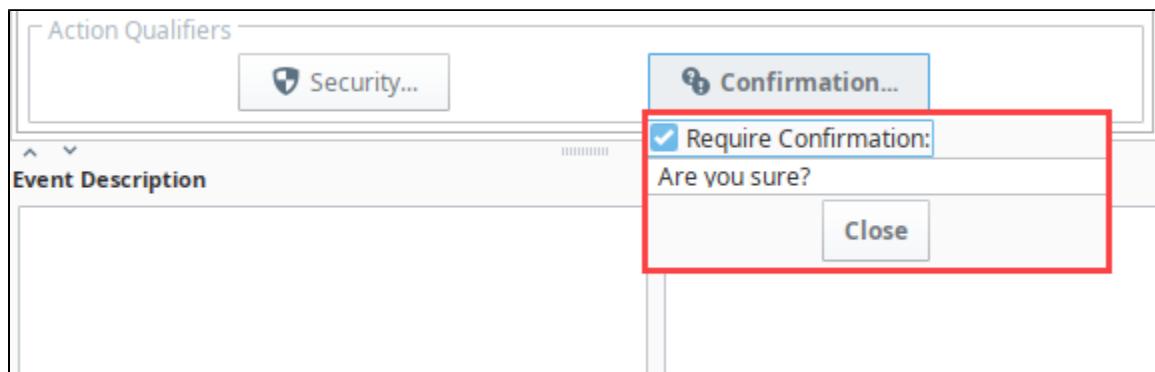
Security Qualifier

The Security Qualifier lets you restrict the event handler from running if the current user does not have one of the required roles highlighted in the Security qualifier dialog box. The roles listed will be all of the roles within the project's default user source. To setup the required roles for an event handler, select one or use **CTRL+click** to select multiple roles. Once the roles are selected, they will be highlighted. To deselect roles, use **CTRL+click** again. The action will only be executed if the **Required Roles** checkbox is enabled. Click **Close**.



Confirmation Qualifier

The Confirmation Qualifier prompts the user with a popup dialog box confirming you want to perform the action. The action will only be executed if the **Require Confirmation** checkbox is enabled. There is a default message, and if you prefer, you can delete it and enter your own message. Click **Close**.



Related Topics ...

- [Parameterized Popup Windows](#)
- [DB Browse Bindings](#)

Component Events

Event Handlers

When running a script on a component, we typically don't want it to be constantly running, but instead want the script to trigger when the user does something on screen such as clicks with the mouse. That something the user does is called an *Event* and can range from a mouse click or a keypress to a window opening or a component property change. When certain events happen, they trigger event handlers, which use a script to *handle* what happens when the event occurs.

This page lists out all of the event handlers that are on Ignition's Vision module. Any third party modules may add new components which may potentially have new event handlers.

Event Object

Every event handler contains an **event** object, which allows you to interact with the component and the entire window hierarchy within your script. While each **event** object has different properties depending on what event handler it resides in, each **event** object contains a **source** property, which is a reference to the component that fired the event. Using `event.source` not only gives us access to all of the properties available on that particular component, such as the `text` property of a text field,

Pseudocode - Event Handler Source Component Properties

```
# Here we start with the event object, then use source to go to the
# component that fired the event,
# and then use the name of the property to access its value. In this case,
# we accessed the text property.
text = event.source.text
```

but it also provides us with a way to navigate to other components within the hierarchy. For example, here we have a script on a button that references a text field.

Pseudocode - Event Handler Other Components

```
# Here again we start with event.source to get to the component that fired
# the event, but now we use
# parent to go up to the root container, and then getComponent to navigate
# back down to a different component.
text = event.source.parent.getComponent("Text Field")
```

On this page ...

- [Event Handlers](#)
- [Action Event Handlers](#)
- [Property Event Handlers](#)
- [Mouse Event Handlers](#)
- [MouseMotion Event Handlers](#)
- [Key Event Handlers](#)
- [Focus Event Handlers](#)
- [VisionWindow Event Handlers](#)
- [InternalFrame Event Handlers](#)
- [Cell Event Handlers](#)
- [Item Event Handlers](#)
- [Paint Event Handlers](#)



INDUCTIVE
UNIVERSITY

Vision Event Scripts Overview

[Watch the Video](#)

Note: Even when components are disabled, most scripting events can still occur. For example, a mouse click can still happen on a disabled component, which is why it is recommended to use the action performed event when placing a script on a button.

The **Action** category of event handlers pertains to components being "used" from the client, such as a button being pressed or a checkbox component being selected. You can access event handlers through the Scripting option.

Action Event Handlers

Events

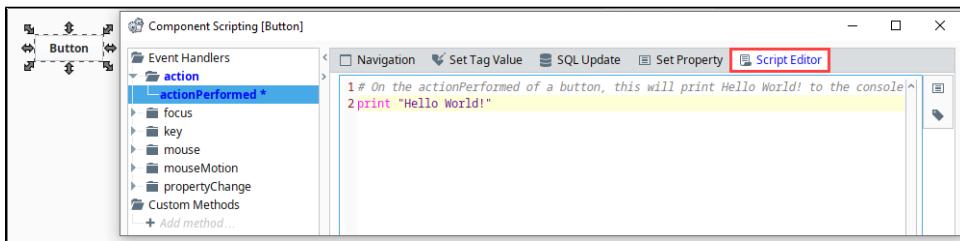
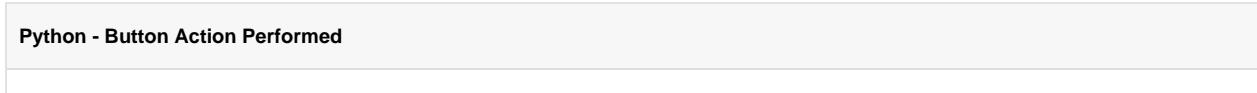
| Events | Description |
|-----------------|---|
| actionPerformed | This event is fired when the 'action' of the component occurs. What this action is depends on the type of the component. Most commonly, this is used with buttons, where the action is that the button was pushed, via a mouse click or a key press. See the component reference for details on what the action means for other components. It is recommended to use this event over mouseClicked whenever possible. |

Event Object Properties

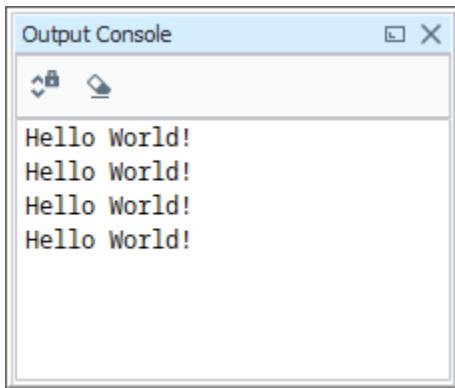
| Properties | Description |
|------------|--------------------------------------|
| source | The component that fired this event. |

In this example, let's use a Button component to print "Hello World!" on the console each time the Button is pressed.

1. Drag a **Button** component to your Designer workspace.
2. Select your Button and right click on **Scripting**.
3. Copy the code in the code block and paste it in the Script Editor tab. Press **OK**.



4. Save your project, and put the Designer in **Preview Mode**.
5. From the top menubar click on **Tools > Console**. Press the button and you'll see "Hello World" printed each time the button is pressed.



Property Event Handlers

Property event handlers typically trigger based on the property of a component.

Events

| Events | Description |
|----------------|--|
| propertyChange | Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties. |

Event Object Properties

| Properties | Description |
|--------------|--|
| source | The component that fired this event. |
| newValue | The new value that this property changed to. |
| oldValue | The value that this property was before it changed. Note that not all components include an accurate oldValue in their events. |
| propertyName | The name of the property that changed. |

Note: Remember to always filter out these events for the property that you are looking for! Components often have many properties that change,

Python - Printing the Changing Property

```
# On the propertyChange of a component, this script will print out the name of the property that is changing.  
print event.propertyName
```

Python - Looking for a Specific Property

```
# It is common to use propertyName to look for specific properties to change. This is a great way to  
restrict how often your scripts execute  
if event.propertyName == 'text':  
    print 'The Text property changed'
```

Mouse Event Handlers

The mouse events all correspond to the clicking and movement of the mouse. They are triggered in the client by an operator interacting with a mouse. Touchscreen monitors will trigger these events when a user touches the screen, but not all touchscreens will fire the mouseEntered and mouseExited events.

Events

| Events | Description |
|---------------|---|
| mouseClicked | <p>This event signifies a mouse click on the source component. A mouse click is the combination of a mouse press and a mouse release, both of which must have occurred over the source component.</p> <p> This event fires after the pressed and released events have fired.</p> <p>Note:</p> <p> This event fires after the pressed and released events have fired.</p> |
| mouseEntered | This event fires when the mouse enters the space over the source component. |
| mouseExited | This event fires when the mouse leaves the space over the source component. |
| mousePressed | This event fires when the mouse presses down on the source component. |
| mouseReleased | This event fires when a mouse button is released, if that mouse button's press happened over this component. |

Event Object Properties

| Properties | Description |
|------------|--|
| source | The component that fired this event. |
| button | The code for the button that caused this event to fire. Use the constants event.BUTTON1, event.BUTTON2, and event.BUTTON3. |
| clickCount | The number of mouse clicks associated with this event. |
| x | The x-coordinate (with respect to the source component) of this mouse event. |
| y | The y-coordinate (with respect to the source component) of this mouse event. |

| | |
|--------------|--|
| popupTrigger | Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists. |
| altDown | True (1) if the Alt key was held down during this event, false (0) otherwise. |
| controlDown | True (1) if the Control key was held down during this event, false (0) otherwise. |
| shiftDown | True (1) if the Shift key was held down during this event, false (0) otherwise. |

Python - Printing on a Mouse Enter

```
# On the mouseEntered event of a component, this script will only fire if the mouse enters the bounds of the component.
print "The mouse is inside the component space!"
```

MouseMotion Event Handlers

The mouseMotion events deal with the motion of the mouse over a component. Not all touchscreen monitors will fire these events.

Caution:

mouseMotion events will not trigger when the project is viewed from a mobile project as these gestures are used by the browser/device to zoom or pan.

Events

| Events | Description |
|--------------|---|
| mouseDragged | Fires when the mouse moves over a component while a mouse button is being held. |
| mouseMoved | Fires when the mouse moves over a component, but no buttons are being held. |

Event Object Properties

| Properties | Descriptions |
|--------------|--|
| source | The component that fired this event. |
| button | The code for the button that caused this event to fire. Use the constants <code>event.BUTTON1</code> , <code>event.BUTTON2</code> , and <code>event.BUTTON3</code> . |
| clickCount | The number of mouse clicks associated with this event. |
| x | The x-coordinate (with respect to the source component) of this mouse event. |
| y | The y-coordinate (with respect to the source component) of this mouse event. |
| popupTrigger | Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists. |
| altDown | True (1) if the Alt key was held down during this event, false (0) otherwise. |
| controlDown | True (1) if the Control key was held down during this event, false (0) otherwise. |
| shiftDown | True (1) if the Shift key was held down during this event, false (0) otherwise |

Python - Printing on a Mouse Movement

```
# From the mouseMotion event on a component, this will print each time the mouse moves when it is over the component.
print "The mouse is moving over the component!"
```

Key Event Handlers

The key events all have to do with the user pressing a key on the keyboard.

Events

| Events | Description |
|-------------|--|
| keyPressed | Fires when a key is pressed and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3. |
| keyReleased | Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3. |
| keyTyped | Fires when a key is pressed and then released when source component has the input focus. Only works for characters that can be printed on the screen. |

Event Object Properties

| Properties | Description |
|-------------|---|
| source | The component that fired this event. |
| keyCode | The key code for this event. Used with the keyPressed and keyReleased events. Uses the standard Java key codes, see below for more information. |
| keyChar | The character that was typed. Used with the keyTyped event. |
| keyLocation | Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN. Uses the standard Java key locations, see below for more information. |
| altDown | True (1) if the Alt key was held down during this event, false (0) otherwise. |
| controlDown | True (1) if the Control key was held down during this event, false (0) otherwise. |
| shiftDown | True (1) if the Shift key was held down during this event, false (0) otherwise. |

Python - Printing the Key Released

```
# On the keyReleased event of a component, this will print out the key code of the key that was hit on the
# keyboard,
# but only on release of the key, and only when the component has focus.
print event.keyCode
```

Java Keys

The key event handlers use the Java KeyEvent class, which has unique identifiers for both keys and locations on the keyboard to help differentiate which key is actually being pressed on the keyboard. The numeric codes for each unique location and character can be called from the event object using a constant. For example, the letter "a" has the constant name VK_A. This can then be used to compare against the keyCode value like this:

Python - Checking Specific Key Codes

```
if event.keyCode == event.VK_A:
    print "The key press was a"
```

We listed the locations and some common codes below, but the full list of codes can be accessed by going to <https://docs.oracle.com/javase/8/docs/api/java/awt/event/KeyEvent.html>.

Note: Some Operating Systems reserve certain keys for certain function, and will capture the key press or release before it gets sent to the Client. For example, many Operating Systems use the TAB key to shift focus to the next field

Key Code Constants

Key Location Constants

| | | | | |
|----------------|------------|------------|----------------|-------------|
| VK_0 - VK_9 | VK_END | VK_PAGE_UP | VK_DOWN | VK_CONTR OL |
| VK_A - VK_Z | VK_ENTER | VK_RIGHT | VK_PAGE_DOW WN | VK_LEFT |
| VK_F1 - VK_F24 | VK_HOME | VK_SHIFT | VK_UP | VK_TAB |
| VK_ALT | VK_INSE RT | VK_SPACE | VK_ESCAPE | |

| | | |
|-----------------------|----------------------|---------------------|
| KEY_LOCATION_LEFT | KEY_LOCATION_RIGHT | KEY_LOCATION_NUMPAD |
| KEY_LOCATION_STANDARD | KEY_LOCATION_UNKNOWN | |

Focus Event Handlers

Focus events deal with focus moving between different components on a window. Opening windows, using tab to move around the screen, or clicking on components will trigger these events. Note that not all components can hold focus.

Events

| Events | Description |
|-------------|--|
| focusGained | This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it. |
| focusLost | This event occurs when a component that had the input focus lost it to another component. |

Event Object Properties

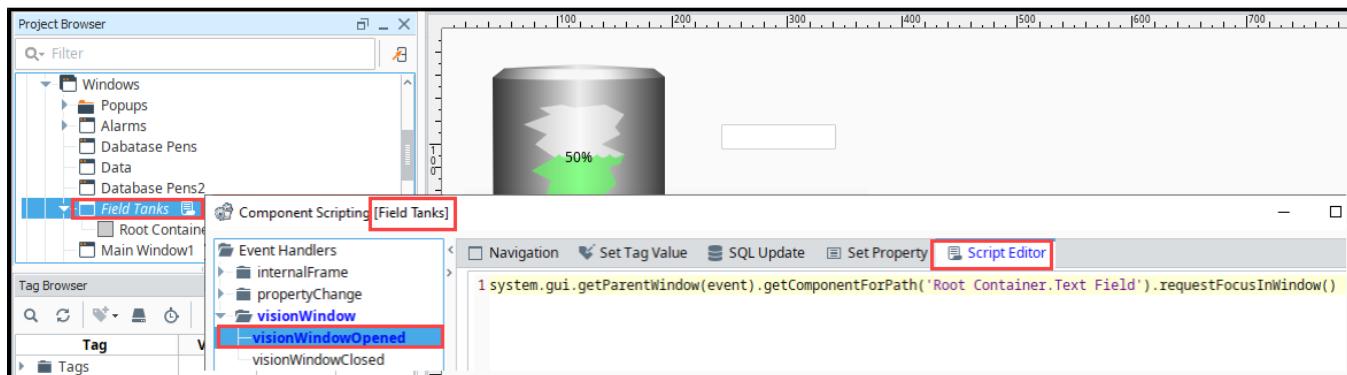
| Properties | Description |
|-------------------|--|
| source | The component that fired this event. |
| oppositeComponent | The other component involved in this focus change. That is, the component that lost focus in order for this one to gain it, or vice versa. |

Python - Printing on Focus Gained

```
# On the focusGained event of a few different components, this script can print out when the component has gained focus.
print "The component name now has focus!"
```

VisionWindow Event Handlers

The visionWindow events are specific to windows and not available elsewhere. Right click on the window name in the Project Browser and select Scripting to get access to these events. These events are triggered by a window opening or closing.



Events

| Events | Description |
|--------|-------------|
| | |

| | |
|--------------------|---|
| visionWindowOpened | This event is fired each time the window is opened and before any bindings are evaluated. |
| visionWindowClosed | This event is fired each time the window is closed. |

Event Object Properties

| Properties | Description |
|------------|--|
| source | The vision window that fired this event. |

Python - Grabbing Focus on Window Opened in Two Different Ways

```
# From a visionWindowOpened event on a window, you can request the focus of components in the window, to start the focus on a component other than the upper left most component.

# Here we grab the reference to the component using the property selector on the upper right side of the script editor.
system.gui.getParentWindow(event).getComponentForPath('Root Container.Text Field').requestFocusInWindow()

# Here we can manually enter in the path to the component using our knowledge of the component hierarchy and # the getRootContainer function. Both of these functions work in the same way.
system.gui.getParentWindow(event).getRootContainer().getComponent("Text Field").requestFocusInWindow()
```

InternalFrame Event Handlers

The internalFrame events are fired by windows: windows are known as "internal frames" in the underlying Java windowing system that the Vision component uses. Note that the source of these events is the window itself, just like the visionWindow events above.

Events

| Events | Descriptions |
|--------------------------|--|
| internalFrameActivated | Fires whenever the window is shown or focused. If you want a script to fire every time a window is opened, use this event. |
| internalFrameClosed | Fires when a window is closed. |
| internalFrameClosing | Fires right before a window is closed. |
| internalFrameDeactivated | Fires when a window loses focus. |
| internalFrameOpened | Fires the first time a window is opened. Note that when windows are closed and cached, next time they are opened this event will not be fired. Use internalFrameActivated instead. |

Event Object Properties

| Properties | Description |
|------------|---|
| source | The window that fired this event. Use source.rootContainer to get the root container. |

Python - Printing on Frame Activation

```
# From the internalFrameActivated event on a window, this will fire each time the window is focused, so clicking between two different windows will trigger it.
print "This window is now in focus!"
```

Cell Event Handlers

The cell event is unique in that it only appears on the [Table](#) component. It will trigger when something within a cell changes, and once for each cell changed.

Events

| Events | Description |
|------------|---|
| cellEdited | This event is fired when one of the cells in a table component has been modified. |

Event Object Properties

| Properties | Description |
|------------|---|
| source | The table component that fired this event. |
| oldValue | The old value in the cell that changed. |
| newValue | The new value in the cell that changed. |
| row | The row of the dataset this cell represents. |
| column | The column of the dataset this cell represents. |

Pseudocode - Updating a Database Table

```
# From the cellEdited event of a table component, this script can update our database table with any new
# data that is entered in the table

# Get the id of the row we edited and the headers
id = event.source.data.getValueAt(event.row, 'id')
headers = system.dataset.getColumnHeaders(event.source.data)

# Build our Update query.
query = "UPDATE User SET %s = ? WHERE id = ?" % (headers[event.column])
args = [event.newValue, id]

# Run the query with the specified arguments.
system.db.runPrepUpdate(query, args)
```

Item Event Handlers

The item event is unique in that it only appears on components that can be "on" or "off", such as with [Radio Buttons](#), [Check Boxes](#), and [Toggle Buttons](#).

Events

| Events | Description |
|------------------|--|
| itemStateChanged | This event fires when the state of the component changed. This event is the best way to respond to the state of that component changing. |

Event Object Properties

| Properties | Description |
|-------------|--|
| source | The component that fired this event. |
| stateChange | An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is. |
| SELECTED | The constant that the stateChange property will be equal to if this event represents a selection. |
| DESELECTED | The constant that the stateChange property will be equal to if this event represents a de-selection. |

Python - Printing on a Radio Button Selection

```
# On the itemStateChanged event of a radio button, this will print when this specific radio button is selected.  
if event.stateChange == event.SELECTED:  
    print "This radio button is selected!"
```

Paint Event Handlers

The paint event is only found on the [Paintable Canvas](#) component, and is used to customize how the component gets painted. This event requires a heavy knowledge of programming using the Java 2D drawing tools, but there is code for a pump shape each time you add a new Paintable Canvas to a window.

Events

| Events | Description |
|---------|--|
| repaint | This event will fire whenever the component needs to repaint itself. It will repaint when any of its custom properties change, or when .repaint() is called on it. When a Paintable Canvas is first dragged onto the screen, the repaint event handler will be filled with an example that draws out a pump. |

Event Object Properties

| Properties | Description |
|------------|---|
| source | The Paintable Canvas component that fired this event. |
| graphics | An instance of java.awt.Graphics2D that can be used to paint this component. The point (0,0) is located at the upper left of the component. |
| width | The width of the paintable area of the component. This takes into account the component's border. |
| height | The height of the paintable area of the component. This takes into account the component's border. |

Python - Painting a Circle

```
# On the repaint event of a paintable canvas component, this will create a circle with a gradient background color of orange and white.  
  
from java.awt import Color  
from java.awt import GradientPaint  
from java.awt.geom import Ellipse2D  
  
g = event.graphics  
  
# Body  
innerBody = Ellipse2D.Float(8,8,72,72)  
  
#### Scale graphics to actual component size  
dX = (event.width-1)/100.0  
dY = (event.height-1)/100.0  
g.scale(dX,dY)  
  
# Paint body  
g.setPaint(GradientPaint(0,40,Color.WHITE, 0,100, Color.ORANGE,1))  
g.fill(innerBody)  
g.setColor(Color.ORANGE)  
g.draw(innerBody)
```

Extension Functions

What Are Extension Functions

Extension Functions are found on the component scripting window of certain components, and they allow for more advanced customization of the component using scripting. These functions are generally more advanced and require a better understanding of Python. Unlike [Event Handlers](#), Extension Functions are not driven by a specific event, but are instead called by the component itself for a specific purpose when appropriate. This may be when the component first loads in the window, or whenever the function receives new input.

From an object-oriented point of view, Extension Functions create a custom "subclass" of the base component type. Your subclass can then override and implement parts of the functionality of the component itself, in Python. Following Python object-oriented methodology, each extension function's first argument is called `self`. That is because these are methods that belong to the component's class itself, instance methods. The value of `self` will always be the component itself. Notice that this is different than Event Handler scripts where you are given an `event` object in your scope and the component that fired the event is under `event.source`. When you write an Extension Function, there is no `event` object so the component is given to you as the `self` object instead.

Each component Extension Function comes with its own documentation built-into the function's default implementation using a standard Python "doc-string". You will find that you are unable to edit the function's signature or docstring. Changing the method's signature (arguments or function name) would prevent the component from calling it correctly. Changing the docstring could be misleading or confusing as you'd lose the documentation for how your implementation of the function should work.

On this page ...

- [What Are Extension Functions](#)
- [Using Extension Functions](#)
- [Example - User Management Component](#)
- [Example - Table Component](#)
- [Example - Power Table Component](#)
- [Example - Ad Hoc Charting](#)



INDUCTIVE
UNIVERSIT

Component Extension Functions

[Watch the Video](#)

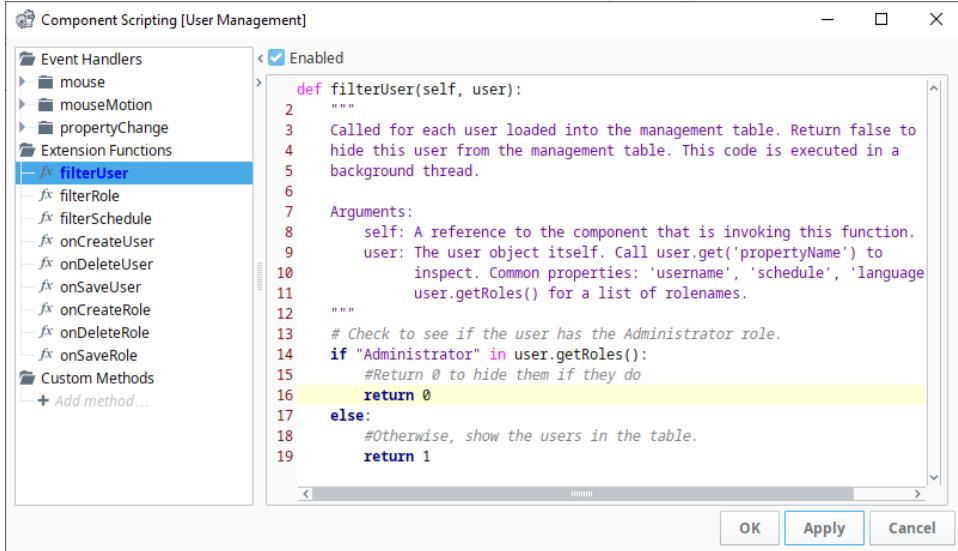
Using Extension Functions

Using an Extension Function works much like using an Event Handler. First select and **Enable** the Extension Function within the component scripting window, and then add in a script. The script will then automatically run when called.

Example - User Management Component

The [User Management](#) component has many Extension Functions that provide a way to customize how the component works. The `filterUser()` extension function is useful for filtering out users you don't want to see in the user source, preventing users from editing that user in the client. We can add a simple script to the `filterUser()` Extension Function that will hide the user from the list if they have the Administrator role.

1. Drag a **User Management** component in to your Designer workspace, and right click on **Scripting**.
2. Under the Extension Functions folder select `filterUser` and click **Enabled**.
3. Copy the code from the code block below and add it to bottom of the script and click **OK**.



Python - User Management Filter User

```

# Check to see if the user has the Administrator role.
if "Administrator" in user.getRoles():
    # Return 0 to hide them if they do.
    return 0
else:
    # Otherwise, show the user in the table.
    return 1

```

- By enabling this script, we now only see the users without the Administrator role in the list of users.

| Username | Name | Roles | Contact Info | Schedule |
|----------|-------------|----------------------|--------------------------|----------|
| operator | Greg Peters | Operator, Supervisor | email: operator@ai.com | Always |
| Jane | Jane Dobson | Operator | email: operator@induc... | Always |
| Sara | Sara Jones | Supervisor | email: supervisor@ind... | Always |

| Role name | # of Members |
|------------------|--------------|
| Administrator | 4 |
| Operator | 5 |
| Supervisor | 5 |
| Upper Management | 1 |

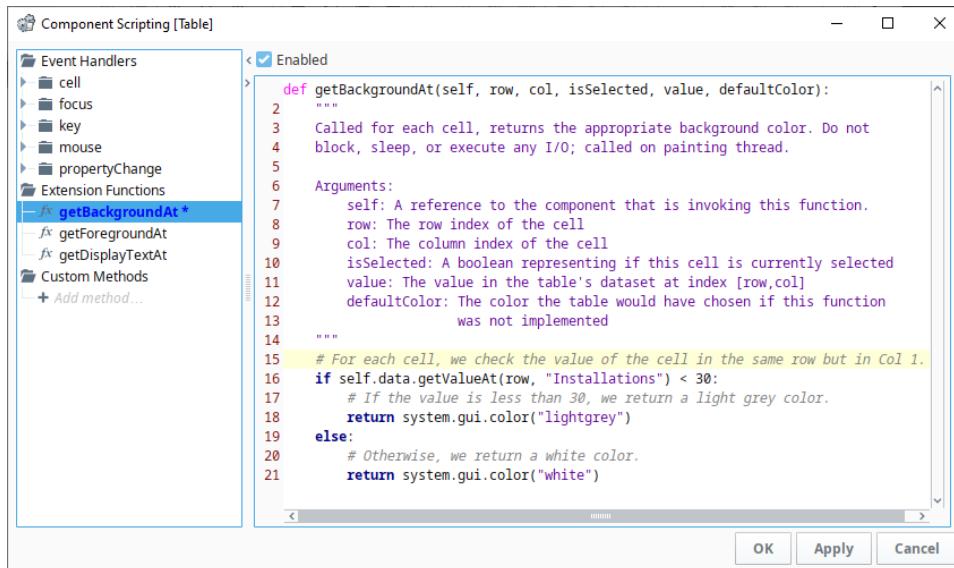
Example - Table Component

The **Table** component exposes an Extension Function called `getBackgroundAt()`. By implementing this function, you can control the background color of each cell of the table component using scripting. Starting with a Table component with some test data, then add the following script to the `getBackgroundAt()` Extension Function.

Here is our sample data. We want to have a clearer indication of what companies have fewer than 30 installations. So, we'll write a script to make those rows have a light grey background color.

| Installations | id | cityName | State | companyName |
|---------------|----|---------------|------------|--------------------|
| 34 | 1 | Sacramento | California | ABC Company |
| 62 | 2 | Phoenix | Arizona | Acme Bird Seed |
| 14 | 3 | Denver | Colorado | High Rocks Ente... |
| 87 | 4 | Omaha | Nebraska | Corners Inc. |
| 13 | 5 | San Francisco | California | Haight Jewelry |
| 99 | 6 | San Antonio | Texas | Ten Gallon Foods |
| 15 | 7 | Chico | California | XYZ Brewing |
| 74 | 8 | Portland | Oregon | City Books |
| 23 | 9 | Iowa City | Iowa | Best Syrups Inc |
| 15 | 10 | San Rafael | California | Retro Redwood |

1. Right click on the **Table** component and choose **Scripting**.
2. Choose the **getBackgroundAt*** extension function.
3. Select the **Enabled** check box.
4. Copy the code from the code block below and add it to bottom of the script and click **OK**.



Python - Table Row Color

```

# For each cell, we check the value of the cell in the same row but in Col 1.
if self.data.getValueAt(row, "Col 1") < 30:
    # If the value is less than 30, we return a light grey color.
    return system.gui.color("lightgrey")
else:
    # Otherwise, we return a white color.
    return system.gui.color("white")

```

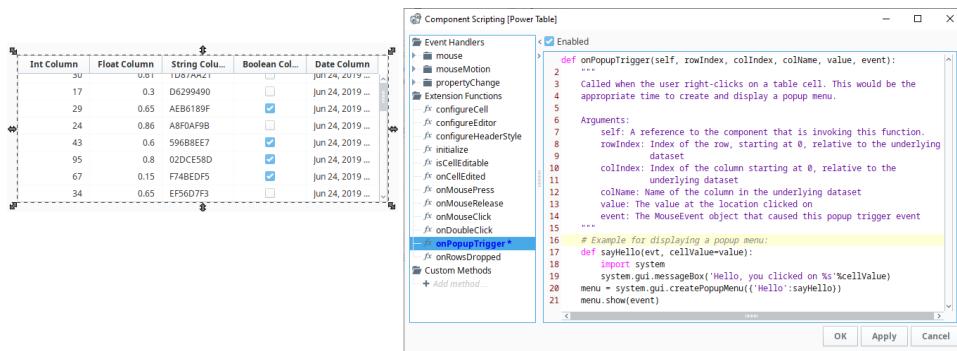
5. We should see the script run automatically, and the background color of the table will change.

| Installations | id | cityName | State | companyName |
|---------------|----|-----------------|------------|--------------------|
| | 34 | 1 Sacramento | California | ABC Company |
| | 62 | 2 Phoenix | Arizona | Acme Bird Seed |
| | 14 | 3 Denver | Colorado | High Rocks Ente... |
| | 87 | 4 Omaha | Nebraska | Corners Inc. |
| | 13 | 5 San Francisco | California | Haight Jewelry |
| | 99 | 6 San Antonio | Texas | Ten Gallon Foods |
| | 15 | 7 Chico | California | XYZ Brewing |
| | 74 | 8 Portland | Oregon | City Books |
| | 23 | 9 Iowa City | Iowa | Best Syrups Inc |
| | 15 | 10 San Rafael | California | Retro Redwood |

Example - Power Table Component

The **Power Table** component has several extension functions on it that change the way the table looks or behaves. One in particular, called **onPopupTrigger()**, makes it easy to implement a right click popup menu as it is called each time a user right clicks on a cell of the table. It can be used in conjunction with the **system.gui.createPopupMenu** function to create your own custom popup menu, as shown in the following example.

1. Drag a Power Table component on to your Designer workspace and set the **Test Data** property to '**true**' so you have some data to test on.
2. With the **Power Table** selected, click **Scripting**, and then click on the **onPopupTrigger** extension function.
3. Select the **Enabled** checkbox.
4. The **onPopupTrigger** extension function will have a pre-built example commented out in the extension function. Uncomment the lines of code to see it in action.



Python - Power Table Popup Trigger

```

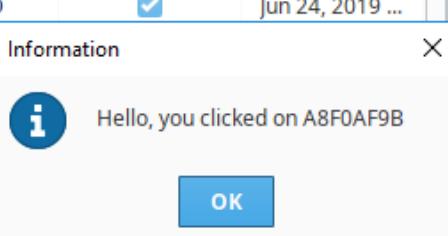
import system
def sayHello(evt, cellValue=value):
    import system
    system.gui.messageBox('Hello, you clicked on %s'%cellValue)
menu = system.gui.createPopupMenu({'Hello':sayHello})
menu.show(event)

```

5. Right click on a cell in the table and a "Hello" menu option will have a message box appear that has the value of the cell that was right clicked.

| Int Column | Float Column | String Colu... | Boolean Col... | Date Column |
|------------|--------------|----------------|-------------------------------------|------------------|
| 30 | 0.01 | TD87AAZT | <input type="checkbox"/> | Jun 24, 2019 ... |
| 17 | 0.3 | D6299490 | <input type="checkbox"/> | Jun 24, 2019 ... |
| 29 | 0.65 | AEB6189F | <input checked="" type="checkbox"/> | Jun 24, 2019 ... |
| 24 | 0.86 | A8F0AF9B | <input type="checkbox"/> | Jun 24, 2019 ... |
| 43 | 0.6 | 596B8EE7 | <input checked="" type="checkbox"/> | Jun 24, 2019 ... |
| 95 | 0.8 | 02DCE58D | <input checked="" type="checkbox"/> | Jun 24, 2019 ... |
| 67 | 0.15 | F74BEDF5 | <input type="checkbox"/> | Jun 24, 2019 ... |
| 34 | 0.65 | EF56D7F3 | <input type="checkbox"/> | Jun 24, 2019 ... |

Hello



Example - Ad Hoc Charting

The Easy Chart component has an Extension Function to allow scripting when a Tag is dropped onto it (see [Using the Tag Browse Tree for Charting](#)). There is a lot of customization possible in the Designer, but any client side changes to the Tag Pens dataset must be done here. Generally, people want to change what axis and subplot a pen goes into based on some other information. Below is a simple example that uses the Tag's name to determine this. For this example to work, you need to have two subplots and a second axis named "HOA".

Component Scripting [Table]

Enabled

```

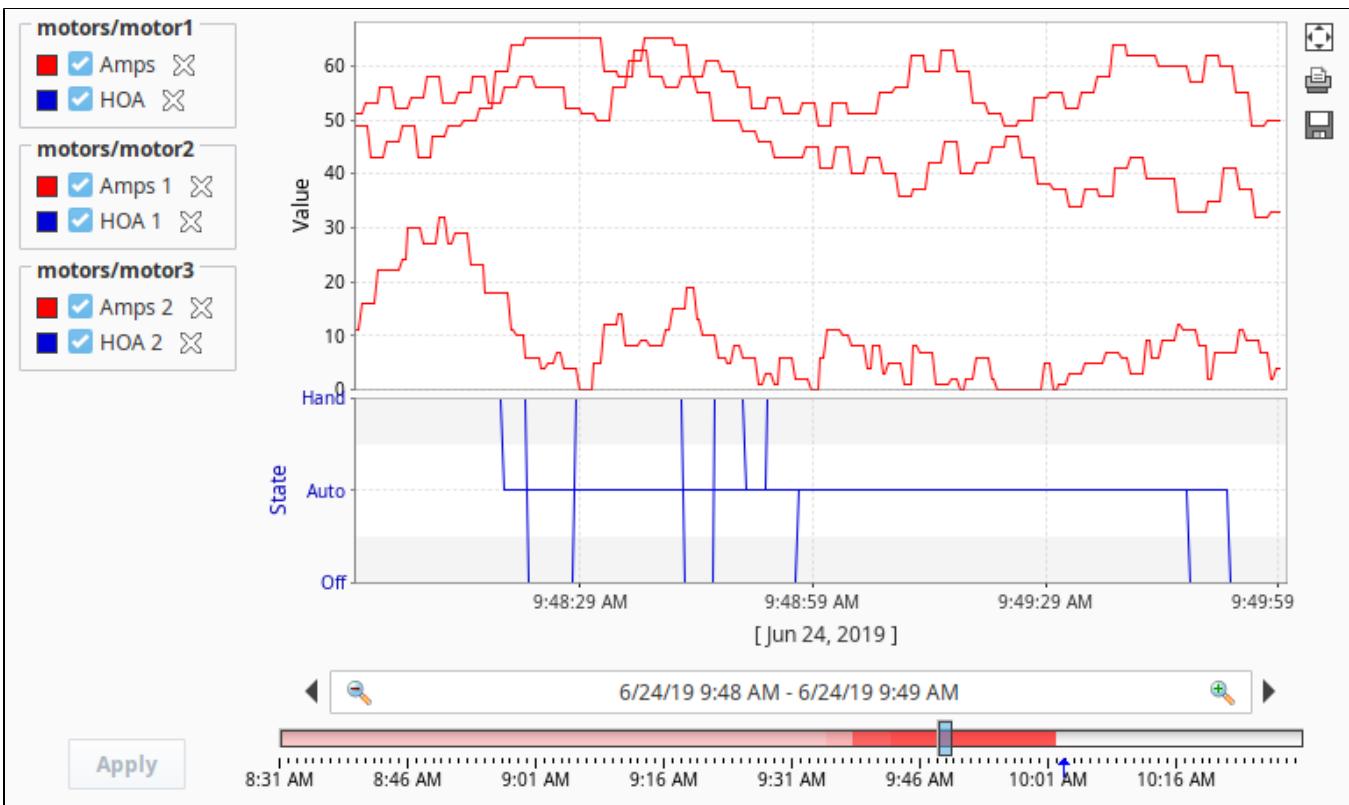
<input checked="" type="checkbox"/> Enabled

def getBackgroundAt(self, row, col, isSelected, value, defaultColor):
    """
    Called for each cell, returns the appropriate background color. Do not
    block, sleep, or execute any I/O; called on painting thread.

    Arguments:
        self: A reference to the component that is invoking this function.
        row: The row index of the cell
        col: The column index of the cell
        isSelected: A boolean representing if this cell is currently selected
        value: The value in the table's dataset at index [row,col]
        defaultColor: The color the table would have chosen if this function
                      was not implemented
    """
    # For each cell, we check the value of the cell in the same row but in Col 1.
    if self.data.getValueAt(row, "Installations") < 30:
        # If the value is less than 30, we return a light grey color.
        return system.gui.color("lightgrey")
    else:
        # Otherwise, we return a white color.
        return system.gui.color("white")

```

OK Apply Cancel



Python - Drop Tags on Easy Chart

```

# Alter chart configuration when dropping pens
# sample data for the Tag Pens property:
#"NAME", "TAG_PATH", "AGGREGATION_MODE", "AXIS", "SUBPLOT", "ENABLED", "COLOR", "DASH_PATTERN", "RENDER_STYLE", "
LINE_WEIGHT", "SHAPE", "FILL_SHAPE", "LABELS", "GROUP_NAME", "DIGITAL", "OVERRIDE_AUTOCOLOR", "HIDDEN", "
USER_SELECTABLE", "SORT_ORDER", "USER_Removable"
#"HOA", "[~]Motors/Motor 1/HOA", "MinMax", "HOA", "2", "true", "color(85,255,85,255)", "", "1", "1.0", "0", "true", "
false", "", "false", "false", "true", "true"

# get old pen data and append new info
oldData = system.dataset.toPyDataSet(self.tagPens)
# get new info
for fullTagPath in paths:
    # get names for everything in the tag path
    lastSlashIndex = fullTagPath.rfind("/")
    closeBracketIndex = fullTagPath.find("]")
    tagName = fullTagPath[lastSlashIndex+1:]
    tagPath = fullTagPath[closeBracketIndex+1:]
    groupName = fullTagPath[closeBracketIndex+1:lastSlashIndex]

    # find which tags are named "hoa" and put them in the HOA subplot.
    if tagName.lower() == "hoa":
        axis = "HOA"
        subplot = 2
        color = "color(255,85,85,255)" #red
        digital = "true"
    else:
        axis = "Default Axis"
        subplot = 1
        color = "color(85,85,255,255)" #blue
        digital = "false"

    # append to the old pen data
    newData = system.dataset.addRow(oldData, [tagName,tagPath,"MinMax",axis,subplot,"true",color,"","1",
1.0,"0","true","false",groupName,digital,"false","false","true","","true"])

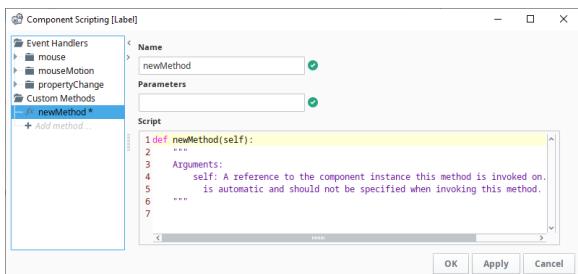
```

```
# push new pens back to the tagPens property  
self.tagPens = newData
```

Custom Component Methods

Custom Methods

Custom Methods function much like [Project Library](#) in that you write a script and call it from somewhere else. However, Custom Methods are written on a component instead of a separate scripting section, and are also automatically passed the value of `self`, just like an [Extension Function](#). The `self` object provides the script with easy access to everything within the window. In addition to `self`, Parameters can be added that you can use to pass in other objects into your Custom Method.



On this page ...

- [Custom Methods](#)
 - [Sample Custom Method](#)



INDUCTIVE
UNIVERSIT

Component Custom Methods

[Watch the Video](#)

Custom Methods can then be called from the same component or from other components. Custom Methods are called just like any other method on a component: `.methodName()`. So if I had a custom method on a text field, and I wanted to call it from the `actionPerformed` event on a button in the same container, I would use:

Python - Custom Method

```
# The name of the custom method in this instance is myCustomMethod.
event.source.parent.getComponent("Text Field").myCustomMethod()
```

Templates are another good use for custom methods. By adding a custom method directly to a template, all the components that make up the template can call the custom method from the template itself. Another advantage of templates is, in the event you want to share your template with another project, your custom methods (scripts) would not have to be exported separately like they would as if they were in a Script Library. When you export the template, the custom methods are included in the export automatically.

Sample Custom Method

A great use for Custom Methods is checking for valid input on a form with a lot of text fields. Instead of checking every text field all within a script on a button press, we can instead build a value check script on each text field that is unique to that Text Field's specific type of input. This keeps each script organized on the appropriate Text Field component. Take this sample code that can go into a custom method on a text field which checks for a valid email address using :

Python - Text Field Input Validation

```
# First, we need to import the Regular Expression library
import re

# We need to grab the value of the text field.
text = self.text

# Here, we put together our regular expression for email addresses.
validAddress = re.compile(r"^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+\$")

# Check the text against the regular expression.
# If the text does not fit in with the regular expression, it returns a value of None, which we use to show
# an error box.
if validAddress.match(text) is None:
    system.gui.errorBox("Please enter a valid email address!", "Email Not Valid")
```

A similar script can be repeated for each Text Field component, but modified to fit the expected input. Then, the another script can be on a button that simply calls each of the Text Field's Custom Methods.

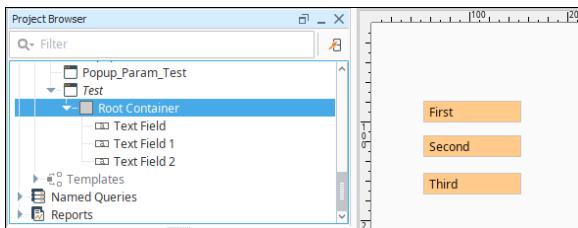
Python - Validate Input on Button Press

```
event.source.parent.getComponent("Email Field").validInputCheck()
```

Focus Manipulation

Focus Order

How components are laid out on a window determines the focus order. When tabbing through components on a window, the focus moves from left to right, then top to bottom. Focus will then cycle back to the first component. When determining order, the top left corner of the component is used. To see for yourself, drag several components into your window. Go to **Preview Mode**, and tab through the components.

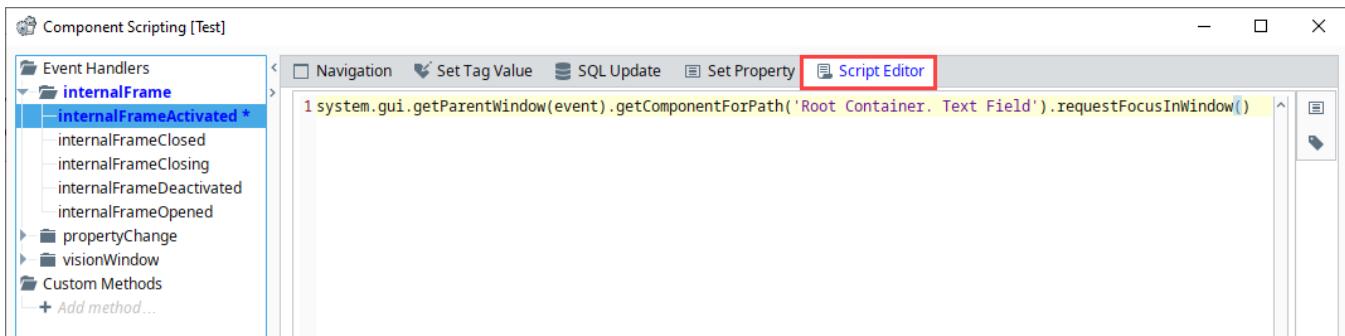


On this page ...

- [Focus Order](#)
- [Requesting Focus in the Window](#)
 - [The getComponentForPath Function](#)
- [Controlling Focus Between Components in a Window](#)
 - [About Invoke Later](#)
 - [Text Areas and Focus Requests](#)

Requesting Focus in the Window

You can programmatically request that focus be given to a component by calling the function `requestFocusInWindow()` on that function. This function is called on a component and will pull the focus to that component so it is selected and ready to use. It is best used with an input component, such as a Text Field, so the user can immediately begin typing into the component. You can use it on the `internalFrameActivated` event to bring focus to the component right when the window opens.



The getComponentForPath Function

The example above references the function **getComponentForPath**. This function can be called from a window object, and allows you to specify the full path to a component inside of the window as a single string, using the following format:

Pseudocode - Get Component For Path

```
getComponentForPath('Root Container.ComponentName')
```

When referencing a component from a window event handler, such as `internalFrameActivated`, clicking the **Property Reference** icon will use the `getComponentForPath` function. While this function is useful, you never have to use the `getComponentForPath` function. Instead you can use the component paths that are seen from event handlers on other components. Below is a comparison of using both `getComponentForPath`, as well as the more traditional

Pseudocode - Component Hierarchy from a Window

```
# Both lines below will return a reference to the root container in the window that this script originates from.

print system.gui.getParentWindow(event).getComponentForPath('Root Container')
print system.gui.getParentWindow(event).getRootContainer()
```

```
# These lines will both reference the text property on a Label that is nested in a Group on the window.

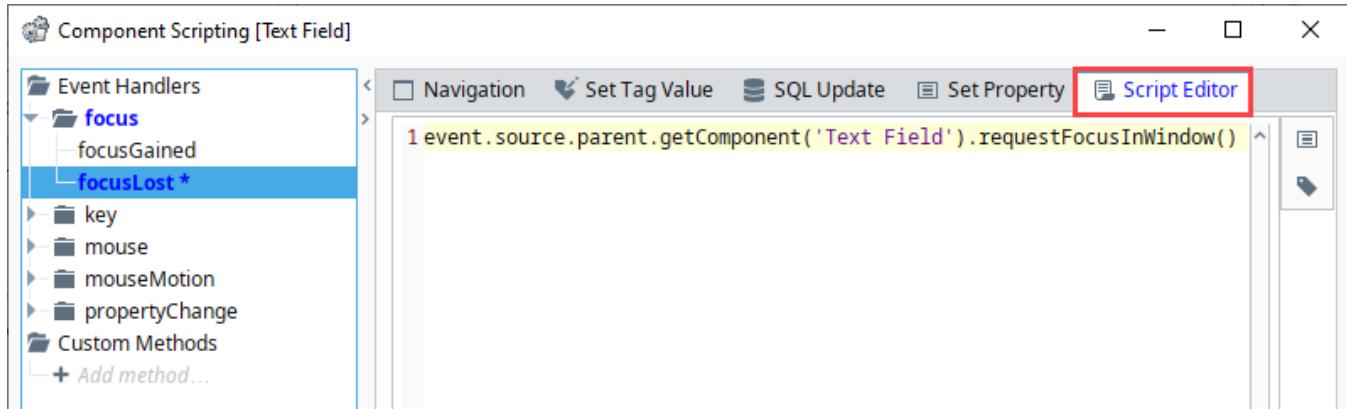
print system.gui.getParentWindow(event).getComponentForPath('Root Container.Group.Label').text
print system.gui.getParentWindow(event).getRootContainer().getComponent('Group').getComponent('Label').text
```

Python - Requesting Focus on Frame Activation

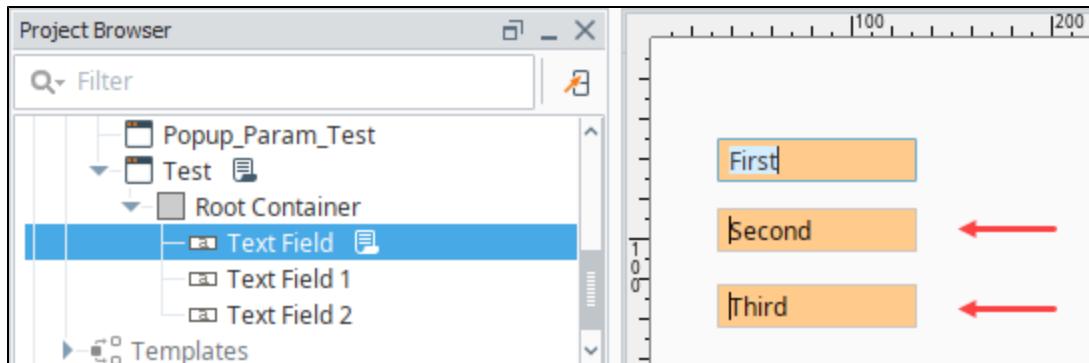
```
system.gui.getParentWindow(event).getComponentForPath('Root Container.Text Field').requestFocusInWindow()
```

Controlling Focus Between Components in a Window

In some cases, you may wish to control which component gains focus when the user clicks on a different component, or tabs away, instead of using the default focus order. You can call `requestFocusInWindow()` from the **focusLost** event to control which component should gain focus next.



However, calling `requestFocusInWindow()` may cause some irregular behavior as shown below. Notice below how the "second" and "third" Text Fields both have a text cursor in the component, but the "first" text field has focus. This is because `requestFocusInWindow()` is being called on the `focusLost` event, which runs when one of our components loses focus. This means that while focus is being pulled to one component (the "second" Text Field), our script changes focus again to a different component.



The solution to the problem above is to have the `requestFocusInWindow()` call occur as the last part of the event trigger. This can be accomplished in one of two ways: using **Invoke Later** under Advanced Settings, or the `invokeLater()` system function.

About Invoke Later

The concept of invoking some code later leads to a broader discussion on event handling and timing, which deviates from the purpose of this page: focus manipulation. The concept of "Invoke Later" simply means to wait for the current event to finish processing before running our `focusLost` script. In the scenario above, clicking from one component to another (or tabbing to a different component) natively calls a focus change event. A script on the `focusLost` event handler that uses `requestFocusInWindow` will also cause a focus change event, except it does so mid-execution of the native focus change event.

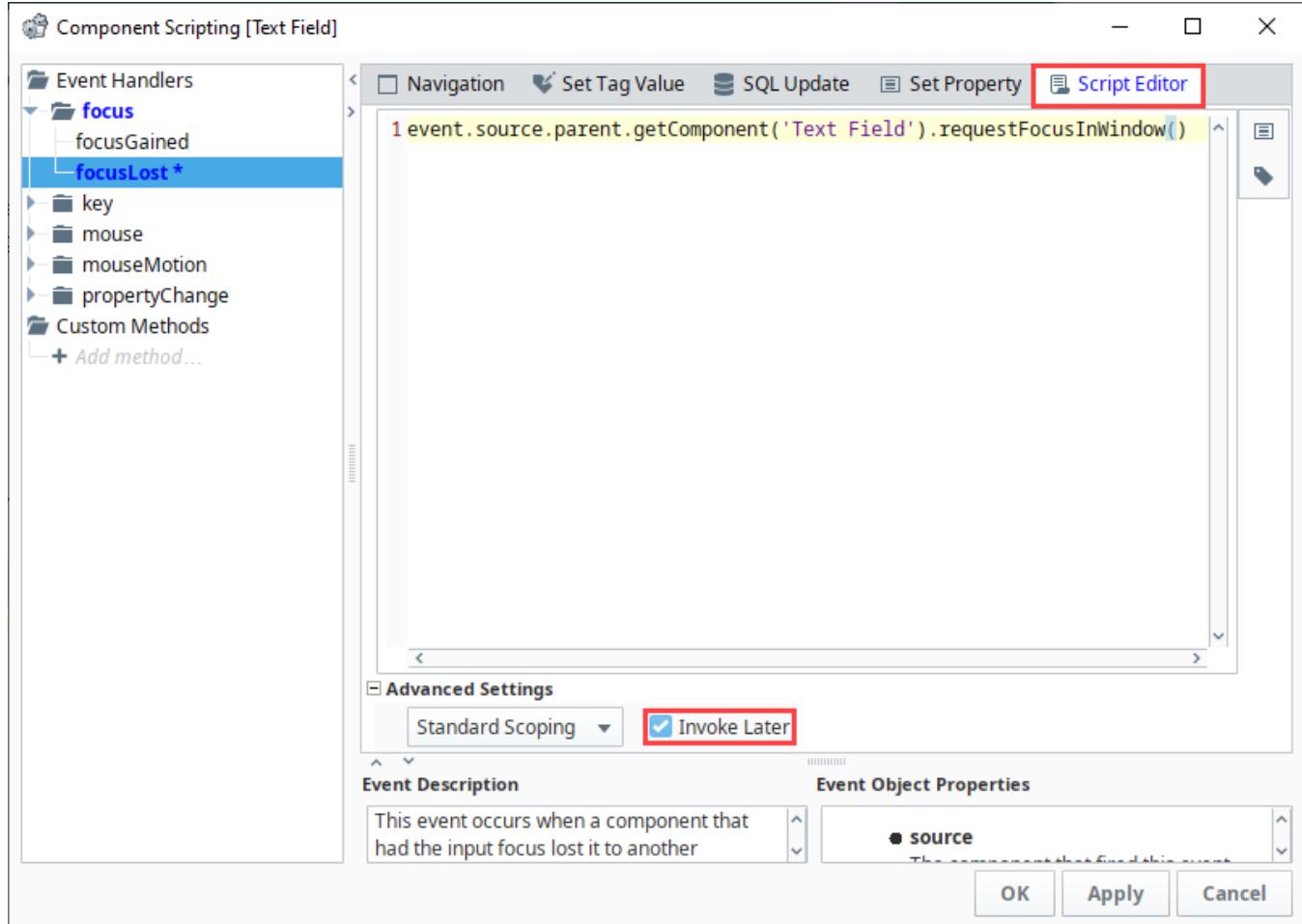
The main issue then is that focus is being moved to two components simultaneously from within the same event stack. The solution then is to have them occur in sequential order instead, with our custom `focusLost` script occurring last, hence using `Invoke Later`.

Running a script with one of the invoke later approaches mentioned here is not required on most scripts in Ignition, but it is common when the script is attempting to set or interact with focus on a window. On a related matter, invoking a script later is not the same as adding a delay mid execution. For more information about scripting a delay, see the [Adding a Delay to a Script](#) page.

To resolve the focus manipulation issue mentioned above, we can take one of two approaches:

Script Editor Advanced Settings

The simplest approach would be to use the **Invoke Later** option under Advanced Settings in the Script Editor.



Using the invokeLater System Function

Alternatively, we could address this by using [system.util.invokeLater](#) to request focus at the end of the event.

Python - Requesting Focus Later

```
# Create a function for invokeLater that requests focus.
def focus():
    event.source.parent.getComponent('Text Field').requestFocusInWindow()

# Call the function once the event has executed.
system.util.invokeLater(focus)
```

Text Areas and Focus Requests

[Text Area](#) components are slightly more complex when it comes to focus requests. Calling `requestFocusInWindow()` directly on the component will not allow the user to immediately start typing into the component. However, we can accomplish this by calling `getViewport()`, and then calling `getViewport().requestFocusInWindow()` first as shown in the code below.

Python - Requesting Focus from a Text Area

```
# Create a reference to the Text Area. This step could be skipped and combined with the line below, but is
# segregated in this example for clarity.
textArea = event.source.parent.getComponent('Text Area')

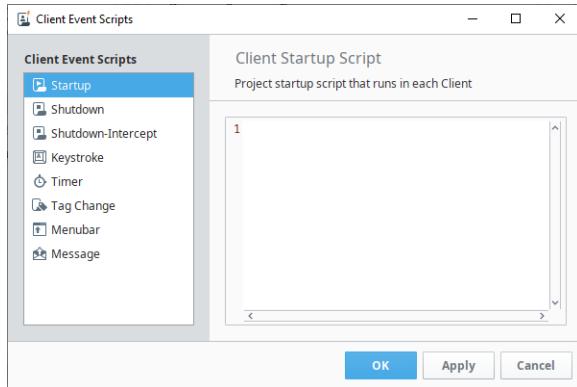
# This line demonstrates how to request focus on a Text Area
textArea.getViewport().getView().requestFocusInWindow()
```

Client Event Scripts

Client Event Scripts Overview

Client Event Scripts execute in the running Client, which means that they may never execute if no clients are running, or they may execute many times if multiple clients are running. Client Event Scripts will also execute in the Designer, but only in Preview Mode. Because Clients are full-fledged applications, Client Event Scripts run on the computer running the Client, not on the Gateway's host server computer. This means that they don't have access to the Gateway's file system, and so on.

Note: System functions are available for both Client Event Scripts and Gateway Event Scripts, but some system functions are specific to either one or the other. When you're writing event scripts, it's important to remember the scope of where you're writing the script: Client or Gateway. You can check [Scripting Functions](#) in the Appendix to see a list of all system functions, their descriptions, and what scope they run in.



On this page ...

- [Client Event Scripts Overview](#)
- [Startup Script](#)
- [Shutdown Script](#)
- [Shutdown-Intercept Script](#)
 - [Preventing Client Shutdown](#)
- [Keystroke Scripts](#)
 - [Client Keystroke Script Interface](#)
 - [Choose Keystroke Window](#)
- [Timer Scripts](#)
- [Tag Change Scripts](#)
- [Menubar Scripts](#)
 - [Menubar Script Interface](#)
- [Message Scripts](#)
 - [Client Message Handler Settings](#)
- [Troubleshooting Client Scripts](#)



Gateway vs Client Event Scripts

[Watch the Video](#)

Startup Script

These trigger when the user logs into the client. These scripts trigger before any windows in the project are opened, so they are ideal to use when you need to dynamically open certain windows based on which user logged on.

Configurations for Client Event Startup Scripts are similar to Gateway Event Startup scripts. See the [Gateway Event Scripts page](#) for more information

Shutdown Script

These trigger when the user closes the client, or logs out.

Configurations for Client Event Shutdown Scripts are similar to Gateway Event Shutdown scripts. See the [Gateway Event Scripts page](#) for more information

Shutdown-Intercept Script

The Shutdown-Intercept Script is unique in that it runs when a user attempts to shutdown a client, but before actual shutdown occurs. The main reason to use a Shutdown-Intercept script is to prevent the client from closing.

Preventing Client Shutdown

There is a special property on the event object inside the Shutdown-Intercept script that can be used to prevent the client from closing: simply type "event.cancel = 1;" somewhere in your code. Doing this will cancel the shutdown event and leave the client open. This allows you to set special

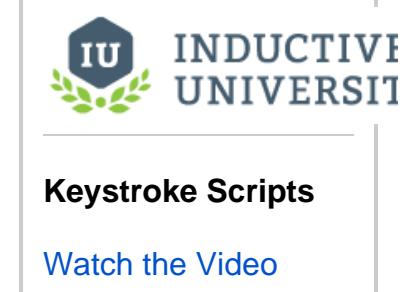
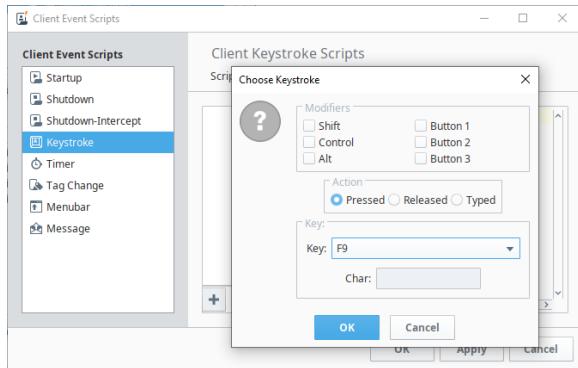
restrictions in regard to when the client is actually allowed to shut down, such as having a certain role, as seen in the example below:

Python - Cancel Application Exit

```
# Check to see if the user has a certain role.  
if "SuperUser" not in system.security.getRoles():  
    # If the role is not present, it will warn the user and cancel the shutdown process.  
    system.gui.warningBox("Only administrators are allowed to shutdown the client.")  
    event.cancel = 1
```

Keystroke Scripts

The Keystroke Scripts let you create different events that will activate on certain key combinations, allowing you to add keyboard shortcuts to your projects.



**INDUCTIVE
UNIVERSITY**

Keystroke Scripts

[Watch the Video](#)

Client Keystroke Script Interface

-  **Add Script** - Adds a new keystroke script.
-  **Delete Script** - Deletes the currently selected keystroke script.
-  **Script Settings** - Opens the **Choose Keystroke** window, allowing you to modify the keystroke that will trigger the script.

Choose Keystroke Window

The following areas are available on the Choose Keystroke window:

- **Modifiers** - Additional keys or mouse buttons that must be held to trigger your script. Modifiers are inclusive, so multiple modifiers must all be held down when the key is typed to trigger the script.
- **Action** - Similar to the **Key Event Handlers**, determines what action must occur to the key to trigger the script:
 - **Pressed** means the key was pressed,
 - **Released** means the key was released. When used in conjunction with a modifier, this action provides the user a means to prevent the script from happening after the key has already been pressed: if the user releases the modifier before releasing the key, then the script will not trigger.
 - **Typed** means the user typed a specific character. Selecting this action enabled the **Char** field under the key section. This provides an easier way to trigger the script based on non-standard ascii characters.
- **Key**- Which key will trigger the script. A dropdown is available when the Action is set to **Pressed** or **Released**. A Text Field is available if the Action is set to **Typed**.

Special keys like the Function keys (F1) or ESC key are only available in the pressed and released actions.



Some operating systems reserve certain keys for certain functions, and will capture the key press or release before it gets sent to the Client. For example, many operating systems use the TAB key to shift focus to the next field.

Timer Scripts

Run on a timer in the same fashion as their Gateway counterpart, except each instance of the project (i.e., client looking at the project containing the timer script) has a separate instance of the timer script running. Timer scripts that insert records into a database, or write to a Tag, are better suited as Gateway Event Scripts, since there will only ever be one running. If there are not any open clients, there will not be an instances of this script running.

Configurations for Client Event Timer Scripts are similar to Gateway Event Timer scripts. See the [Gateway Event Scripts page](#) for more information

Tag Change Scripts

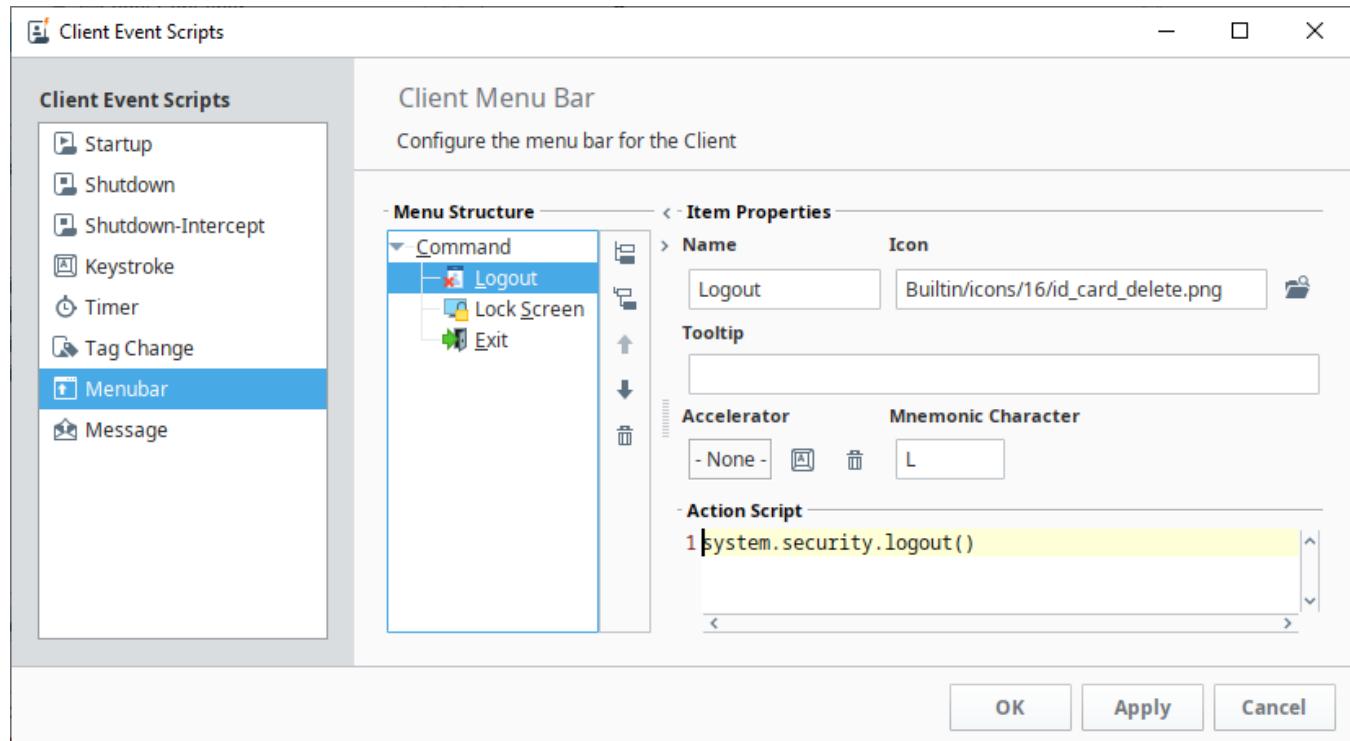
Monitor one or more Tags, and trigger a script in each instance of the client on Tag change. Unlike the Gateway Tag Change Script, Client Tag Change Scripts can monitor a Client Tag. Much like Timer Scripts, they only run in each instance of the client, so if there aren't any open clients, then the script will never execute.

Configurations for Client Event Tag Change Scripts are similar to Gateway Event Tag Change scripts. See the [Gateway Event Scripts page](#) for more information

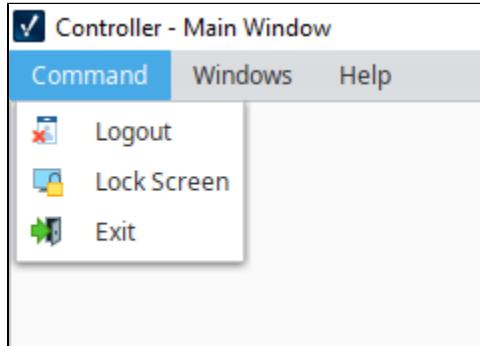
Menubar Scripts

The Client Menubar Scripts create and control the options available in the menubar of the Client. As such, these scripts are only available in the Client Scope. By default, a Client will have three menus: Command, Windows, and Help. The Windows and Help Menu are separate, and controlled through the project properties, but the Command menu is actually created in the Client Menubar Scripts.

Below we see a configured **Menu Structure** list.



Next we see the results of the structure in the client.



A screenshot of a website for "INDUCTIVE UNIVERSITY". The header features the university's logo (IU) and name. Below the header is a "Menu Bar" section containing a "Watch the Video" link. Other menu items are visible but partially cut off.

Menubar Script Interface

The user interface on the Menubar script is divided into two sections.

Menu Structure

A tree representing the layout of the menu bar. Items at the root of the tree will appear on the menu bar in the client, and nested items will appear as subitems in the client. In the image of the Client Event Script above, notice that the **Logout**, **Lock Screen**, and **Exit** items are children of the **Command** item. When looking at the image of the menubar in the Client, these three items appear under the Command Item.

Because the Menu Structure ultimately impacts the order, the following buttons are available to help sort each item.

- **Add Sibling** - Adds a new sibling, or peer item, to the selected item.
- **Add Child** - Adds a new child to the selected item.
- **Move Up** and **Move Down** - Moves the selected item up or down in the list. The order in the list determines the order that items appear in the menu, so these buttons can be used to group meaningful items together.
- **Delete** - Deletes the selected item, removing it from the menu.

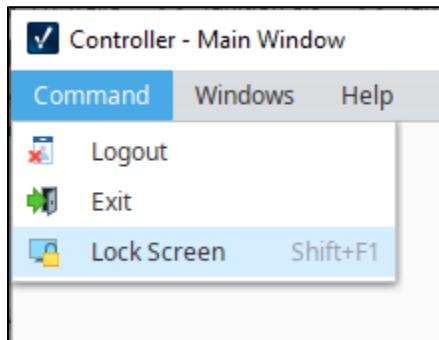
Item Properties

- **Name** - The text on the item in the menu.
- **Icon** - What image should appear next to the item, if any.
- **Tooltip** - Optional property allowing you to specify a tooltip when the user hovers the mouse cursor on top of the item.
- **Accelerator** - Allows you to define a keyboard shortcut that will quickly select the item. Please see the Accelerator section below for more details.
- **Mnemonic Character** - Allows you to define a character key that will trigger the option when the menu is open. Please see the Mnemonics section below for more details.
- **Action Script** - The script that will run when the user selects the item. Every item, even those at the root and branches may have a script defined.

Note: It is uncommon to have a script defined on a branch, as they usually act as a means to list other items.

Accelerators

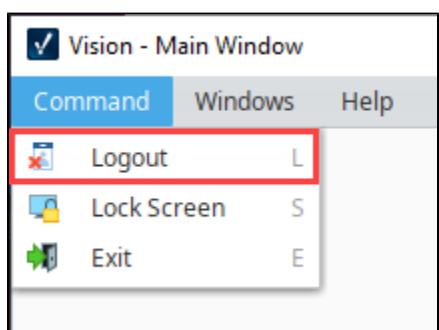
An accelerator is a key or key combination that can be pressed at any time in the client to initiate that menu item's event. If an accelerator has been configured for an item, then it will be listed on the menu in the client. Below we see our initial menu bar has been modified with the accelerator **Shift+F1**. Now the Lock Screen item may be called anywhere in the client by holding **Shift** and pressing the **F1** key.



Mnemonics

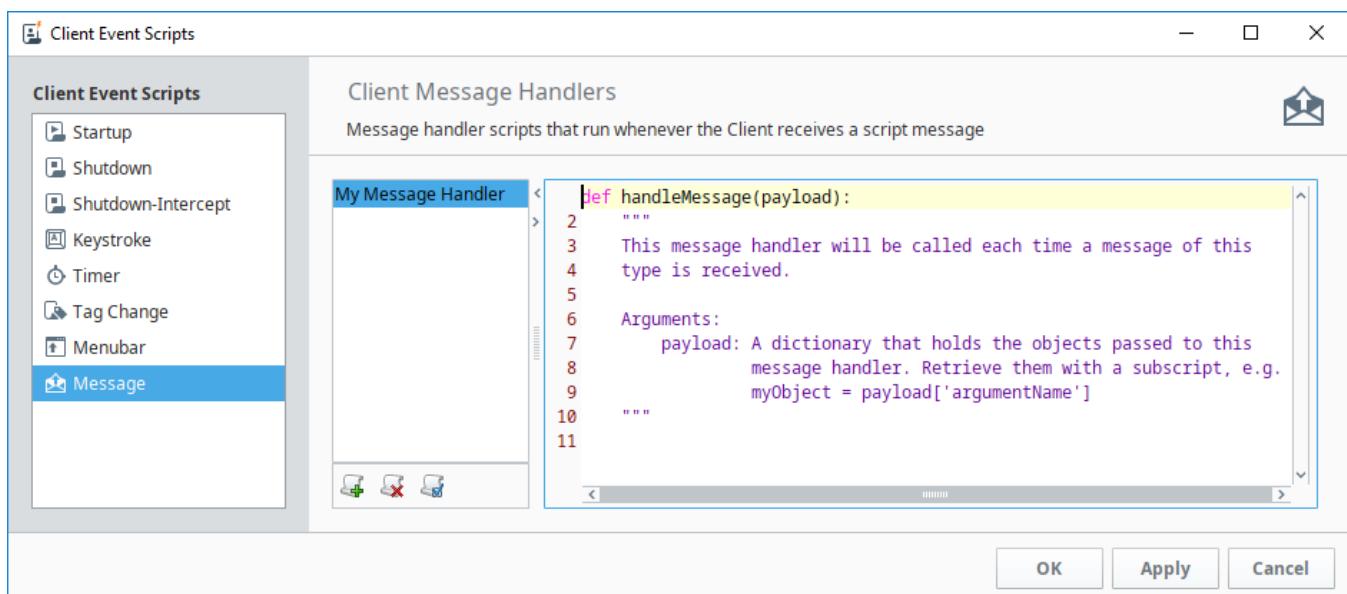
The mnemonic character is a key that can be pressed when the menu is opened. This is functionally similar to an Accelerator, in that it allows the user to select an item in the menu without clicking on it. However, mnemonics differ in that they only call an item when the menu is open, and the item is visible on the screen.

Users can identify mnemonics by a character to the right of the command in the menu. In the image below, we see that the **Logout** item has an "L" character. This means the user can now press the "L" key to select the Logout item. However, this will not work unless the menu is open, so if the user accidentally presses the L key while the menu is closed, the script will not trigger. Notice, how each command also has a mnemonic character defined for each command.



Message Scripts

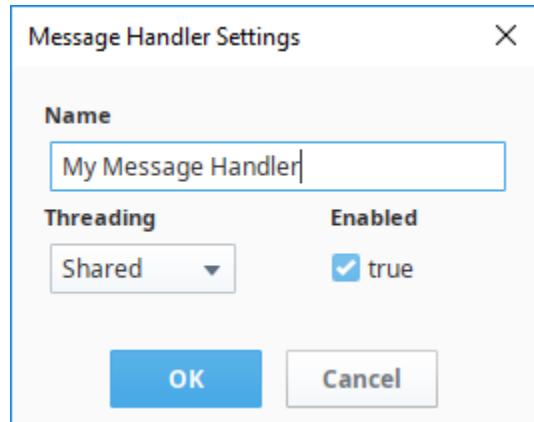
Client Message Handlers are created and called using the same mechanisms as [Gateway Event Scripts](#). There are two main differences that make Client Message Handlers stand out from Gateway Message Handlers: they run in the Client, and they have different settings.



Script Messaging

[Watch the Video](#)

Client Message Handler Settings



Client Message Handlers have the following settings:

- **Name** - The name of the message handler. Each message handler must have a unique name per project.
- **Threading** - Determines the threading for the message handler. Contains the following options:
 - **Shared** - The default way of running a message handler. Will execute the handler on a shared pool of threads in the order that they are invoked. If too many message handlers are called all at once and they take long periods of time to execute, there may be delays before each message handler gets to execute.
 - **Dedicated** - The message handler will run on its own dedicated thread. This is useful when a message handler will take a long time to execute, so that it does not hinder the execution of other message handlers. Threads have a bit of overhead, so this option uses more of the Gateway's resources, but is desirable if you want the message handler to not be impeded by the execution of other message handlers.
 - **EDT** - This will run the message handler on the **Event Dispatch Thread** (EDT) which also updates the GUI. If a message handler were to take a long time to execute, it would block the GUI from running which may lock up your client. This is helpful when your message handler will be interacting with the GUI in some way, as the GUI will not be able to update until the message handler finishes.

For more information on Message Handlers, such as working with the Payload argument, or calling them, please see the [Gateway Event Scripts](#) page.

Troubleshooting Client Scripts

The Console is very a important tool in Ignition for troubleshooting Client scripts. You can check to see if your script is working directly from the Client window, or the Designer while in Preview Mode. Any client scripting errors along with printouts go to the Console. The Console will identify the script name, error message, what line the script error is in, and a description of the problem.

To access the Console from a Client, go to the menubar and select **Help > Diagnostics > Console**. To access the Console from Preview Mode in the Designer, go to the menubar **Tools > Console**.

Related Topics ...

- [Gateway Event Scripts](#)

Read a Cell from a Table

After data has been populated in a Table or Power Table component, it may be useful to read or extract a particular cell from the Table, especially if users can select rows in the Table. On this page, we'll take a look at how to retrieve information from a particular cell on a Table.

The example on this page utilizes a simple Power Table and Button component. Users select a row from the table to extract one of the cells from the highlighted row in the Power Table and press the Sign-In button.

| Mechanic_ID | Mechanic_Name |
|-------------|---------------|
| 1 | Waylon |
| 2 | Monty |
| 3 | Kurt |
| 4 | Lewis |

On this page ...

- [Power Table Example](#)
 - Retrieve a single cell requires that we reference the Power Table a couple of times in the same script. Because of this, we can create a variable that references the table, and use the variable later on. Be aware that a Button and Power Table can be in the same container or separate containers. If your Button and Power Table are in separate containers your path may differ.
 - [Selecting a Single Cell](#)
 - [Selecting Multiple Cells - Same Column](#)
 - [Test Your Script](#)

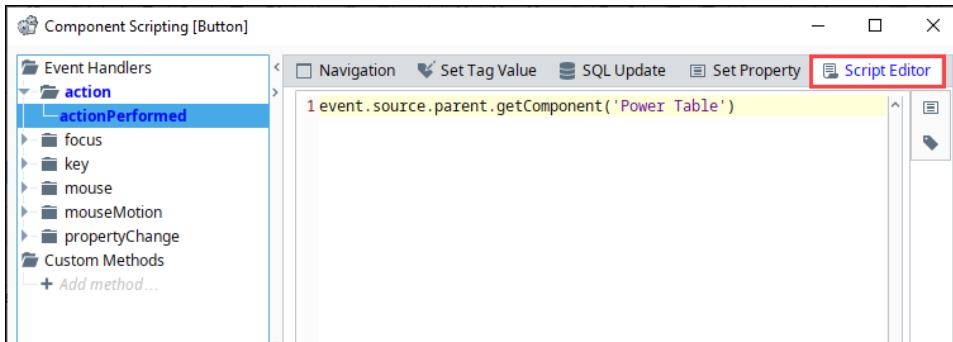
Power Table Example

This example provides the code that you can use for selecting a single cell and multiple cells in a Table.

Note: Before we get started, it is important to understand that a cell in a Table is actually a cell in a dataset. Data in a Table is stored in a property on the component (the **Data** property), and the script needs to interact with that property.

Retrieve a single cell requires that we reference the Power Table a couple of times in the same script. Because of this, we can create a variable that references the table, and use the variable later on. Be aware that a Button and Power Table can be in the same container or separate containers. If your Button and Power Table are in separate containers your path may differ. **Selecting a Single Cell**

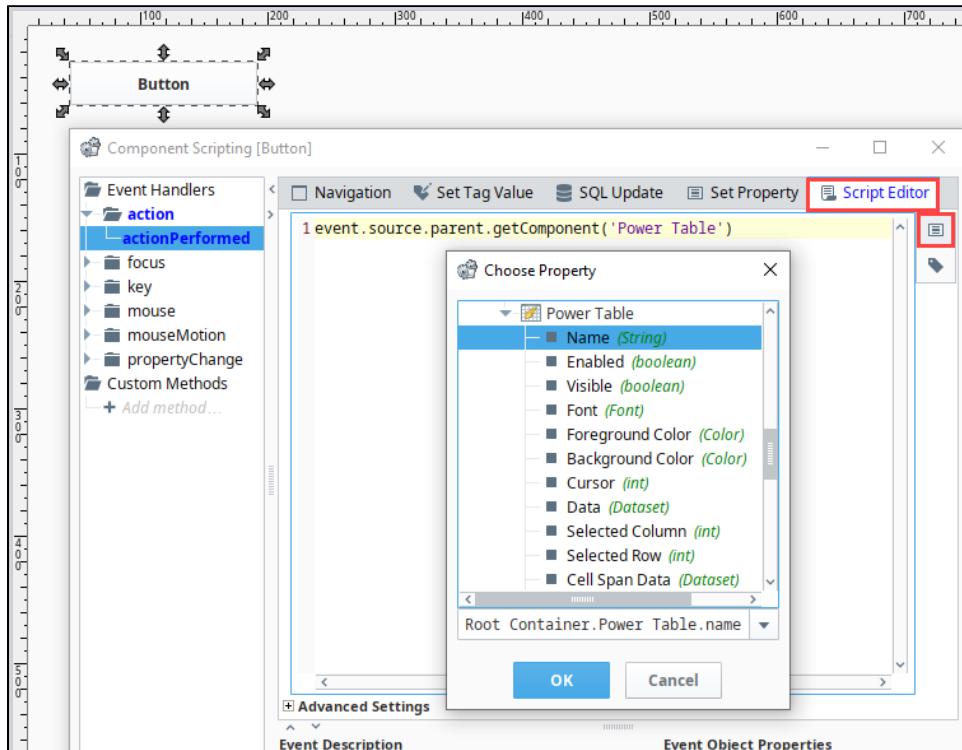
1. Drag a **Button** and **Power Table** components to your Designer workspace. Assuming the Power Table has not been renamed, we can reference the Power Table component with the line of code below as shown in the image below.



Python - Reference the Power Table

```
# Grab a reference to the Power Table.  
table = event.source.parent.getComponent('Power Table')
```

- As mentioned above, if the Button and Table are in separate containers, the path will be different. Double click the **Button** component to open the **Scripting** window, select the **action > actionPerformed** event handler, and click the **Property Reference** icon  to generate the path to a property, but not the component.
- Next, select a property from the Power Table in the **Choose Property** window and click **OK**.
- Remove the ".propertyName" portion at the end of the script in the Script Editor, and click **OK**.



- Next, we simply need to figure out which row contains the cell we want to read from. In our example, the user will select the row for us, so we simply need to know which row is selected when the Button is pressed. Fortunately, the Power Table contains a **Selected Row** property that can be used to determine the row that is selected. Furthermore, we can use the [getValueAt\(\)](#) function that is built into datasets.

Python - Reference the Name of the Column

```
# Here the "Mechanic_Name" argument references the name of the column.
table.data.getValueAt(table.selectedRow, "Mechanic_Name")
```

- Alternatively, we can use an integer as the last argument to specify the index of the column our cell is located in.

Python - Reference the Index of the Column

```
# Here the '1' references the index of the column in the Power Table's raw dataset (Data property).
# Remember, indexes are zero-based, so this would retrieve the second column from the left.
mechanicName = table.data.getValueAt(table.selectedRow, 1)
```

- Lastly, we will need to account for scenarios where the user did not select a row, otherwise, this will throw an exception. An if-statement can be used here to check for a -1 value on the Power Table's **Selected Row** property, and an else-clause can be used to notify the user that a row needs to be selected. There is a property on the Power Table named **Selection Mode** that allows users to select multiple rows. By default it is set to only allow a single row to be selected. Change it to test your button.

Here is all the code that you'll need to read a single cell from a Power Table in the same container.

Python - Putting it all Together

```
# Grab a reference to the table.
table = event.source.parent.getComponent('Power Table')

# Make sure the user selected something before doing the rest of the work.
```

```

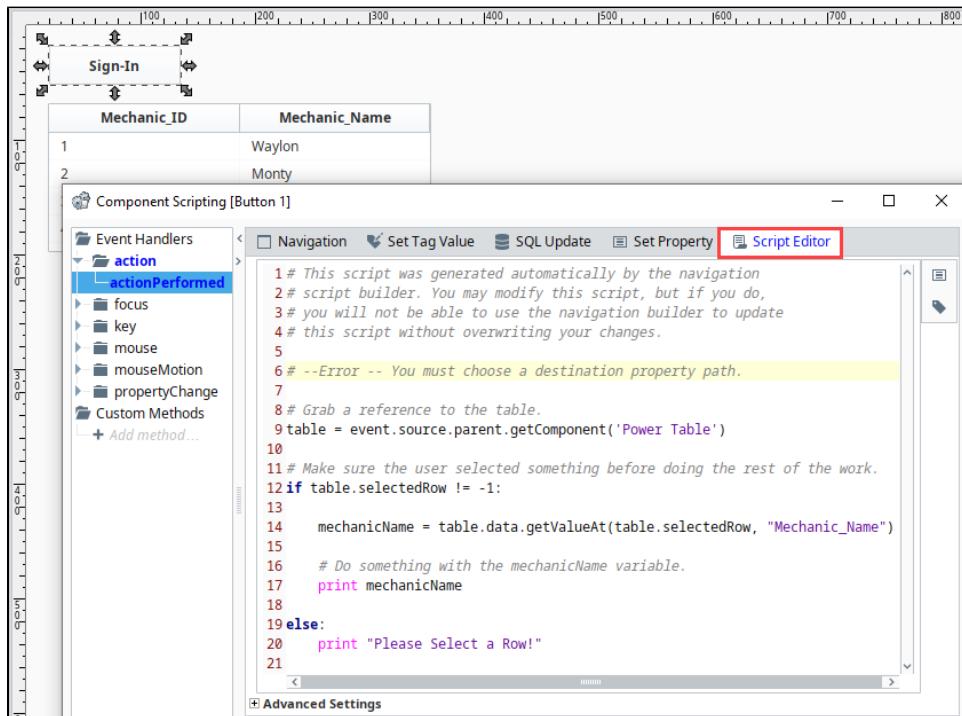
if table.selectedRow != -1:

    mechanicName = table.data.getValueAt(table.selectedRow, "Mechanic_Name")

    # Do something with the mechanicName variable.
    print mechanicName

else:
    print "Please Select a Row!"

```



- To test your script, follow the steps in the [Test Your Script](#) section.

Selecting Multiple Cells - Same Column

By default, the Power Table allows for multiple rows to be selected. However, the Selected Row property only shows the row index for the first row selected. Fortunately, the Power Table also has a built-in `getSelectedRows()` function that can be used to return all of the indices. We simply need to iterate over each index.

We can still use the Power Table's **Selected Row** property to test if any rows are selected, but we could instead check the length of the object returned by `getSelectedRows()`:

Here is all the code you'll need to read multiple cells from a Table in the same container. Copy the code in the code block below and replace the code in previous example to allow for multiple rows to be selected.

Python - Reading Multiple Cells

```

# Grab a reference to the table.
table = event.source.parent.getComponent('Power Table')

selectedIndices = table.getSelectedRows()

# If the length of selectedIndices is greater than 0, then at least one row is selected.
if len(selectedIndices) > 0:

    for index in selectedIndices:

        mechanicName = table.data.getValueAt(index, "Mechanic_Name")

```

```

# We can do something with the value as we iterate, or append to a list and
# do something with the entire list after the for-loop completes.
print mechanicName

else:
    print "Please Select a Row!"

```

Test Your Script

Now you're ready to test your script whether you are selecting one row or multiple rows.

1. Open the **Output Console** and select **Tools > Console** in the menubar.
2. In **Preview Mode**, select a row in the **Power Table**, and click the **Sign-In** button. You will see the selected Mechanic_Name displayed in the console.
3. To test selecting multiple rows, shift click a couple of rows and press the **Sign-In** button, and you'll see the selected Mechanic_Names displayed in the console.

The screenshot displays two windows side-by-side. On the left is a 'Sign-In' interface featuring a 'Power Table'. The table has two columns: 'Mechanic_ID' and 'Mechanic_Name'. It contains four rows with data: (1, Waylon), (2, Monty), (3, Kurt), and (4, Lewis). The fourth row, which corresponds to Mechanic ID 4, is highlighted with a blue background. On the right is an 'Output Console' window, which shows the name 'Lewis' in its text area, indicating that the selected row was successfully processed.

Related Topics ...

- [Vision - Table](#)
- [Vision - Power Table](#)

Historian in Vision

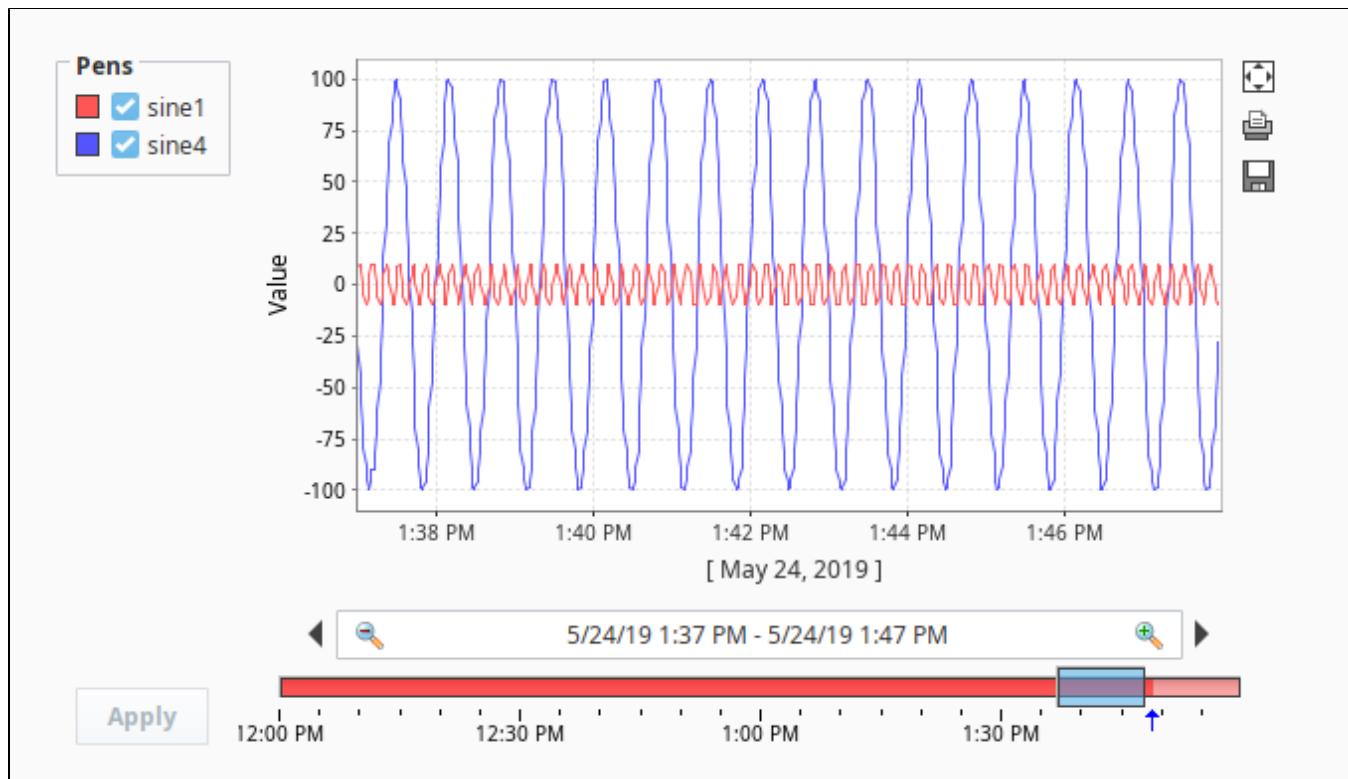
The [Tag Historian](#) is a powerful system that can easily be set up to store Tag data to a database to be accessed at a later time. The Vision system has many components that are designed to easily pull the information out of the database and display it, most commonly in chart format.

The Easy Chart

The most popular Vision component that trends historical data would be the [Easy Chart](#). This component is simple to initially configure and contains many ways to customize the look and behavior of the chart. The Easy Chart also features a customizer that enables you to change the many different settings of the component.

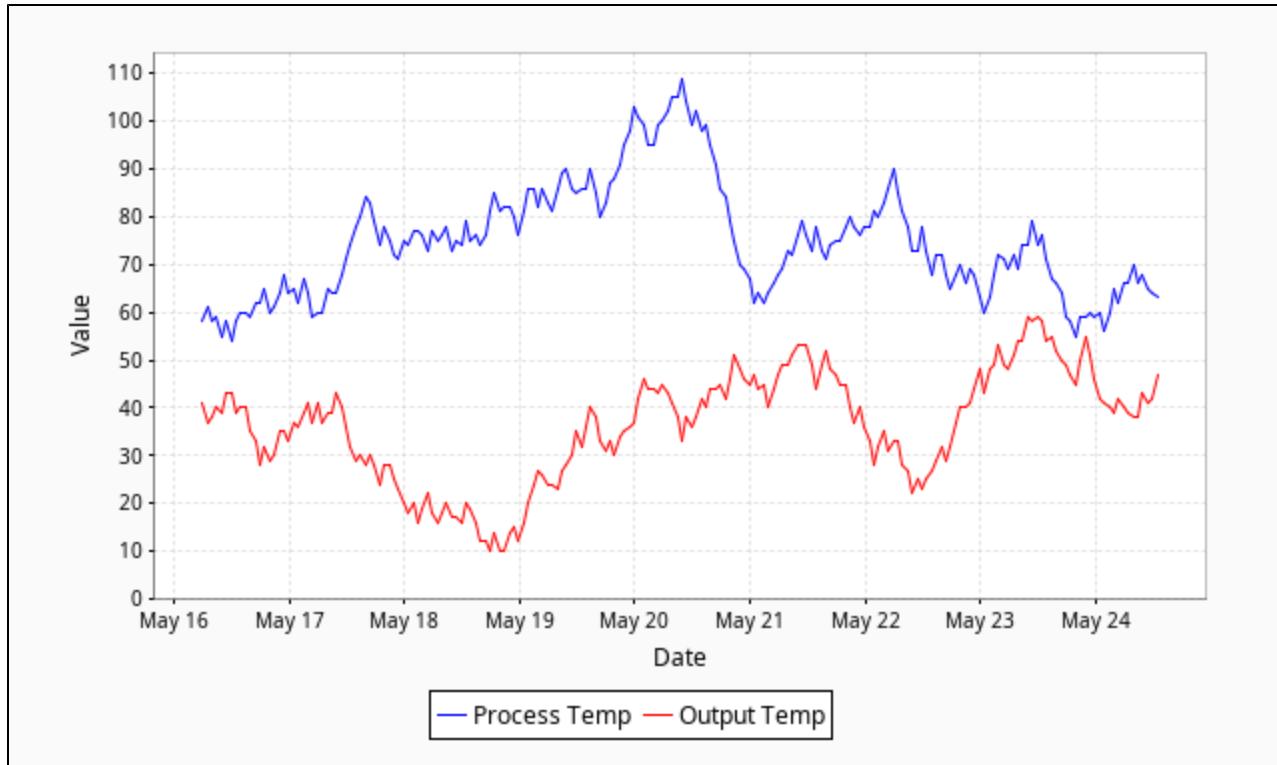
On this page ...

- [The Easy Chart](#)
- [The Classic Chart](#)
- [The Sparkline Chart](#)
- [The Status Chart](#)



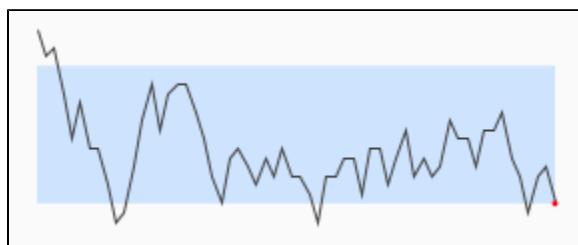
The Classic Chart

The [Classic Chart](#) pulls in data from the Tag History system and can display it in a variety of ways. While not as simple to get started with as the Easy Chart, the Classic Chart provides the unique ability to trend data that isn't based on a timestamp, but instead something like a category.



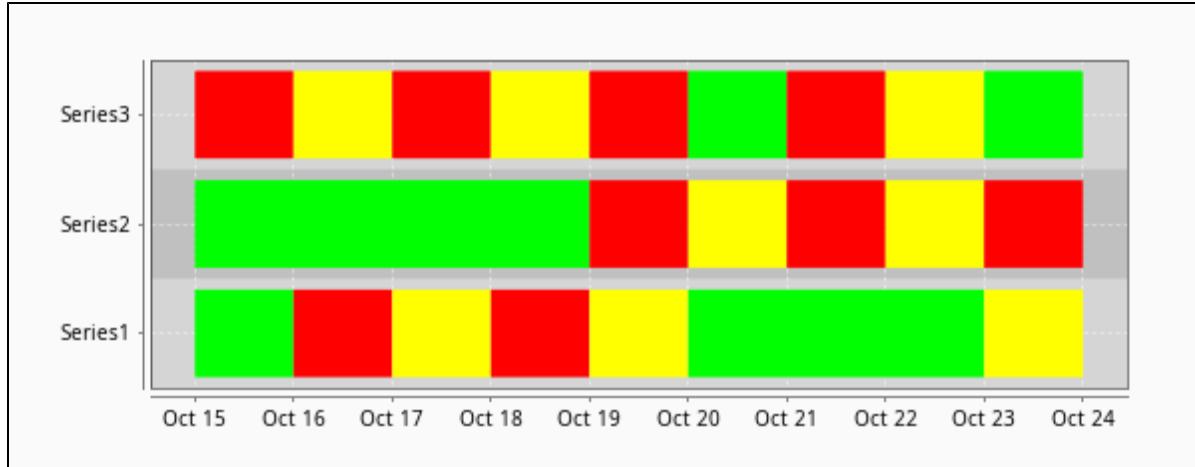
The Sparkline Chart

The [Sparkline Chart](#) is a simple chart that strives for minimalism rather than many fancy settings. Even with its simplicity, it remains a powerful tool. For example, the Sparkline Chart works great when used in [High Performance HMI](#) environments, where muted colors and lines are used to help draw the users attention to the places that matter.



The Status Chart

The [Status Chart](#) displays discrete data over a period of time. This provides a great way of displaying status information for various machines. It can also accept data in both wide and tall format, making it easy to use with any type of stored historical data.



In This Section ...

Using the Vision Easy Chart

Trends Made Easy

The [Easy Chart](#) was developed with the [Tag Historian](#) system in mind. Once an Easy Chart is created, you can drag and drop historical Tags onto the chart. The chart will immediately retrieve the results and trend the history. Data that is not set up with Tag Historian can also be displayed on the chart, as long as the data has timestamps associated with the values. For this type of data, database pens are created and displayed.

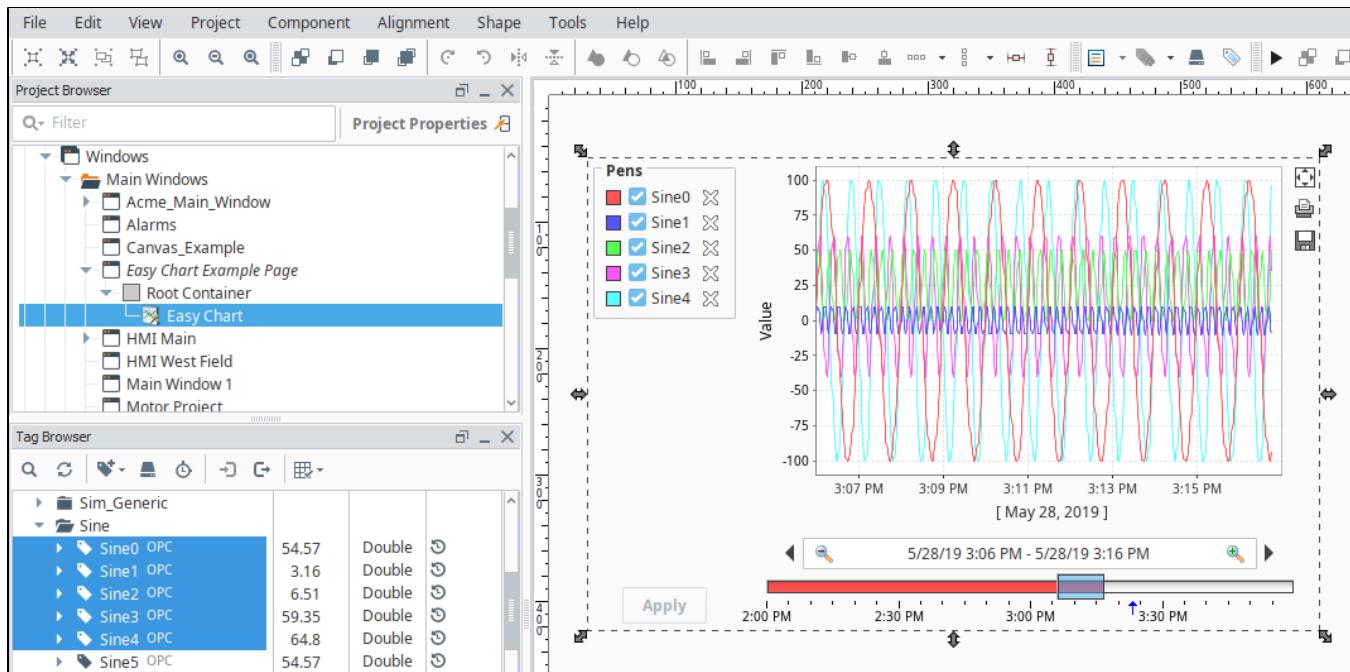
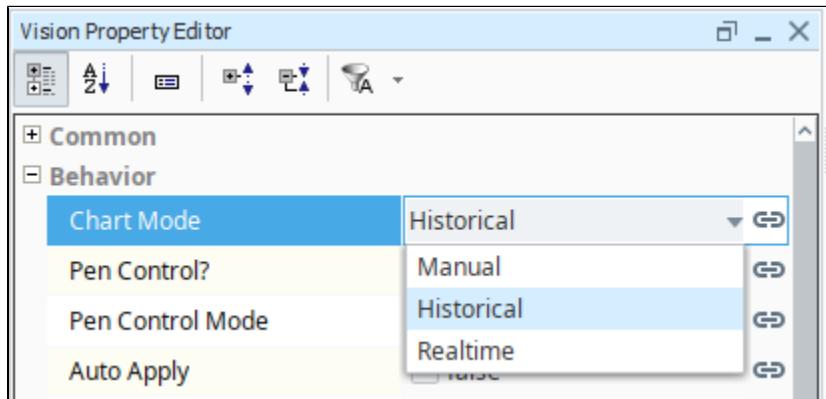


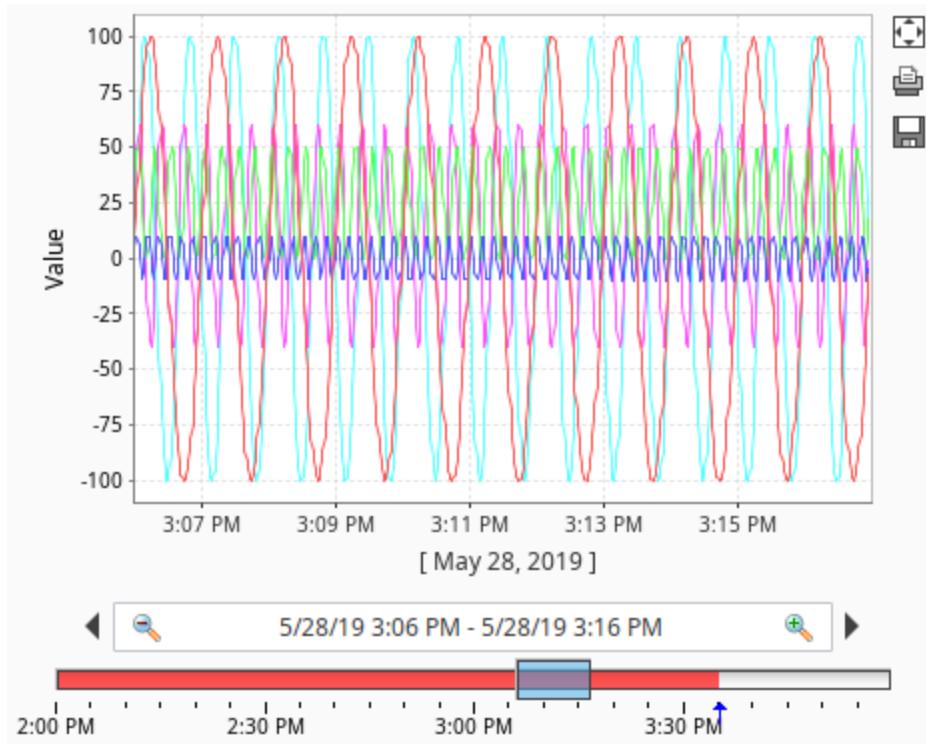
Chart Modes

The Easy Chart has a Chart Modes property that changes the behavior of the chart in several ways. The mode is set in the Vision Property Editor in the Chart Mode Property.



Historical

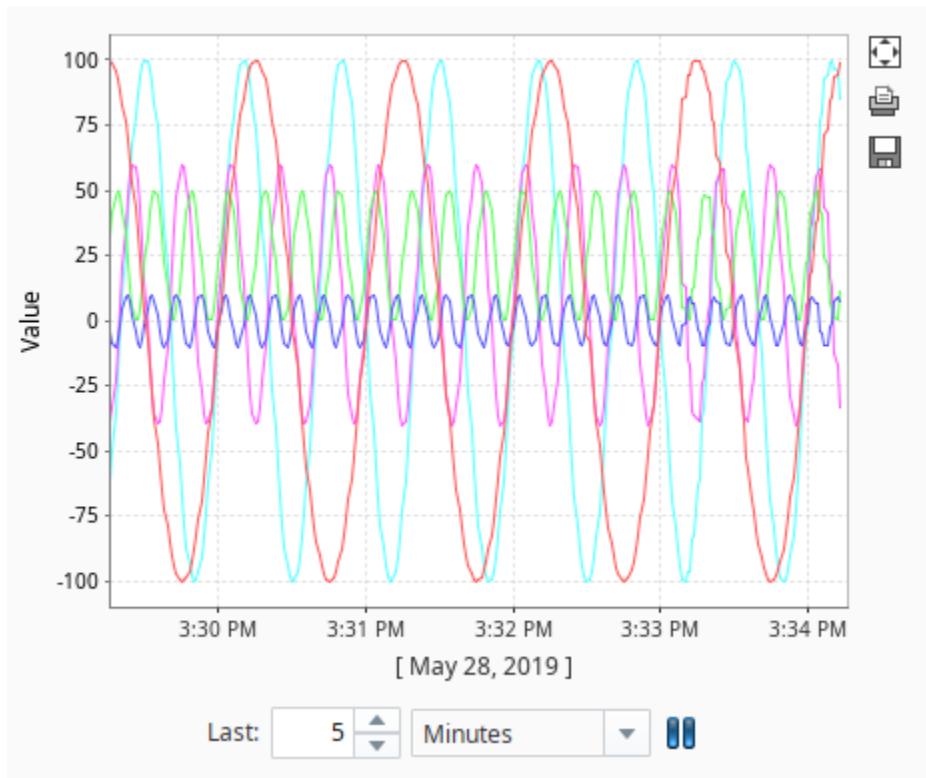
This mode places a [Date Range Selector](#) component at the bottom of the chart. This mode allows users to select a start and end date for the trends. The data density is shown at the bottom of the chart: the more vibrant the color, the higher the density. This is the default mode, and commonly used in situations where users need to look at specific date ranges. It is important to remember that the chart does not poll in this mode. New values are only added to the chart when the selection box is moved or re-sized.



Realtime

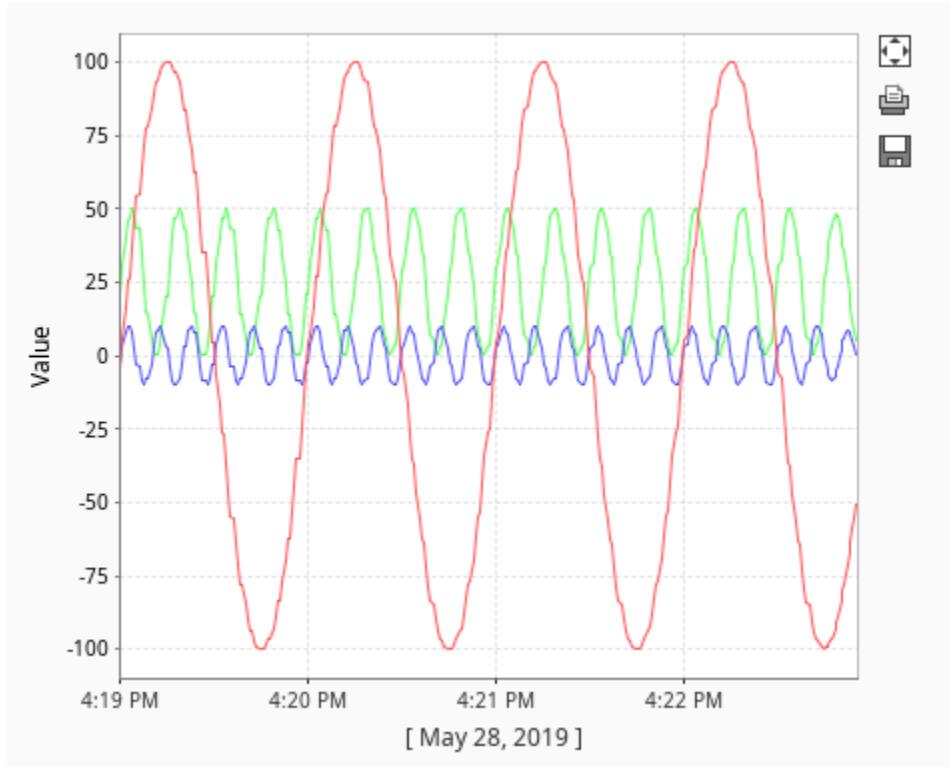
Displays the most recent data for each pen. Users are able determine how far back in time the trend should display with the [Spinner](#) and [Dropdown](#) components at the bottom of the chart. In this mode, the chart polls for data at the rate specified by the Poll Rate property.

In some cases, you may notice that the most recent values on tag pens tend to flat-line, and then 'snap' to a different value. This is generally due to how often the chart polls versus how often history is being generated. If the chart polls at a 1,000ms rate, but history is only recorded at a 10,000ms rate, then the chart will extrapolate the last recorded value for 9,000ms. After a new value is recorded, the next poll will return the latest value, and the flat-line will change position.



Manual

Similar to the Historical mode in that trends range from two points in time. However, there is not a built-in method for users to change the data range. Instead, some sort of binding can be applied to the chart's Start Date and End Date property. This mode is generally used in situations where only certain date ranges should be shown, such as values recorded during the previous day, or shift.



Pens

Pens on the chart, or each series of data points on the chart, can be customized to take on a number of different styles and colors. There are three types of pens, and each pen functions in a similar manner. What makes them different is how their data is collected.

- **Tag Pens** - These pens are driven by the Tag history system. Data from any historical provider can be used, and Tag history from different databases can be shown on the same chart. These are the type of pens that are created when Tags are dragged onto the chart. Since the Tag History system is being used, an Aggregation Mode must be selected, and the Tag Path needs to be specified for each pen.
- **Database Pens** - These pens are driven by a SQL query, so they are ideal to use when trending Transaction Group data. However, they can query for data in any connected SQL database, so it is possible to show historical data recorded by other systems on the Easy Chart.
- **Calculated Pens** - Pens that derive their data from calculations performed on other pens. Data for calculated pens is not stored directly into a database, but rather calculated in the runtime based on data from another pen. These type of pens are great for display running totals, control limits, or specification limits.

Easy Chart Customizer

Pens **Axes** **Subplots** **Dynamic Groups**

Tag History Pens

| Name | Tag Path | Color | Preview | |
|-------|---------------|-------|---------|--|
| Sine0 | [~]Sine/Sine0 | Red | | |
| Sine1 | [~]Sine/Sine1 | Blue | | |
| Sine2 | [~]Sine/Sine2 | Green | | |

Database Pens

| Name | Y Column | Table | Color | Preview | |
|------|-------------|--------------|-------|---------|--|
| RV | Tank_Number | Tank_History | Green | | |

Calculated Pens

| Name | Function | Pen | Parameter | Color | Preview | |
|-----------|------------|-------|-----------|--------|---------|--|
| sine0 Sum | RunningSum | Sine0 | | Orange | | |

OK **Cancel**

Pens can be added manually to the chart with the [Easy Chart Customizer](#) or added dynamically by modifying the various pens properties listed under Chart Configuration in the Vision Property Editor. These properties contain the configuration of each type of pen and can make use of the binding system. Because of this, pen preferences can be saved to a database table and then queried in the runtime with a SQL binding. Additional adjustments can be made with [Cell Update bindings](#) or scripting to create a dynamic-yet-robust chart.

Vision Property Editor

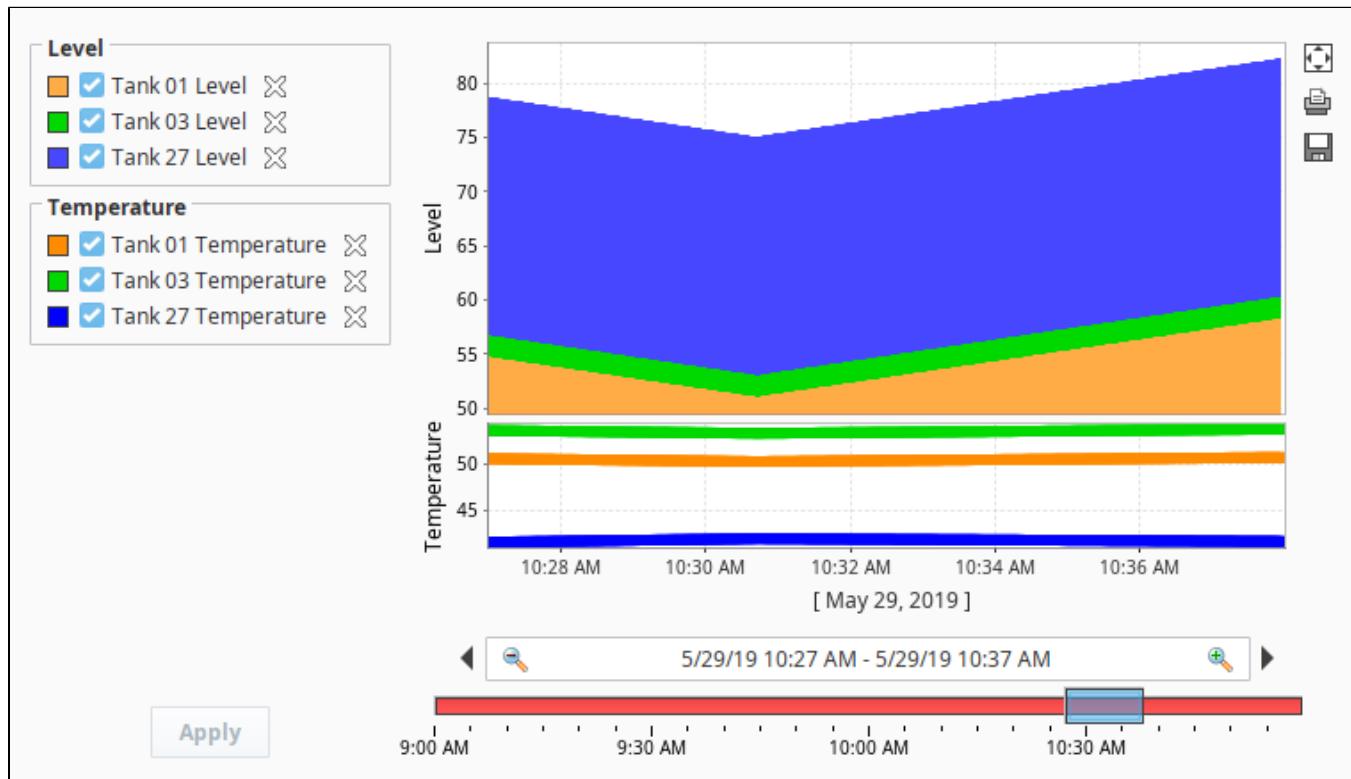
Chart Configuration

| | | |
|-----------------|--------------------|--|
| DB Pens | Dataset [2R x 23C] | |
| Tag Pens | Dataset [3R x 20C] | |
| Calculated Pens | Dataset [1R x 21C] | |
| Axes | Dataset [1R x 22C] | |
| Subplots | Dataset [1R x 3C] | |

Note: When you add pens to the Easy Chart, all pens will show up with a white X next to their name. This exists for creating an [ad hoc chart](#) used with the Tag Browse Tree component. To remove the ability to delete pens, you must edit the Tag Pens dataset property. Click on the dataset viewer icon, then in the last column "User Removable", deselect the checkbox and click OK.

Easy Chart Customizer

Aside from the properties on the component, the [Easy Chart Customizer](#) allows modifications to be made to the chart. Along with configuring pens, the customizer can be used to create [subplots](#), new [axes](#), and dynamic groups. Once created, each pen can be assigned to any available axes or subplots. This way different values can be shown on different plots with an axis that is specific to data at hand.



Related Topics ...

- [Easy Chart Customizer](#)

In This Section ...

Easy Chart - Axes

Configuring Multiple Axes on an Easy Chart

The [Easy Chart](#) supports the use of multiple axes for displaying data from the Tag Historian.

i This section assumes that Tags and Tag History have been configured

To learn more, go to the [Tag](#) and [Configuring Tag History](#) pages.

The examples below use OPC Tags from the [Programmable Device Simulator](#) driver, but Memory Tags can be used instead.

On this page ...

- Configuring Multiple Axes on an Easy Chart
- Hiding Pens
- Configuring an Easy Chart using the Symbol Axis

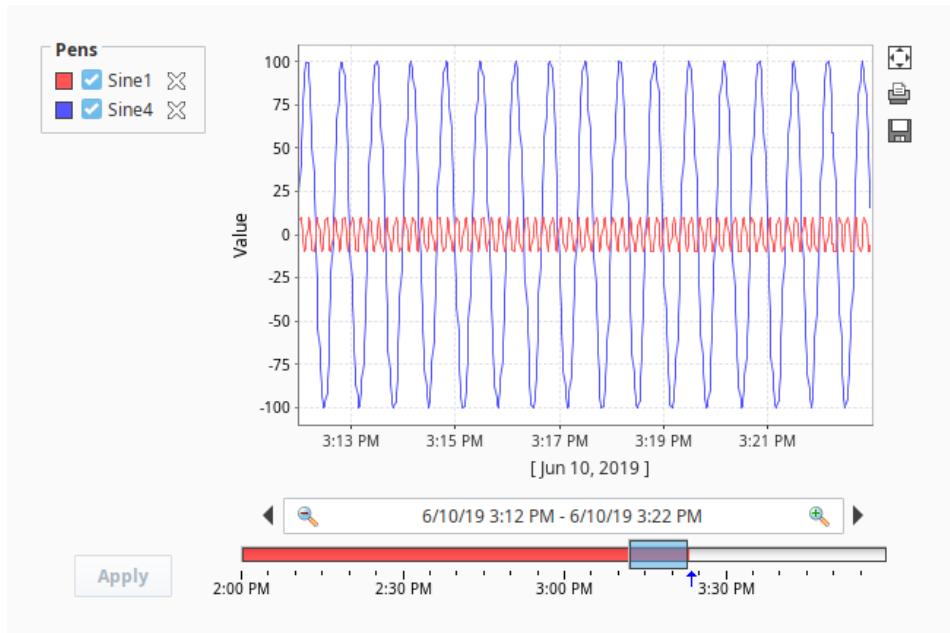


Easy Chart - Axes

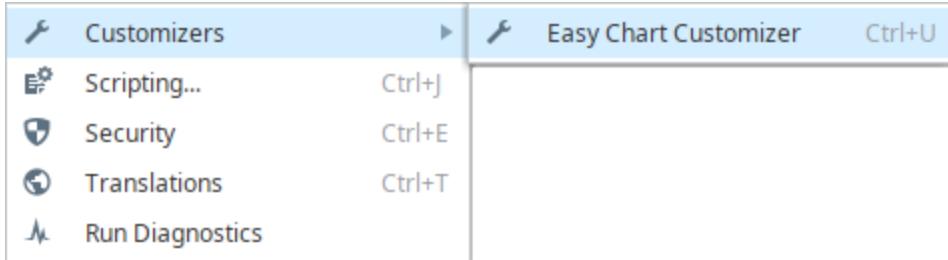
[Watch the Video](#)

Now, let's configure multiple axes on an Easy Chart component.

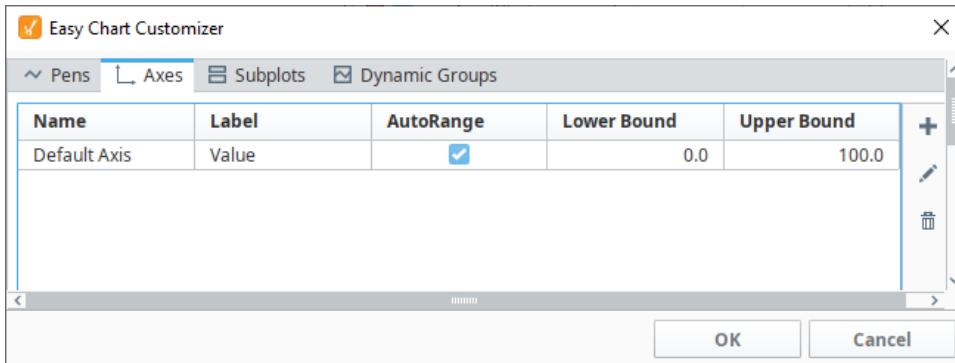
1. From the Component palette, drag an **Easy Chart** component to your workspace.
2. Next, drag your Tags over from the Tag Browser onto your Easy Chart. For the example, we used Sine 1 and Sine 4. Sine 1 and Sine 4 have completely different value ranges. Sine 1 values range between -10 and 10. Sine 4 values range between -100 and 100. Since both sines are on the same axis, it is hard to see the details of Sine 1 values because Sine 4 is throwing off the axis due to its wide range of values.



3. You have the option of putting Tags into different axes. You can do that in the Customizer of the Easy Chart component. Right click on the Easy Chart component and choose **Customizers > Easy Chart Customizer**.



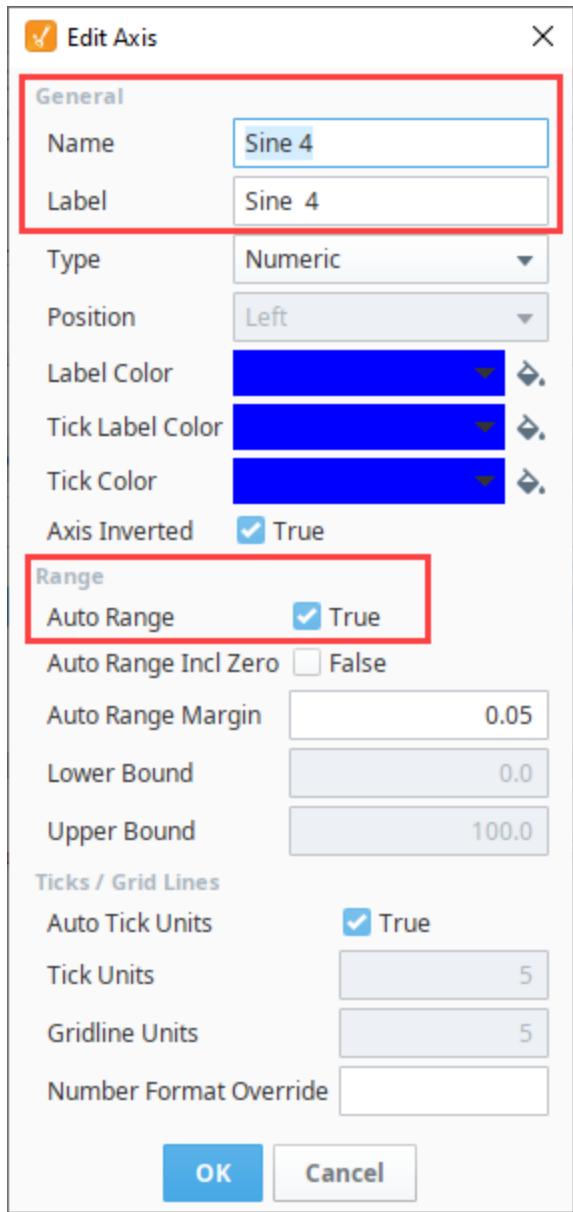
4. Click on the **Axes** tab. You'll notice that there is already one axis showing called Default Axis, which both Tags are sharing.



5. To add an axis, click on the **Add** icon.
 6. The Edit Axis window is displayed. Assign the **Axis** a name. In this example, it's the name of the Tag that is being used.
 7. Enter a **Label** name, which is a name that you want users to see on your chart.
 8. Select the **Type** of axis from the dropdown: Numeric, Logarithmic or Symbol. This example uses the default, **Numeric**.
 9. If desired, select the **Label**, **Tick Label** and **Tick Color** that you want to set for your axis.

The **Position** property determines which side of the chart the Axis should be drawn on. By default, this property is disabled because the Easy Chart automatically attempts to position each Axis. To manually determine the position of an Axis, locate the **Auto Axis Positioning** property in the Property Editor of the Easy Chart component, and set it to '**False**'.

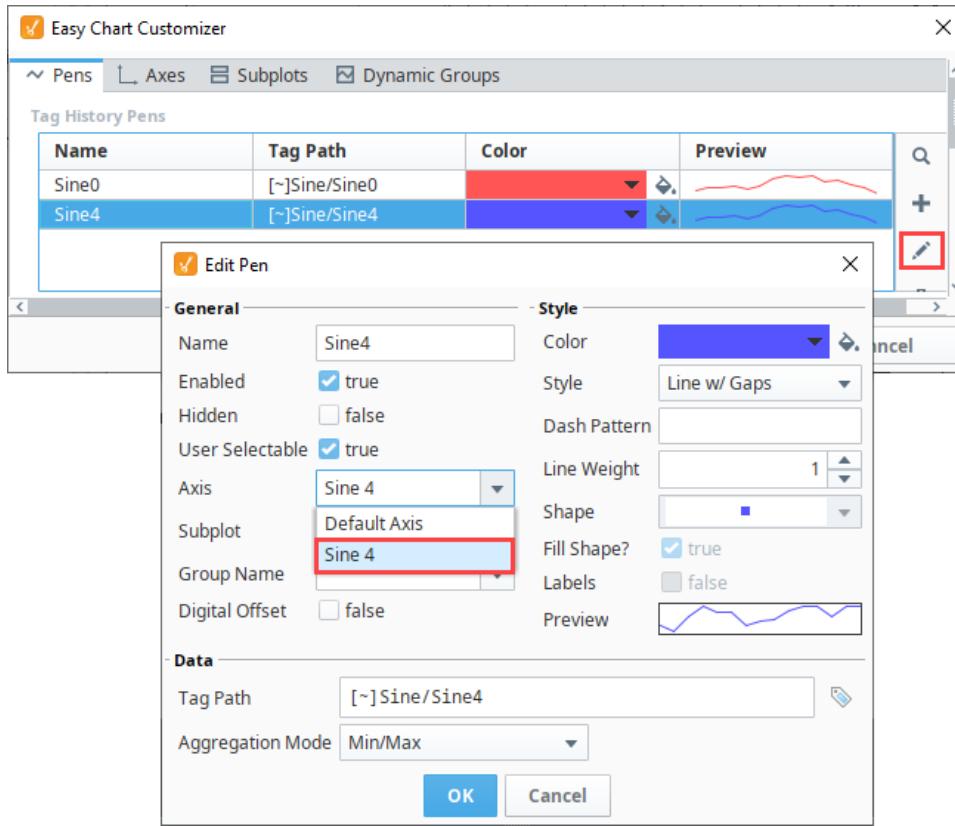
By default, the **Auto Range** is set to '**true**' and will apply padding so the pens do not draw at the top and bottom of the axis. Instead of having the Easy Chart automatically determine the range, Auto Range could be set to '**false**', in which case the **Lower Bound** and **Upper Bound** properties will determine the full range of the axis.



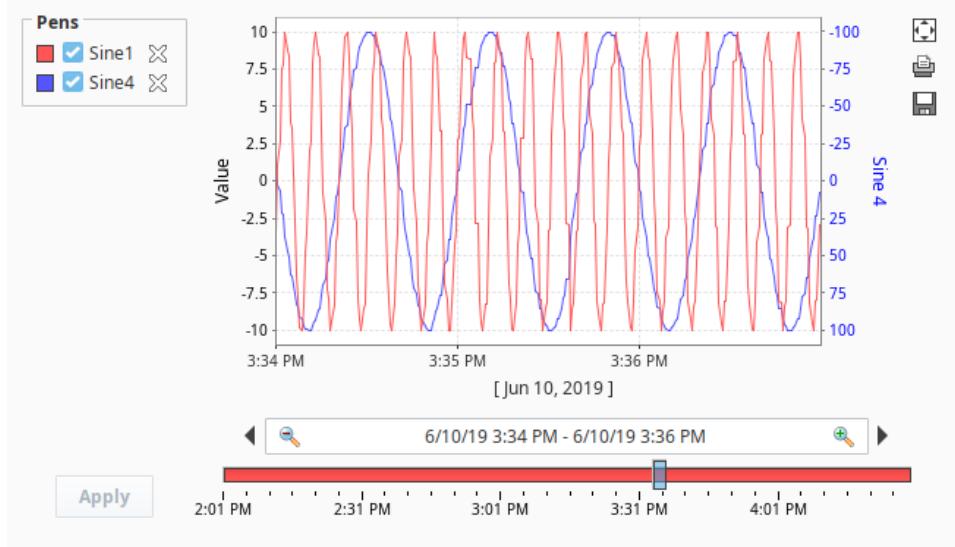
10. Now, you have two axes: Default Axis and Sine 4 axis.

| Easy Chart Customizer | | | | | |
|--|--------|-------------------------------------|-----------------------------------|--|---|
| <input checked="" type="checkbox"/> Pens | | <input type="checkbox"/> Axes | <input type="checkbox"/> Subplots | <input checked="" type="checkbox"/> Dynamic Groups | X |
| Name | Label | AutoRange | Lower Bound | Upper Bound | <input type="button" value="+"/> |
| Default Axis | Value | <input checked="" type="checkbox"/> | 0.0 | 100.0 | <input type="button" value="Pens"/> <input type="button" value="Axes"/> <input type="button" value="Subplots"/> <input type="button" value="Dynamic Groups"/> |
| Sine 4 | Sine 4 | <input checked="" type="checkbox"/> | 0.0 | 100.0 | <input type="button" value="Edit"/> <input type="button" value="Delete"/> <input type="button" value="OK"/> <input type="button" value="Cancel"/> |

11. Once a new axis has been created, you need to assign a pen to the axis. Select the **Pens** tab, select the pen row you want to change, and click the  icon. This example uses the 'Sine 4' pen. In the **Axis** field, select the 'Sine 4' axis from the dropdown menu, and click **OK** to save the pen.
12. Click **OK** again to close the Easy Chart Customizer.



13. You'll notice that each pen is now in a different axis: Sine 1 is in the Default Axis, and Sine 4 is in the Sine 4 axis. On the right side, you can see Sine 4 axis and its values. On the left side, you can see the Default Axis (Sine 1) axis and its values. Now, you are using a different axes for each pen, and one pen is not going to throw off the values for the other pen.

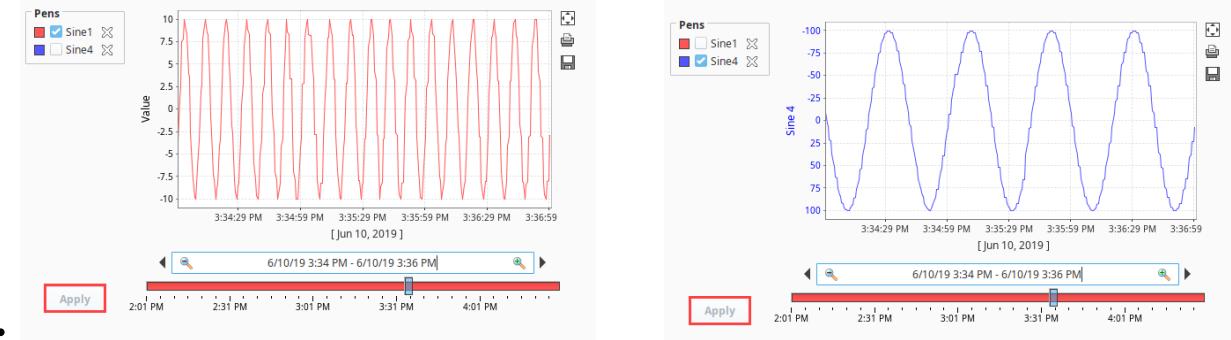


Hiding Pens

You can also hide or turn off pens so they are not displayed on the Easy Chart. To only see values for the Sine 1 axis, go to **Preview Mode**, uncheck the **Sine 4** pen, and click **Apply**. To see the values for only the Sine 4 axis, check the **Sine 4** pen, uncheck the **Sine 1** pen, and click **Apply**. Also important to note, auto positioning on the Easy Chart will automatically move the axis should pens assigned to an axis be removed.

Sine 1 Pen

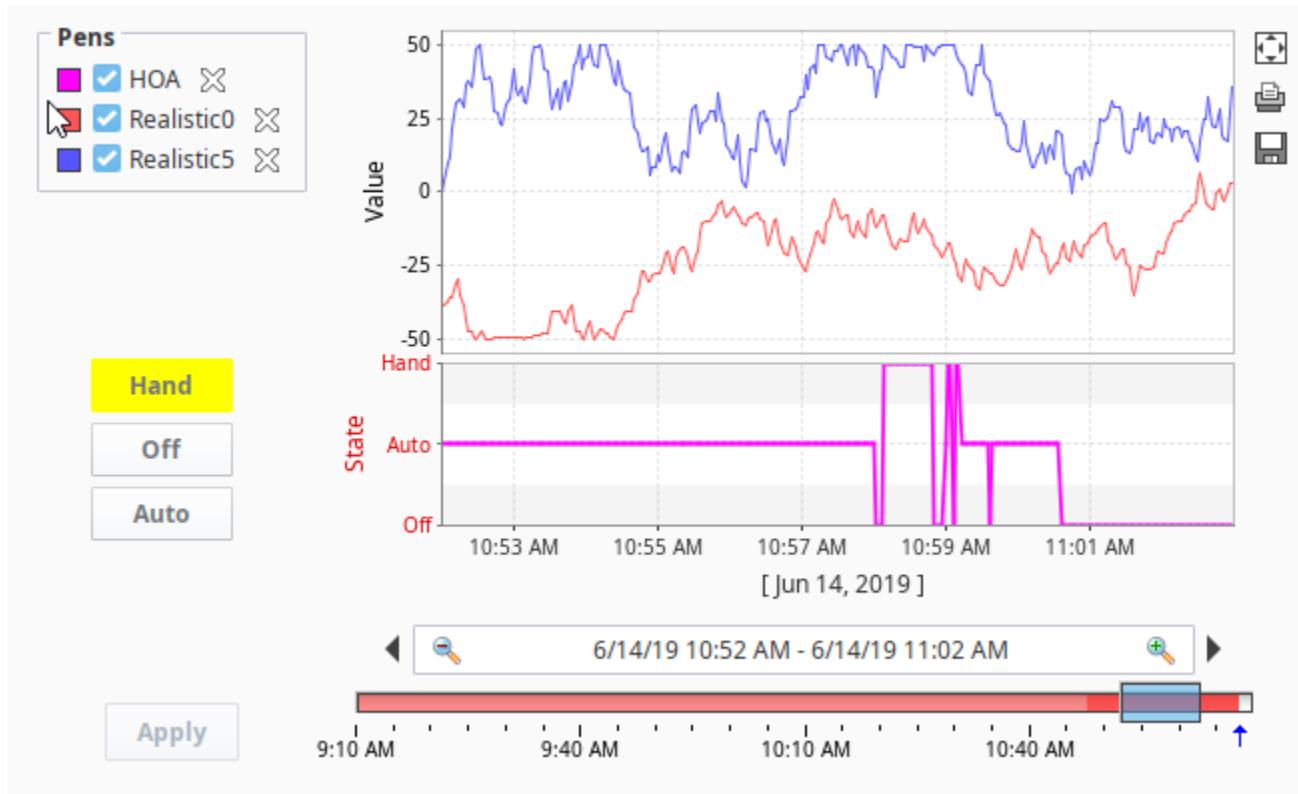
Sine 4 Pen



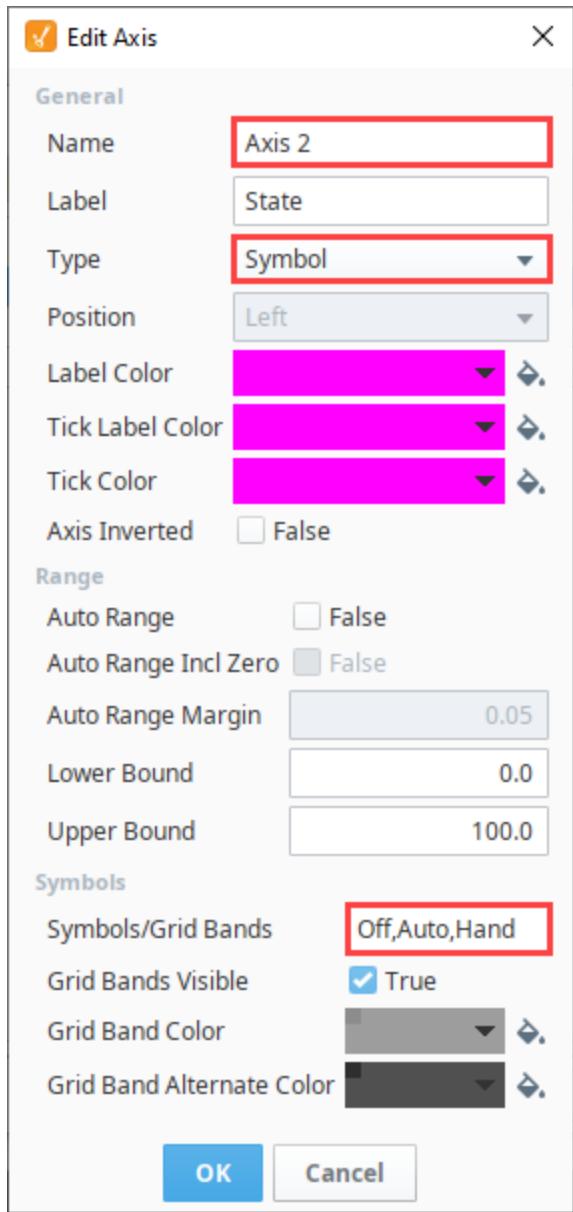
Configuring an Easy Chart using the Symbol Axis

Another feature of the Easy Chart is the use of the **Symbol Axis** type. Instead of showing numerical values on the axis, the Symbol Axis type can show plain text on the axis.

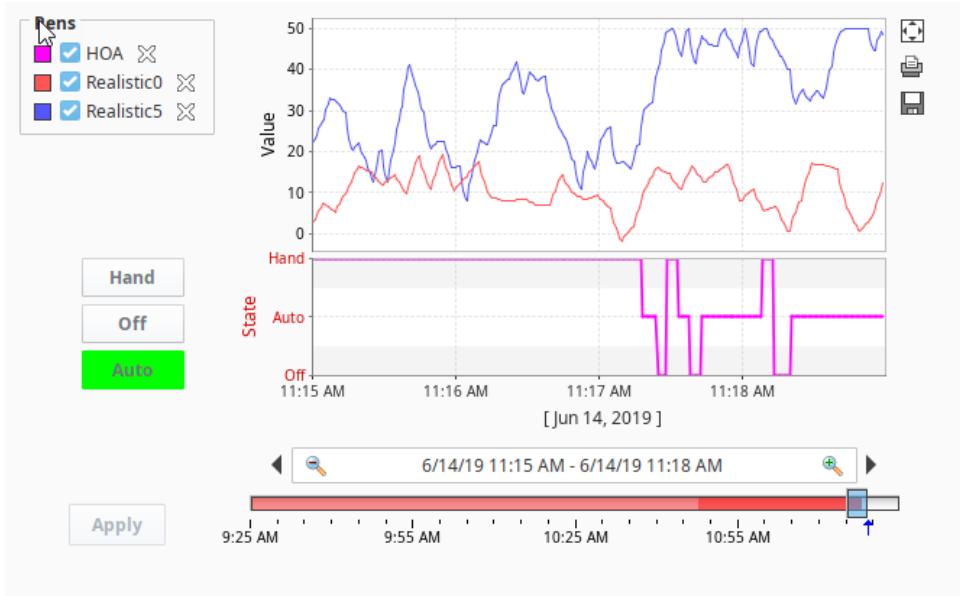
The second subplot in this Easy Chart uses a Multi-State Button component to demonstrate the use of the **Symbol Axis** type. The Multi-State Button component is bound to an OPC Tag, and the value of the Tag is stored in the Tag History system. Instead of showing the numerical values '0,' '1,' and '2,' you can use plain text such as 'Hand,' 'Off,' and 'Auto.' This is helpful to an operator who immediately knows the state of the equipment instead of having to learn what the numeric values mean.



1. Click on your Easy Chart component, and select **Customizers > Easy Chart Customizer**.
2. Click on the **Axes** tab, then click the **Add** icon to add an axis.
3. Enter a **Name**, we chose 'Axis 2'. Enter a **Label**, we called 'State'.
4. In the **Type** field, select '**Symbol**' from the dropdown.
5. In the **Symbols/Grid Bands** field enter '**Auto**,' '**Off**,' and '**Hand**' separated by commas, and no spaces. (Note: The order of the symbols when you type them in, will be ascending order on the axis).
6. Click **OK** two times.



7. Put the Designer in **Preview Mode** ►.
8. Toggle the Multi-State buttons to begin logging data to your Easy Chart. Data for the Multi-State Button component is collected in the second subplot which is using the Symbol Axis type. The first subplot is data for another Tag using the Numeric Axis type.



This Easy Chart example above shows the [subplot feature](#) of the Easy Chart component and how easy it is to break up the chart plot area into multiple distinct subplots sharing the same X axis. It is a good way to display lots of data from different Tags in one Easy Chart.

Related Topics ...

- [Easy Chart Customizer](#)
- [Easy Chart - Subplots](#)

Easy Chart - Subplots

i This section assumes that Tags and Tag History have been configured

To learn more, go to the [Tag](#) and [Configuring Tag History](#) pages.

The examples below use OPC PC Tags from the [Programmable Device Simulator](#) driver, but Memory Tags can be used instead.

Subplot Overview

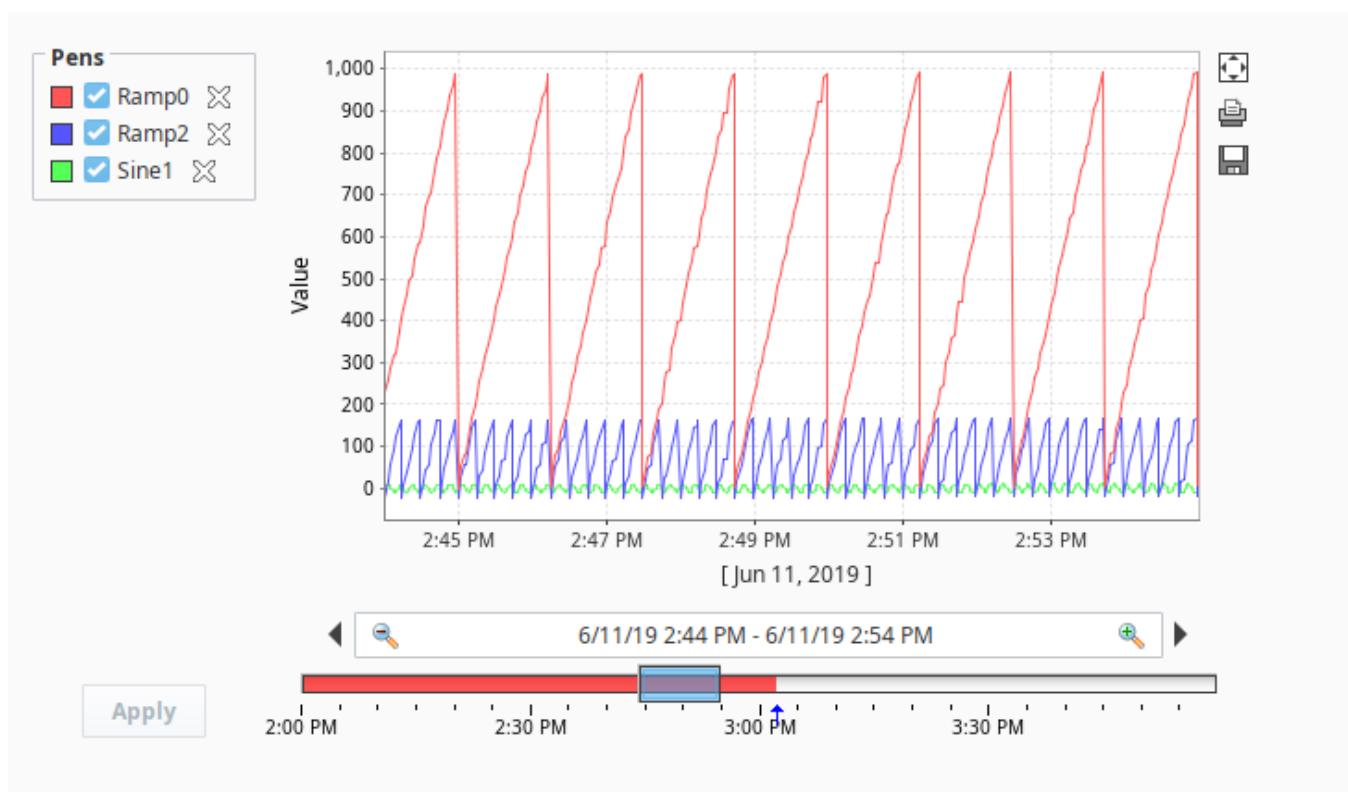
The subplot feature of the Easy Chart component allows you to break up the chart plot area into multiple distinct subplots sharing the 'X' axis, but they each have their own 'Y' axis. It is a good way to display lots of data from different Tags in one Easy Chart.

By default, the Easy Chart has one subplot which is the main white area. In this example, there are three Tags inside the chart, yet it's difficult to see the details of the data. It's possible to break up your Tags into multiple subplots which is often useful for discrete data.



Easy Chart - Subplots

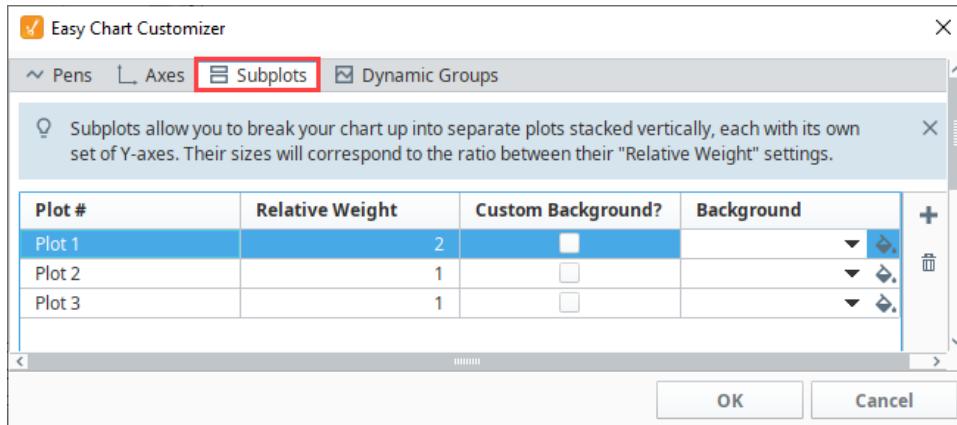
[Watch the Video](#)



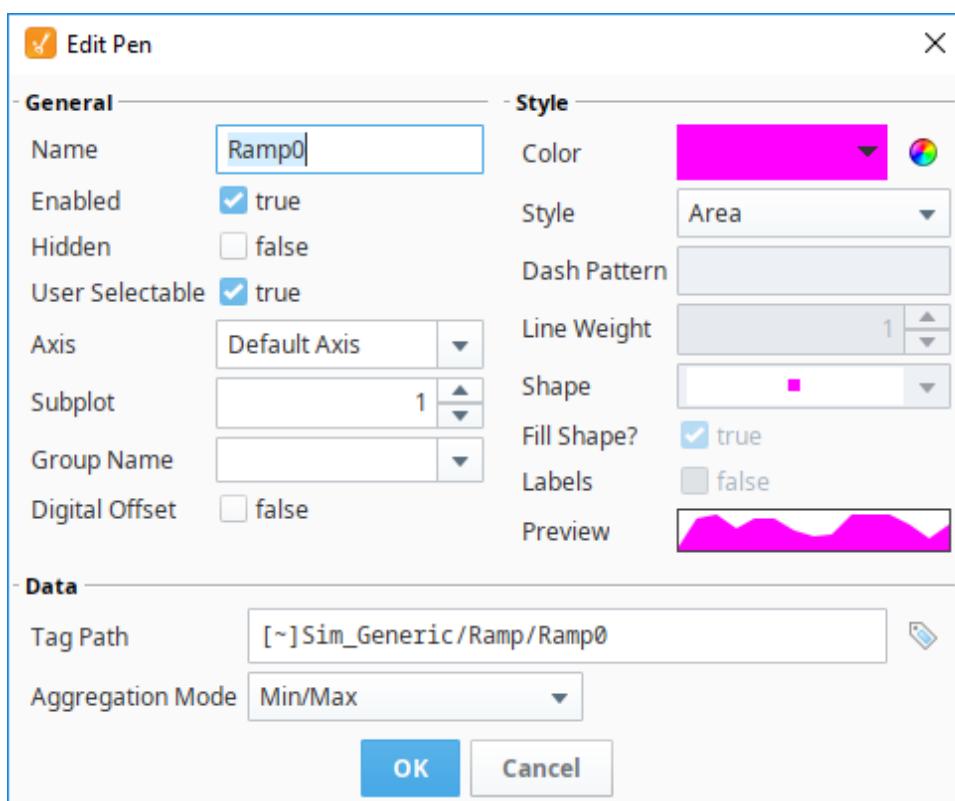
Configuring Easy Chart Subplots

For each Tag in the Easy Chart example above, let's create its own subplot so the data is easier to view and analyze.

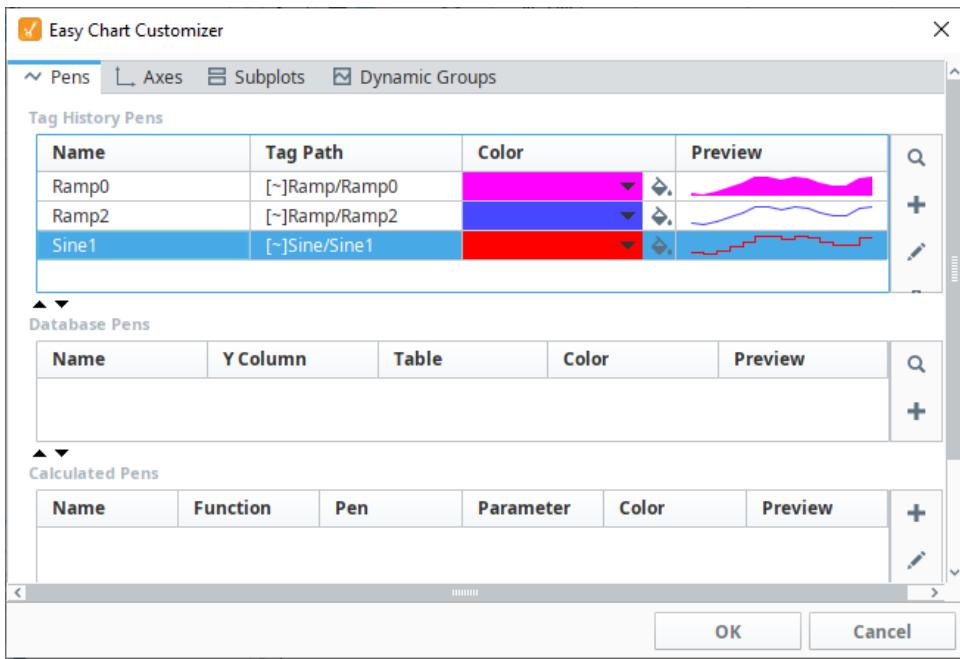
1. Drag an Easy Chart component onto your window.
2. Drag three Tags onto the chart. We used Ramp0, Ramp2, and Sine1 from [Programmable Device Simulator](#).
3. Right click on the Easy Chart component and choose **Customizers > Easy Chart Customizer**. The Easy Chart Customizer window opens displaying four tabs.
4. Click on the **Subplots** tab.
5. The Subplots tab lets you add one or more subplots to the Easy Chart. Create two more subplots by clicking the **Add +** icon two more times.
6. The size of each subplot corresponds to the ratio between their "Relative Weight" settings. By default, each subplot is assigned a weight of 1, meaning each subplot will share an equal percentage of space on the chart. In this example, we set **Subplot 1** has a weight of '2', and **Subplots 2 and 3** have a weight of '1'. Subplot 1 is going to be 2 times larger than Subplots 2 and 3.



7. Now that we have subplots, we'll put each of the different pens into a different subplot. Click on the **Pens** tab, select the row for the Ramp0 Pen, and click the **Edit** icon. For this example, we chose the following settings:
- Chose **Subplot 1**.
 - Set the color to pink.
 - Set the style to **Area**.
 - Click **OK**.

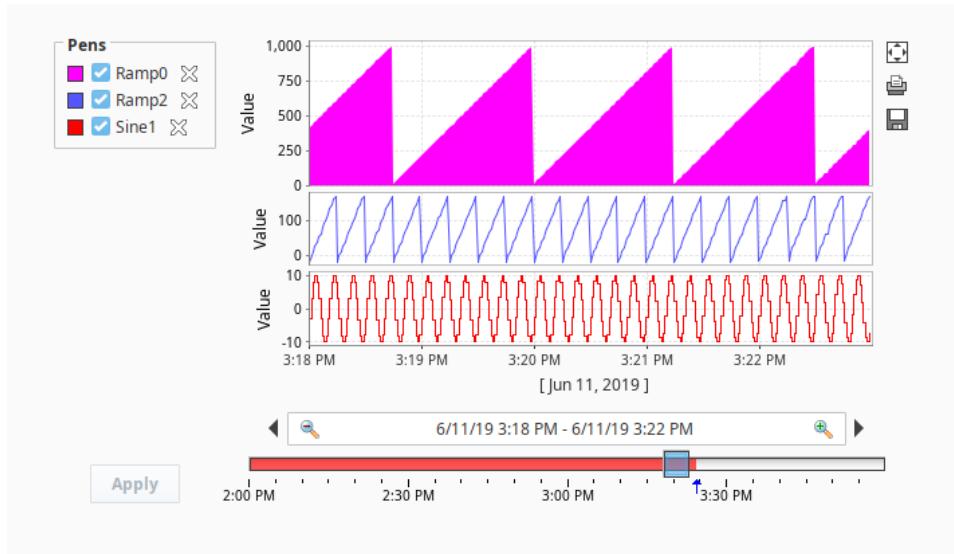


8. For the Ramp2 pen, we selected the following:
- **Subplot:** 2
 - **Color:** Blue
 - **Style:** Line w/Gaps
9. For the Sine1 pen, we selected the following:
- **Subplot:** 3
 - **Color:** Red
 - **Style:** Digital



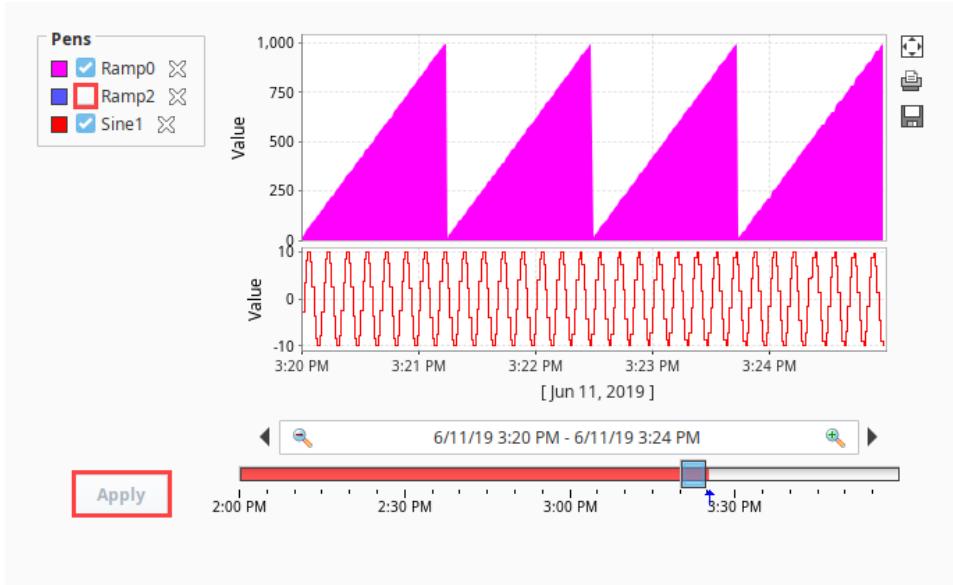
10. Click **OK**.

11. Now, you have three distinct subplots on one Easy Chart. You are not limited to the number of subplots on one Easy Chart.



12. You can be selective about what subplots you want to view. Go to **Preview Mode**, uncheck the Pens you don't want to see, and click **Apply**.

13. Notice how the Ramp2 pen is unchecked and is no longer displayed on the Easy Chart. To add the Ramp2 pen back to the Easy Chart, check the Ramp2 pen, and click **Apply**. The Easy Chart only displays subplots that have active Tags.



Related Topics ...

- [Easy Chart Customizer](#)
- [Easy Chart - Pen Names and Groups](#)
- [Easy Chart - Pen Renderer](#)

Easy Chart - Pen Names and Groups

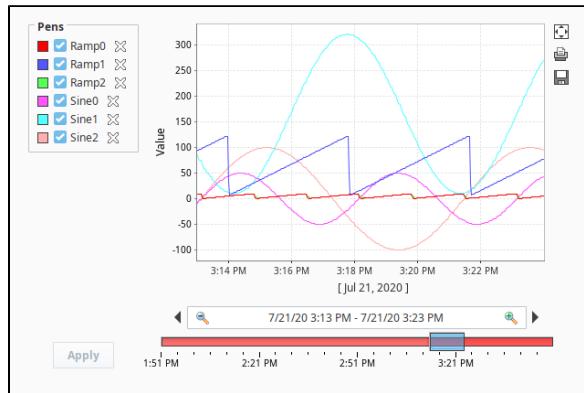
i This section assumes that Tags and Tag History have been configured

To learn more, go to the [Tag](#) and [Configuring Tag History](#) pages.

The examples below use OPC Tags from the [Programmable Device Simulator](#) driver, but Memory Tags can be used instead.

Pen Names and Group Overview

You can organize pens on the Easy Chart by creating custom names and groups for each pen. By default, when you drag Tags from the Tag Browser on to the Easy Chart component, the pen name is the same as the Tag name and organized into a single group called 'Pens.' One of the great things about pens is you can change pen names and organize pens into different groups making it easier for the operator to quickly analyze the data.



Configuring Pen Names and Groups

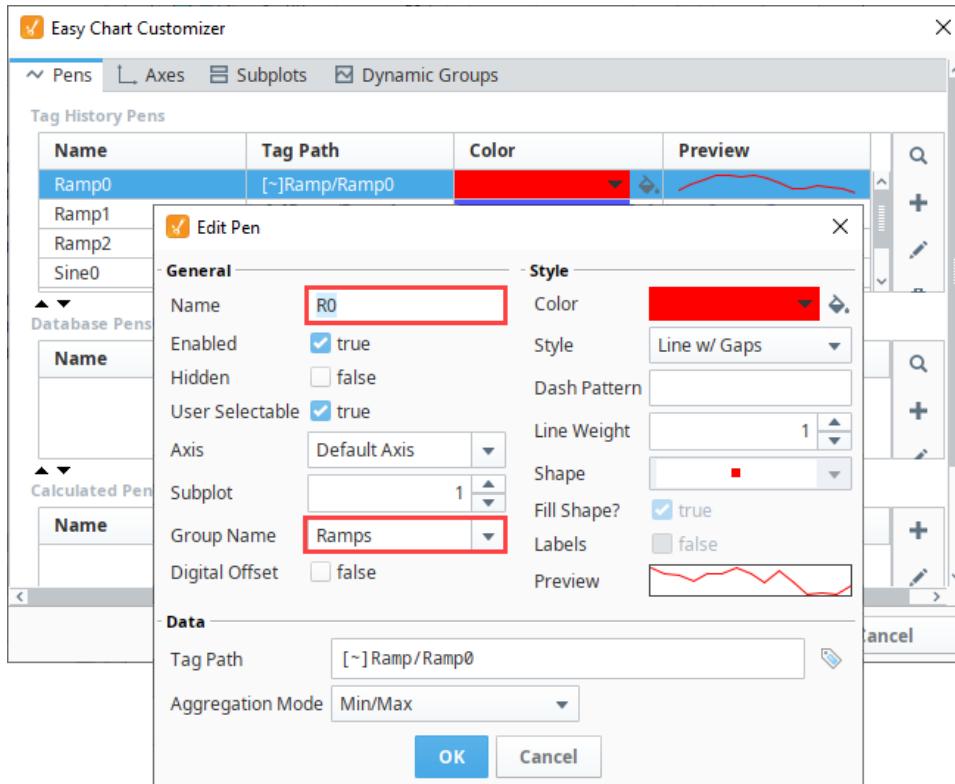
For each Tag on the Easy Chart, let's create unique pen names and organize each pen into a group using the Easy Chart Customizer.

1. Right click on the Easy Chart component, and choose **Customizers > Easy Chart Customizer**.
2. On the Pens tab, we'll edit each of the individual pens and give them a different name as well as put them inside a different group.
 - a. Select the first pen row, click the icon, and rename the first Tag from 'Ramp0' to 'R0' as shown in the following example.
 - b. At the bottom of the Edit Pen window, next to **Group Name**, create a new group called 'Ramps.'
 - c. If other groups exist, you can select one from the dropdown list or enter your own. In this example, this is the first group to be created, so simply type 'Ramps,' and **press OK**.
3. Repeat this step for each Ramp pen assigning each pen a new name.



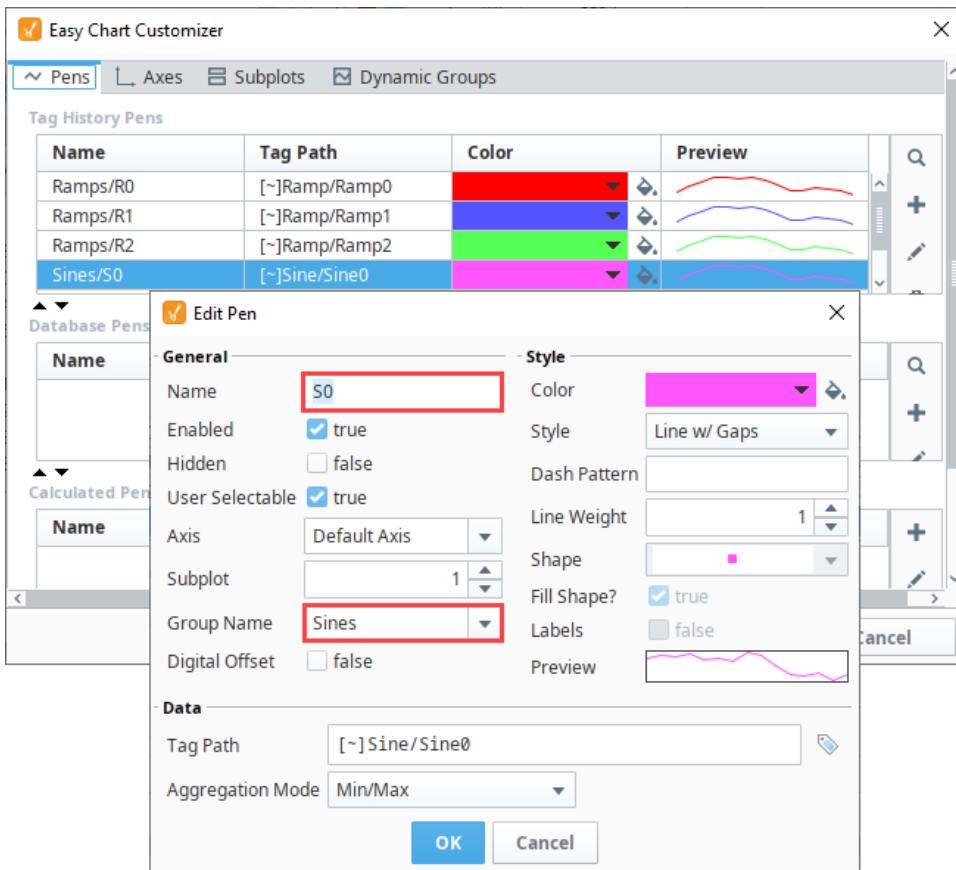
Easy Chart - Pen Names and Groups

[Watch the Video](#)

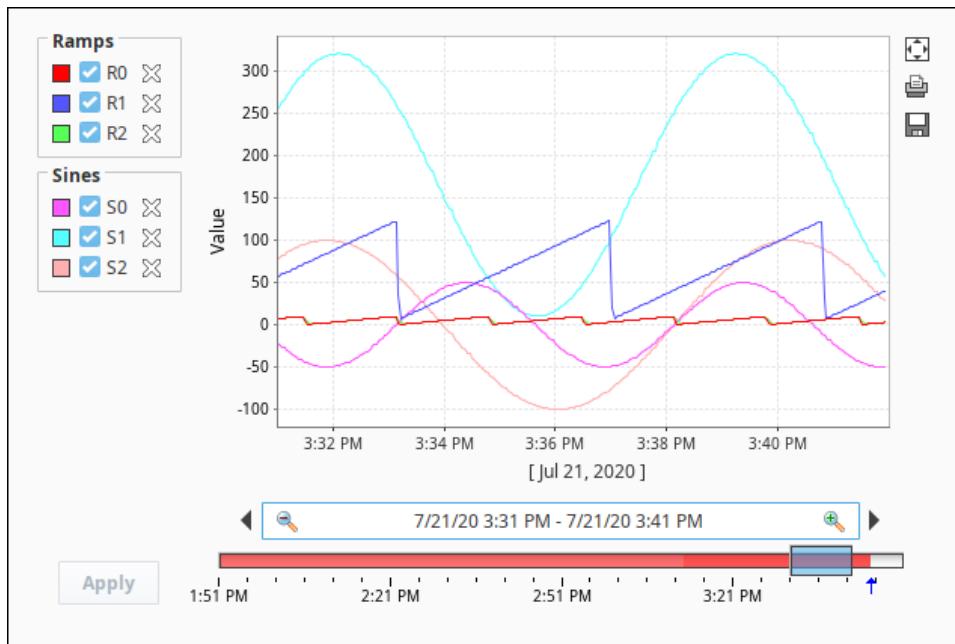


4. Next, let's keep the Sine pen names the same as the Tag name, but add them to a group.

- Select the **Sine0** pen row, and click the **...**
- Enter a new group name called '**Sines**'.
- Click **OK**.
- Repeat this steps a and c. to add each Sine pen to the Sines group.



5. Once you have all your pens configured, click **OK**. Operators will see the Pen and Group names organized into two legends on the Easy Chart. You can update the pen and group names on this view by double clicking on any of the fields. It's a faster way to edit this information than having to go to the Edit icon for each pen.



Tip for Viewing Specific Pen Values

To see specific Pen values on the Easy Chart, uncheck the Pen and click the Apply button. To see all Pen values, make sure all Pens are checked, and hit the Apply button.

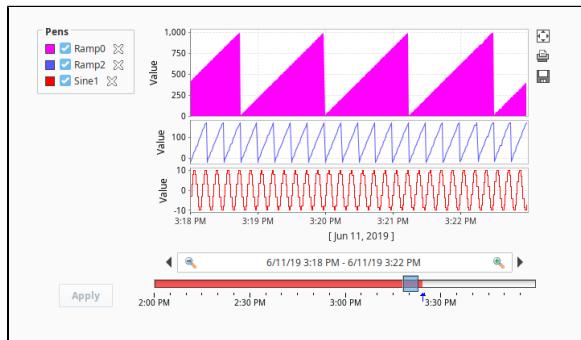
Related Topics ...

- [Easy Chart Customizer](#)
- [Easy Chart - Pen Renderer](#)

Easy Chart - Pen Renderer

Customizing the Pen Renderer

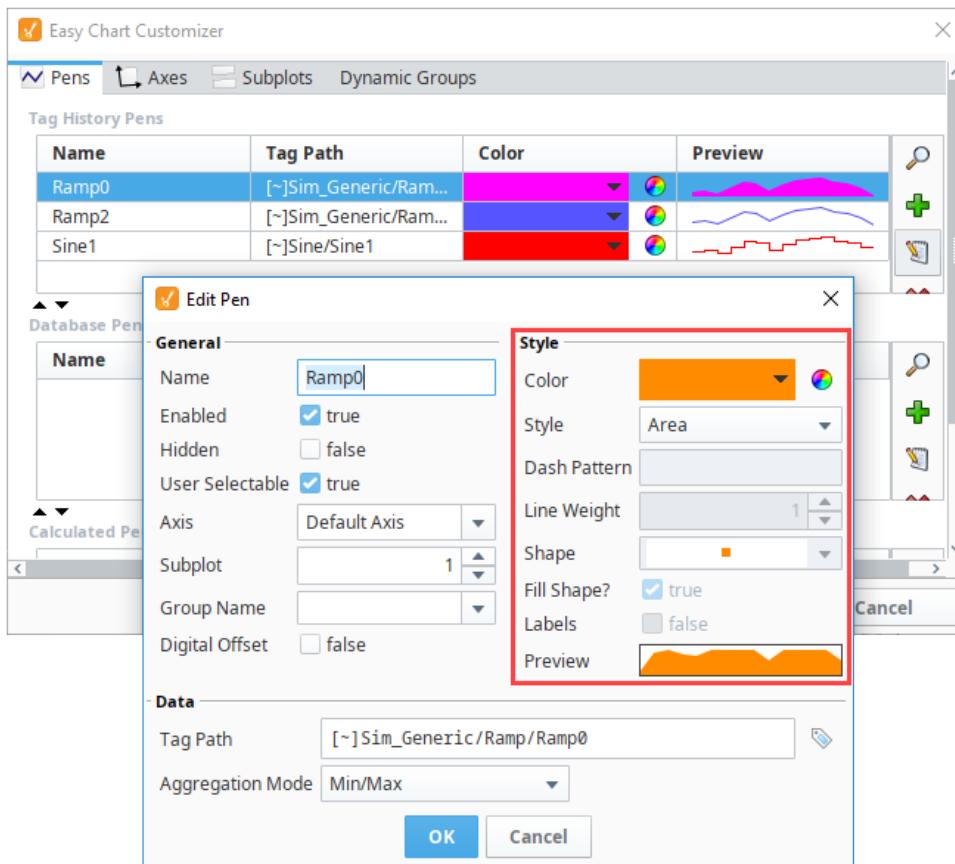
You can customize the renderer of each pen on the Easy Chart to change its style, shape, weight and color. When adding Tags to an Easy Chart component, the default renderer or style of each pen is a simple line, but it can be customized for each pen.



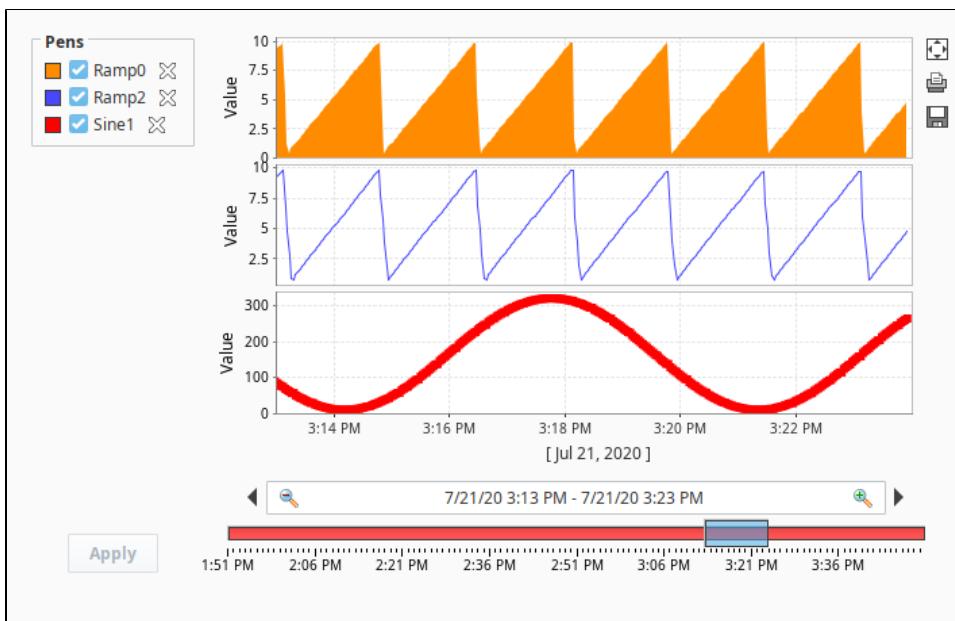
The image shows the Inductive University logo, which consists of a blue hexagon with 'IU' in white and a green laurel wreath. To the right, the text 'INDUCTIVE UNIVERSITY' is written in a bold, sans-serif font. Below it, 'Easy Chart - Pen Renderer' is displayed in a smaller bold font, followed by a 'Watch the Video' button.

In this example, we will use the Pen Renderer to customize several pens. What's nice is the Easy Chart Customizer allows you to preview the style, shape, weight and color. If you don't like it, you can easily change it.

1. Click on the Easy Chart component, and scroll down to **Customizers > Easy Chart Customizer**.
2. From the **Pens** tab, select the pen row, and click the **Edit** icon. On the right side of the Edit Pen window under Style, you can change the line color, style, weight, and shape. The line style, by default, is 'Line with Gaps,' which means that if you lose communication to the PLC or don't have data for a particular time period, you will see a gap. You can change the Style from 'Line w/ Gaps' to any other style type listed in the dropdown that fits your needs and preferences.
3. Edit each style property. If you don't like the result in the Preview window, select another style. You can edit it as many times as you like. When you're finished, press **OK**.



4. Repeat steps 2 and 3 to customize the style for all the pens you want to change. When you're finished, **press OK**.
5. The pens on the Easy Chart now reflect your style changes.



[Related Topics ...](#)

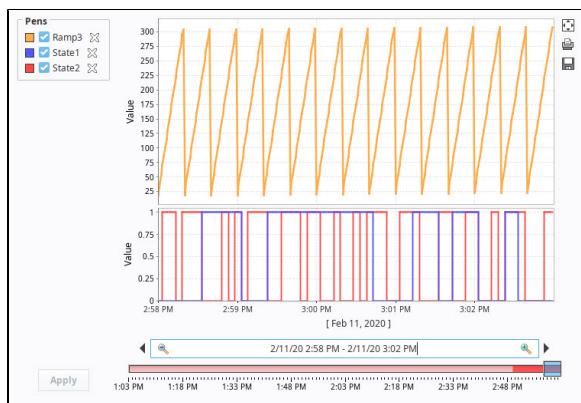
- [Easy Chart Customizer](#)

- Easy Chart - Digital Offset

Easy Chart - Digital Offset

Digital pens often share the same subplot on the [Easy Chart](#) component. When you have multiple digital pens on the same Easy Chart subplot, it's hard to see what the values are of each pen because they may overlap each other. There is a digital offset pen setting that can be set which prevents the values from overlapping and enables them to be seen better in the subplot.

The following example shows two digital pens on the same subplot: State1 and State2. The values are a little difficult to see because they are on top of each other.



On this page ...

- [Adding a Digital Offset](#)



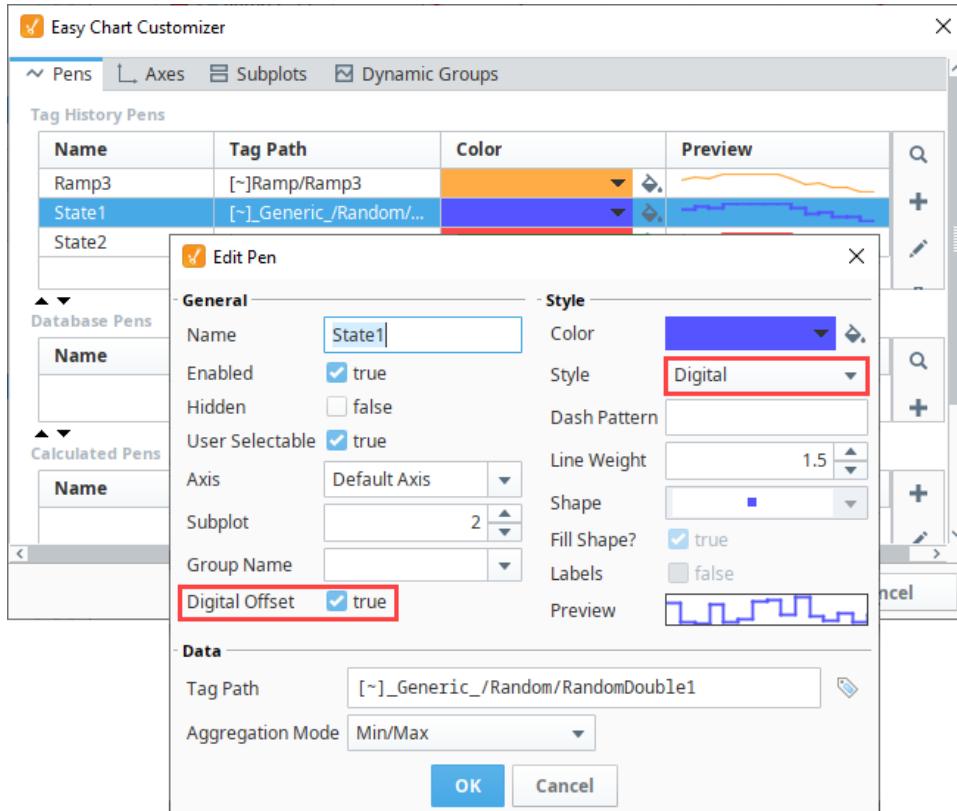
Easy Chart - Digital Offset

[Watch the Video](#)

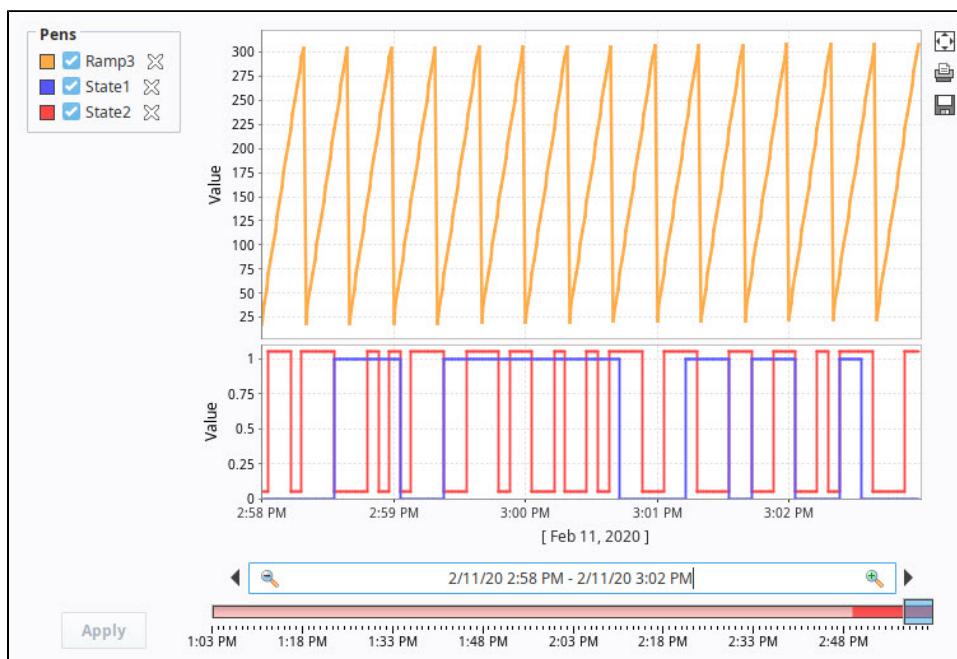
Adding a Digital Offset

In this example, we will apply a digital offset on the pen renderer so you can see the values better.

1. Right click on the Easy Chart component and choose [Customizers > Easy Chart Customizer](#).
2. On the Pens tab, select the **State1** row pen, and click the icon.
3. Set the **Style** is set to 'Digital'.
4. Set the **Digital Offset** to 'true'. Click **OK** to save the changes to the pen.



5. Repeat steps 2-4 for the **State2** pen.
6. When you're finished editing your pens, click **OK** to return to the Easy Chart. You will see a little offset in the values between the State pens in the 2nd subplot so they don't overlap each other, making it a lot easier for the operator to read the digital values.



Related Topics ...

- [Easy Chart Customizer](#)
- [Easy Chart - Calculated Pens](#)

Easy Chart - Calculated Pens

Calculated Pens

This section assumes that Tags and Tag History have been configured

To learn more, go to the [Tag](#) and [Configuring Tag History](#) pages.

The examples below use OPC Tags, but Memory Tags could be used instead.

Calculated pens display a value that is dynamically calculated based on another pen. This can be used to calculate certain values for a pen and graph them alongside the original pen values, allowing you to gain valuable insight into your data. There are many unique calculations that can be used, with some of them containing unique customization. Almost all of the Calculated pens require a driving pen, which is a tag or database pen that you have already setup.

Note: You cannot bind the Calculated Pen values inside the Easy Chart Customizer. To bind the function values, use the [Cell Update Binding](#)

On this page ...

- [Calculated Pens](#)
- [Calculated Pen Functions](#)
- [Configuring Calculated Pens](#)
- [Hide Driving Pens](#)



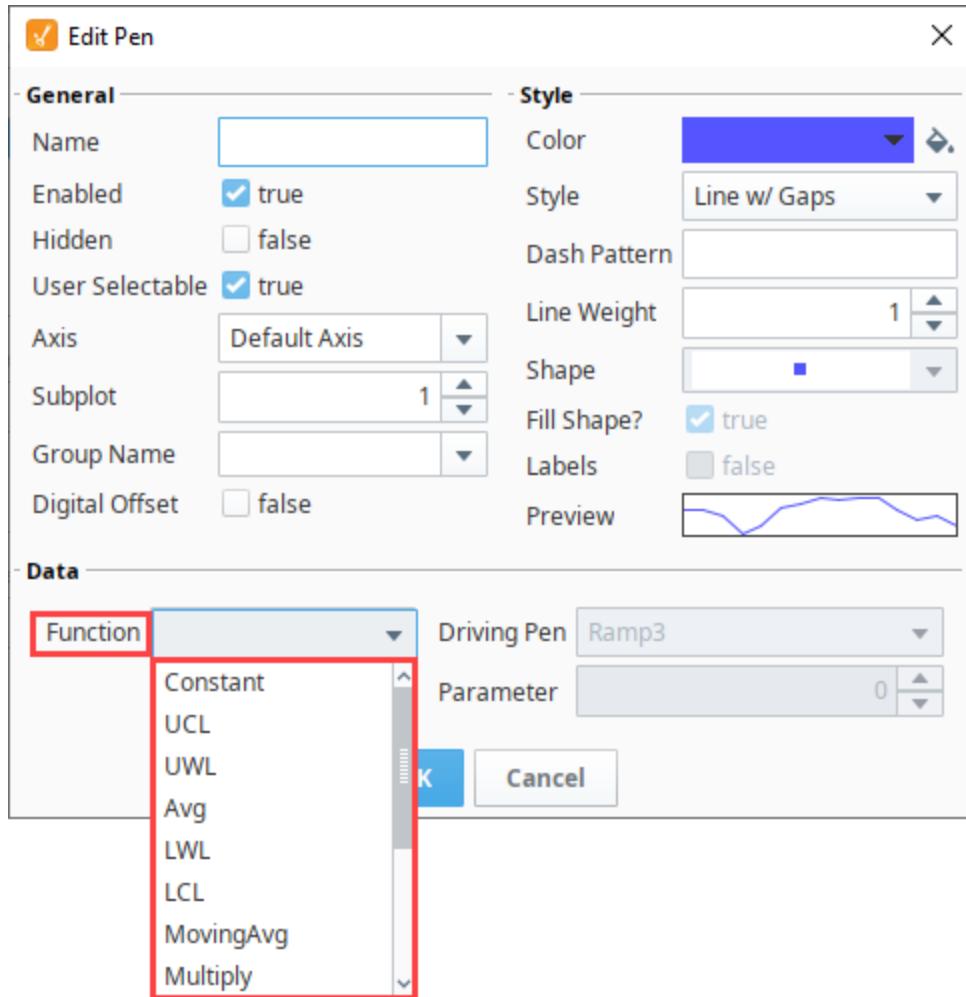
INDUCTIVE
UNIVERSITY

Easy Chart - Calculated Pens

[Watch the Video](#)

Calculated Pen Functions

There are a variety of functions that can be used to calculate the pen value. The pen functions are located on the Edit Pen screen in the Function dropdown list. The table below defines the available functions.



| Function | Display Name | Description | Extra Properties |
|---------------------|--------------|---|---|
| Constant | Constant | A constant value on the chart. | Constant Value - The constant value of the pen |
| Upper Control Limit | UCL | The upper control limit of the driving pen, which is three standard deviations above the mean of the Driving Pen. | |
| Upper Warning Limit | UWL | The upper warning limit of the driving pen, which is two standard deviations above the mean of the Driving Pen. | |
| Average | Avg | The average of the driving pen. | |
| Lower Warning Limit | LWL | The lower warning limit of the driving pen, which is two standard deviations below the mean of the Driving Pen. | |
| Lower Control Limit | LCL | The lower control limit of the driving pen, which is three standard deviations below the mean of the Driving Pen. | |
| Moving Average | MovingAvg | A series of averages based on subsets of the driving pen. The subsets are determined by the window size. | Window Size - The size of the moving average window, specified as a multiplier of the chart's date range. |
| Multiply Pen | Multiply | Multiply each data point of the driving pen by a factor. | Factor - The factor that each data point of the driving pen is multiplied by. |
| Minimum Value | Min | The minimum value of the driving pen. | |
| Maximum | Max | The maximum value of the driving pen. | |