

Ignition 7.9 User Manual — Designer, Clients and Visualization

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

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

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

Designer

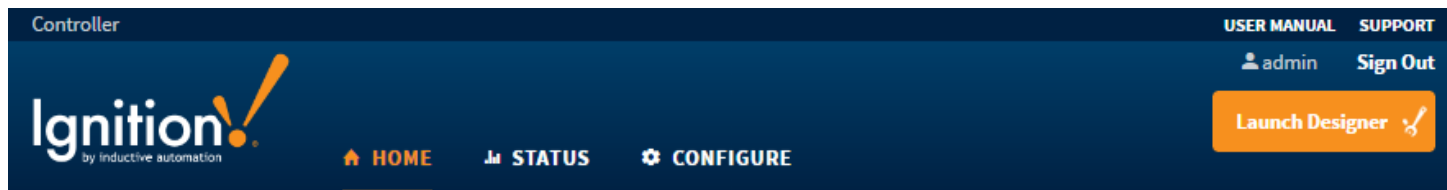
The Ignition Designer is where the majority of configuration and design work is done. Designer uses **web-launch** technology which is one of the best parts about Ignition. This is how you launch both the Designer (where you'll configure your projects) and the Ignition Vision Clients (where you'll view your projects). The Designer is included in the platform, so it is not a separate install and doesn't require any additional license.

The Designer provides all the firepower to bring your projects to life. It uses a drag-and-drop configuration making screen development quick and intuitive. You can create user interfaces by dragging **components on your window**, and Tags onto your components **to instantly bind data** to tables, charts, graphs, and more. You can set up Tags and **Transaction Groups** to log data to your databases. You can set up **Reports** to generate and save data however you'd like. The Designer saves all your projects to the Gateway so everything is controlled in one place.

Launching Designer

Web-launch technology lets you launch a full-fledged application with zero installation by only clicking a link on a webpage. All you need to do is install the Gateway. All your Designers and Clients do not need to be installed, and they are always kept up-to-date. Once you start using web-launched Clients, you'll wonder how you ever did without them.

The Designer is launched from the Gateway homepage by clicking on the **Launch Designer** button on the top-right of the Gateway webpage.

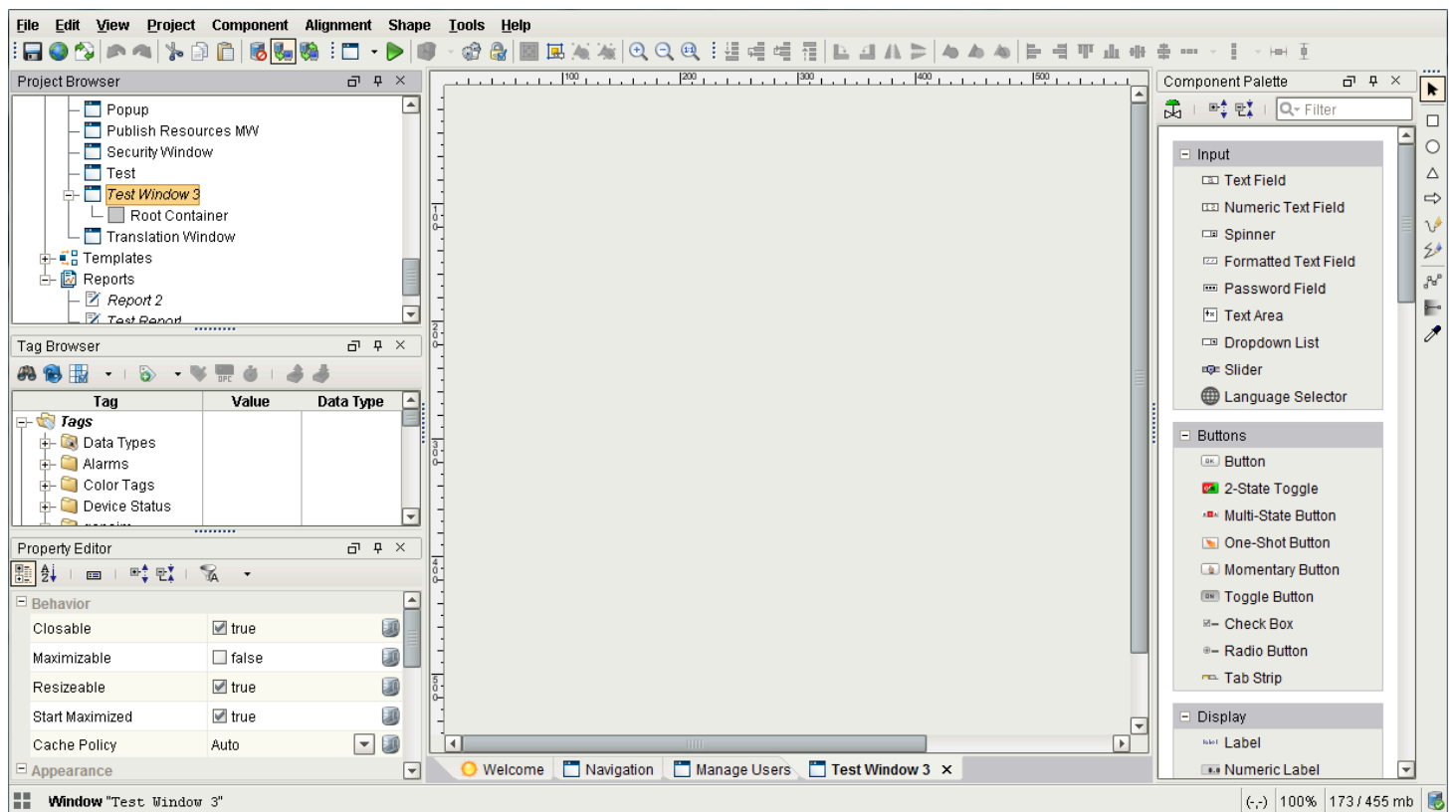


Once the Designer is started, you will be prompted with the login window. Enter your username and password. Unless it was changed by your System Admin, your default Username is '**admin**,' and default Password is '**password**.'

Once logged in, you can either open an existing project, or create a new project. To learn more, go to the section on [Launching Designer and Opening a Project](#). This section will give you lots of good information about naming projects, selecting from a variety of project templates to use for your project, and also provide troubleshooting tips for web-launch problems.

Designer's Workspace

The **Designer workspace** is centrally located and organized by panels. Some of the panels include a Project Browser, Tag Browser, Property Editor, and Component Palette. These panels can change depending on the type of resource you are currently editing. For example, if you are editing a Window, your workspace will be the Window Designer. If you're editing an Alarm Notification Pipeline, your window will be the Pipeline Block Editor. If you are editing a Report, your window will be the Report Designer. Each type of workspace has panels that are only valid when that workspace is active.



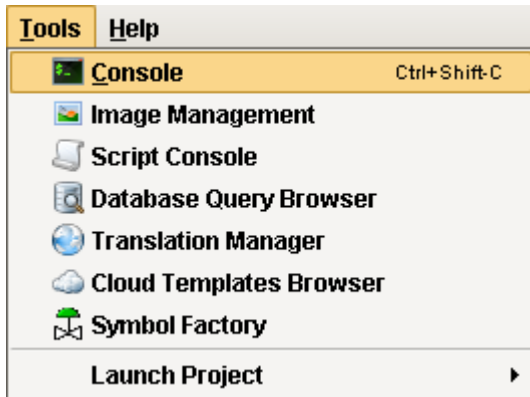
Designer Tools

Designer has a host of tools to help you accelerate building, testing, and deploying your project.

Tools Menu

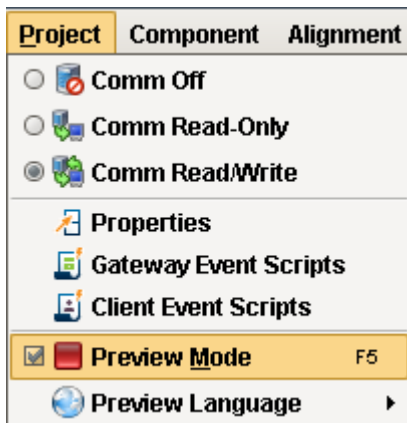
In addition to all the panels available in the Designer workspace, there is also a **Tools menu** to help you create your projects.

- **Console** - used to test and debug Python scripts.
- **Image Management** - manages and stores images used for your projects.
- **Symbol Factory** - (separate module) contains a variety of high quality vector graphics symbols that can bring your projects to life.



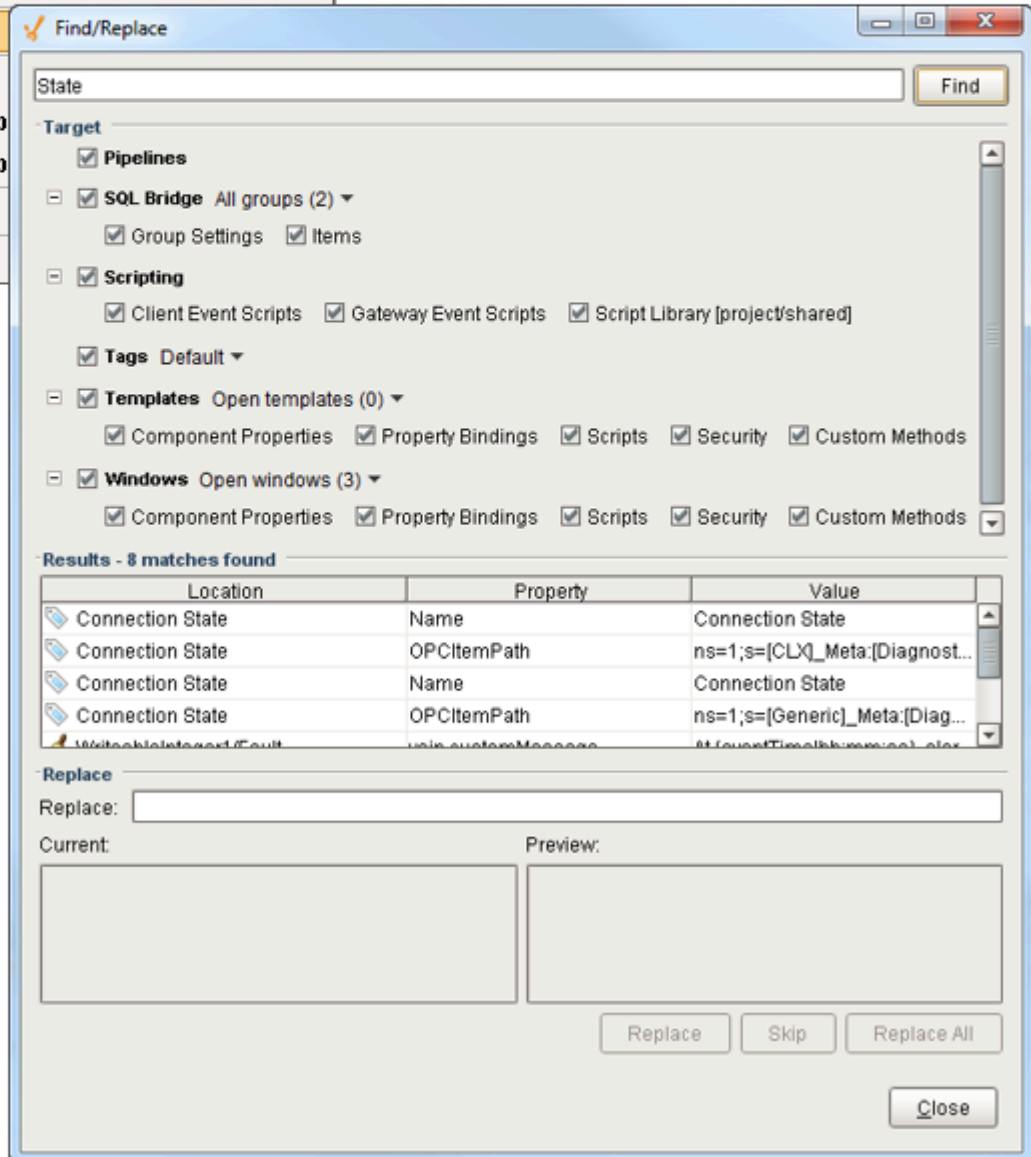
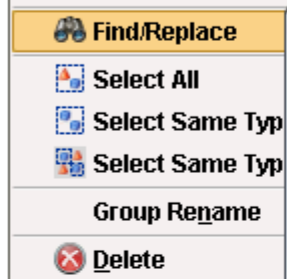
Previewing the Project

Designer provides the capability to preview, test, and interact with the screens and functionality before you deploy your project. The Designer workspace operates in two distinct modes: design mode and **preview mode**. Designers can easily switch between these modes to make sure their project is working as expected during the course of development. Simply select **Project > Preview Mode** to test and interact with the functionality of your project.



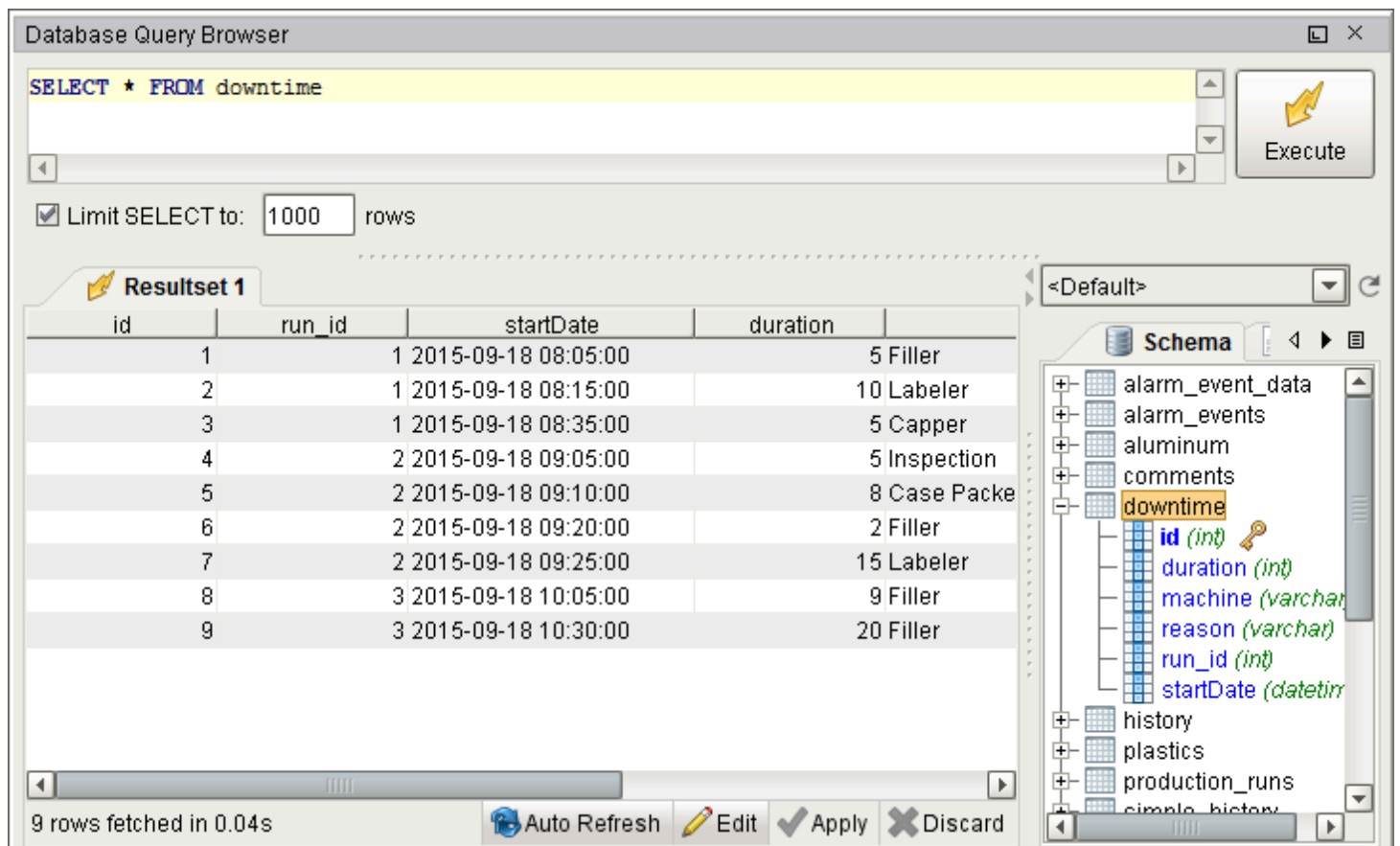
Find and Replace

The **Find and Replace** is a handy tool in the Designer workspace. You can search your entire project for specific components, properties, scripts, and more. You can even use the replace command to make mass changes expeditiously to a project with very little effort.



Query Browser

The **Database Query Browser** is a very convenient tool that lets you make simple edits in a database table, and interact with all of the databases that Ignition is connected to (i.e., running queries, browsing tables and schemas). It is very common during the course of project design to inspect the database directly, or to experiment with a SQL query to get it just right.



Keyboard Shortcuts

There are a lot of ways to speed up your development once you are familiar with how Ignition works. There are many **keyboard shortcuts** in Designer that are listed throughout the Designer interface alongside menu options.

Cut	Ctrl-X
Copy	Ctrl-C
Paste	Ctrl-V
Delete	Delete
Group	
Ungroup	
Convert to Container	
Lock	
Layout...	Ctrl-L
Size & Position...	Ctrl-P
Customizers	
Scripting...	Ctrl-J
Security	Ctrl-E
Translations	Ctrl-T
Run Diagnostics	

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

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

Launching Designer and Opening a Project



INDUCTIVE
UNIVERSITY

Launching Designer and
Opening a Project

[Watch the video](#)

Getting Started with the Designer

Web-launching is one of the best parts about Ignition. This is how we launch both the Designer, where you'll configure your projects, and the Ignition Vision Clients, where you can view the projects.

Web-launch technology lets you launch a full-fledged application with zero installation just by clicking a link on a webpage. This means that with Ignition, you'll only need to install the Gateway. All of your Designers and Clients do not need to be installed and they are always kept up-to-date.

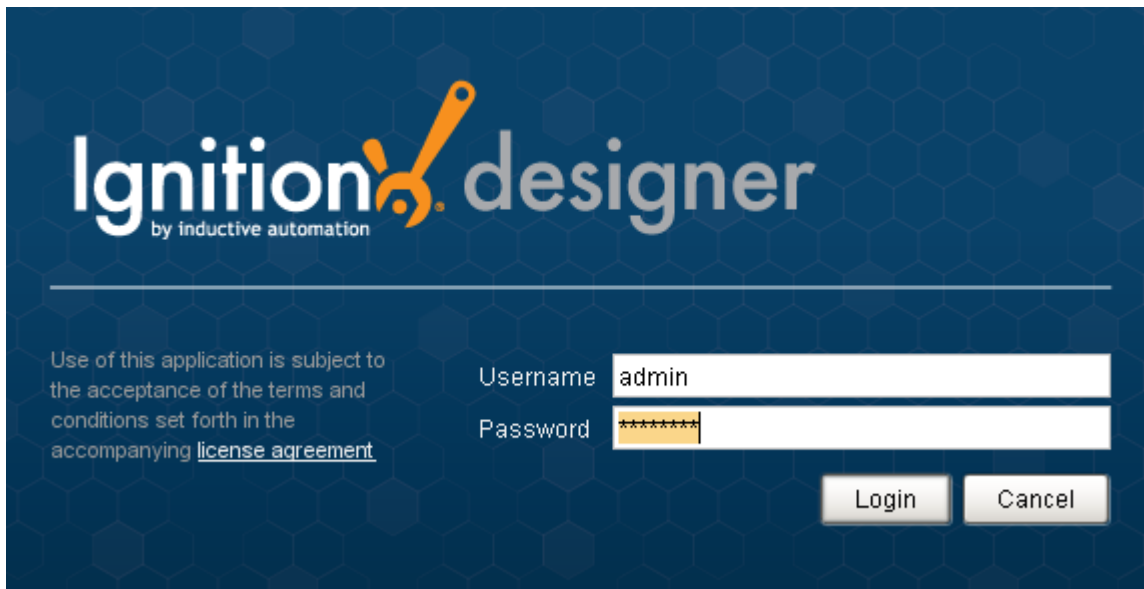
! INFO

Before you can successfully web-launch the Designer, make sure you have Java installed.

Launching the Designer

There are a few simple steps to follow to open the Designer that will help you get started with Ignition.

1. Click **Launch Designer** on the upper-right of any Gateway page. Depending on your web browser, it will ask you if you want to save or run a **designer.jnlp** file.
2. Run the **designer.jnlp** file to start the Designer. The Designer starts up and the login window is displayed.



3. Enter the default username and password for the Designer which is the same for the Gateway Configuration section.

Username: **admin**

Password: **password**

INFO

This assumes that this username and password were not yet changed. If this username and password does not work, then the default username and password are no longer valid. Please refer to Security to understand further how Ignition manages its users and their passwords.

4. Click on **Login**. The **Open/Create Project** window is displayed giving you two options, to create a new project or open an existing one. If you don't have any existing projects yet, we will create a new project.

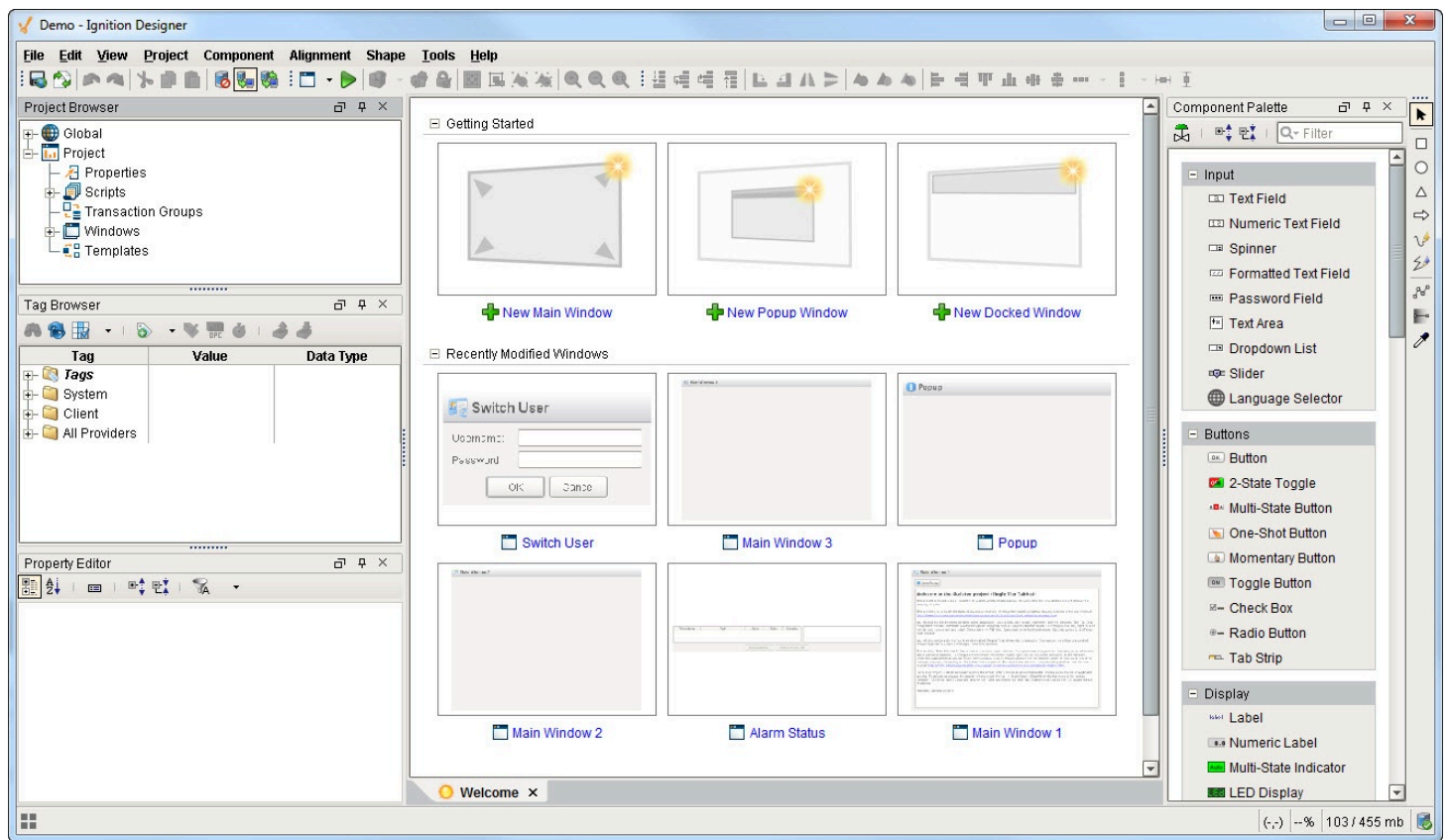
The screenshot shows the 'New Project Setup' window in Ignition. On the left, there is a sidebar with 'Create New' (containing 'New Project') and 'Open Recent' (listing 'My First Project', 'Training', 'Training 2', 'Training3', and 'Training 4'). The main area is titled 'New Project Setup' and contains the following fields:

- Project Name:** A text input field with a red warning icon to its right.
- Project Title:** A text input field.
- Authentication Profile:** A dropdown menu with 'default' selected.
- Default Database:** A dropdown menu with 'MySQL' selected.
- Default Tags Provider:** A dropdown menu with 'default' selected.
- Project Template:** A dropdown menu with 'Blank' selected.
- Description:** A large text area.

At the bottom center of the main area is a button labeled 'Create New Project'.

Inductive Automation logo is visible in the bottom right corner of the interface.

5. In **Project Name**, enter a name for your new project (no spaces!).
6. For **Project Template**, select a template from the drop-down list, for example, you can select the **Single-Tier Nav, West & North** template. Rather than starting from a blank project, the templates let you start from one of the skeleton projects that already have navigation windows setup. These projects are completely customizable and are designed to help you get started faster.
7. Click on **Create New Project**. The Designer is now launched and you can work on your project.



Once in the Designer, you can create or open a project using the Designer's **File > Open** menu. The **Open/Create Project** window is displayed and you can follow the same steps as above to open existing projects or create a new one.

Troubleshooting

Sometimes, the Designer may not launch properly. There are a few things that you need to understand and can try to fix any issues encountered when launching the Designer.

What is Web-Launching

Web-launching is the mechanism by which Clients and the Designers are opened on a computer. They are launched from the Ignition Gateway. You download and run them without requiring any installation steps.

How Web-Launching Works

Web-launching relies on Java Web Start technology. When the user clicks on a project or the Designer link in the Ignition Gateway (or embedded in a separate website), they download a small JNLP file that describes the application. When this file is opened (usually automatically), Java is invoked on the user's machine and directed to the remote application. The application is downloaded and cached, and then executed. The running application (an Ignition Client or the Designer) communicates with the Gateway via HTTP. It is cached for increased subsequent launch speed, and

can optionally install a link in the Start menu and on the desktop for easy access. By default, a shortcut is created for you, but this setting can be changed in the Java Control Panel.

Troubleshooting Web-Launch Problems

There are a few common problems that can cause difficulties with web-launching. Fortunately, they are often easy to fix.

No Java Installed

Before you successfully web-launch the Designer, make sure you have **Java 8** installed, see <http://www.java.com>.

If you're on the computer that's running the Ignition Gateway, you already have Java installed, the Ignition installer makes sure of that. If you're on a computer that is accessing the Gateway over the network, you can check to see if Java is running by going to the **Status** page and selecting **Systems > Overview**. The Java version is shown under the **Environment** table.

No Network Connection

Web-launched clients depend on network connectivity to connect to the server. If the network is unavailable, the client cannot be launched. This is often a problem with clients and the Designers launched directly from desktop/start menu links.

Cached References to Modified Servers

The cached projects/launch shortcuts contain the address of the Gateway machine. If the server is relocated, these links will no longer work. They can be updated by re-launching from the Gateway.

Still Having Problems?

You can also see [Native Client Launchers](#).

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

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

Project Templates



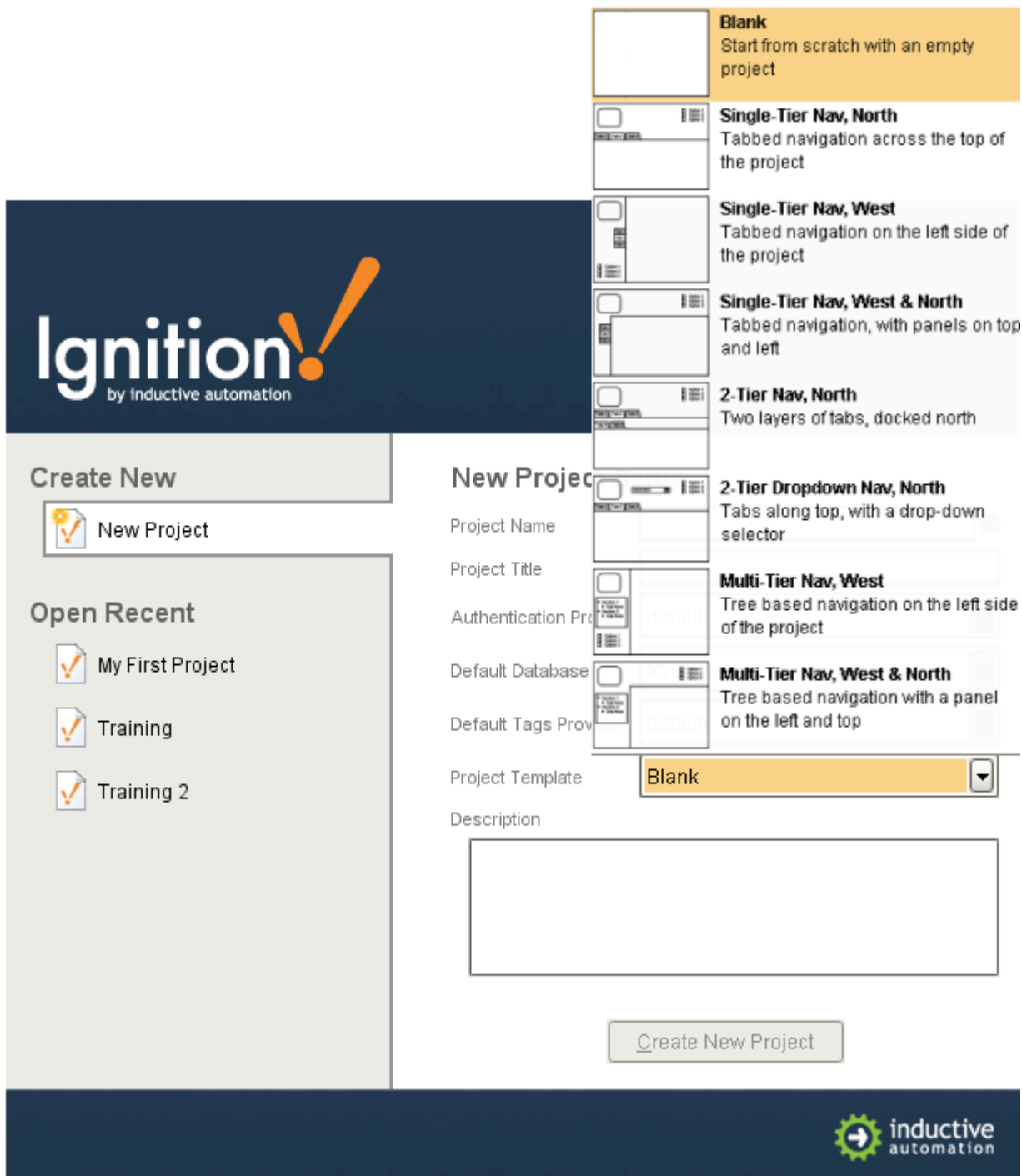
INDUCTIVE
UNIVERSITY

About Project Templates

[Watch the video](#)

Getting Jump Start with Project Templates

When creating a **New Project** in the Ignition Designer, you are given the option to choose from a number of project templates in the **Project Template** field of the **Open/Create Project** window. Project templates are sets of pre-built windows mainly focused on navigation. They provide an easy way to quickly get started with different windows in the project instead of worrying about navigation. They can be used as is, or modified to fit the needs of the project.



Different Types of Project Templates

- **Blank (Default)**
Start from scratch with an empty project.
- **Single-Tier Nav, North**
Tabbed navigation across the top of the project.
- **Single-Tier Nav, West**
Tabbed navigation on the left side of the project.
- **Single-Tier Nav, West & North**
Tabbed navigation on the left side of the project with a header.

- **2-Tier Nav, North**

Two layers of tabs docked north of the project.

- **2-Tier Dropdown Nav, North**

Tabs along the top of the project, with a dropdown selector to swap tab sets.

- **Multi-Tier Nav, West**

Tree-based navigation on the left side of the project.

- **Multi-Tier Nav, West & North**

Tree-based navigation on the left side of the project with a header.

- **Mobile Project**

A project template optimized for a small Mobile device (ie: small windows and large buttons).

Note: You don't have to use this template to launch in a mobile device. If you have the Mobile module, you can use all of the project templates with your mobile device.

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

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

Designer User Interface



INDUCTIVE
UNIVERSITY

Using Designer User Interface

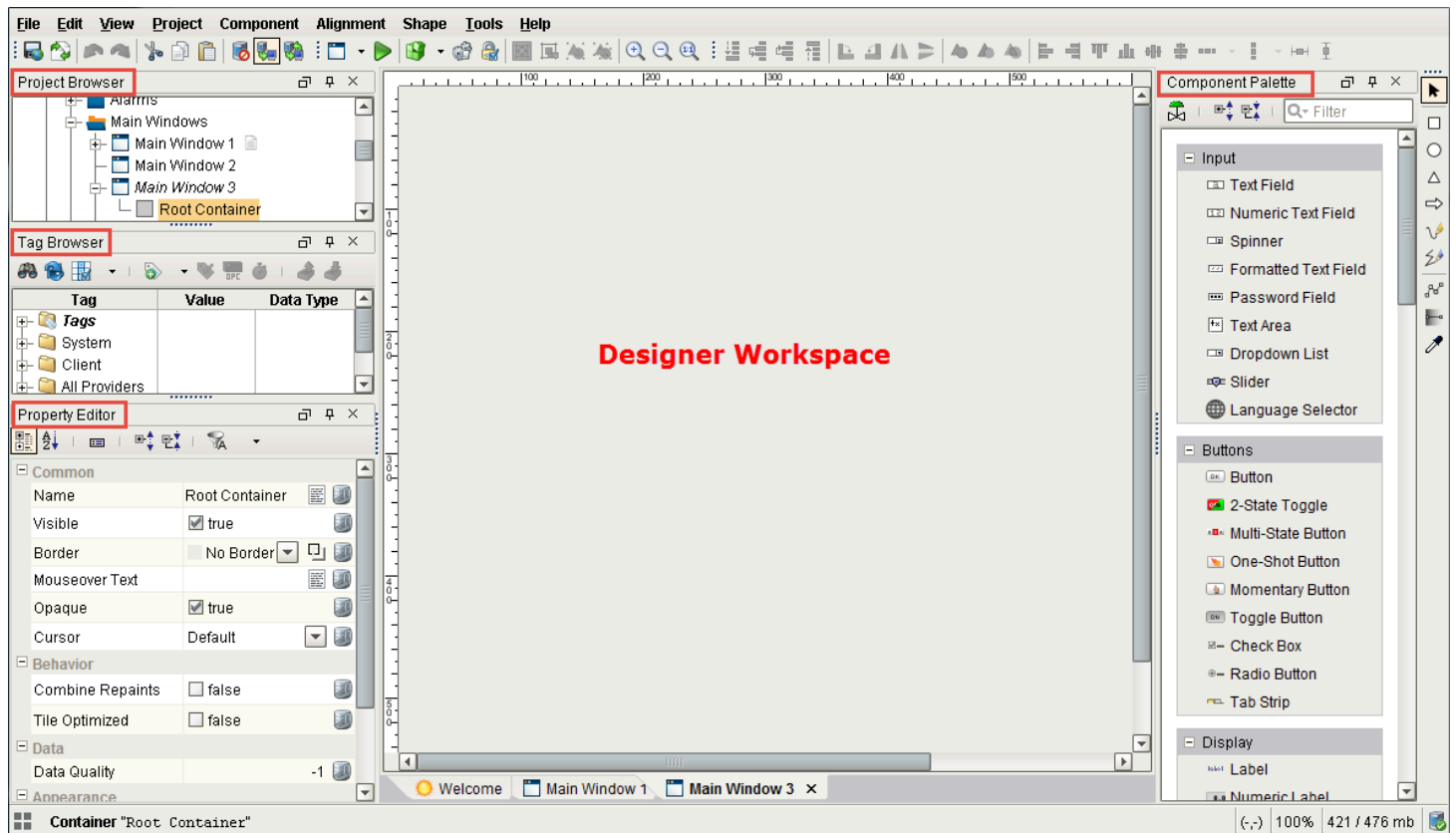
[Watch the video](#)

Designer Workspace

The Designer is organized in panels around a central workspace. Some of these panels include: Project Browser, Tag Browser, and Property Editor. The workspace changes depending on the type of resource that you are currently editing. For example, if you are editing a Window, your workspace will be the Window Designer. If you are editing a Transaction Group, your workspace will be the Transaction Group Editor, and so on.

There are many dockable and draggable panels that surround the workspace, as well as the familiar menu bars and toolbars. The dockable panels can be rearranged as you wish and will snap into place as you move them around the screen. Each type of workspace can have panels that are only valid when that workspace is active. Each workspace remembers its own perspective, which is the docking arrangement of the panels around it.

If you have closed a panel and want to get it back, re-enable it from the View > Panels submenu. The View > Reset Panels will reset the user interface back to its default window arrangement.



Docking System

The Designer's docking system provides a very flexible user interface, allowing you to customize the layout as you wish. To re-arrange the dockable panels, simply drag on their title bars. As you are dragging the panel, it will try to snap into place and show you a highlighted border. Use the highlighted border that appears to gauge where the panel will be moved to. Hold the CTRL key as you drag these panels to keep them from snapping into place. You can also drag these panels outside of the designer or onto a second monitor.

Dockable panels can be in one of four modes:

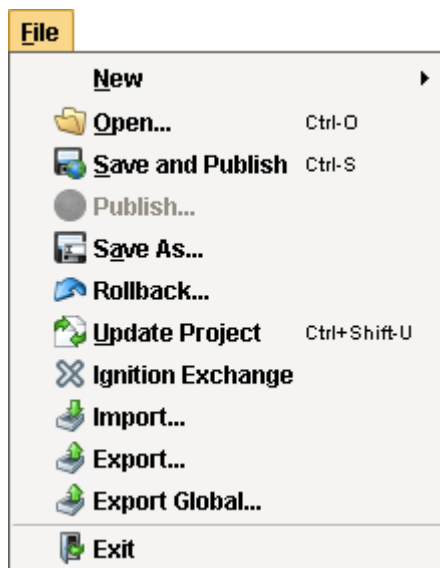
1. **Docked** - The panel is visible, and located somewhere around the perimeter of the workspace. If two panels are docked in the same location, a tab strip will appear to switch between the two panels.
2. **Floating** - A panel can be dragged outside of the workspace perimeter to be floated. The panel can now be positioned anywhere on your desktop.
3. **Pinned** - Pinning a panel makes it minimize to one of the four sides of the Designer, represented by a small tab. Hover over the tab to use the panel.
4. **Hidden** - A hidden panel is not shown. You can open it again by selecting it in the **View > Panels** menu.

Toolbars can also be rearranged and floated to your liking. Simply drag on the "textured" left edge of the toolbar. If you have re-arranged your panels into a layout that you don't like, you can quickly revert back to the default by selecting the **View > Reset Panels** option from the menu bar.

 TIP

Your docking preferences are stored under `%USER_HOME%/.ignition/*.layout`. If you really want to reset your preferences, remove these files and restart the Designer.

File Menu



The File Menu contains most of your basic options related to saving, similar to many other software applications on your PC.

Function	Description
New	The New menu option allows you to create a new Object. That Object could be a new window, a new template, or even a new project. It can vary depending on what modules you have installed, also providing the ability to make new transaction groups, reports, Sequential Function Charts, and more.
Open	Allows you to Open a previously created project.
Save and Publish and Rollback	By default, saving and publishing are all done together. However, this can be changed to allow for just saving, where the changes will be saved but not pushed out to clients, and just publishing, where the new saved changes will be pushed out to the client. Rollback will allow you to 'roll back' to one of the 10 most recently saved versions of the project. See also: Versioning
Update Project	Merges any changes from the gateway to the project, if for example there were multiple people working on the same project.
Ignition Exchange	New in 7.9.13 Opens the Ignition Exchange website.
Import and Export	Allows you to import or export specific resources into or out of the project. Export global will export globally scoped resources, such as Alarm Pipelines.

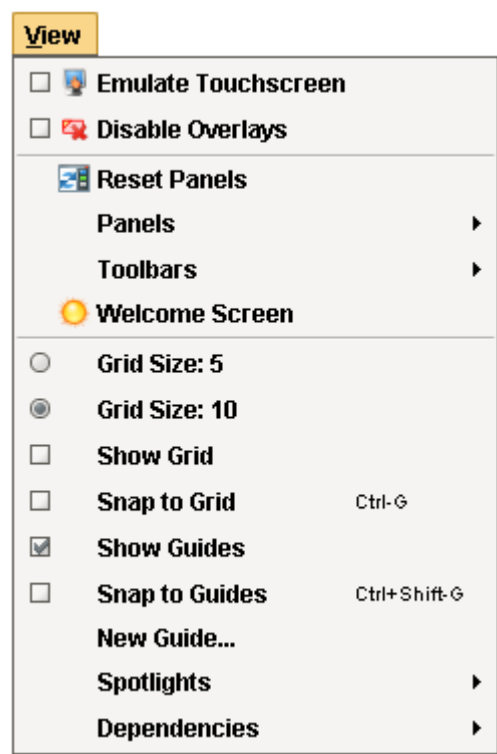
Edit Menu

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

The Edit Menu is also similar to other applications edit menus, in that it provides much of the basic copy/paste functionality.

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. ie: Tag edits cannot be undone.
Cut/Copy/Paste/Duplicate	These function 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, 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
Fine/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

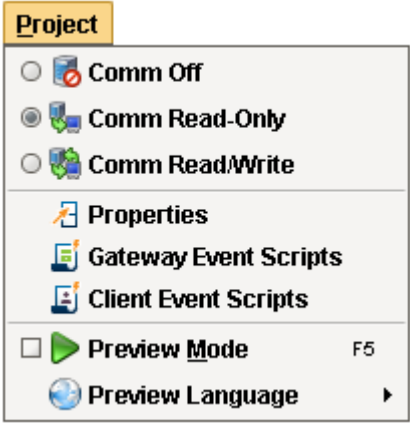


The View Menu allows you to manipulate how various objects look or act in the designer.

Functions	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

Functions	Description
	all components that are bound to the currently selected component, and Show All will show all of the bindings, regardless of the selected components.

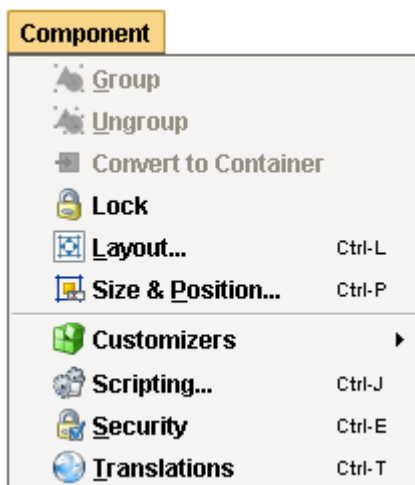
Project Menu



The Project Menu is where many project specific settings can be changed.

Functions	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 turn 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 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 and Languages

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.

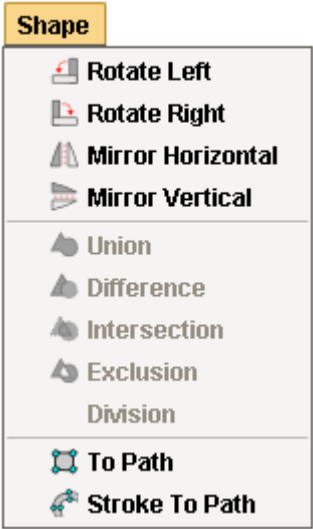
Alignment Menu

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	

These options allow you to adjust the alignment of components relative to other components.

Function	Description
Move	The move commands move a component on its z axis, as in in front of or behind other components. It can be moved to the far back or front, or done in single steps.
Align	The align commands move all selected components to the edge of the furthest component. So if Align Left was selected, it would move all components to the leftmost component's edge.
Align Centers	Aligns all of the selected components either vertically or horizontally on their centers.
Align as Row	Aligns all of the the components on their centers as a row or 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	Centers the currently selected components relative to their parents either vertically or horizontally.

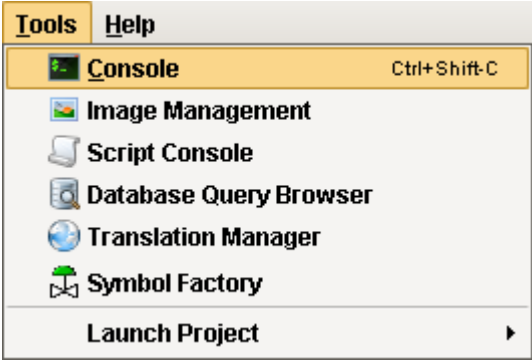
Shape Menu



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 consiting of the area they share.
Exclusion	Alters the first shape to become a new shape consiting 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	Converts the selected shape into a new shape defined by its stroke.

Tools Menu



The **Tools** menu in the Designer provides some tools to help you when creating projects. The **Console**, **Image Management**, and **Symbol Factory** tools are described here.

Function	Description
Console	The Output Console is the script-writers best friend. It is a dockable panel, and you can open it via the Tools > Console menu or the Ctrl-Shift-C keyboard shortcut. The output console is most frequently used to test and debug Python scripts in Ignition. For example, by using the print function in your script, you can observe the inner workings of your script as it executes. If you executed the following script: see Console - Example Script The output console is also available in the Vision Client from Help > Diagnostics. See also: Scripting, Help Menu
Image Management	The Image Manager is available from the Tools > Image Management menu. This tool is a drag-and-drop browser that helps manage the images that are stored on the Gateway. It is important to realize that these images are shared across all projects: they are not stored inside a project itself. Use the toolbar at the top to do common tasks like uploading new images and creating folders. You can drag images and folders from your computer's desktop or hard drive into this window to easily upload new images to the Gateway. You can also get to this tool by putting an Component Palette > Display > Image component on a window, and clicking the browse button on the image's Image Path property. See also: Images (png, jpg, gif)
Script Console	Opens up the Scripting Console window. This is where you can test scripts out. See also: Script Console
Database Query Browser	Opens up the Database Query Browser Panel, which will allow you to run SQL queries against your database connections. See also: Database Query Browser
Translation Manager	Opens up the Translation Manager Panel, which will allow you to configure translations. See also: Localization and Languages
Cloud Templates Browser	Changed in 7.9.13 Cloud Templates have been moved to the new Ignition Exchange and are still available.

Function	Description
Symbol Factory	If you have the Symbol Factory module installed, you'll be able to open the Symbol Factory browser under the Tools menu in the Designer. You can browse through the symbols or use the convenient search function to find the symbol you need. Once you find a symbol, you can drag-and-drop it into a window. Each symbol is dropped as a shape group. You will be able to un-group it or double-click into the group just as if you had drawn the symbol yourself using fundamental shapes. This means that you can alter the shape if you need to, or bind any colors inside the shape to a tag to make the shape dynamic.
Launch Project	Allows you to launch the project directly from the designer. You can either launch a staging or published windowed client, or a published full screen client.

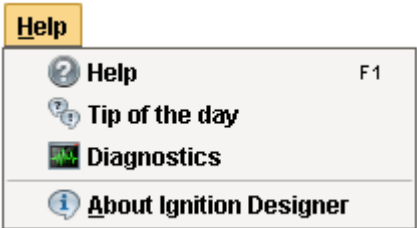
Console - Example Script

Console - Example Script

```
# A python script that demonstrates the print statement
print "Print me first"
x=10
z=2
print x, y, x/y

# Result looks like this:
#Print me first
#10 2 5
```

Help Menu



Functions	Description
Help	Opens up your web browser and takes you to this User Manual, for quick reference.
Tip of the Day	Reopens the Tip of the Day popup that by default appears when you first open the Designer.
Diagnostics	The Help menu in the Designer and the Vision Client has a Diagnostics window that contains a number of tabs each providing a useful troubleshooting feature. You can right-click on any of the tabs to show or hide the other tabs. These tabs or the troubleshooting features are as follows: Performance Tab Displays a number of small realtime charts that show various aspects of the currently executing Designer or Client's performance. These charts can be very useful to help troubleshoot performance issues, especially slow queries. One of the most common causes of query slowdown is simply running too many queries too frequently, and the # of Select Queries (Queries/Second) chart can help identify when this is occurring. The statistics shown on this page apply to the current session only, you can see the Gateway statistics from the Status section of the Gateway. Console Tab Displays the Ignition Output Console. Log Viewer Displays the logged events for the current Designer or Client session. Whenever errors occur, they are logged and displayed in this tab. This is a good place to go when troubleshooting an issue, as any errors shown here may illuminate the cause of the problem. To view entries across all categories chronologically, uncheck the Group Categories checkbox. Logging Levels Determines the verbosity of a host of internal loggers. Most users will not use this tab unless prompted by a technical support representative. Thread Viewer

Functions	Description
	Shows information about the currently running threads. Most users will not use this tab unless prompted by a technical support representative. Connections Displays the Gateway connection status as being connected or not.
About Ignition Designer	Provides information about the Designer such as the versions of the modules, java version, and gateway IP Address that the designer is using.

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

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

Communication Modes



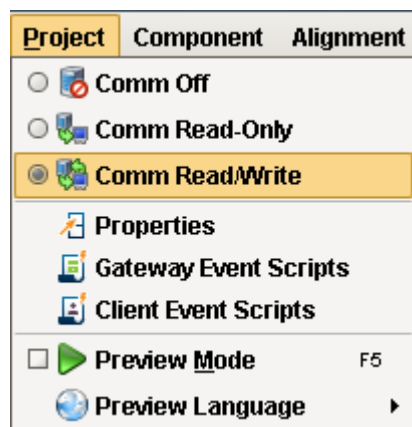
INDUCTIVE
UNIVERSITY

About Communication Modes

[Watch the video](#)

Controlling Communication Between the Designer and Gateway

The Designer offers three data communication modes for your projects: **Comm Off**, **Comm Read-Only**, or **Comm Read/Write**. Comm Read-Only is the default mode, which does not allow writing from the Designer to Tag or Database sources.



- **Comm Off**

In this mode all database query traffic and Tag subscriptions and writes are blocked.

Experts often use the **Comm Off** mode while designing a window to temporarily shut off data flow so that they can manipulate component's bound properties without the values being overwritten by the data bindings. This is useful to set the values that they want to serialize into the window.

This can be important for windows with large datasets, clearing the datasets before saving the window can significantly reduce the size of the window, improving performance.

This setting does not affect the execution of a project's Transaction Groups, because Transaction Groups execute on the Gateway, not in the Designer.

- **Comm Read-Only** (default)

Tag subscriptions and SELECT queries work, but Tag writes and UPDATE/INSERT/DELETE queries are blocked.

The Designer starts up in **Comm Read-Only** mode as a safety mechanism so that you don't inadvertently write to a Tag as you are designing. You can customize the Designer's startup mode, see the [Designer/General Properties](#) section.

- **Comm Read/Write**

All data is passed through to the Gateway. The mode can be switched at any time via the tri-state toggle selection in the main toolbar, or the radio buttons in the Project menu.

A common beginner mistake is to forget to switch the mode to **Comm Read/Write** when attempting to test a window's functionality in **Preview** mode.

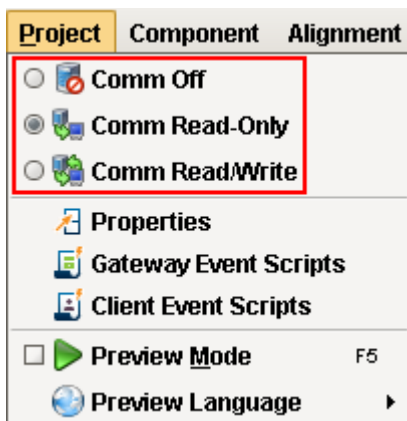
To Set or Change the Communication Mode

There are three ways to set or change the comm mode in the Designer:

- Using the **Project** menu
- The buttons located on the **Main Toolbar**
- Using the **Project Properties** window

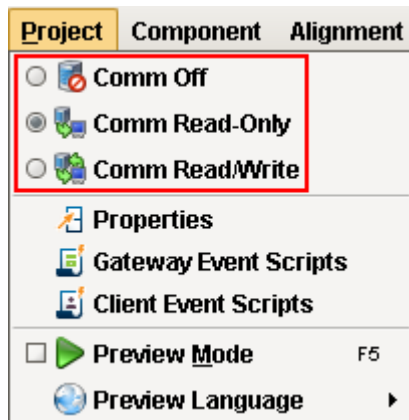
Using the Project Menu

On the Designer toolbar, go to the **Project** menu and choose: **Comm Off**, **Comm Read-Only**, or **Comm Read/Write**.



Using the Main Toolbar

If the Main Toolbar is enabled, go to **Project**, and you'll see the corresponding button for each Comm Mode. The currently selected Comm Mode will have its radio button filled in.



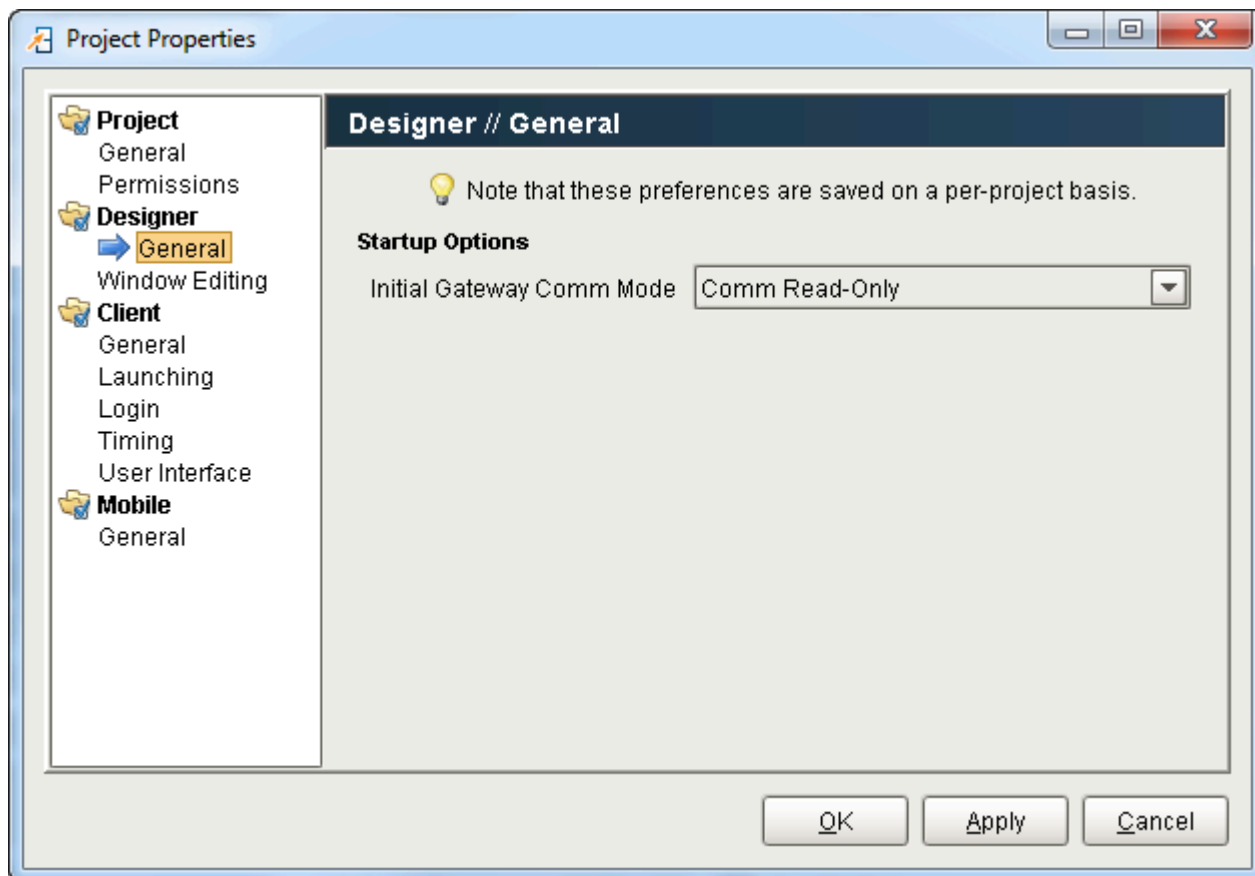
Using the Project Properties Window

This will change the default Comm Mode that the designer starts up in for the current project.

1. In the Designer, either double-click on the **Project > Properties** node in **Project Browser**, or click on the **Project > Properties** command on the top menu.
The **Project Properties** window is displayed.
2. Go to the **Designer > General** setting section.
3. Under **Startup Options**, for the **Initial Gateway Comm Mode**, choose from the dropdown **Comm Off**, **Comm Read-Only**, or **Comm Read/Write**.

NOTE

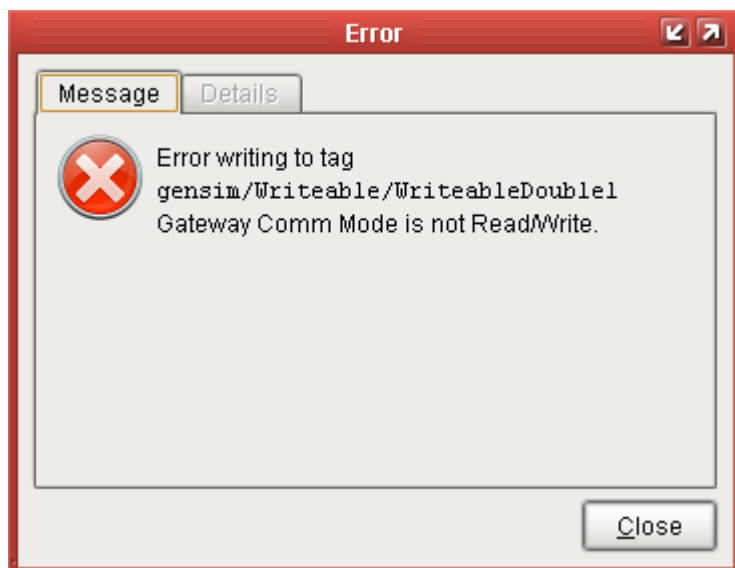
These property settings are saved on a per-project basis.



Communication Error Message

If a user is attempting to write to a project in the Designer or from a Client, and the Comm Mode is not enabled for Read/Write, they will get an error message. There are a couple places to check:

- From Main Toolbar go to **Project**, and make sure the Comm Mode is set to Read/Write.
- If Comm Read/Write is checked, you may have a Client Event Script enabled that could be preventing you from writing to a project or a Tag, as shown in the example below. From the menubar, go to **Project > Client Event Scripts**. You may need to edit your Client Event Script.



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

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

Previewing the Project

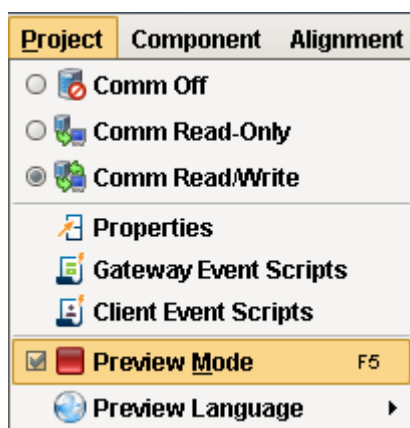


Previewing the Project

[Watch the video](#)

Testing in the Designer

Many times, it is useful to test the components on the screen to ensure that certain bindings or scripts are working the way that was intended. The Designer can go into **Preview Mode** that will allow you to interact with the currently opened window as though you were working in a client. This means that instead of clicking between components and seeing their properties, you will be able to interact directly with the components such as clicking on a button to execute its script, or entering a value into a text field to update a Tag value.



Preview Mode

The window workspace operates in two distinct modes: design mode and preview mode. You can switch between these modes with the play/stop buttons in the toolbar or the **Project > Preview Mode** menu item. You can also use the **F5** key to toggle between the two modes.

In design mode, your mouse is used to manipulate components in a window. You can select, drag, and resize them. You can alter data bindings and event script configuration. Data bindings are active

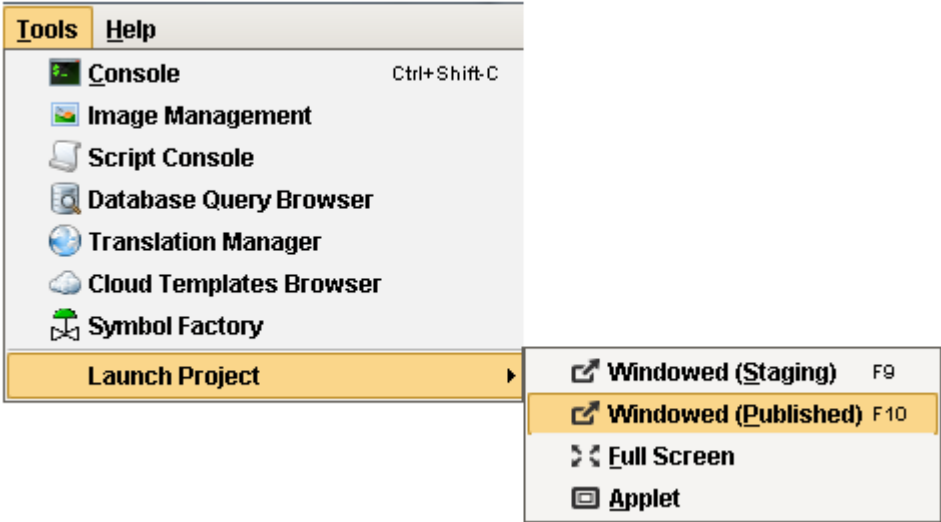
in design mode, but event handlers are not.

In preview mode, you are interacting with a "live" version of the window. Property bindings and event handlers will run, just like in the Client.

⚠ CAUTION

Some of Ignition's functionality will not work in Preview Mode. ie: the retarget and openWindowInstance scripting functions must be tested in a Client.

Preview mode is useful for a quick check of the operation of a window, but it becomes cumbersome when trying to test a whole project. For that, we recommend having a launched Client up as well, and doing testing in the true Client. You can quickly launch a client in one of the three launch modes via the **Tools > Launch Project** menu.



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

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

Find and Replace



INDUCTIVE
UNIVERSITY

Using Find and Replace

[Watch the video](#)

Searching Through the Designer

The **Edit > Find/Replace** menu command in the Designer is a very handy tool. You can search your entire project for specific components, properties, scripts, Transaction Groups and more. You can then use the replace feature to make mass changes to a project with very little effort.

The Find/Replace dialog box is shown with the following sections:

- Find:** A search input field and a 'Find' button.
- Target:** A tree view of search targets with checkboxes for selection.
 - ☒ **Pipelines**
 - ☒ **SQL Bridge** All groups (6) ▾
 - ☒ Group Settings
 - ☒ Items
 - ☒ **Scripting**
 - ☒ Client Event Scripts
 - ☒ Gateway Event Scripts
 - ☒ Script Library [project/shared]
 - ☒ **Tags** Default ▾
 - ☒ **Templates** Open templates (0) ▾
 - ☒ Component Properties
 - ☒ Property Bindings
 - ☒ Scripts
 - ☒ Security
 - ☒ Custom Methods
 - ☒ **Windows** Open windows (1) ▾
 - ☒ Component Properties
 - ☒ Property Bindings
 - ☒ Scripts
 - ☒ Security
 - ☒ Custom Methods
- Results:** A table with columns: Location, Property, Value.

Location	Property	Value
----------	----------	-------
- Replace:** A 'Replace:' input field.
 - Current:** A text input field.
 - Preview:** A text input field.
- Buttons:** 'Replace', 'Skip', 'Replace All', and 'Close'.

Search Options

In the Target section, the Find/Replace tool has options for searching through many different parts of a project.

- Pipelines
- SQL Bridge Transaction Groups
- Scripting
- Templates
- Windows
- **New in 7.9.17** Tags - (Includes Tag property values as well as substrings in Tag Event Scripts)

To Find and Replace

1. In the Designer, go to **Edit > Find/Replace** or use the shortcut **Ctrl-F**. The Find/Replace window is displayed.
2. Type what you're searching for in the textbox at the top to search through your project. The wildcard character (*) which will match anything, and the single-character wildcard character (?).

For example, to find all references to a tag that started with the string `Motor`, you could search for `Motor*`. This would match things like `Motor15`, `MotorH0A`. The search query `Valve?Status` would return `Valve1Status`, but not `Valve38Status` because multiple characters are present between `Valve` and `Status`.

3. In the **Target** section, narrow down your search by choosing the options you are interested in. It is often useful to specify a narrow search target. Find/Replace searches through many different parts of a project as well as through Tags. The target settings let you specify exactly what Find/Replace should search through. By unchecking boxes in the Target section, you can avoid search results that you aren't interested in.
4. Click the **Find** button at the top right to execute the search. All matching items appear in the **Results** section. Double-click on an item in the Results table to bring that item into editing focus in the Designer.
5. To replace a value, select an entry in the Results section and you'll see the Value of that entry in the **Current** box of the **Replace** section.
6. Enter the text you want to use as a replacement in the **Replace** textbox and a preview of the new value is shown in the **Preview** box.

7. Click the **Replace** button to execute the replace. This will move your selection down in the Results table so that you can rapidly execute multiple replacements. If you're satisfied and you'd like to make the identical replacement to many items, select them all in the Results table and click the **Replace All** button.

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

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

Keyboard Shortcuts



INDUCTIVE
UNIVERSITY

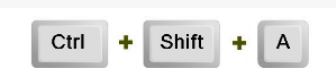
**Using the Popular Keyboard
Shortcuts**

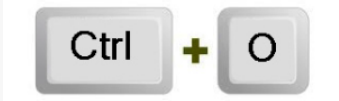
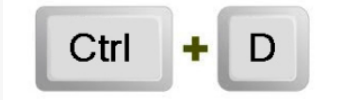
[Watch the video](#)

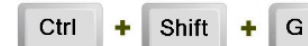
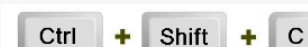
Using Keyboard Shortcuts in the Designer

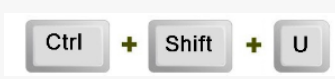
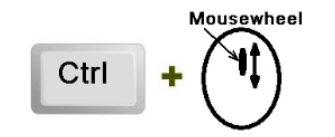
You can interact with screens and more by using some of the popular keyboard shortcuts in the Designer.

Action	Description	Keyboard Shortcut
Nudge	Moves selected component(s) in the direction of the arrow key by the default nudge distance.	
Alt-Nudge	Same as Nudge, but uses the "alt-nudge" distance.	
Resize Right	Moves the right edge of the component left or right. Add Alt to use the alt-nudge distance.	
Resize Bottom	Moves the bottom edge of the component to top or bottom.	
Resize Left	Moves the left edge of the component left or right.	
Resize Top	Moves the top edge of the component to top or bottom.	
Move Forward/Move Backward	Moves selected component(s) in the Z-order.	
Move to Front/Move to Back	Moves selected component(s) to Front or Back.	
Copy-Move	Copies the component when holding Ctrl while doing a mousemove.	
Orthogonal-Move	Restricts to only moving straight up, down, left, or right when holding Shift while doing a mousemove.	
Selection-Move	Drags the current selection when holding Alt while dragging.	

Action	Description	Keyboard Shortcut
Copy-Axis-Move	Combines copy move with axis move.	
Proportional Resize	Resizes a component while maintaining its aspect ratio.	
On-Center Resize	Resizes the selection using the center as the anchor point.	
Select All	Selects all components that are siblings of the selected component.	
Select Same Type	Selects all components that are siblings of the selected component and the same component type.	
Select Same Type in Window	Selects all components in a window that are the same type as the selected component.	
Layout	Opens Layout Constraints to let you specify layout for the component(s).	
Size & Position	Opens Size & Position to let you specify exact size and position for selected component or window.	
Customizer	Opens Custom Properties to let you configure complex component properties.	
Customizer 2	Opens the second Customizer if available.	
Jython	Opens Component Scripting to let you configure actions for component or window.	

Action	Description	Keyboard Shortcut
Security	Opens Security Settings to let you set security for Component(s) or window.	
Script Modules	Opens Script Module Editor for reusable Jython functions.	
Save	Saves the project.	
Open	Opens the Open/Create Project to let you select a different project.	
Undo	Undoes the last action.	
Redo	Gets rid of the last undo action.	
Copy	Copies selected component(s) or window.	
Duplicate	Duplicates selected component(s) or window.	
Cut	Cuts selected component(s) or window.	
Paste	Pastes selected component(s) or window in clipboard. Pasted component(s) wait for position before pasting.	
Immediately Paste	Places pasted component(s) at the same location where they were copied/cut.	

Action	Description	Keyboard Shortcut
Comment/Uncomment Lines of Code	Quickly comment or uncomment lines of a script or query in Designer.	
Cancel	Cancels a pending paste operation, deselects the current row of a table, cancels dragging components onto window.	
Find/Replace	Opens Find/Replace to let you Search and replace the project based on string, pattern, or regex.	
Delete	Deletes the current selection.	
Snap to Grid	Toggles whether or not moving and resizing components snaps to the grid.	
Snap to Guides	Toggles whether or not moving and resizing components snaps to guides.	
Console	Opens the Output Console.	
Help	Launches the Ignition User Manual in a web browser.	
Preview Mode	Toggles preview/design mode.	

Action	Description	Keyboard Shortcut
Rename	Renames the selected item (tag, window, transaction group, component, and so on).	
Launch Staging Project	Launches the staging client from the Designer.	
Launch Published Project	Launches published client from the Designer.	
Update Project	Updates project to server. Receives concurrent edits from other Designers.	
Zoom	Zooms in/out in the Designer.	
Lasso Select	Draws a lasso and every component in the path will be selected.	
Select Through	Click on a component that is beneath another component. You can do this multiple times if there are several layers of components.	
Code Folding/Unfolding	Select a line of code, and this command will collapse the selection. Press again while the folded code is selected, and the code will unfold. The Script Console has a separate command for code folding.	

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

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

Saving and Publishing Projects



INDUCTIVE
UNIVERSITY

**Project Creation and
Publishing**

[Watch the video](#)

Once you created the windows, Tags, templates, and anything else that goes into your client, you need to save and publish the work. Saving your progress means pushing the changes from your Designer into the Gateway. Publishing the changes means telling all of the clients that they can use the new updates to your project. By default, these two items are linked, but that can be changed so they happen independently. If they are not linked, it's possible to publish individual resources on a project as they are finished instead of publishing the entire project. This is particularly helpful if you want to get resource updates to users quickly, or parts of a project into production sooner rather than waiting for the entire project to be completed.

In the Project Properties of the Designer, there are two Publish Modes to choose from:

- **Auto** - saving performs both a save and a publish of the entire project. This is the simplest way to push your changes, and is the default setting.
- **Manual** - splits the save and publish commands into two operations. Save will take any changes you have made and push them to the Gateway, and Publish will push any saved changes out to the Clients. Additionally, Manual mode gives you the option to select certain resources and perform a publish on just those.

You also have options on how to perform Client updates. You can choose between notifying the operator of an available update, or automatically pushing updates as soon as it is published making it transparent to the operator. This page assumes you are using the default Notify mode, see [Client Update Modes](#) for more information.

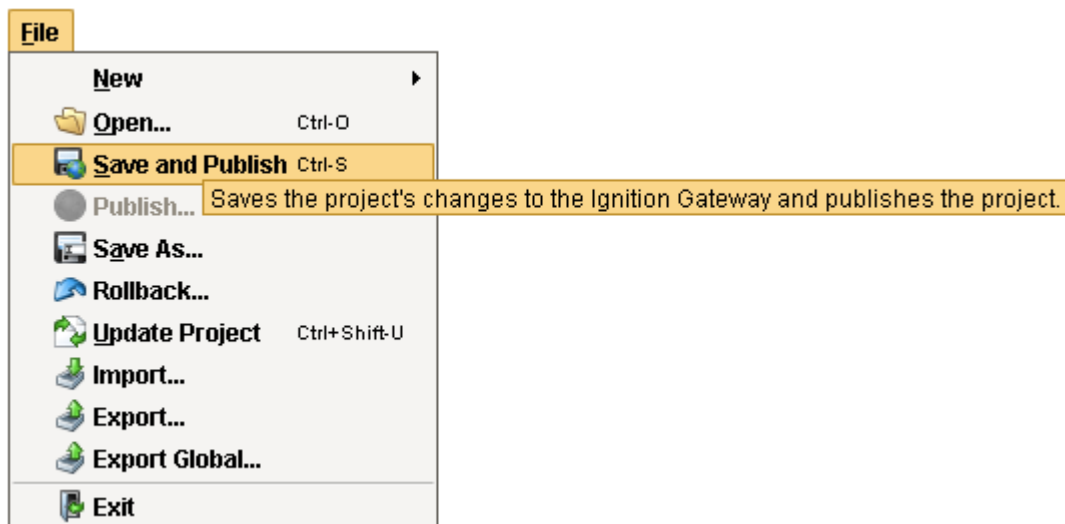
Auto Publish Mode

Publishing in Auto Mode allows you to save and publish an entire project with one button press. You'll notice that the **Save and Publish** option (under **File**) is available when the Publish Mode is set to Auto. By default, the Publish Mode is set to Auto.

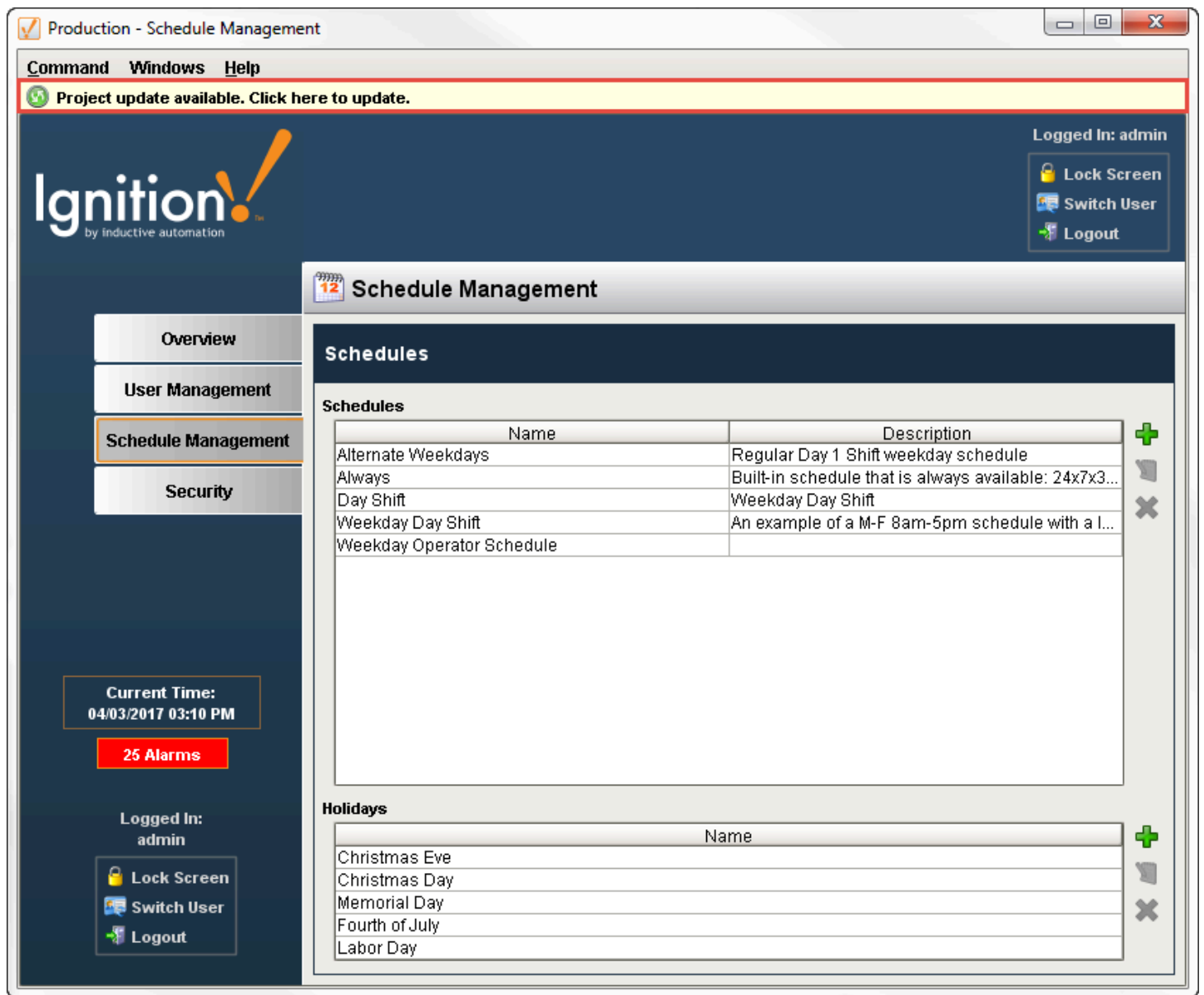
Publishing in a Project in Auto Mode

Let's publish an entire project using Auto Mode.

1. Open your Designer and a client. If you don't have a project already, create a new project when opening the Designer, then open a client of it.
2. In Designer, open one of the windows (or create a new one) and make a few edits. You can drag components onto the window for obvious changes. When you're ready to publish your project, go to the menubar, and select File > Save and Publish. Unlike Manual Mode, Auto lets you Save and Publish using one command.



3. Now go to your opened client and you'll see the Update bar at the top of the screen. Click it to pull the new changes into the client. Depending on your changes, it may not show the additions automatically. You may have to logout and log back in again.



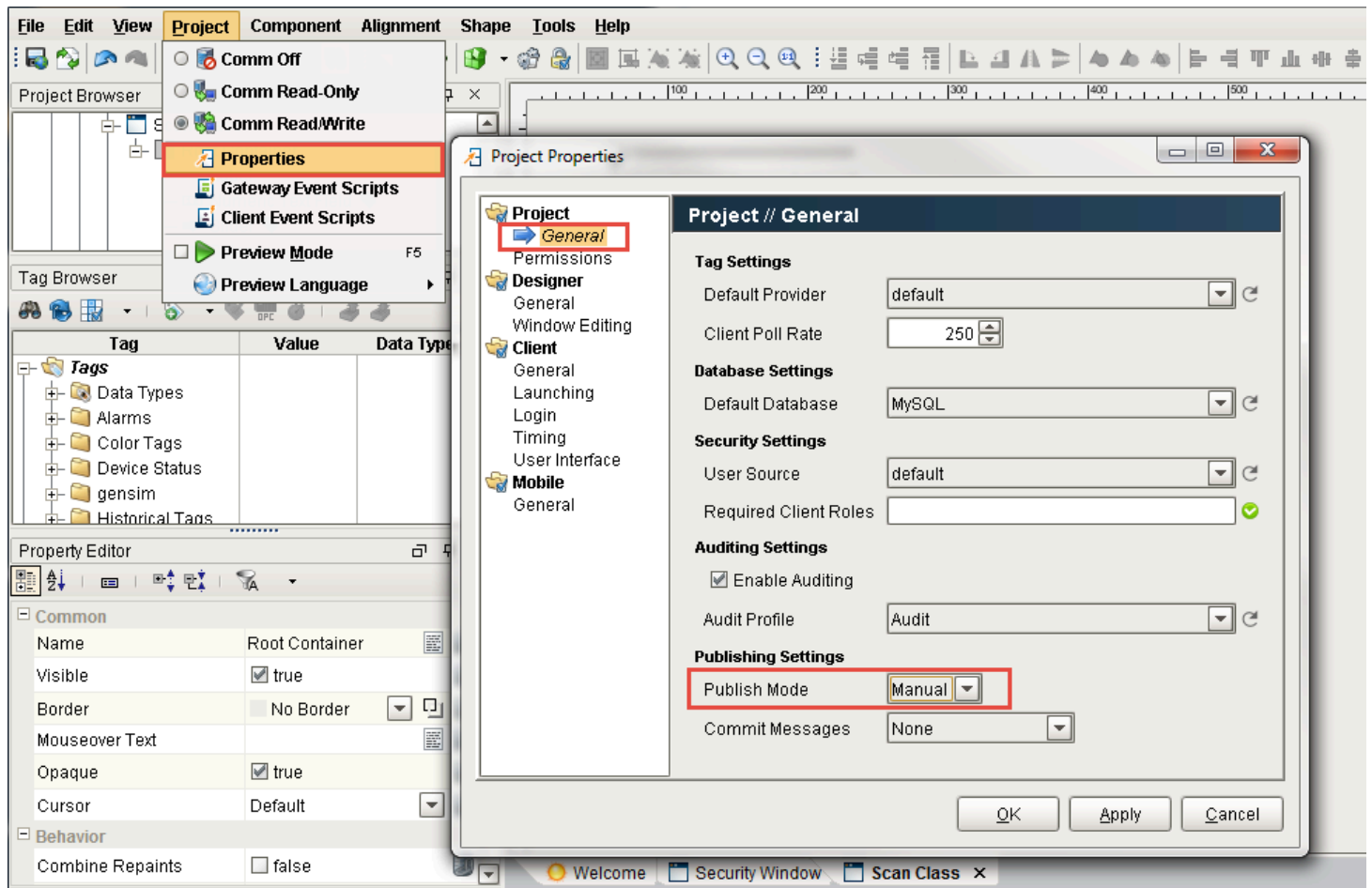
4. That's all there is to it! Depending on how Clients are setup to receive project updates, operators can either be notified that a project update is available and install it themselves, or an update can be pushed/installed automatically which is transparent to the operator. Client updates are described later in this section.

Manual Publish Mode

Publishing in Manual Mode allows you to publish individual global and project resources. It's important to note that the **Publish** option (under **File**) is only available when the Publish Mode is set to Manual. If it is set to Auto, you will not be able to publish your project resources individually.

To Set the Publish Mode to Manual

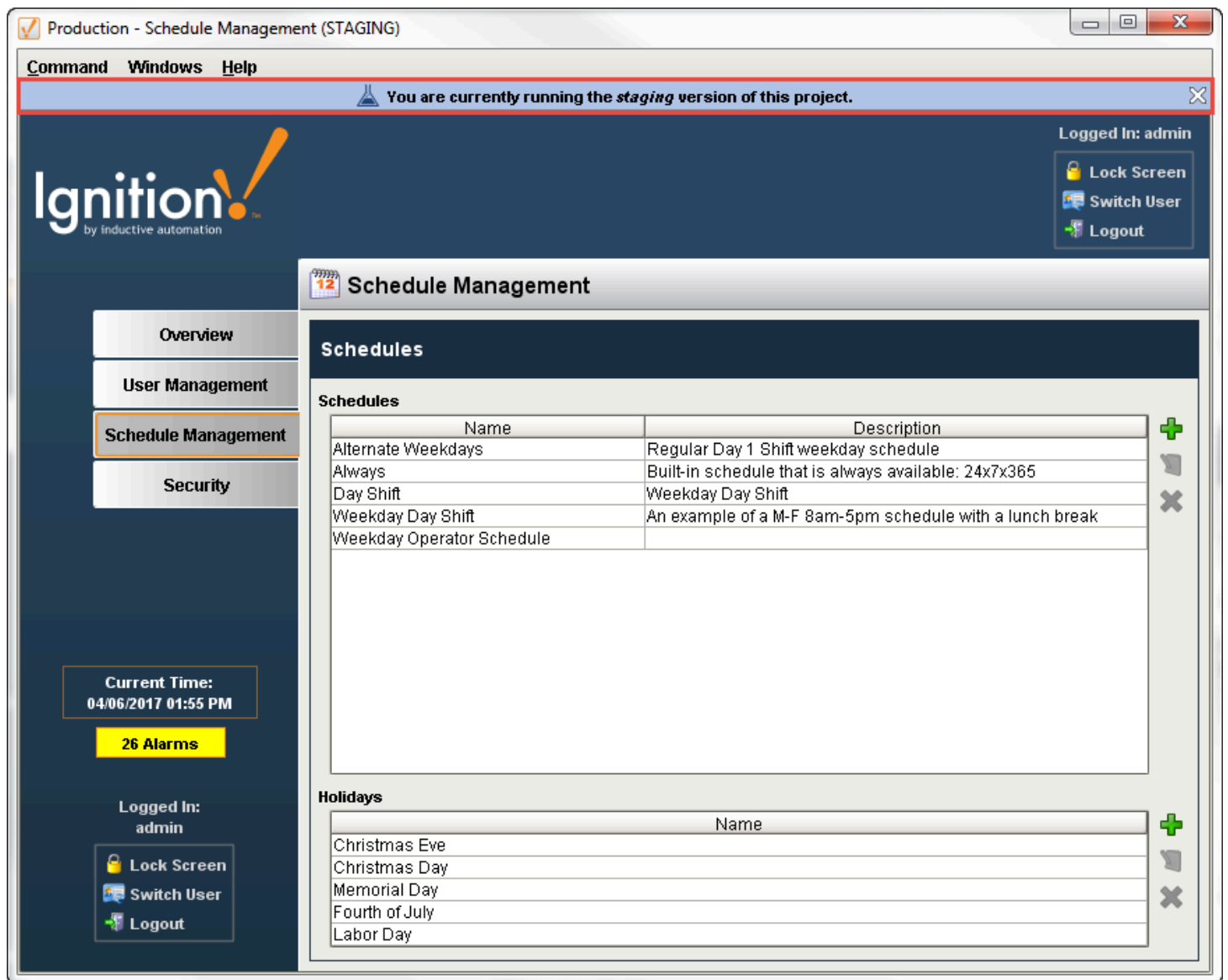
1. In Designer, go to **Project > Properties**, and select **Project > General**.
2. Set the **Publish Mode** to **Manual**, and press **OK**.



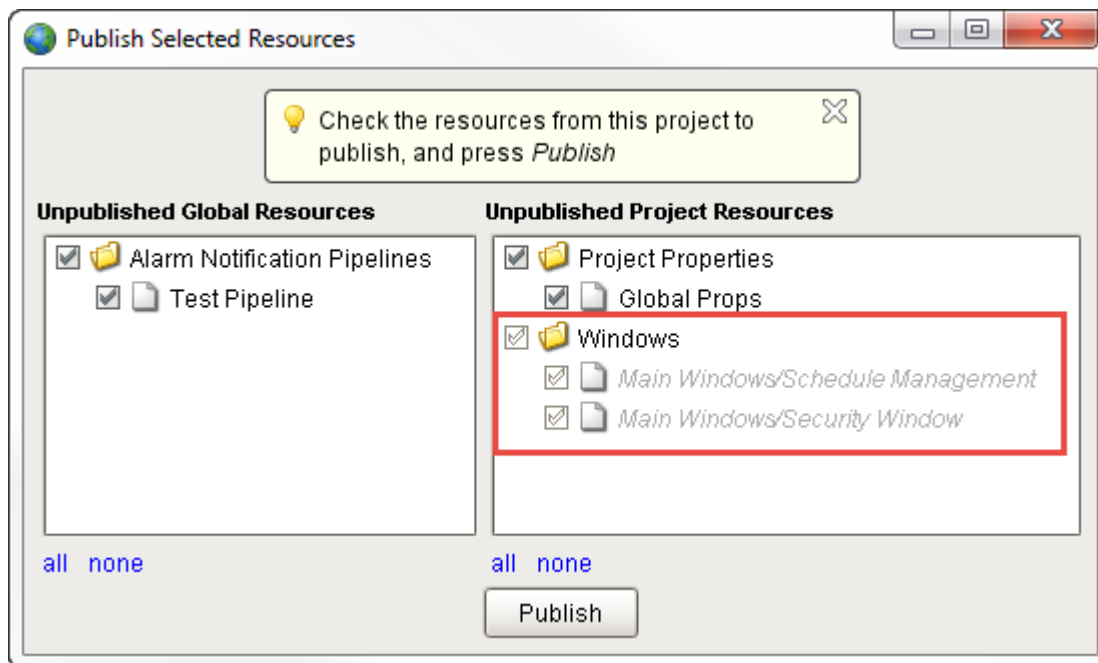
Publishing a Project in Manual Mode

Let's manually publish several individual resources.

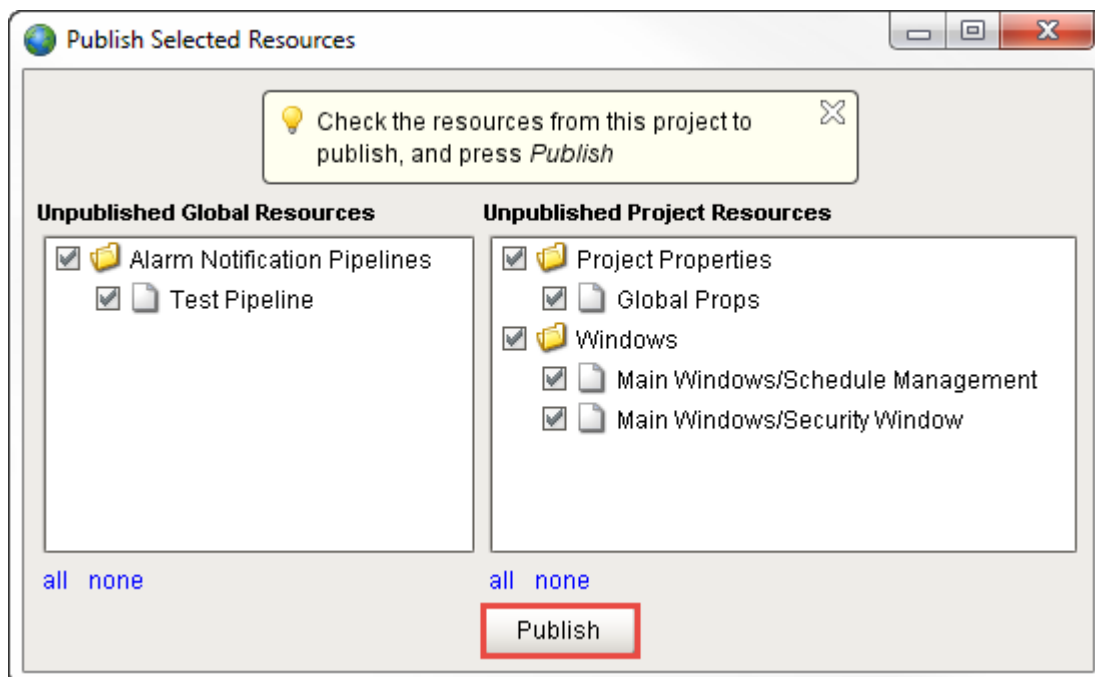
1. Open your Designer and a Client. If you don't have a project already, create a new project when opening the Designer, then open a client of it.
2. In the Designer, open a window and make a few edits to it. Before you can publish your selected resources, you must Save your project. Go to the menubar, select File > Save.
Note: Closing a window (and selecting commit changes) is not the same as saving the project. You must save the project before you can publish your changes.
3. At this point, we have saved but unpublished resources. If you would like to see and test these changes before publishing them, you can open a staged Client from the Designer. Go to the menubar and select Tools > Launch Project > Windowed (Staging). This will give you a Client with a blue banner at the top showing that you are running a Staging Client instead of a normal Client.



- Now that you have saved but unpublished resources, we can publish them. Go to the menubar and select **File > Publish**.
- The Publish Selected Resources window will open. By default, Ignition will select all the resources, both Global and Project Resources, that were saved since the last time any items were published. If you see any resources that are grayed out, it means that those items were committed but not saved. If you want to publish the grayed out items, close the Publish Selected Resources window, save your project, and then go back to the menubar, and select **File > Publish**.



6. If there are any resources you don't want to publish, unselect those resources. When you're ready, click **Publish**.



7. Now go to your standard **Client** (not Staging) and you'll see the yellow bar at the top showing there is an update. Click it to pull the new published resources from the Gateway.

i NOTE

Your Staging Client will not receive an update when you Publish.



Logged In: admin

- Lock Screen
- Switch User
- Logout

Schedule Management

Overview

User Management

Schedule Management

Security

Current Time:
04/03/2017 03:10 PM

25 Alarms

Logged In:
admin

- Lock Screen
- Switch User
- Logout

Schedules

Schedules

Name	Description
Alternate Weekdays	Regular Day 1 Shift weekday schedule
Always	Built-in schedule that is always available: 24x7x3...
Day Shift	Weekday Day Shift
Weekday Day Shift	An example of a M-F 8am-5pm schedule with a l...
Weekday Operator Schedule	

Holidays

Name
Christmas Eve
Christmas Day
Memorial Day
Fourth of July
Labor Day

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

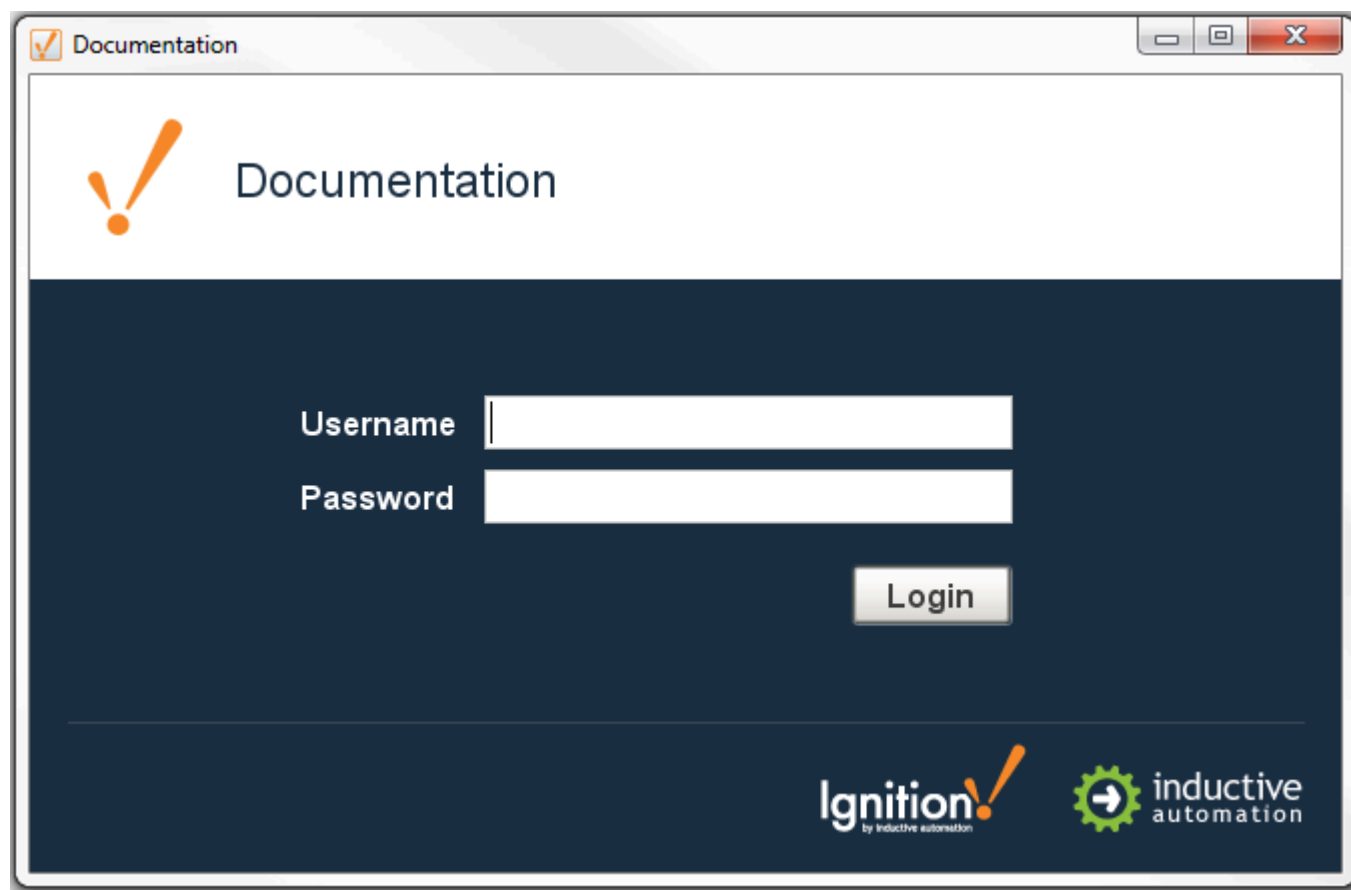
For documentation on the latest stable release, see the [latest version](#) (8.3).

Clients

Ignition Clients

The web-launched Vision Clients in Ignition are the "runtimes" of the Vision module. You can launch one or more Clients to display the projects you created in the Designer. They run as full applications and feel like a traditional installed Client, without the need to install and manually synchronize projects.

Clients are launched from the Gateway Homepage for a specific project.



NOTE

Ignition Clients may look differently on Windows than on Linux. This is because Linux does not have the default Arial font that Windows uses. To have Linux use the same font, it will need to be

download. Check out a forum post where a user goes over how to do this:

<https://forum.inductiveautomation.com/t/java-fonts-and-linux/1112>

There are two ways to run Clients: **Windowed** and **Full Screen**. You can choose the mode from the dropdown next to the project name. By just clicking on the project name, the project is launched in the default mode. Certain modes may not be available, depending on project settings. You can launch as many Clients as you want! If you've got other computers on the same network as the Gateway computer try launching a client on them too. Make sure that your Gateway computer doesn't have a Firewall enabled and it is allowing traffic on port 8088 - the default port for the Ignition web server.

Launch Projects ?



Windowed

The **Windowed** launch mode is the **Standard** launch method. The Client will be web-launched using Java Web-Start and will have its own window. In this mode, the Client runs as a full, independent application. After being launched, the browser can be closed and the project can be launched from a shortcut on the desktop.

Full Screen

The **Full Screen** launch mode is similar to the Windowed mode, and will also use web-launching to run the Client as a full, independent application. In this mode, however, the Client will occupy the full screen, and will not have a title bar. This mode is ideal for touch-screen display panels, and other displays where the Ignition project is the sole focus on the screen.

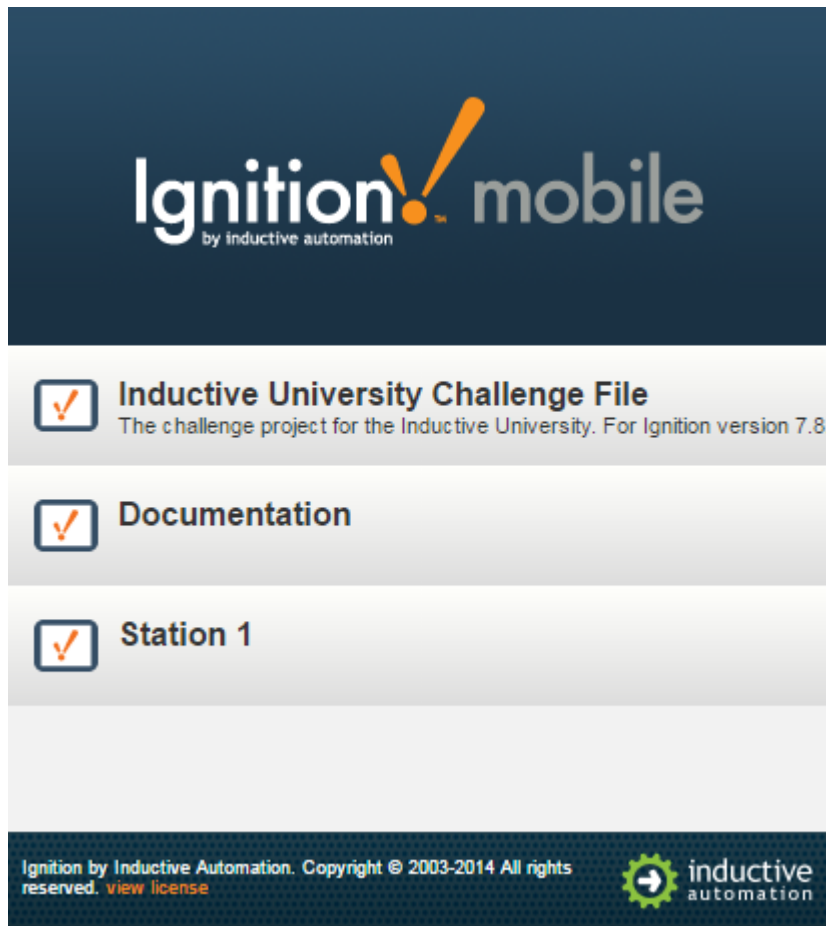
Can Clients Switch Projects?

It is possible to change which project an active client is viewing. This is known as "Retargeting", and is a great way to allow users access to multiple projects without forcing them to launch multiple clients.

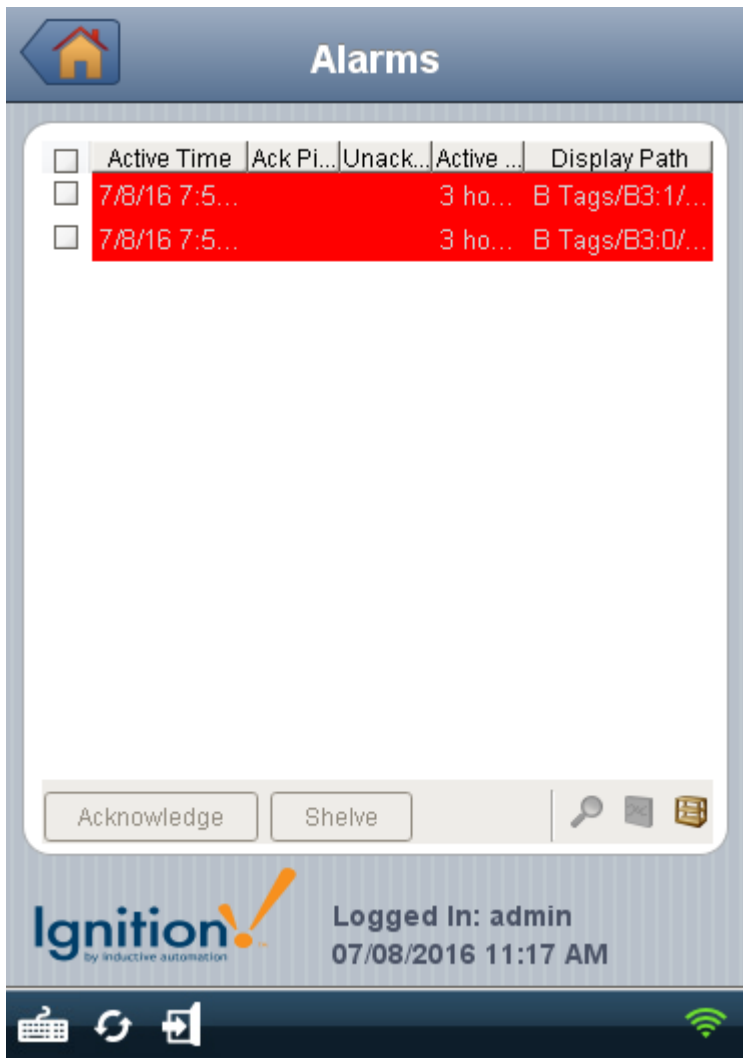
More information can be found on the [Retargeting](#) page.

Mobile Clients

If you have the **Mobile** module installed, you can launch projects on your smartphone or tablet as well. All the user has to do to launch a mobile Client is to connect their mobile device to the wireless network and point the web-browser to the Gateway's LAN address. At this point, they'll be presented with a mobile-optimized version of the Ignition Gateway homepage, where they can select a project to launch. Note, that projects must have at least one window defined and be enabled for mobile launching in order to show up in this list.



After selecting a project and logging in, they can use the project like a normal project. A Utility bar will be present on the side or bottom of the mobile client to provide access to common actions such as logging out, or invoking a touchscreen keyboard.



See [Mobile Client Settings](#) for more details on configuring mobile clients.

Native Client Launchers

The Ignition Gateway contains downloadable native executables called Native Client Launchers. These executables can launch Clients directly without calling Java Web Start. Native Client Launchers also allow you to select from projects on multiple gateways. A single shortcut can launch any project on your network.

More information can be found on the [Native Client Launchers](#) page.



Gateway URL 10.20.7.236:8088:8043 [CHANGE](#)

☒ Create desktop icon
LAUNCH DESIGNER 

Launch Projects



Inductive University Challenge File
The challenge project for the Inductive University. For Ignition version 7.8.0.

☒ Create desktop icon
LAUNCH  



Documentation

☒ Create desktop icon
LAUNCH 



Station 1

☒ Create desktop icon
LAUNCH  

Ignition Client Launcher v7.8.3 by Inductive Automation. Copyright © 2003-2016. All rights reserved.





Local Client Fallback

Ignition Clients are fully dependent on being able to communicate with a Gateway. If Gateway communication is lost, the Client suspends operation while it attempts to reconnect with the Gateway. This can be a problem when you need the Client to monitor critical operations on a plant floor.

Ignition provides a local Client fallback mechanism that lets you use a Gateway running on the local machine. In normal operation, your Client can connect to a central Gateway located somewhere on the network. The central Gateway would be responsible for all data aggregation, such as storing historical data in a database. But if communication to the central Gateway is lost, the Client can automatically retarget to a project that you specify in the local Gateway. This project should contain the minimal realtime information that you need to keep your operation running. Note, that in order to use local fallback, port 6501 must be open on the local machine.

To enable local fallback, do the following steps:

1. Go to **Configure > Gateway Settings** in the local Gateway.
2. Scroll down to the **Local Client Fallback** section and select the Enable Local Fallback.

Local Client Fallback	
Enable Local Fallback	☐ Enables a client to fall back to a project in a local Gateway if communication is lost to the central Gateway. Note that port 6501 must be open on the local machine. (default: false)
Seconds Before Failover	60 The number of seconds to wait before switching to the local Gateway project after comm loss. (default: 60)
Fallback Project	- none - ▼ The local project to use during fallback.

3. Select a Fallback Project from the dropdown list. Note, that the selected project must be published in the local Gateway, and it must have at least one main window.
4. Optionally, you can change the Seconds to Failover setting to a value other than 60 seconds.

This setting controls the number of seconds to wait before fallback automatically starts. During comm failure, you can also click a button to load the local fallback project immediately.

When local fallback is enabled, the Client attempts to open port 6501 on the local machine. If the port can be opened successfully, the Client reads fallback settings from the local Gateway and shows a **Fallback Project** button on the bottom of the **Gateway Connection Lost** window. You can click this button at any time to load the fallback project, or simply wait for the fallback project to automatically load. You may want to set the local Client to automatically log in to avoid typing in a username and password when the Client loads. This can be set in the Login section of the project's properties.

Testing Local Fallback

Testing local fallback is highly recommended before you start to depend on it in a production setting. The easiest way to test fallback is to simply unplug the network cable to the Client machine, or disable the network card on the machine. If the **Fallback Project** button is not visible on the **Gateway Connection Lost** window, check your local Gateway console and verify that the message **Started Fallback Socket on port 6501** is present in the console. Any other error message related to the **Fallback Socket Controller** indicates that some other problem has occurred (most likely the port cannot be reserved) and local fallback is not available to Clients.

Automatically Transferring Back

Local Client Fallback will not automatically transfer back to the main project when it is running again, it is a purposefully simple system to protect against errors or complications. You can, however, provide

your own solution depending on your needs. One example is to add a retarget script to your **Timer** Client Event Scripts to silently try to reconnect.

Automatically Retarget Back

```
# add this to a Client Timer Script running every 30 seconds  
# change the ipaddress and project names to match your system  
  
# allow the main copy of Ignition 10 seconds to give a response  
status = system.util.getGatewayStatus("ipaddress:8088/main", 10000)  
# if it's running again, retarget  
if status == "RUNNING":  
    system.util.retarget("my_project", "ipaddress:8088:8043/main")
```

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

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

About Mobile Clients



INDUCTIVE
UNIVERSITY

About Mobile

[Watch the video](#)

The Mobile module adds the ability to launch Vision module projects on modern smartphones and tablets. This lets you keep track of your control system while moving around your facility. The Mobile module can be combined with a remote-access networking architecture to allow global on-the-go access to your control system.

How it Works

Normally, you can't launch Vision projects on mobile devices. This is because of the technical limitation of Java SE (Standard Edition) which does not run on mobile devices. The Mobile module gets around this limitation by launching the Client on the Gateway in a special headless (invisible) mode, and then using HTML5 and AJAX to show the Client's screen on the mobile device's browser.

Networking

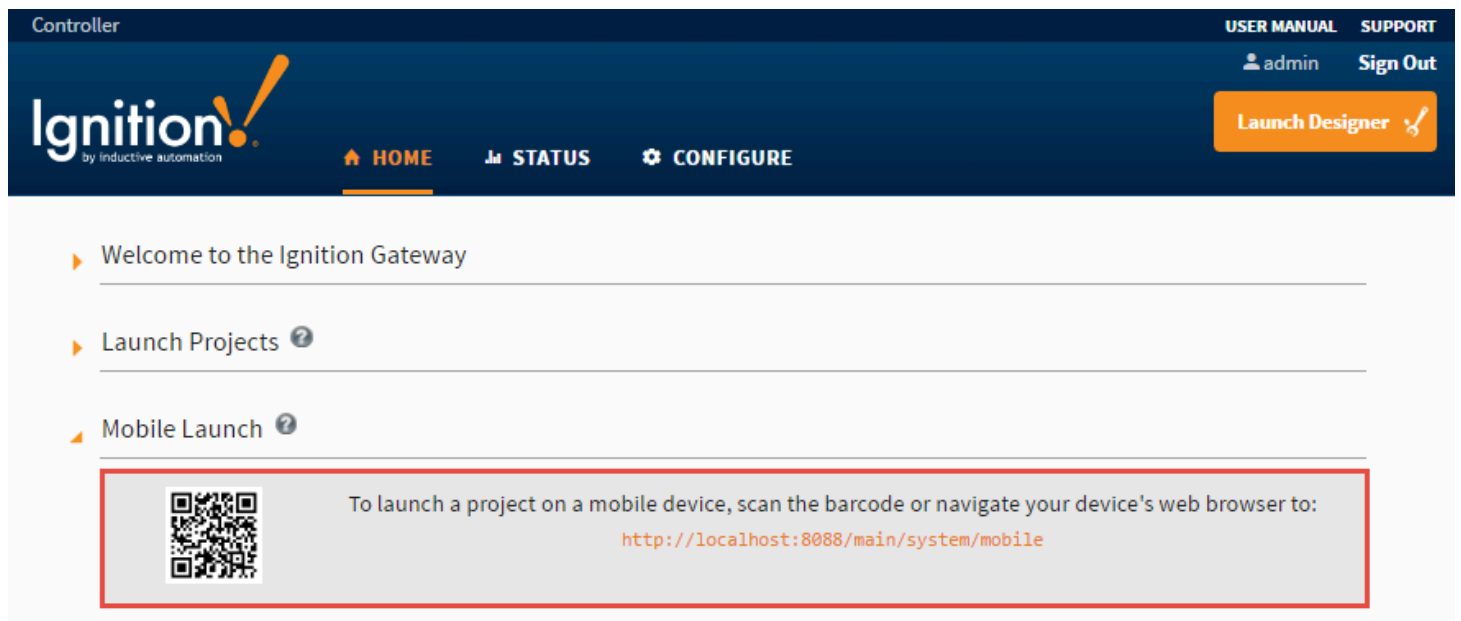
Typically, the mobile device connects to the Ignition Gateway via the facility's wireless LAN (802.11) infrastructure. To launch a mobile client, the mobile device simply connects to the Ignition Gateway by pointing its web browser to the Gateway's LAN address. It is important to understand that normally, the traffic is not going over the device's cellular connection because the cellular connection connects to the internet, and without explicit setup, an Ignition Gateway is not accessible from the outside internet.

Remote (as in, beyond the reach of wifi) mobile access can be enabled through the same networking strategies that enable remote access for standard Vision clients. For this, the mobile device must be able to access the Ignition Gateway via its cellular connection. One strategy would be to set up a VPN router and configure the mobile device as a VPN client. This way, the mobile device could directly access the LAN address of the Gateway as if it were on-site. Another technique would be to put the

Ignition Gateway in a DMZ so that at least one NIC had a public IP address. Or, an edge router could be configured to port-forward the HTTP and HTTPS ports to the Gateway. Coordination with your IT is advised when attempting to set up remote access.

To Launch the Mobile Client

1. Go to the Gateway **Home** page.
2. In the **Mobile Launch** section, click on `http://ipaddress:8088/main/system/mobile`. This address will need to be changed to reflect the computer that your Ignition system is installed on. The Client launches on your mobile device.



3. Choose the project you want to open, then login with your username and password. The Project is displayed. If there is only one project in the mobile project list, the list will be skipped and you will be sent directly to the login screen.
4. To check how many Clients are launched, go to the **Status > Vision Clients** page of the Gateway.

Controller

USER MANUAL SUPPORT

admin Sign Out

Launch Designer

HOME STATUS CONFIGURE

Search...

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

SYSTEMS

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

CONNECTIONS

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

Connections

Vision Clients

Client Stats

Active Clients: 1

Requests / sec: 6.9

Client Details

<u>Id</u> ▼	<u>User</u>	<u>Project</u>	<u>Uptime</u>	<u>Activity</u>	<u>Address</u>	<u>Memory</u>	<u>Actions</u>
10301C5A	admin	Parameters	15 minutes	✓ Active	mipsaro-ws (127.0.0.1)	38mb	More ▼ Details

Launch Mobile Client without viewing the mobile project selection page

It is possible to bypass the mobile project selection page by adding the project name to the URL:
<http://ipaddress:8088/main/system/mobile?project=projectName>

Viewing the Gateway Web Interface on a Mobile Device

The Gateway will automatically redirect mobile devices to the Mobile Project Selection screen. In some cases, it may be preferable to view the normal Gateway Web Pages on the mobile device, such as the Gateway Logs, or the Performance page. To bypass the mobile project selection screen, simply type the address of a specific page on your gateway into your mobile device's browser. For example, viewing the Overview page in the Status section on a mobile device would look like the following:

`http://ipAddress:port/main/web/status/`

Mobile Project Launch Links

You can add **mobile project launch links** to the home screen of a mobile device. Links can point to either the project listing page or directly to a single project. You can even type the launch link into the browser of your mobile device.

Mobile Client Settings

In the **project properties** in the Designer, there is a page specific to Mobile clients if you have the mobile module installed. Here you can enable the project for Mobile, set fit, auto-login, and more.

When designing or viewing your project from a mobile device, whether it is a tablet or a mobile phone, you want to make sure your device's pixel resolution is picture perfect. You can use the default **mobile client size setting** to "Fit to Device," or you can set a custom pixel size.

The screenshot shows the 'Project Properties' dialog box with the 'Mobile // General' tab selected. The left sidebar lists the following categories: Project (General, Permissions), Designer (General, Window Editing), Client (General, Launching, Login, Timing, User Interface), Mobile (General), and Mobile (General). The 'Mobile // General' tab contains the following settings:

- General**
 - ☒ Enable Mobile Launch
 - ☐ Hide from Launch Page
 - ☐ Enable Geolocation
 - Image Encoding: PNG (dropdown)
- Viewport**
 - ☒ Fit to Device
 - ☐ Custom: 800 (width) x 600 (height)
- Utility Bar**
 - ☒ Auto ☐ Hidden
- Auto Login**
 - ☐ Enable Auto-Login
 - Username: [text box]
 - Password: [text box]
 - Password (repeat): [text box]

At the bottom of the dialog are three buttons: OK, Apply, and Cancel.

Mobile Module Tags

Using the Mobile Module adds additional (client scoped) tags to the System Tags folder under **System** > **Mobile**. This adds Remote Address and User Agent values. If location is enabled in the mobile browser, the Location tags will show Latitude, Longitude, etc.

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

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

Mobile Client Settings

Common Mobile Settings

With the Mobile module installed, you can launch your projects on any modern smartphone or tablet. This ability does not require any re-design of your projects – a mobile Client launches the same projects that the Vision Clients launch.

See the [Mobile Module](#) for more information.

There is very little setup involved with the mobile module. Usually, with the exception of the the Server Address setting, you can keep all the the default settings.

1. Go to the **Configure** section of the Gateway.
2. Scroll down and go to **Mobile > Settings** page.

Setting the Java Path



INDUCTIVE
UNIVERSITY

Setting Java Path

[Watch the video](#)

The mobile clients can either use the same Java version as the Ignition server or it might be necessary to set a specific path to the Java version that you want the mobile clients to access. A likely scenario for this is if the Ignition server is upgraded to run on Java 9 but the underlying technology that the Mobile client uses to run can be limited to Java 8 pending technological upgrades in the future. Regardless, setting the Java path will result in more control over which version of Java runs the mobile clients. The settings take either the system path such as **Java** or a full path to the **java.exe** file.

To Set the Java Path

1. Go to the **Configure** section of the Gateway.

2. Scroll down, go to **Mobile > Settings** page. See the **Java Path** setting with **java** as the default path to the Java executable.
3. Enter the path in Java Path, for example, for Java 8 JRE you enter: `C:\Program Files\Java\jre8\bin\java.exe`

Java Path	C:\Program Files\Java\jre8\bin\java.exe The path to the Java executable. Must be a Java 8 JRE. The default value assumes that Java is on the path. (default: java)
------------------	---

DANGER

The mobile module will not work with Java 9+. If your Gateway is running on Java 9, you will need to install Java 8 and set the Java Path property of the mobile settings to it.

Setting Max Mobile Clients



INDUCTIVE
UNIVERSITY

Setting Max Mobile Clients

[Watch the video](#)

You can set the max number of concurrent mobile Clients in the Gateway **Home** page. This is a safety measure, because each Client consumes resources on the Gateway. Limiting the number of Clients to be launched prevents overloading the Gateway server.

To Set the Max Number of Mobile Clients

1. Go to the **Configure** section of the Gateway.
2. Scroll down, go to Mobile > Settings page. You will see the Max Clients setting, which is set to 15 by default. Max Clients is the max number of concurrent mobile clients. This number is set as a safety measure, because each client consumes resources on the Gateway, and limiting clients prevents overloading the Gateway server.

Max Clients	15 The max number of concurrent mobile clients. This is a safety measure, because each client consumes resources on the Gateway. Limiting clients prevents overloading the Gateway server. (default: 15)
--------------------	---

Available Mobile Settings

Settings

Setting	Description
Java Path	The path to the Java executable on the Ignition Gateway server machine. The Java 8 JRE is required for the mobile module. Mobile will not work with java 9. The default is java , which assumes that Java 8 is on the path and can be invoked merely with the java keyword.
Client Memory	The max amount of heap space allowed for each mobile client VM that is launched. Mobile clients are virtual clients that are launched on the server. All of the work is done on the server and transmitted to the mobile device so keep in mind that more mobile clients means more memory and CPU consumption on the server.
JVM Options	Command-line JVM options to use when launching mobile client VM's. Multiple options are separated with spaces. This option is made available mostly for troubleshooting by technical support staff, but if you are familiar with java and comfortable with command-line arguments, you can specify ones you may find useful.
Environment Variables	The environment variables to pass to the mobile client VM process. Use the NAME=VALUE format, one per line.
Max Clients	The max number of concurrent mobile clients allowed. This is a safety measure, because each client consumes resources on the Gateway. Limiting clients prevents overloading the Gateway server. (default: 15)
Server Address	This is the address to use when launching from the QR code on the launch page. This is a setting that often causes confusion. Initially this value is blank by default which results in the QR code pointing to the address of <code>http://localhost:8088/</code> . Since Ignition is not running on your mobile device, this address will not actually launch the mobile homepage. This should be set to the Ignition server address that can be reached by the mobile device.
Required Roles	A comma separated list of roles required to view the mobile homepage. If blank, no login will be required to view the project-listing page. (default:

Setting	Description
	blank)
Auto-select Single Project	Boolean. If there is only one mobile-enabled project, skip the project listing screen and automatically choose that project. (default: true)
Client Debug	Boolean. Displays debugging information in the client web interface.

Networking

Setting	Description
Callback Port	The port that the mobile Client VMs use to communicate to the Gateway on. The default is 45900, but if this port is already in use, you need to change this to an available port.
Callback Interface	The interface that mobile client VMs should use to communicate back to the Gateway on. The default, localhost, makes use of the loopback adapter, however if this host doesn't have a loopback adapter or if there are two network cards, set this to the IP address of the NIC that should be used for local loopback.
AJAX Timeout	The max time, in milliseconds, that each request has to complete. The default is 10,000 (10sec).
Image Fetch Strategy	The way in which images are retrieved. URL will fetch each image via a separate URL, and Base64 will encode each image in-line with the rectangle fetch command. (default: Base64)
Refresh Delay	The delay, in milliseconds, between web client screen-refresh requests. (default: 150)
Session Timeout	The timeout (in sec) for idle mobile web sessions. (default: 120)
Session Invalidation Retry Count	New in 7.9.9 When set to a number greater than zero, allows the client attempt to reconnection should its session become invalidated. Note that this property requires that auto-login is enabled on the project, as the retry will use the auto-login credentials.

Advanced Properties

The Mobile Module also has an option to allow VNC connections. This allows certain thin clients that do not support the Java Runtime Environment and also do not have an HTML 5 compatible browser to launch Ignition clients. The settings listed under the advanced properties section all have to do with configuring the VNC connection.

VNC Settings

Setting	Description
Enable VNC	Allows direct thin-client connection over VNC (also known as, RFB) protocol.
VNC Port	The port used for the VNC connections.
Project Name	The Mobile module only allows one of the projects on the Ignition Gateway to be viewed through VNC so you have to specify that project here. Unlike the normal mobile launch screen that allows you to choose a project, the project that you specify in this setting will be automatically launched when you connect via a VNC viewer application.
Project Width	The width of the project when it's launched.
Project Height	The height of the project when it is launched.

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

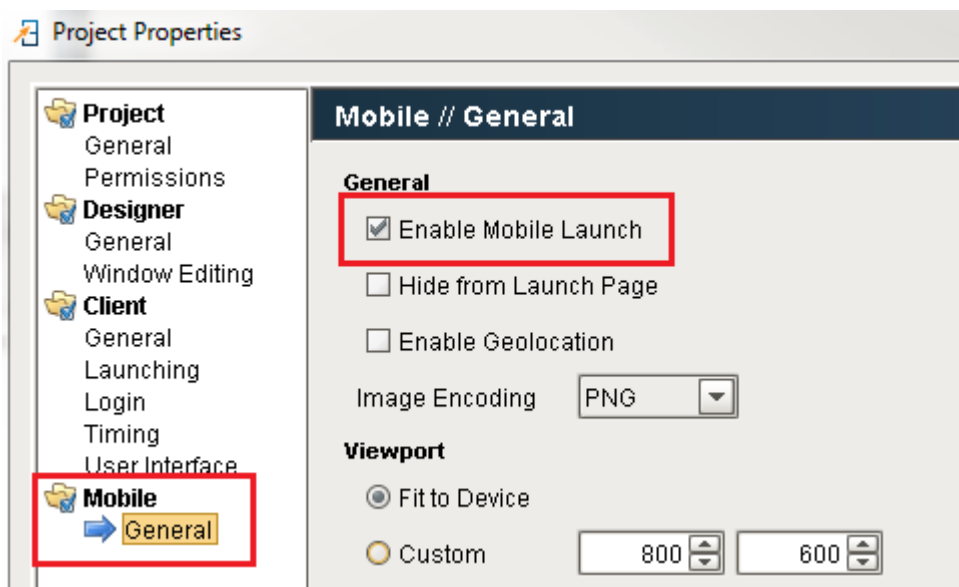
For documentation on the latest stable release, see the [latest version](#) (8.3).

Mobile Project Launch Links

Adding a Shortcut to the Home Screen of a Mobile Device

Mobile project shortcuts can easily be added to the home screen of a mobile device. The links can point to either the project listing page, or directly to a single project.

The following requires that the mobile device has **network access** to the Ignition Gateway. Additionally, at least a single project exists on the Gateway, and the project must be enabled for mobile launch. This property can be found under **Project Properties**:



Step 1 - Launch the Project on the Mobile Device

The Home tab of the Gateway Webpage contains a Mobile Launch link.

Controller

USER MANUAL SUPPORT

Sign In

Launch Designer

Ignition!
by inductive automation

HOME STATUS CONFIGURE

Launch Projects ?

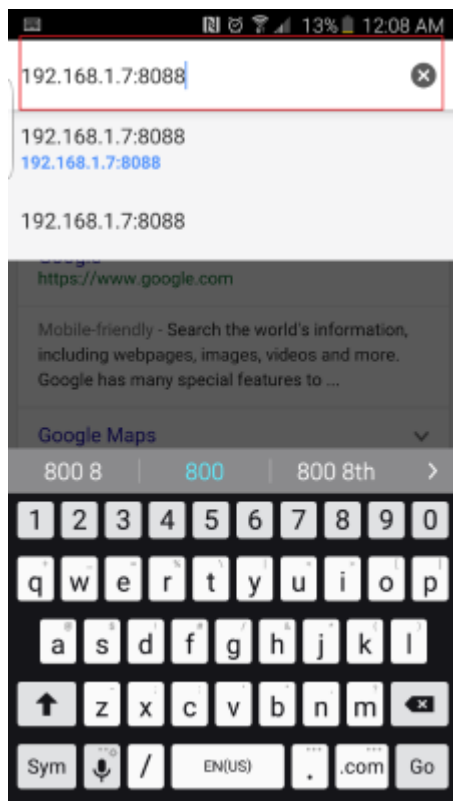
Mobile Launch ?

To launch a project on a mobile device, scan the barcode or navigate your device's web browser to:

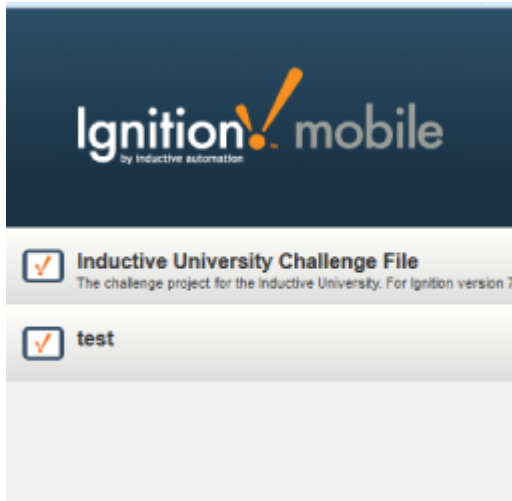
<http://10.20.7.236:8088/main/system/mobile>

Launch a mobile project on the device by either using a barcode scanner app, or type the launch link into a browser on the device.

Since the mobile device has network access to the Ignition Gateway, the browser on the device can navigate to the Gateway Webpage directly. This way the project can be launched by clicking on the link from the device.



If multiple mobile-enabled projects are present, a project listing page will appear.



NOTE

It is possible to bypass the mobile project selection page by adding the project name to the URL: <http://ipaddress:8088/main/system/mobile?project=projectName>. This will immediately launch the mobile project, and bypass the project listing page entirely.

If only a single mobile-enabled project exists on the Gateway, then it will automatically jump to the login page.

A screenshot of a mobile application login screen. The title bar is dark blue with the word 'Login' in white and a small orange checkmark icon on the right. Below the title bar, the text 'Turbine Farm' is displayed. There are two input fields: 'Username' and 'Password'. Below the 'Password' field is a 'Login' button and a small lock icon. The background is a light gray.

Step 2 - Add a Shortcut to the Device's Home Screen

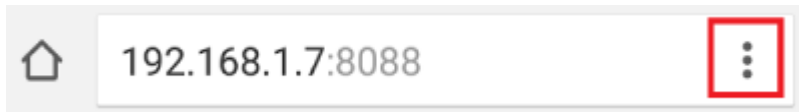
The second step differs depending on which mobile device you are using.

Android iOS

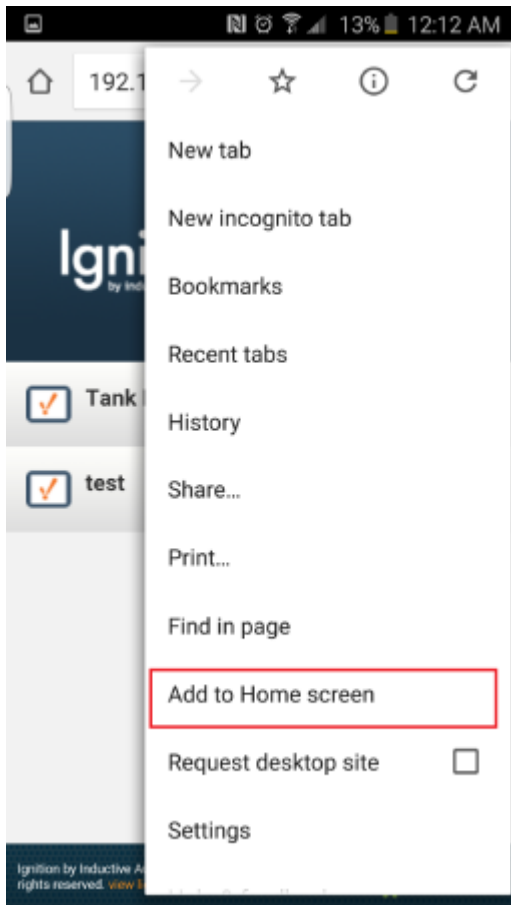
NOTE

The following instructions assumes Chrome will be used. If using a different browser, please see the browser's documentation for adding a link to the Home screen.

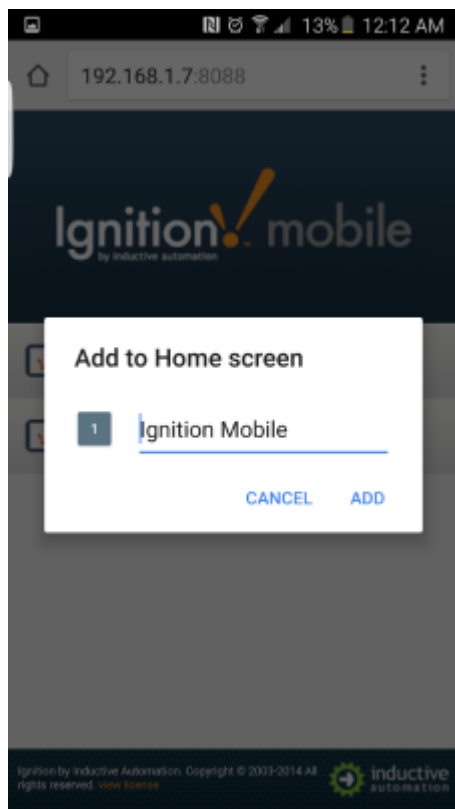
Next to the page's URL press the More icon.



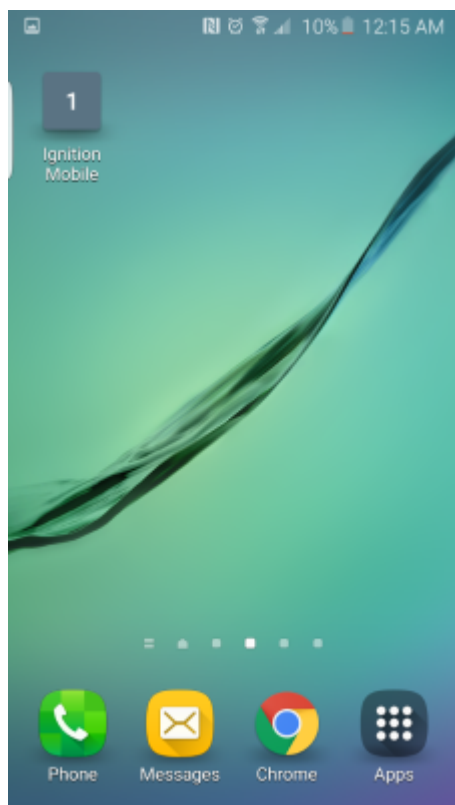
A list of options will appear. Press the **Add to Home screen** option.



Name the link that will appear on the Home screen and press **Add**.



The link will now be available on the Home screen.



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

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

Setting Mobile Client Size and Fit to Device



INDUCTIVE
UNIVERSITY

Setting mobile Size and Fit to
Device

[Watch the video](#)

Adjusting the Mobile Client Size

Mobile Clients can be set to specific sizes or told to fit themselves to the device. By default, they are set to Fit to Device, to allow for the many varied mobile phone and tablet screen sizes, and to allow for automatic adjustments when rotating the mobile device. However, if the project needs to be set to a specific size, that can be done as well.

In the Designer, go to the **Project > Properties**, and then to **Mobile > General**. You can see that the **Fit to Device** is selected by default. **Fit to Device** will resize itself when the device is rotated, as well as when the client is initially launched. You can also choose **Custom** and specify any size you wish.

Project Properties

Project
General
Permissions

Designer
General
Window Editing

Client
General
Launching
Login
Timing
User Interface

Mobile
General

Mobile // General

General

☒ Enable Mobile Launch

☐ Hide from Launch Page

☐ Enable Geolocation

Image Encoding: PNG

Viewport

☒ Fit to Device

☐ Custom: 800 x 600

Utility Bar

☒ Auto ☐ Hidden

Auto Login

☐ Enable Auto-Login

Username:

Password:

Password (repeat):

OK Apply Cancel

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

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

Direct Launch Links



INDUCTIVE
UNIVERSITY

Using the Direct Launch Links

[Watch the video](#)

Clients can be opened with the Direct Launch Links in addition to using the Gateway Home page. These links can be used in any browser on the network and will launch the **.jnlp** files without a desktop shortcut.

To Open a Client with the Direct Launch Link

1. Go to the **Configure** section of the Gateway.
2. Select **System > Projects**, and you will see a list of your projects on the **Projects** page.
3. Click the **More** link to the far right of the row for the project you want to launch, and select **details**. You will see the **Project details for 'your_project_name'**, and the **Published** and **Staging** launch links.

The screenshot shows the Ignition Gateway web interface. The top navigation bar includes 'HOME', 'STATUS', and 'CONFIGURE' (which is active). A green banner at the top of the main content area indicates 'License Incomplete' with a timer at 0:00:00 and buttons for 'Reset Trial' and 'View Modules'. The left sidebar shows the 'SYSTEM' menu with 'Projects' selected. The main content area displays a table of projects. The 'Development' project row is highlighted, and its 'More' dropdown menu is open, showing options like 'delete', 'details' (highlighted with a red box), 'copy', and 'export'.

Name	Description	Authentication	Default Database	Default Tag Provider	
Development		default	MySQL	default	More edit
DrillDown		default	MySQL		More edit
Maria		default	MySQL		More edit

4. Right-click on **Published/Windowed Mode**. If you are using Google Chrome, click **Keep** on the small dialog box that appears on the bottom-left of your window, then click on the **.jnlp** file to start the Client.

Project details for 'Facility 1'

Launch Links

Published

[Windowed Mode](#)

[Fullscreen Mode](#)

Staging

[Windowed Mode](#)

[Fullscreen Mode](#)

Other browsers should start running immediately or may ask you if you want to save/run the .jnlp file. Select '**Run**' if prompted.

Passing Parameters to a Client

Client Tag values in Ignition clients can be set when the client launches. You can do this by adding parameters to the end of launch links. This allows parameters to be passed into the client either from a URL in a website, through sending a co-worker a link, or by updating the properties on a desktop shortcut.

NOTE

Client Tags that are set by the client launcher should not contain spaces in the Tag names.

Using the Launch Link

The example below assumes that you have two Client Tags in the root folder of your project, called **LaunchTag** and **LaunchTag2**.

```
http://1.1.1.1:8088/main/system/launch/client/myterminal.jnlp?
LaunchTag=AAA&LaunchTag2=ZZZ
```

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

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

Launching Clients

Window Option



INDUCTIVE
UNIVERSITY

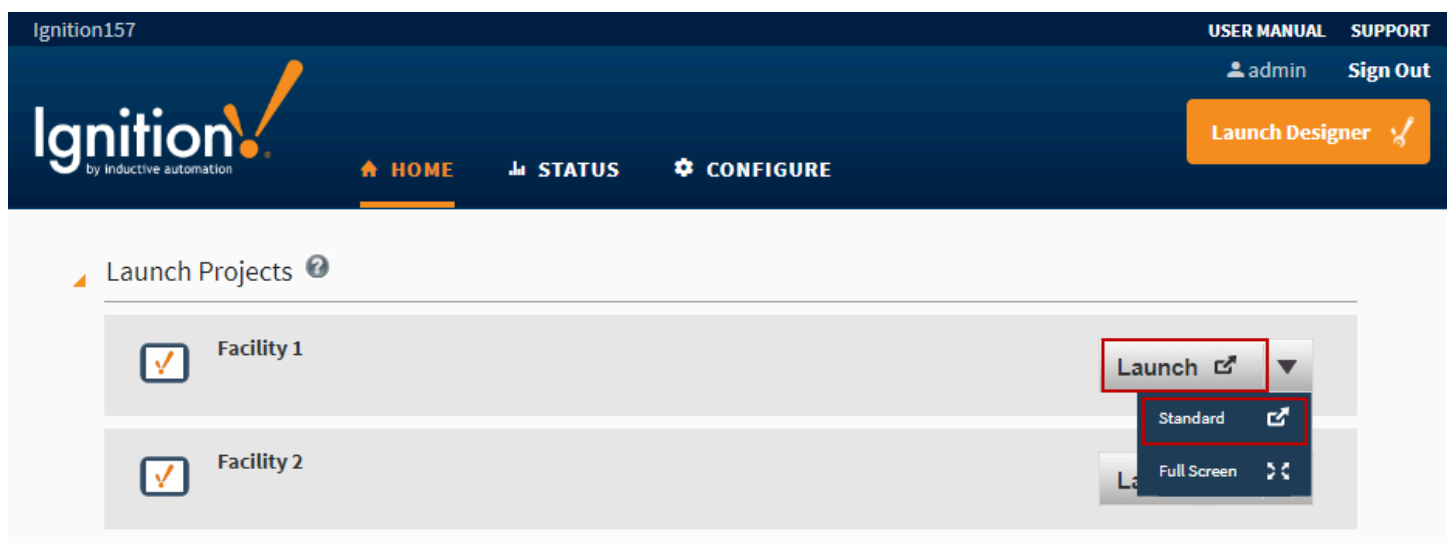
Using the Windowed Option

[Watch the video](#)

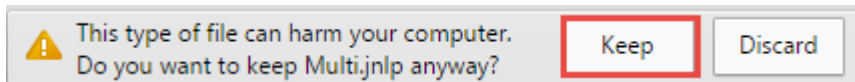
Clients can be launched with Java Web Start in Windowed mode from the Gateway Webpage or the Designer. Windowed mode creates a Client that appears as a normal windows program.

To launch a Client in Windowed mode

1. Go to the Gateway Home page.
2. Click on the **Launch** button for the project you want to open in the Client. This will launch the Client in the default **Windowed** mode. Alternately, from the dropdown you can click the Standard option.



3. If you are using the Google Chrome web browser, click **Keep** on the small dialog box that appears on the bottom-left of your window, and then click on the **.jnlp** file to start the Client. The login screen for the Client is displayed. You can keep the Client screen as large as the whole screen or as small as you wish.



Other browsers should start running file immediately or may ask you if you want to save/run the .jnlp file. Select 'Run' if prompted.

4. Type in the **Username** and **Password** to see your project.

Your project is now displayed in the Client.

Full Screen Option



INDUCTIVE
UNIVERSITY

Using the Full Screen Option

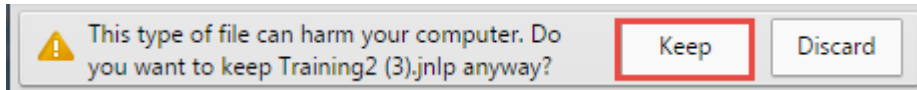
[Watch the video](#)

Clients can be launched with Java Web Start in Full Screen mode from the Gateway Webpage or the Designer. Full Screen mode creates a Client that takes up the entire screen including the Windows Taskbar. Special keys like the Tab, Alt, Ctrl, and Window/Option keys can be used in Full Screen mode. To prevent users from getting to their desktop, these keys can be blocked by enabling window settings and programs that are not part of Ignition. This is typically done when a keyboard is available, but often omitted when only using a Touchscreen monitor.

To launch a Client in Full Screen mode

1. Go to the Gateway Home page.
2. From dropdown of the **Launch** button, select the **Full Screen** mode to launch the project you want to open in the Client.

3. If you are using the Google Chrome web browser, click **Keep** on the small dialog box that appears on the bottom-left of your window, then click on the **.jnlp** file to start the Client. The login screen for the Client is displayed covering the full size of your screen. In the **Full Screen** mode, you can not make the screen smaller as you could do with the **Windowed** mode.



Other browsers should start running file immediately or may ask you if you want to save/run the .jnlp file. Select 'Run' if prompted.

4. Type in the **Username** and **Password** to see your project.

Your project is now displayed in the Client.

Mobile Option



INDUCTIVE
UNIVERSITY

Using the Mobile Option

[Watch the video](#)

Clients can be launched on a mobile device using the Mobile Module. They are opened behind the scenes on the server and sent using the mobile device's web browser, not as a native App. A desktop environment (that is, X Window) is required for any headless server installation (common on Linux servers). A "headless" server installation means the computer Ignition is installed on has no desktop to interact with, ie: you have to use a command prompt to access that computer. Most installations will not use this method.

To launch a Client on a Mobile device

1. Go to the Gateway Home page.
2. From the Mobile Launch section, click on the <http://localhost:8088/main/system/mobile> link.

The Ignition Mobile launch page is displayed showing a list of all your projects.

Controller

USER MANUALSUPPORT

admin

Sign Out

Launch Designer

Ignition
by inductive automation

HOMESTATUSCONFIGURE

Welcome to the Ignition Gateway

Launch Projects ?

Mobile Launch ?



To launch a project on a mobile device, scan the barcode or navigate your device's web browser to:

<http://localhost:8088/main/system/mobile>

3. Click on your project name.

This downloads and displays your project.

NOTE

The "localhost" address will only work on the computer Ignition is installed on. If you are accessing this webpage from a mobile device, you will automatically be redirected to the mobile launch page.

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

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

Multi-Monitor Clients



INDUCTIVE
UNIVERSITY

Multi-Monitor Clients

[Watch the video](#)

Multiple Desktops

In some situations, such as control rooms, or workstations with multiple monitors, it may be preferable to have clients open on several monitors so that different windows are simultaneously in view. Instead of opening several different clients, it is possible to open a single client, and spawn multiple Desktops through scripting. Desktops are additional workspaces where windows may be opened. They are similar in functionality to a standalone Client in that they may be positioned and resized independently of other Desktops and Clients, but share a session ID with the initial Client that spawned the Desktop.

Client Tags and Property Values

The value of Client Tags are shared between each Desktop. This provides an easy method to change values on one desktop from another without interacting with other Clients: simply write to a Client Tag.

Desktops act as separate clients in regard to property values. For example, if a Text Field is placed on a window, and multiple Desktops open that same window, values entered into one Desktop will not overwrite the other Text Fields. If synchronization on these components is preferred, then simply bind the property to a Tag.

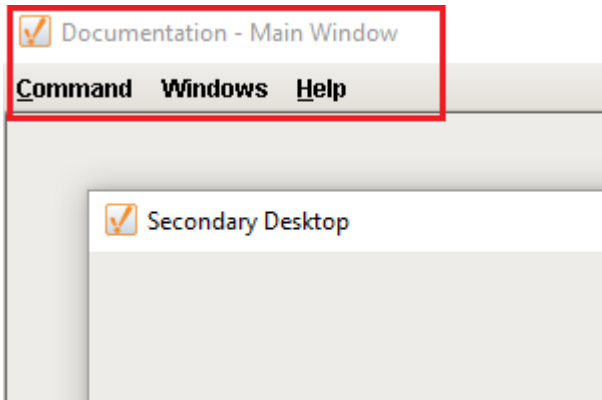
Handles

When creating a new Desktop, an optional handle may be assigned to the Desktop. This acts as a name, or reference to the Desktop. If a handle is not provided, then the Desktop may be referenced by the screen index. Handles and indices are useful when trying to interact with specific Desktops from a Python script.

Spotting the Primary Desktop

When multiple Desktops are open, it is important to know that only the Primary Desktop will have a menu bar. Additionally, the title bar on the client will show the name of the project. Additional Desktops will instead show the handle or index of the Desktop by default. However, a custom title may be used when the Desktop is invoked.

Below we see two Desktops. The highlighted Desktop is displaying the title of the project (Documentation). Because this is the Primary Desktop, a menu bar is present. The other Desktop was given a title of "Secondary Desktop". A menu bar is not present because this Desktop is not the primary.



Project Updates

When a project update is pushed to a Client, the Update banner will only appear on the Primary Desktop (assuming the Update Mode of the project is set to **Notify**). Updating the Primary Desktop will also push the changes to all other local Desktops, so there is no need to update each Desktop individually.

Opening Another Desktop

Another Desktop may be opened by calling `system.gui.openDesktop`:

```
#This will open a new desktop without any windows, and a name of "0"  
system.gui.openDesktop()
```

However, without specifying which windows to open, the Desktop will open without any opened windows. It is recommended to specify at least one window, a title, and a handle for the new desktop. Assuming a window exists at the path "Main Windows/Main Window", the following would open a new Desktop, open the specified window, and specify a title and handle for the window.

```
#Create a list of window paths to open in the new desktop
windowToOpen = ["Main Windows/Main Window"]

#Defines a name for the Desktop, which will be used as both the the title and
handle of the window.
name = "2nd Desktop"

#Creates a new desktop. The desktop will open the windows listed above.
system.gui.openDesktop(windows=windowToOpen, title=name, handle=name)
```

Navigating Windows in Desktops

Functions for the **gui** and **nav** scripting modules will execute in the Desktop that originated the call: If the Primary Desktop calls **system.nav.swapTo**, then the Primary Desktop will swap to a new window, but all other desktops will remain unaffected. However, it is possible for a script on one Desktop to force a navigation or GUI change on another Desktop with the following functions:

- **system.gui.desktop**
- **system.nav.desktop**

```
# to open a popup in your second desktop
# if you are identifying desktops by number, they are zero indexed
system.nav.desktop(1).openWindow('Popups/Popup')
```

Additional scripting functions that interact with desktops exist in the gui and nav scripting modules. Please see the **Scripting Appendix** for more details.

Opening a Desktop on Each Monitor

Sometimes you may want your client to open a new desktop on each of your other monitors. It's pretty simple to get all of your monitors and open a client on each, but then you will have two on your main monitor. The following code block shows you how to skip the primary monitor and even how to open specific windows on each new desktop. This example assumes you have a **'Main Window/Overview'** window, and that window has a custom string property in the root container named **'Display'** to pass values into. Bind a label component to that custom property to easily check your script. This script is best placed in a Client Startup Script to open a client for each monitor on startup.

```
# Get the screen information for all of your monitors.
# The getScreens() function returns a dataset of the index (0 based), width, and
height for each monitor.
screensDataset = system.gui.getScreens()

# Open the first window of the project in the (current) primary monitor.
screenIndex = screensDataset[0][0]
monitorNum = screenIndex + 1
primaryScreenText = 'This is Monitor %d' %monitorNum
system.nav.swapTo('Main Windows/Overview', {'Display':primaryScreenText})

# Step through all of the screen information, starting with index 1 instead of 0.
for screenDetails in screensDataset[1:]:
    # unpacks the tuple that is returned for each of the monitors present.
    # Consists of screen index, width, and height of the screen.
    screenIndex, screenWidth, screenHeight = screenDetails
    monitorNum = screenIndex + 1
    screenText = "This is Monitor %d" %monitorNum

    # Open an empty frame on the next monitor.
    # Assign a handle and apply the width/height for the monitor you are opening
    on
    handleName = "Monitor %d" %monitorNum
    system.gui.openDesktop(screen=screenIndex, handle=handleName,
width=screenWidth, height=screenHeight)

    # Open the Main Window on this new desktop and pass the parameters needed.
    system.nav.desktop(handleName).swapTo('Main Windows/Overview',
{'Display':screenText})
```

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

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

Native Client Launchers



INDUCTIVE
UNIVERSITY

**Using the Native Client
Launchers**

[Watch the video](#)

The Ignition Gateway contains downloadable native executables that can launch Clients directly without invoking Java Web Start. Using the native executables you can set up Clients in ways that are impossible to achieve with Java Web Start, such as being able to launch Clients automatically as part of a machine startup script.

The Native Client Launcher automatically searches the network for existing Ignition Gateways and lists existing projects. It is a more robust shortcut than the default desktop shortcuts created by Java.

NOTE

The Native Client Launchers require Java to be installed on the Client machine.

New in 7.9.10 Native Client Launchers now have Java 9 embedded in them, so Java no longer needs to be installed on the Client machine.

Download and Installation

 Native Client Launchers 



Client Launcher for Windows
Save the .exe file to your local machine and double-click to get started.

[↓ Download](#)



Client Launcher for Linux
Save the .tar.gz file to your local machine, extract using 'tar -xvf' and run the executable .sh file.

[↓ Download](#)



Client Launcher for Mac OSX
Save the .dmg file to your local machine, double-click to open and drag Client Launcher to your Applications folder.

[↓ Download](#)

Native client launchers for Windows, Linux, and Mac OSX are available on the Gateway Home page. Click on the **Download** link to download the native launcher that you need for your operating system. You can hide the native client launcher section on the Gateway **Configure** page by disabling the section under the Gateway **Settings** > **Homepage Config** tab.

Gateway Settings

[Settings](#) [Homepage Config](#)

✓ Homepage panel "Native Client Launchers" will now start expanded.

Panel	Start Expanded	Enabled	
Welcome to the Ignition Gateway	☒	☒	[down]
Launch Projects	☒	☒	[up] [down]
Mobile Launch	☒	☒	[up] [down]
Native Client Launchers	☒	☐	[up]

Windows

macOS

Linux

Move **clientlauncher.exe** to a convenient location after it has downloaded to your local machine. Double-click **clientlauncher.exe** to start.

Configuration

When the client launcher opens for the first time, you must select a Gateway on the Gateway Configuration screen. If your Gateways have multicast enabled, and multicast transmission is allowed on your network, you can see a list of Gateways under the Available Gateways tab. Select a Gateway from the list and click the **Select Gateway** button. The client launcher will attempt to connect to the Gateway that you selected.

If you do not see your Gateway, you can still manually enter Gateway address under the Manually Input Gateway tab. The format is `host:port`.

NOTE

If you are using SSL, the Gateway address will need to specify both the Ignition Gateway port and the SSL port using the format `host:port:port`. See [Gateway Port Reference](#) for more information.

Examples

```
// The following will connect to a Gateway using an IP address
10.10.10.1:8088

// The following will connect to a Gateway running on the local machine
localhost:8088

// The following will connect to a Gateway using SSL
10.10.10.1:8088:8043
```

After you entered your address, click the **Apply** button. The client launcher will attempt to connect to the Gateway that you selected.

After you selected a Gateway, the client launcher will attempt to contact the Gateway and will update the **Status** field at the top of the screen. If there was an error while attempting to contact the Gateway, you can hold your mouse over the **Status** field to see what the error was. You can also check `<user home folder>/.ignition/clientlauncher-data/clientlauncher.log` for error information.

NOTE

You cannot use a client launcher to connect to older Gateways.

The minimum Gateway versions that can be used by client launchers are 7.5.11 for the 7.5

Ignition platform and 7.6.4 for the 7.6 Ignition platform.

Any 7.7 or later Ignition platform can be used with the client launchers.

Temporary Configuration

The client launcher is typically used to open different projects residing on a single configured Gateway. However, you can also create temporary connections to other Gateways by checking the **Temporary Connection** checkbox on the Gateway Configuration screen. This prevents the selected Gateway from being saved in `launch.xml`, which may be helpful if you have already configured a default Gateway and you do not want to lose its connection settings. After the connection has been established and saved, the Projects screen will be shown for the temporary Gateway. You can then launch a project or the Designer as you normally would. If the **Create desktop icon** checkbox is checked for the launched project, the temporary Gateway network address is saved in the desktop shortcut.

Pre-configure for Remote Clients

The client launcher can be pre-configured locally and then deployed to remote client machines via a zip file. Using this capability allows a client launcher to immediately connect to a Gateway on the very first launch.

1. Configure a client launcher on a local machine to point to a Gateway.
2. Copy the `<user home folder>/.ignition/clientlauncher-data` folder next to the launcher executable.
3. Zip everything together and distribute the zip to other machines.

4. **Windows & Linux** macOS

The launcher can use the `clientlauncher-data/` folder that was placed next to the executable.

Redundancy

The 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 Master Gateway and the Backup Gateway until one responds or the user closes the launcher.

Managing Multiple Gateways

It is possible to configure the client launcher to always start on the Gateway Configuration screen. This allows you to connect to any number of Gateways using the same launcher. To enable this configuration, open the `launch.xml` file (details on this file are in the Custom Launch Settings section below). Set the `initconnect` property to **false** and save the file.

Gateway Favorites

If you are managing multiple Gateways, it may be helpful to mark some Gateways as favorites. These Gateways are always listed first when accessing the Gateway Configuration screen. You can also filter out non-favorite Gateways by checking the Favorites Only checkbox on the Gateway Configuration screen. By setting the `initconnect` property to false (as documented above in the Managing Multiple Gateways section), you can force the client launcher to always display your favorite Gateways list any time you start the launcher.

Using the Native Client Launchers

1. Go to the **Home** page on the Gateway webpage.
2. Scroll to the bottom of the page to see the **Native Client Launchers** section.
3. Click on the **Download** button to download the native launcher that you need for your operating system.
 - For Windows, you will see a **clientlauncher.exe** file.
 - For macOS, double-click **clientlauncher.dmg**, drag **Client Launcher.app** into the Applications folder.
 - For Linux, you will see a **clientlauncher.sh** file.
4. Click on the **.exe**, **.app**, or **.sh** file (depending on your OS) to open the page that shows URLs for available Gateways.
5. Choose the Gateway you want, click **SELECT Gateway** button, and click **Save & Close**.
 - On the window that is displayed you can launch your project.
 - If your Gateways have multicast enabled (**Multicast Enable** check box is selected on the **Configure > Gateway Settings** page), and multicast transmission is allowed on your network, you can see a list of Gateways under the **Available Gateways** tab.
6. Select a Gateway from the list and click the **SELECT Gateway** button.

- The Client launcher will attempt to connect to the Gateway you selected.

7. If you don't see your Gateway, you can manually enter the Gateway address under the **Manually Input Gateway** tab with the format `host:port`. After you have entered your address, click **Apply**.

- The Client launcher will attempt to connect to the Gateway you selected.
- After you selected a Gateway, the Client launcher will attempt to contact the Gateway and will update the **Status** field at the top of the screen. If there was an error while attempting to contact the Gateway, you can hold your mouse over the **Status** field to see what the error was.
- You can also check `<user home folder>/.ignition/Clientlauncher-data/Clientlauncher.log` for error information.

You can hide the **Native Client Launcher** section on the **Gateway Configure** page by disabling it in the **Gateway Settings > Homepage Config** tab.

Launching Projects

After the Gateway has been configured, you can launch a client project or the Designer from the Projects screen. After a project has been launched, a desktop shortcut is created. You can use the desktop shortcut to directly launch the project from the client launcher without needing to select the project again from the Projects screen. A client project will not be visible for launch unless all the conditions below are satisfied:

- The project has at least one main window
- The project is not disabled in the Gateway
- The project is not hidden from launch via the project properties
- The project's configured window launch mode matches the client launcher window mode. For example, if a project's default launch mode is Windowed mode and the "Full Screen" button is not enabled in the project's properties, this means that the project cannot be launched when the client launcher is running in fullscreen mode.

If a project is no longer available in the Gateway, launching the desktop shortcut for the old project will cause the client launcher to display the Projects screen along with an error message at the top of the screen.

Custom Launch Settings

Client launcher configuration settings, log files and temp files are stored by default in `<user home folder>/.ignition/clientlauncher-data/`. The following settings can be configured in `launch.xml`:

Setting Name	Setting Description	Example Setting
gateway.addr	The gateway that you have configured which contains your launchable projects.	10.20.7.122:8088
gateway.backup.addr	Automatically set by client launcher when using a redundant setup.	10.20.7.122:8088
gateway.fallback.addr	Allows a fallback Gateway to be specified if neither the master Gateway nor the backup Gateway cannot be reached after a specified number of attempts. This setting is particularly useful for the local client fallback scenario, where you can still launch a local client if remote Gateways cannot be reached. Note that you must still specify the fallback.project setting, and your retries setting must be 0 or greater.	localhost:8088/main
fallback.project	The project to launch if the native launcher is attempting to launch using the specified fallback Gateway.	myproject
timeoutsecs	The maximum number of seconds allow for any gateway communication. Any communication that exceeds this amount will cause the client launcher to abort the communication and try again if configured.	30
retries	How many times to attempt to contact a gateway again if an error occurred during communication. The default setting of -1 cause the client launcher to try again forever until the client launcher is manually closed. A setting of 0 will cause the client launcher to abort communication after the first failure and display the Gateway Configuration screen. A setting of 1 or more will cause the client launcher to make X more attempts before aborting and showing the Gateway	-1, 0, 1

Setting Name	Setting Description	Example Setting
	Configuration screen, or attempting to load the fallback project if configured for fallback.	
windowmode	Controls whether or not to launch the client launcher in fullscreen mode. The fullscreen setting will cause the client launcher to take over the entire screen. When launching a client that also has fullscreen mode configured, the fullscreen client will replace the fullscreen client launcher after launch. The default setting is "window", which launches the client launcher in windowed mode.	window, fullscreen
screen	The screen index indicates which monitor to use when launching in fullscreen mode.	0,1
show.closebutton	When this is set to false, all close buttons on the client launcher are hidden. Use this setting along with the fullscreen mode to set up a dedicated touch-screen terminal.	true, false
multicast.addr	This setting must match the gateway's multicast address setting in order to see a list of available gateways on the Gateway Configurations screen.	231.1.1.1
multicast.receive.port	This setting must match the gateway's multicast receive port in order to see a list of available gateways on the Gateway Configurations screen.	4446
initconnect	When this is set to false, the client launcher will not attempt to contact the configured Gateway, but instead will immediately open the Gateway Configuration screen. This allows you to choose different Gateways using a single client launcher.	true, false

Setting Name	Setting Description	Example Setting
tempconnection	When this is set to true, the Temporary Connection checkbox on the Gateway Configuration screen will be checked when the client launcher is started. By setting this property to true and setting the initconnect property to false, you can use the client launcher to access any number of Gateways without saving the selected Gateway address in launch.xml.	true, false
jvm-args	Allows JVM-specific settings to be set on the client that is launched, such as MaxPermSize memory allocation.	
other-args	See the "Setting client tags" section below.	- XX:MaxPermSize=512m

Startup Parameters

The settings below can also be passed to a client launcher executable as startup parameters. Note that many of the settings overlap with the custom launch settings above, and if present, they will override the settings in launch.xml.

Setting Name	Setting Description	Example Setting
gateway.addr	The gateway that you have configured which contains your launchable projects.	10.20.7.122:8088
gateway.backup.addr	Automatically set by client launcher when using a redundant setup.	10.20.7.122:8088
gateway.fallback.addr	Allows a fallback Gateway to be specified if neither the master Gateway nor the backup Gateway cannot be reached after a specified number of attempts. This setting is particularly useful for the local client fallback scenario, where you can still launch a local client if remote Gateways cannot be reached. Note that you must still specify the fallback.project setting, and your retries setting must be 0 or greater.	localhost:8088/main
fallback.project	The project to launch if the native launcher is attempting to launch using the specified fallback Gateway.	myproject
timeoutsecs	The maximum number of seconds allow for any gateway communication. Any communication that exceeds this amount will cause the client launcher to abort the communication and try again if configured.	30
retries	How many times to attempt to contact a gateway again if an error occurred during communication. The default setting of -1 cause the client launcher to try again forever until the client launcher is manually closed. A setting of 0 will cause the client launcher to abort communication after the first failure and display the Gateway Configuration screen. A setting of 1 or more will cause the client launcher to make X more attempts before aborting and showing the Gateway Configuration screen, or attempting to load the fallback project if configured for fallback.	-1, 0, 1

Setting Name	Setting Description	Example Setting
windowmode	Controls whether or not to launch the client launcher in fullscreen mode. The fullscreen setting will cause the client launcher to take over the entire screen. When launching a client that also has fullscreen mode configured, the fullscreen client will replace the fullscreen client launcher after launch. The default setting is "window", which launches the client launcher in windowed mode.	window, fullscreen
screen	The screen index indicates which monitor to use when launching in fullscreen mode.	0,1
show.closebutton	When this is set to false, all close buttons on the client launcher are hidden. Use this setting along with the fullscreen mode to set up a dedicated touch-screen terminal.	true, false
scope	Indicates whether the project being launched is a client (scope C) or a Designer (scope D)	C, D
project	The project name. This field is required if the scope is set to C, indicating that a client is being launched.	myproject
initconnect	When this is set to false, the client launcher will not attempt to contact the configured Gateway, but instead will immediately open the Gateway Configuration screen. This allows you to choose different Gateways using a single client launcher.	true, false
jvm-args	Allows JVM-specific settings to be set on the client that is launched, such as Xmx, which is the maximum size of the Client/Designer.	-Xmx1024M
tempconnection	When this is set to true, the Temporary Connection checkbox on the Gateway Configuration screen will be checked when the	true, false

Setting Name	Setting Description	Example Setting
	client launcher is started. By setting this property to true and setting the initconnect property to false, you can use the client launcher to access any number of Gateways without saving the selected Gateway address in launch.xml.	

Client Startup Examples

Windows:

```
"C:\ClientLauncher\clientlauncher.exe" scope=C project=myproject windowmode=window
```

Linux:

```
./clientlauncher.sh scope=C project=myterminal windowmode=fullscreen screen=0  
show.closebutton=false
```

NOTE

For another example of automatically launching clients with the Native Client Launcher, please see the following Knowledge Base Article: [Launch Project to Multiple Monitors](#)

Setting Client Tags

Client Tag values in Ignition clients can be set by the client launcher upon startup. There are two ways you can do this. The examples below assume that you have two client tags in the root folder of your project, called **LaunchTag** and **LaunchTag2**.

Using the launch.xml

The first way is to add the tag info to the other-args section in launch.xml, as shown below. These settings would take effect for all clients started from the client launcher.

```
<other-args>  
<arg>-Djavaws.launchparams=LaunchTag;LaunchTag2</arg>  
<arg>-Djavaws.launchparam.LaunchTag=AAA</arg>  
<arg>-Djavaws.launchparam.LaunchTag2=ZZZ</arg>  
</other-args>
```


Using the Command Line to Launch

The second way is run the client launcher from a command line or a desktop shortcut and add these lines to the end of the command:

```
-Djavaws.launchparams="LaunchTag;LaunchTag2" -Djavaws.launchparam.LaunchTag="AAA" -  
Djavaws.launchparam.LaunchTag2="ZZZ"
```

Starting the client launcher from a command line in this case would look like the example below (Linux example shown).

```
./clientlauncher.sh scope=C project=myterminal windowmode=window -  
Djavaws.launchparams="LaunchTag;LaunchTag2" -Djavaws.launchparam.LaunchTag="AAA" -  
Djavaws.launchparam.LaunchTag2="ZZZ"
```

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

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

Projects

What is a project?

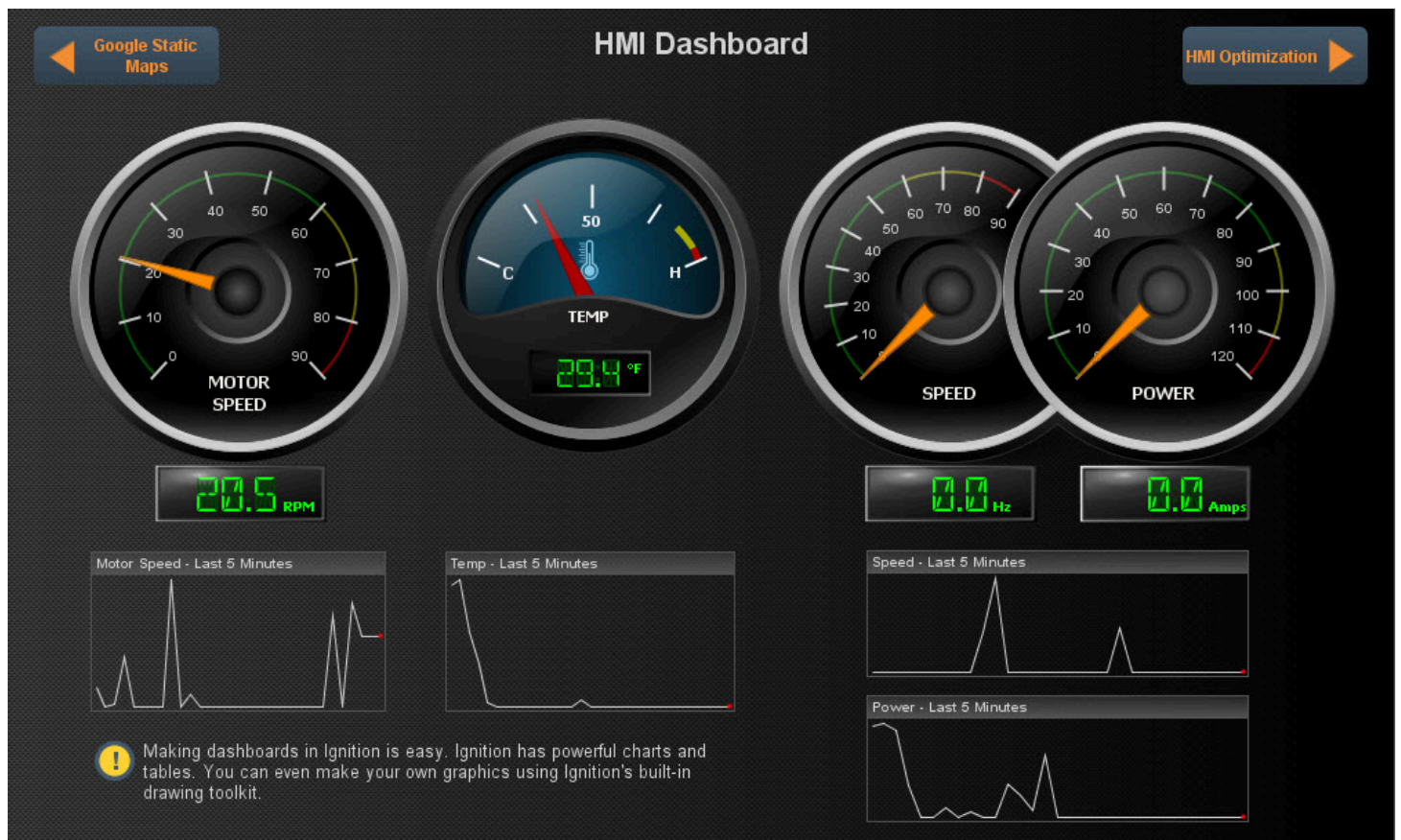
Ignition has two main parts, the Gateway and your projects. The Gateway holds all the shared information like database connections, device connections, Tags, and more. Projects hold all the designed elements that do work. Your projects can hold both interactive elements (like controls, charts, reports, entry forms, and more) and persistent elements (like historical loggers, automated reports, etc).

Projects are predominantly used to create the screens that your users can interact with, the **Visualization** part of Ignition. Here you can create any system you want, from copying existing HMI/SCADA applications to whole new systems with anything you could want to do. The windows in your project can be used for history charts, reports, database forms, alarms, drawing components, scripting, templates and much more.

In Ignition, a project is a unit of configuration that contains:

- **Windows and components:** The HMI and SCADA controls to interact with Tags and databases
- **Transaction Groups:** A bi-directional link between databases and PLCs
- **Templates:** A collection of components that can be re-used and quickly updated
- **Reports:** PDF reports for displaying and recording data
- **Scripts:** Timer and event based scripts used throughout the system
- **General settings and properties:** The settings that control access, resource connections, layout, timing, and more

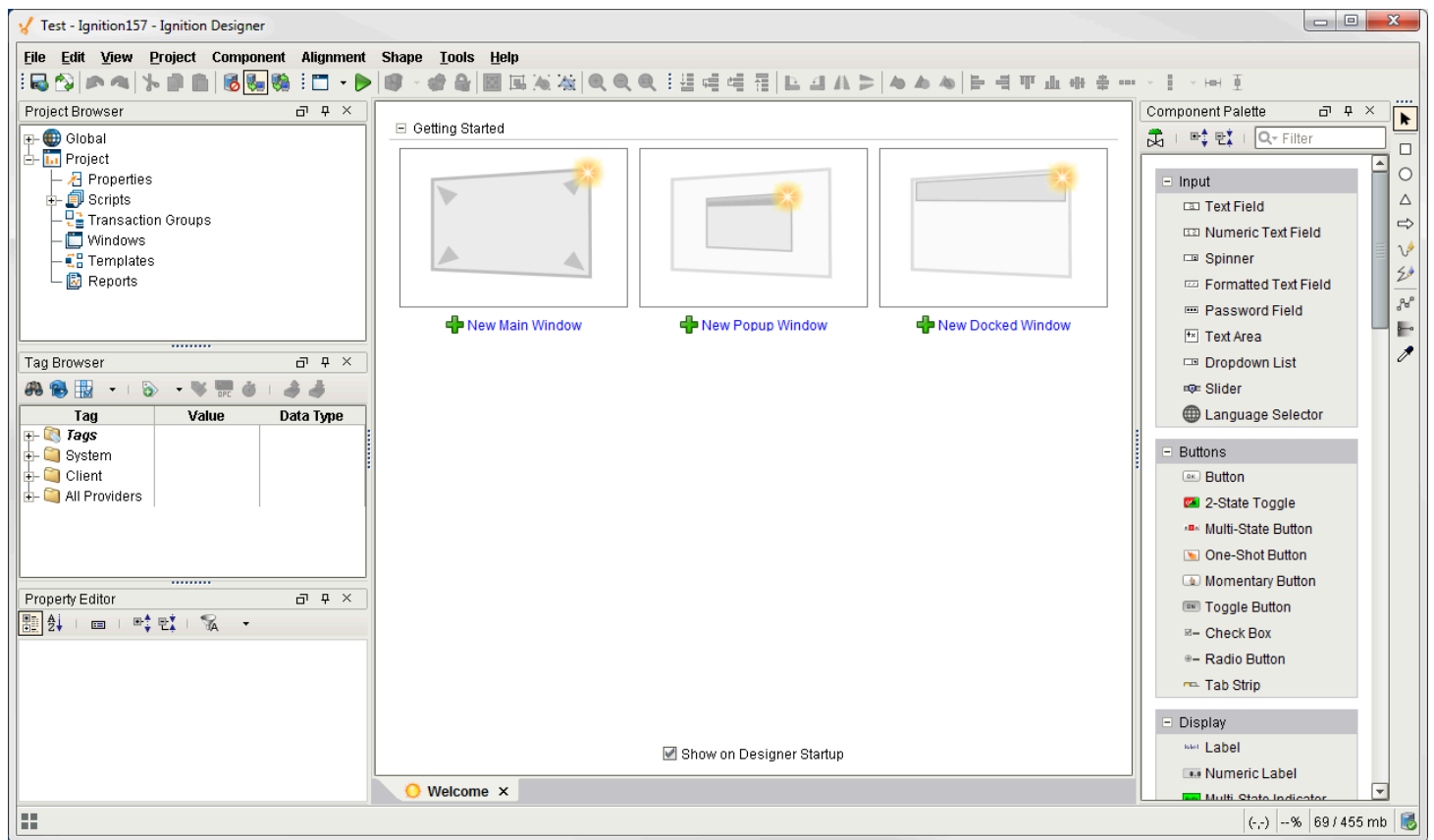
You use the Designer to configure and create projects. The projects are then viewed in the runtime Clients (if windows were created). You can create as many projects as you want, and users can easily jump between projects on the fly or open multiple projects at the same time.



Designing a Project

When you launch the Designer, you're asked to select or create a project. The Designer then launches the project and you can set and modify the different types of project settings and resources. In the Designer, you can create any number of projects. Projects that contains viewable elements, such as windows or reports, will have a launch link on the Gateway homepage. Otherwise, the project runs in the Gateway and will not have a Client runtime.

There is no limits to the number of projects that can be created on a Gateway, but each runtime Client or the Designer can operate only on one project at a time.



You use the different tools in Ignition such as components, shapes, images, Symbol Factory graphics, and Scalable Vector Graphics (SVGs) to create the components. Configuring components is the bulk of the designer's work when designing a Vision project. The basic workflow is to take a component from the palette and drop it into a container on a window. you can use the **Property Editor** panel to alter the component's properties, which changes the component's appearance and behavior.

To make the component do something useful, like display dynamic information or control a device register, you configure property bindings for the component. To make the component react to user interaction, you configure event handlers for it.

It is primarily through property bindings that you bring windows to life, and have them do useful things. A property binding simply links one component's property to another on the same window.

The power of property 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 (`glyphForeground`) color to a tag's `AlertCurrentSeverity` 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 quite literally, endless.

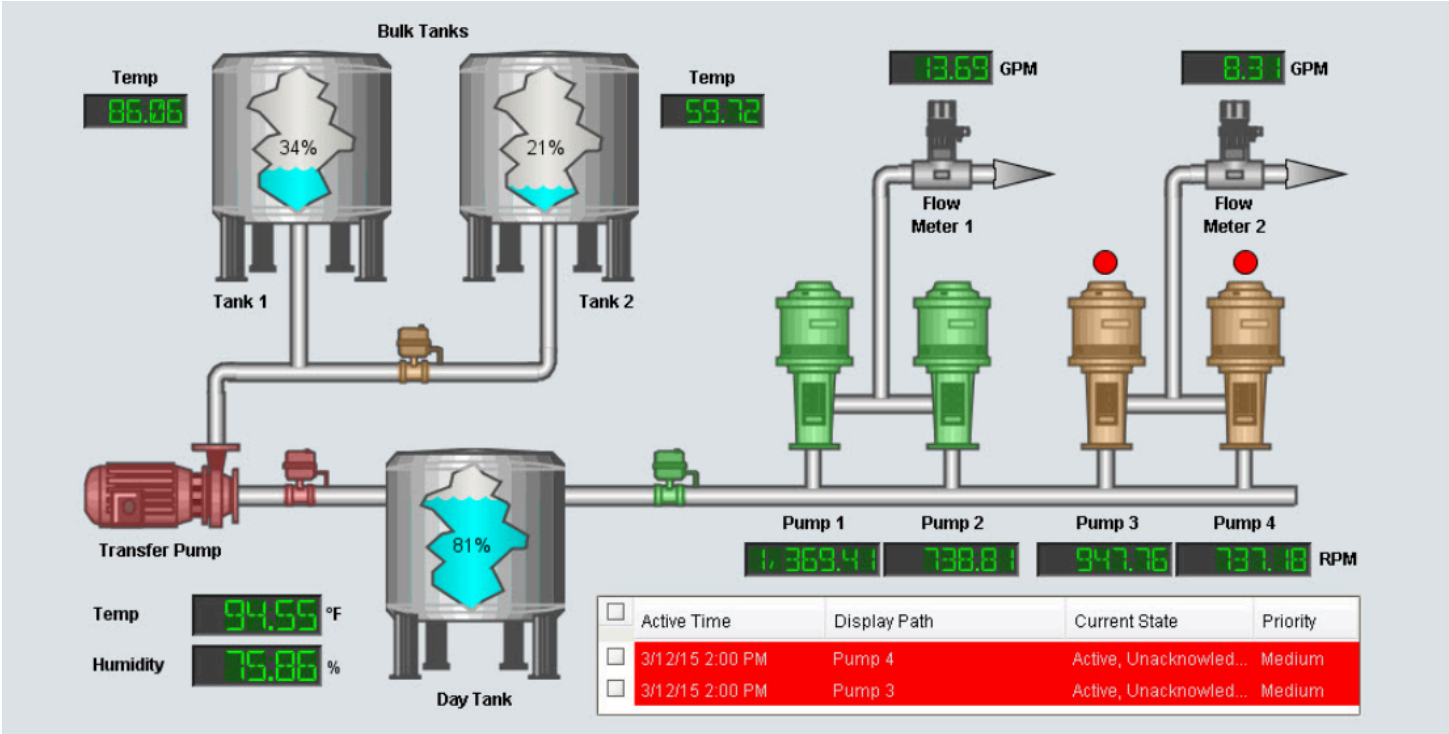
Switching Between Projects

When you launch an Ignition client, it opens a single project to display. If you want to open multiple projects you can have multiple clients open on the same computer, or you can use Ignition's Retargeting system to make seamless transitions between projects. The Retargeting feature allows you to jump from one project to another without closing the client. This allows your users to jump from area to area while still allowing you to keep your designs compartmentalized in multiple projects.

Your projects may all use different authentication sources or role sets for [security](#). This means as your users jump from project to project, they may be able to use some or all of the controls on one project, but only see what is happening on another project.

Project Examples

Ignition is such a diverse application that it can work in any field you can imagine and more. Just a few examples: Oil and gas, food and beverage, automotive, medical, air traffic control, water - waste water, add many more. You can get anything you want set up in Ignition. To see a few live example screens, check out our [public demo project](#). It's packed with samples of the features and functions you might want to use from reporting, to history, to HMI optimization.



Project Workflow

Getting up and running quickly with your project is simple:

1. Open the Designer.
2. Start Designing!

The only real problem is figuring out exactly what you want to make with your system. If you're new to Ignition, we have a basic workflow with a few common tasks for you. See our [Quick Start Guide](#) for simple step-by-step instructions of a sample workflow.

Quick Start Guide

What is NOT in a project?

The designer allows you to create and modify several types of resources that are shared by ALL projects. Depending on the resource, this means that either they run independently of the projects, or they are available for use by any project. Here are a few of the

- **Alarm Pipelines:** Control alarm notification (runs independently)
- **Sequential Function Charts (SFC):** Logic to step through a process (runs independently)
- **Shared Scripts:** Blocks of scripting code (available to all projects)
- **Tags:** Basic or UDT Tags provide realtime data (available to all projects)
- **Alarming:** Alarms exist on Tags, and so are not in a project (available to all projects)

In addition to these resources, the Gateway connections and settings are available to all projects and are set up in the [Gateway Webpage](#). These include resources such as [database connections](#), [OPC server and device connections](#), and more.

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

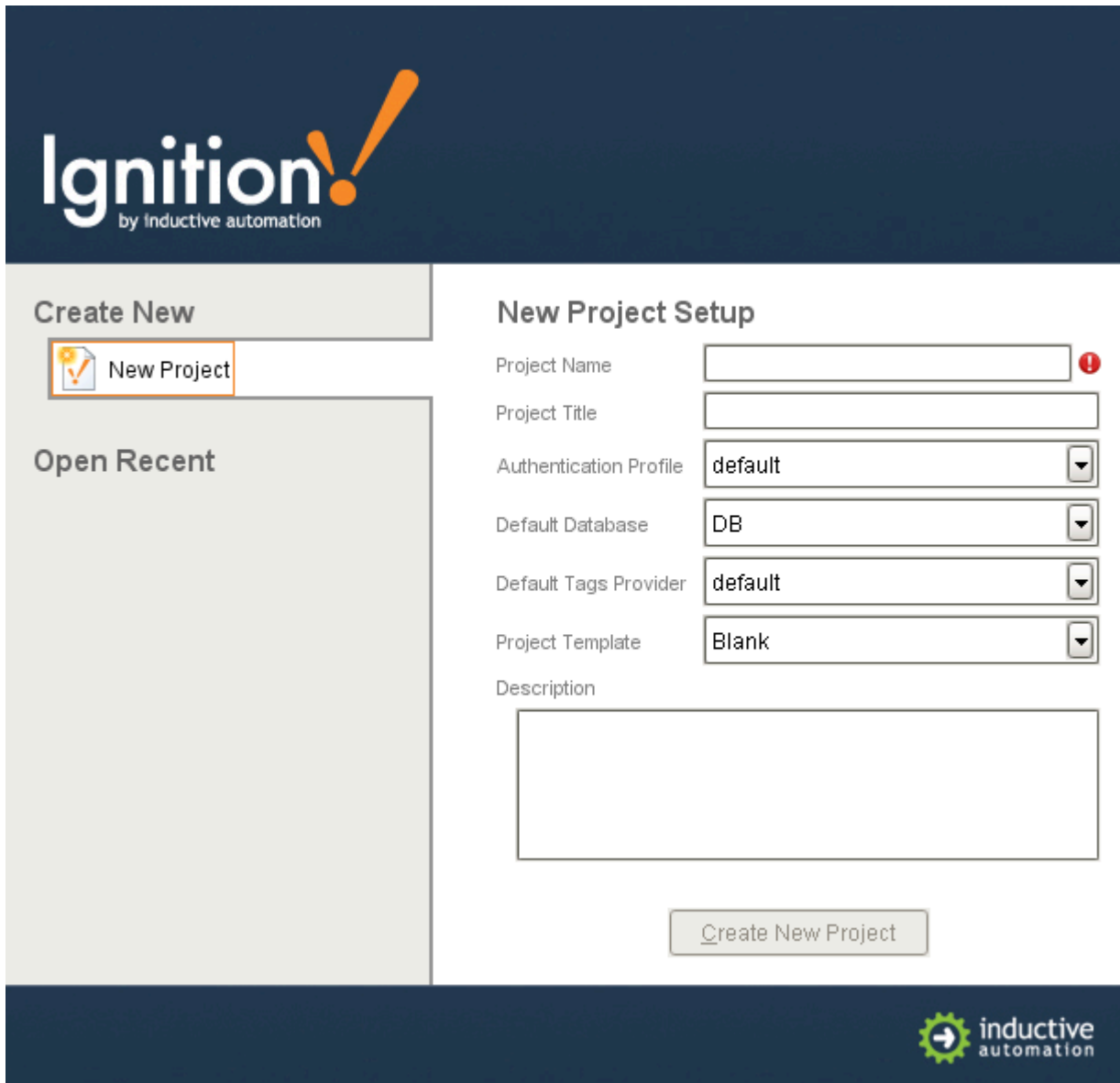
For documentation on the latest stable release, see the [latest version](#) (8.3).

Add/Edit/Delete Projects

Creating a Project

The first step in working with a project is creating one! When you [launch the Designer](#), the **Open/Create Project** window is displayed. You will have the option to create a new project or open an existing project.

For a quick walk-through to create a new project and start developing, see the [Quick Start Guide](#).



The screenshot shows the 'New Project Setup' window in Ignition. The window has a dark blue header with the 'Ignition! by Inductive automation' logo. On the left, there is a sidebar with 'Create New' and 'Open Recent' sections. The 'Create New' section has a 'New Project' button with a folder icon. The 'Open Recent' section is currently empty. The main area is titled 'New Project Setup' and contains several fields: 'Project Name' (text input with a red warning icon), 'Project Title' (text input), 'Authentication Profile' (dropdown menu with 'default' selected), 'Default Database' (dropdown menu with 'DB' selected), 'Default Tags Provider' (dropdown menu with 'default' selected), 'Project Template' (dropdown menu with 'Blank' selected), and 'Description' (text area). At the bottom right, there is a 'Create New Project' button. The footer of the window features the 'inductive automation' logo.

To create a new project, click on **New Project** and define following information:

- **Project Name:** A unique name for the project in the system. The name can only consist of alphanumeric characters and the '_' character. Spaces and other special characters are not supported.

NOTE

It is not advisable to change the name after it's been created. Instead, change the **Project Title** if you want to change how the project appears later.

- **Project Title:** This is the name that will be displayed on the launch page of the Gateway and in the runtime Client. There are no restrictions to special characters or spaces.
- **Authentication Profile:** The profile to use for granting access to the project. For more information or to add users, see [Security](#).
- **Default Database:** Any queries to the database will use this default [database connection](#) unless explicitly specified otherwise.
- **Default Tags Provider:** The primary [Tag provider](#) for the project. Most installations will likely only have one provider, but in situations where there are more than one, this is the provider that will be used unless explicitly specified otherwise.
- **Project Template:** Gives you the option to choose a quick start template for your project. Most people start with a simple 1-tier navigation option. Click the dropdown to see all the available options.

Managing Projects

Once you have a project or two (or twenty) setup, you might want to change some of the base settings. You can manage the projects from the **Configure** section of Gateway by going to **System > Projects**. The **Project** window is displayed showing all your projects. To the right of the project name, click the **More** button, and here you can **edit**, **delete**, view **details**, **copy**, or **export** your project.

NOTE

You can manage many of these things in the Designer, and much more (such as the Client Poll Rate, Publish Mode, etc.,). See the [Project Properties page](#) for more information.

Controller

USER MANUAL SUPPORT

admin Sign Out

Launch Designer

Ignition
by Inductive automation

HOME STATUS CONFIGURE

Search...

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

SYSTEM

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

NETWORKING

- Gateway Network
- Email Settings

SECURITY

- Auditing
- Users, Roles

Projects

✓ Successfully updated Project "Navigation"

Name	Description	Authentication	Default Database	Default Tag Provider
Development		default	MySQL	delete More ▾ edit
DrillDown		default	MySQL	details More ▾ edit
Maria		default	MySQL	copy export More ▾ edit

Viewing the Details of a Project

To view the project details, go to the Configure section of the Gateway webpage, and select System > Projects. Find your project, press the More button to the right of the project name, and select details. The details link on the Project window, takes you to the Project details for `<Project Name>` page where you can choose to view the Published or Staging version of the project, and view the project's edit history.

i NOTE

Have you ever seen `<object name>` with those weird greater than/less than signs before? We do this a lot in the user manual because Ignition will often use your specific names. Don't worry about it, this just means you should expect to see whatever your name is in that spot. IE: If your project name is "Project 1", then you will see Project details for 'Project 1'.

Editing a Project

To edit some of the project settings, go to the **Configure** section of Gateway, and then to **System > Projects**. You will see a list of all your projects. Click **edit** to the right of project name in the list. The **Edit Project** page is displayed. You can now change some of the project settings such as project name, description, and title. You can also change connections to the Authentication profile, default database, and default tag provider.

Many of these settings can be modified in the Designer too, but it is important to note that you cannot change any of these project settings from the Gateway if the project is currently open in a Designer.

Deleting a Project

Be aware that once you delete a project, it cannot be undone, a deleted project is gone forever (unless it can be recovered from a [backup](#) or [auto-backup](#)). Always make a project export or Gateway Backup before deleting a project.

To delete a project, go to the **Configure** section of Gateway, and then to **System > Projects**. You will now see a list of all your projects. To the right of the project name in the list, look for **delete** and click on it. This deletes your project.

Copying a Project

Copying your project is useful when you need a *snapshot* of a project before starting major changes, or for creating a starting point for a new project based on an old one.

To copy a project, go to the **Configure** section of Gateway, and then to **System > Projects**. You will now see a list of all your projects. To the right of the project name in the list, click **copy**. This creates a copy of your project.

Project Settings

Published vs Staging

Each project can have two distinct versions at the same time: the [Staging version](#) and the [Published version](#). Having a staging version allows you to make changes to a project, save the changes, and test out the changes before "publishing" it to a production environment.

Normal Vision module Clients run the published version of a project. However, by launching a Client in a special mode (from the Designer or from the Configure section of the Gateway), you can launch a Client that runs the **staging version** of that project. This staging Client will receive updates on every save, where the production Clients receive updates only on publish. This lets you test out your changes to the project in an actual Client, which is more realistic than the Designer's preview mode.

Not all aspects of a project use this system. It is primarily intended for systems such as the Vision module's Clients. Features that run persistently on the Gateway such as Tags, the SQL Bridge's Transaction Groups, and Gateway-side scripting always run the most recently saved changes (the Staging version). Since these features by definition must run in exactly one place, they cannot be effectively "tested out" by simultaneously running a staging version alongside a published version.

In the Designer, choose the **Project > Properties** menu item, then go to the **Project > General** page to set the **Publish Mode**. By default, a new project is configured to be in **Auto** publish mode, however

you can change it to **Manual** publish mode.

- In the default **Auto** mode, when you want to save your project, Designer lets you do a **File > Save and Publish**, which means both the Staging and the Published versions will be the same and identical.
- In **Manual** mode, when you want to save your project, Designer lets you do both a **File > Save**, or a **File > Publish**. **File > Save** updates the Staging version, and **File > Publish** updates the Published version.

The screenshot shows the 'Project // General' settings dialog. On the left is a tree view with categories: Project (General, Permissions), Designer (General, Window Editing), Client (General, Launching, Login, Timing, User Interface), and Mobile (General). The 'General' option under 'Project' is selected. The main area is titled 'Project // General' and contains several sections of settings:

- Tag Settings:** Default Provider (dropdown: default), Client Poll Rate (spin box: 250).
- Database Settings:** Default Database (dropdown: MySQL).
- Security Settings:** User Source (dropdown: default), Required Client Roles (text box: Operator, with a green checkmark icon).
- Auditing Settings:** ☒ Enable Auditing, Audit Profile (dropdown: Audit).
- Publishing Settings:** Publish Mode (dropdown: Manual, with a list showing Manual, Auto, Manual), Commit Messages (text box).

At the bottom are three buttons: OK, Apply, and Cancel.

Project Versioning and History

Each project keeps a log of recent changes. These include both saves and publishes. Every save increments a number called the "edit count" for the project, which can be used like a serial number. The user, time, affected resources, and a commit messages (described next) are logged as well. To learn more about versioning in Ignition, refer to the section on [Versioning](#).

Commit Messages

A project may be configured to prompt the user making changes to describe those changes, either on every publish event, or on every save and publish event. These messages, called commit messages, are used to describe the changes that have been made to the project. By inspecting the project's

history and reading these commit messages, you can learn what changes have been made to the project, for what reason, and by whom.

Concurrent Project Development

Often in large facilities there are multiple people working on the same project. Multiple Ignition Designers can be launched concurrently, and everyone can work on the same project at the same time. Because everything is controlled in the Gateway, there is no merging or syncing required when you save your changes. When a particular resource (window, template, etc) is being edited, it will be locked, and the other designers won't be able to modify the locked resources. When the resource is closed and saved, it's automatically unlocked. Concurrent edits are pushed to other designers only when you update your project. You can do this by going to the menubar in the Designer and selecting **File > Update Project**.

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

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

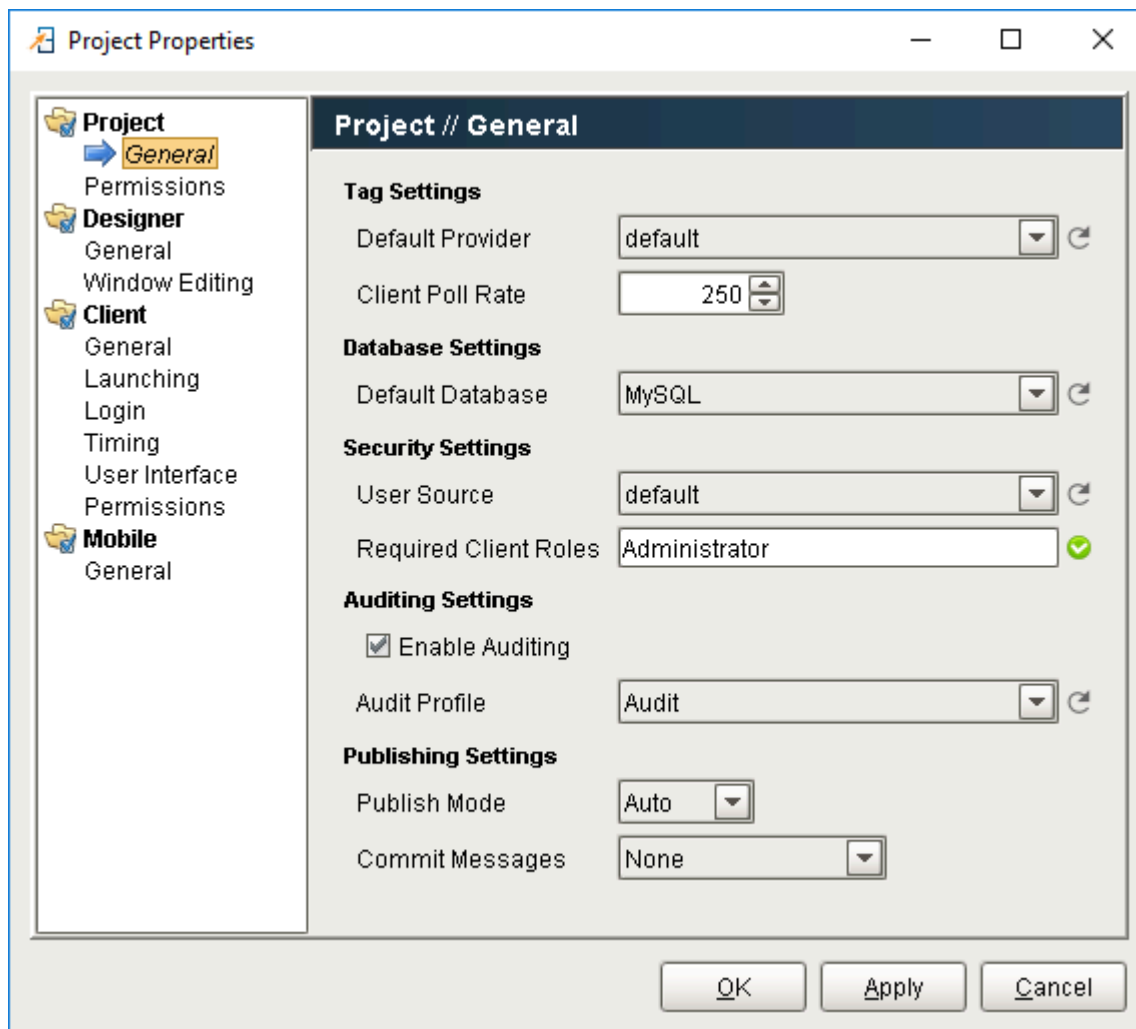
Project Properties

There are a number of properties you can set for your projects within the Designer, Vision Clients, and Mobile devices. For example, there are properties for setting the touchscreen mode, customizing a client's auto-login, or configuring how the clients receive updates, and many more.

The property settings on the **Project Property** window apply to the whole project. This page identifies and describes all the available project properties.

Accessing Project Properties

In the Designer, either double-click on the **Project > Properties** node in **Project Browser**, or click on **Project > Properties** on the menu bar at the top of the Designer. Project properties span several functional areas each containing settings applicable to that area.



Project Properties



INDUCTIVE
UNIVERSITY

About Project Properties

[Watch the video](#)

A project's General properties apply to the project as a whole, across all module functionality. For a new project, there are many default settings that you can use (see next). For example, there are default settings for the Tag provider, database, publish mode, initial comm mode, window editing, and Client launching.

Note: There are a few properties of a project, such as its name, description, and title that are not available from the Designer and you need to set them in the Gateway. To do so, go to the [Gateway Webpage](#), under the **Configuration > Projects** section, click on the **edit** link next to the project. You cannot edit these settings while the project is open in the Designer.

Under **Project > General** is where you set the project's **Default Database** and its **Default Tag Provider**. It is important to understand how to use defaults effectively for proper project design.

Wherever you use a database connection or a Tag in a project, you are always given the option to use the project's default, or an explicitly named connection or provider. If your project is like most typical projects, it primarily uses a single database and a single Tag provider. By consistently using the **default** option, you make your project more resilient to change.

For example, suppose you have a project, and it has a database connection named `Production_DB`. Now you want to adapt the project to a new, similar plant, while leaving the existing project intact. You copy the project and create a new database connection, named `New_DB`. If your project consistently used its default database connection, the switchover will be as simple as changing the copied project's default database. However, if you used the explicit `Production_DB` connection in your groups and screens, you will need to laboriously switch the bindings over to `New_DB`.

Project - General Properties

Tag Settings

Property	Description
Default Provider	The Tag provider chosen here will act as the project's default tag provider. To use the default provider, simply omit the source section of a Tag path, or leave it blank, for example: <code>Path/To/MyTag</code> or <code>[]Path/To/MyTag</code> . See also: Keeping Tags Organized
Client Poll Rate	The Client poll rate (in milliseconds) is the rate at which a Vision Client or Ignition Designer polls the Gateway for updates to its subscribed Tags.

Database Settings

Property	Description
Default Database	This is the default database connection to use for this project. To use the default database connection, use the special <code><default></code> connection, or in scripting, the empty-string connection <code>""</code> .

Security Settings

Property	Description
User Source	Choose the User Source that governs this project's security: specifically, which group of users (User Sources) is allowed to log into the client. The User Sources are all defined in Gateway's Configure section under the Security > User, Roles page.
Required Client Roles	This property is for Client logins, and determines what role(s) a user must have before they can log into the Client. You can optionally specify a list of roles that are required for a user to log into this project. Use commas to separate the roles. Users must have at least one of the roles in order to log in. If no roles are specified, the user only needs to correctly authenticate with the User Source in order to log in. To define the roles, go to the Gateway's Configure section under the Security > User, Roles page, click the manage users link, and then go to the Roles tab. See also: Security

Audit Settings

Property	Description
Enable Auditing	If auditing is enabled, audit events that relate to this project will be stored in the chosen audit profile.
Audit Profile	The audit profile stores the audit events when auditing is enabled.

Publishing Settings

Property	Description
Publish Mode	Here you specify how project publishing is handled. By default, the project is set to Auto, which means both the staging and published version of the project will always be synchronized. • Manual: In this mode, you can do a File > Save to update the Staging version, and then a File > Publish to update the Published version. • Auto: In this mode, the only command available for saving your project is File > Save and Publish which saves and then publishes, so the two Staging and Published versions are always the same. See also: Published vs Staging.
Commit Messages	You can specify whether or not commit messages are required, and if so, under what conditions. You can specify the condition as: None, OnPublish, or OnSaveandPublish.

Project - Permissions Properties

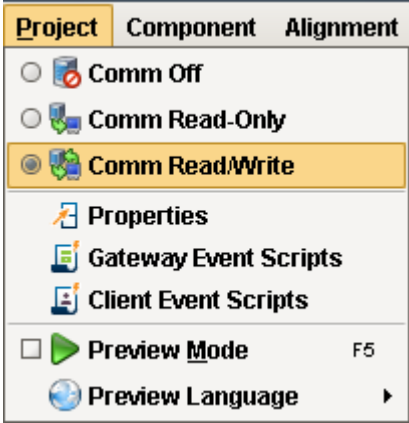
Property	Required Designer Roles Description
Permissions	When opening the project in the Designer, these properties determine which roles are required when making certain changes to the project. • Publish - Users must have at least one of these roles to publish the project. • View - Users must have at least one of these roles to view the project in the Designer. • Save - Users must have at least one of these roles to save the project. • Delete - Users must have at least one of these roles to delete the project. • Protected Resources - Users must have at least one of these roles to access protected resources. More details can be found on the Project Security in Designer and Gateway page.

Designer Properties

The designer property settings are saved on a per-project basis.

Designer - General

Startup Options

Property	Description
Initial Gateway Comm Mode	The Designer starts up in the default Comm Read-Only mode. The property allows you to change the mode the Designer starts in when viewing the project. The options are: Comm Off, Comm Ready-Only, Comm Read/Write. • Comm Off - In this mode, all database query traffic and Tag subscriptions and writes are blocked. • Comm Read-Only - Tag subscriptions and SELECT queries work, but Tag writes and UPDATE/INSERT/DELETE queries are blocked. • Comm Read/Write - The Designer may freely request tag and database values from the Gateway, as well as write or change these values. For more information, see the Communication Modes page. 

Designer - Window Editing Properties

These options affect the operation of the Designer as it applies to the Vision module's window design.

Window & Template Committing

Property	Description
Commit on Close	• Prompt - By default, every time you close a window, you are prompted whether or not you wish to commit the window: i.e., maintain any changes that have been made in the Designer session. • Always - This option allows you to skip the commit prompt, opting to always commit the window on close.
Template Auto-Commit	• On - Automatically commits template definitions when switching to another window or template. • Off - In this option you commit the templates manually, it reverts all changes to the last time the window was committed.

Component Manipulation - Affects how the user interface acts in manipulating components

Property	Description
Constrain to Parent Container Bounds	Disabling the constraint on parent bounds allows you to position components outside of their parents bounds, which can be helpful in advanced layouts.
Nudge Distance	The number in this box is the distance (in pixels) that a nudge moves (when using the arrow keys) or resizes a component.
Alt-Nudge Distance	The number in this box is the distance (in pixels) that an alt-nudge moves (when using the arrow keys plus the Alt key) or resizes a component.

Default Component Layout - Used for all newly added components

Property	Description
Layout Mode	• Relative - All newly created components will be configured with a Relative layout, further configured by the Relative Layout Options property. • Anchored - All newly created components will be configured with an Anchored layout, further configured by the Anchored Layout Options property.
Relative Layout Options	When the Layout Mode property is set to Relative , these options determine the layout options of new components. For more information, refer to Component Layout .
Anchored Layout Options	When the Layout Mode property is set to Anchored , these options determine the layout options of new components. For more information, refer to Component Layout .
Default Color Mapping	The initial color mapping when configuring a new number-to-color binding.

Client Properties

These properties apply to the Vision Client in general.

Client - General

Property	Description
Timezone Behavior	The Vision Client can emulate any timezone. By default, it will appear to be in the same timezone as the Gateway Timezone . This has the effect of all Clients behaving the same, regardless of the timezone setting on the Client's host operating system. Depending on your project's requirements, this may not be optimal. You can have the Client use the host's timezone by choosing the Client Timezone option, or you may specify any explicit timezone for all Clients to emulate.

Update Mode - Affects how Clients receive updates when the project is saved/published

Property	Description
Notify	This is the default option, which means that a notification bar will appear at the top of the Client when an update is available. The update gets applied when the user clicks on the notification bar.
Push	This option automatically pushes all project changes and updates to all running Clients with no user interaction. This is often desirable when a Client is running in a situation where mouse access is inconvenient, such as a large overhead display.

Touch Screen

Property	Description
Touch Screen Mode Enabled	All Clients can operate in touch-screen mode. When enabled, clicking on editable numeric and text entry fields (i.e., Text Fields, Numeric Text Fields, etc) will pop up on-screen keyboards that can be used for data entry. You can optionally set the width of any scrollbars (number of pixels wide/tall).
Touch Screen Mode Active on Startup	Configures the Clients to start up with the touch-screen mode active. More details can be found on the Using Touch Screen Mode page.

Data

Property	Description
Disable Tag History Data Cache	The Clients normally maintain a cache of data retrieved from Tags History, improving repeat operations on graphs and tables. When this option is disabled, no data is cached, and the full queries execute against the Gateway each time data is required.

Client - Launching

These properties apply to the Vision Client launch process.

Gateway Launch Page

Property	Description
Default Launch Mode	Determines the mode on Clients launched from the Launch button that appears next to the project in the Gateway Home page. Available modes are Windowed or Full Screen.
Windowed Button/Full Screen Button	Each launch mode can also be enabled individually, which allows that mode to appear in the Dropdown list next to the Launch button on the Gateway Home page.

Launch Icon

Property	Description
Launch Icon	The image specified here is used to represent the project on the launch page and desktop shortcut. This needs to be a path to an image that has been uploaded to the Gateway . Use the browse button to choose or upload a new image.

Windowed Properties

Property	Description
Vendor	This property is displayed as the project is launching through Java Web Start, as well as in the JWS application manager.
Homepage	A URL that is displayed in the JWS application manager.
Width	The width of the Client window when launched in Windowed mode.
Height	The height of the Client window when launched in Windowed mode.
Screen Index	The number here is the index of the screen to use for Full Screen mode, it starts at 0. This property is not supported on all operating systems.
Start Maximized	If the box is selected, when the Client is launched in Windowed mode, it starts maximized. Note that this is not the same thing as Full Screen mode, which is only available when the Client is launched in Full Screen mode. In Full Screen mode, the width, height, and start maximized properties have no effect. When launched in Full Screen mode, the user is given an Exit button on the login screen by default. For terminals where the application should not be exited, this button can be removed by checking the Hide Exit Button box.
Hide Exit Button	If the box is selected, when the Client is launched in Full Screen mode, the exit button is hidden to prevent the application from closing.

Client Memory - Governs how the Client use RAM resources on its host machine

Property	Description
Initial	The initial memory setting is how much memory the Client will require on startup. While this is typically left alone, boosting it a bit can improve performance somewhat.
Maximum	The maximum memory setting sets a cap on how much memory the Java VM is allowed to use. When you launch a Client on a machine with plenty of RAM, you'll also need to boost this setting to allow the Client to use more RAM.

Client - Login



INDUCTIVE
UNIVERSITY

Customizing Client Login
Screen

[Watch the video](#)

These properties affect how the Vision Client's login process behaves and appears.

Login Screen

Property	Description
Welcome Message	The message that appears in the upper-left corner of the Login screen. If left blank, no message is displayed.
Welcome Image	The image that appears in the upper-left corner of the Login screen. If left blank, uses a default image.
Username Text	The text that appears next to the username field.
Password Text	The text that appears next to the password field.
Login Button Text	The text that appears on the Login button.
Show Locale Selector	Determines if the Locale Selector should appear on the Login screen. This property interacts with Ignition's Localization system. • Automatic - Show the selector if more than a single Languages exists in the project. • Show - Always show the selector, regardless of how many Languages exist. • Hide - Never show the selector.

Auto Login

Property	Description
Enable Auto-Login	By enabling auto-login, you can have the launched Client skip the login process. The Client will log in behind the scenes using the credentials supplied here. If they fail, the login screen will be presented. See also: Setting up Auto Login

SSO Login

Property	Description
Enable SSO Login	Enable Single Sign-On for the project. The Project's default Authentication Profile must use Active Directory, and SSO must be enabled in the Profile. See Active Directory Authentication for more details.

Client - Timing



INDUCTIVE
UNIVERSITY

Setting Project Polling Base
Rate

[Watch the video](#)

Property	Description
Polling Base Rate	The base rate, in milliseconds, for all polling bindings.
Connect Timeout	The maximum amount of time to wait for connections to the Gateway to be established. Specified in milliseconds.
Read Timeout	The maximum amount of time for socket connection to the Gateway to remain open. Specified in milliseconds.
Connection Concurrency	By default, Clients are not limited by the number of concurrent connections to the Gateway. These connections are used to send Tag writes, return database results, as well as any other action that requires information to be passed between the Gateway and the Client. Depending on what is running in the Client, your network's bandwidth could be hindered. Enabling this property will limit the amount of concurrent connections the Client can maintain. Note that this may negatively impact Client performance, but is usually preferable on busy networks.

Client - User Interface



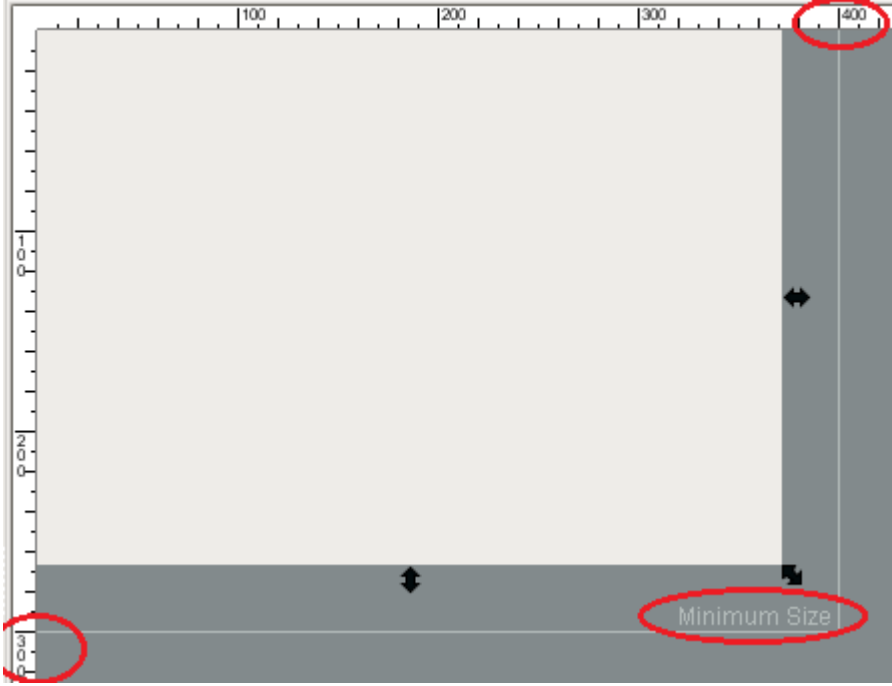
INDUCTIVE
UNIVERSITY

Setting Client Minimum Size

[Watch the video](#)

These properties affect how the Vision Client appears and behaves while it is running.

Minimum Size

Property	Description
Width and Height	Typically, a Vision Client is designed to run on multiple different resolution and sizes of monitors. The various component layout features help design elastic screens, but sometimes you need to set a lower bound as to how small you'll allow the Client's usable area to shrink. This is what the Minimum Size settings are for. You can see these settings visually represented in the Designer as lines on the Vision workspace when the Root Container is smaller than the configured Minimum Size. Whenever the usable space shrinks smaller than these bounds, scrollbars will appear, capping the width and height to these minimums. This defaults to 800x600. In the image below, the project was set to a minimum size of 400 x 300. Since the window is smaller, the outline is visible. 
Client Background Color	This option allows you to specify the color of the Vision workspace which will be visible when not obscured by windows.

Client Menu - Options to alter the appearance, or remove completely, the menu bar that appears in a running Vision Client



INDUCTIVE
UNIVERSITY

Hiding Client Menu Bar

[Watch the video](#)

Property	Description
Font Button	Change the font of the Menu Bar.
Hide Menu Bar	Hide the entire Menu Bar in the Client. Usually enabled in situations where users should not be able to close the client.
Hide Windows Menu	Hide just the Windows menu option. Enabled when users should not be able to close windows in the Client. See also: Menubar .
Touch Screen Keyboard Width	Determines how wide the Touch Screen Keyboard should appear in the Client.

Docking

Property	Description
Axis Precedence	When windows are docked on adjacent sides, this property determines which sides should take precedence. When set to "East/West", windows docked to the East or West sides will expand vertically from the top to the bottom of the Client, and will push any North or South docked windows out of the way.
Prevent Popup/Docking Overlap	By default, popup windows are not allowed to overlap any docked window. Disabling this property will allow Popup windows to be placed on top of docked windows in the client.
Infinite Desktop	By default, the desktop area will be expanded if floating windows are dragged out of frames. If false, popups are prevented from being dragged beyond the bounds of a window so they don't get distorted.

Client - Permissions

New in 7.9.4



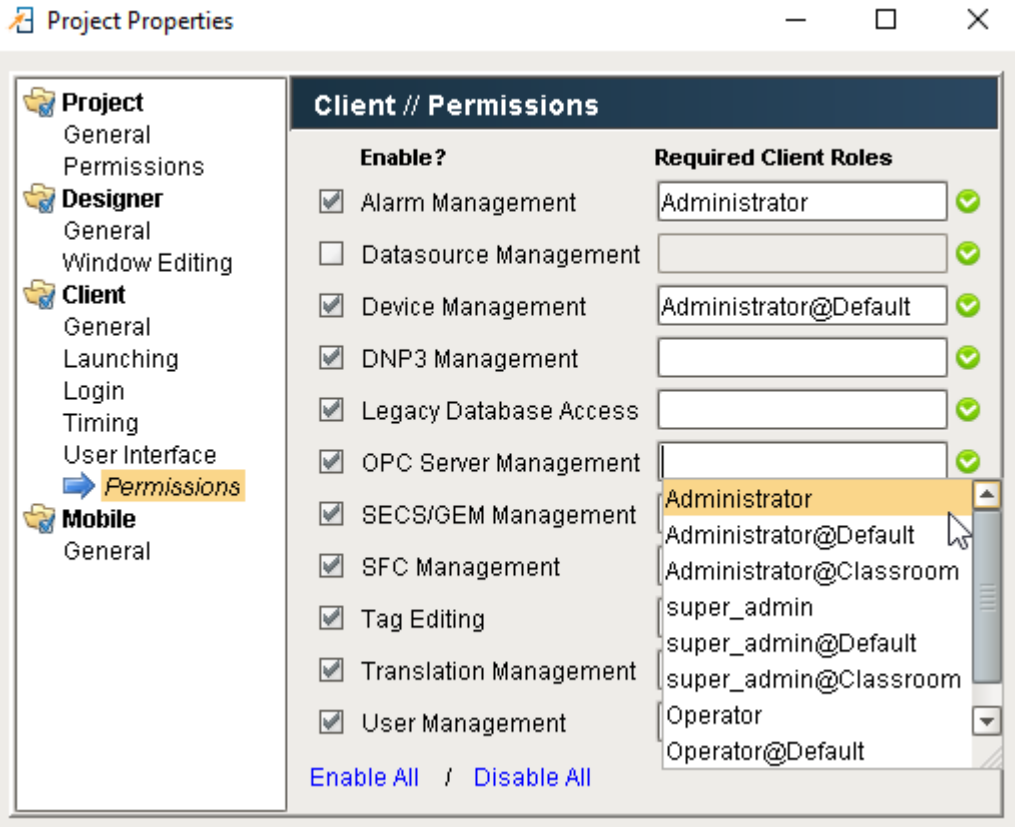
INDUCTIVE
UNIVERSITY

Client Permissions

[Watch the video](#)

These properties allow you to limit the Client's ability to perform certain tasks. The tasks are grouped by category. Access can be configured statically for all users, or require specific roles.

Property	Description
Enable?	Determines if the Client has access to this category. If unselected, the category will be disabled in the Client for all users. If selected, the Required Client Roles text field will also be used to determine if the Client has access.
Required Client Roles	A comma separated list of Managing Users and Roles and/or Security Zones that have access to the category while the Enabled checkbox is selected. While these fields have focus, the down arrow key on the keyboard may be pressed to make a list of available roles appear. There are several ways to use roles and zones in this field, and they can be used together in the comma separated list. • Blank Field - anyone can use this, regardless of their roles. • Role name - users with this role can use this regardless of their zone. • roleName@zoneName - users must have this role AND be logged in from this zone. Note that when you create a new project, all of these settings will be disabled by default.
Category Descriptions	The following is a list of the initial categories. Note that the categories you see in your Designer are dependent on which modules are installed on the Gateway. Additionally, third-party modules can add to this list. • Alarm Management - Allows the Client to cancel, shelve, and acknowledge alarms. Applies to both local and remote Alarms. • Datasource Management - Allows the Client to modify datasources. • Device Management - Allows the Client to modify devices. • DNP3 Management - Allows the Client to freeze DNP3 operations via scripting. • EAM Task Execution - New in 7.9.12 Allows the Client to execute system.eam functions. • Legacy Database Access - Allows Client database access outside of named queries ad Transaction Groups. • OPC Server Management - Allows the Client to modify OPC server connections. • SECS/GEM Management - Allows the Client to disable equipment or delete programs via SECS/GEM scripting.

Property	Description
	• SFC Management - Allows the Client to start or stop Sequential Function Charts. • Tag Editing - Allows the Client to add, edit, or delete Tags through scripting. • Translation Management - Allows the Client to modify translations in the localization system. • User Management - Allows the Client to modify schedules, holidays, and users through scripting or components. The image below contains the following configuration: • Alarm Management is only available if the logged in user has the "Administrator" role (regardless of their current zone). • Datasource Management has been disabled for all users. • Device Management is available if the logged in user has the "Administrator" role and the Client is in the Security Zone named "Default".  The screenshot shows the 'Project Properties' dialog box with the 'Client // Permissions' tab selected. The left sidebar shows a tree view with 'Project', 'Designer', 'Client', and 'Mobile'. The 'Client' node is expanded, and the 'Permissions' sub-item is selected. The main area shows a table of permissions with columns 'Enable?' and 'Required Client Roles'. The 'Required Client Roles' column has a dropdown menu open, showing a list of roles including 'Administrator', 'Administrator@Default', 'Administrator@Classroom', 'super_admin', 'super_admin@Default', 'super_admin@Classroom', 'Operator', and 'Operator@Default'. The 'Administrator' role is currently selected in the dropdown.

Mobile Properties

These properties apply to Mobile Clients.

Mobile - General

Property	Description
Enable Mobile Launch	Enables the project to be launched as a mobile project.
Hide from Launch Page	Hide this project from the mobile launch page.
Enable Geolocation	Enables geolocation which will ask the device for latitude and longitude.
Image Encoding	The image encoding preference for mobile clients. PNGs offer the best quality, while JPEGs offer a trade-off between quality and bandwidth.

Viewport - Defines how the mobile project should fit to the device

Property	Description
Fit to Device	Fits the project to the available size of the mobile browser.
Custom	Sizes the project to explicit dimensions, allows the device to pan and zoom.

Utility Bar - Determines if the Utility Bar should be present on mobile clients.

Property	Description
Auto	Automatically places the utility bar along the bottom or side depending on device orientation.
Hidden	Hides the utility bar entirely.

Auto Login

Property	Description
Enable Auto Login	Automatically logs into all mobile versions of the project. To complete the setup, enter the following: • Username • Password • Password (repeat) See Setting up Auto Login for more details.

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

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

Client Update Modes



INDUCTIVE
UNIVERSITY

About Client Updates

[Watch the video](#)

Notify Versus Push

When a Client is launched, the most recent version of the project is used. If you make changes to the project and save the changes (by doing a **File > Save**) while a Client is open, the Client is notified about the updates. The Clients can receive the updates in two different mode: **Notify** or **Publish** mode.

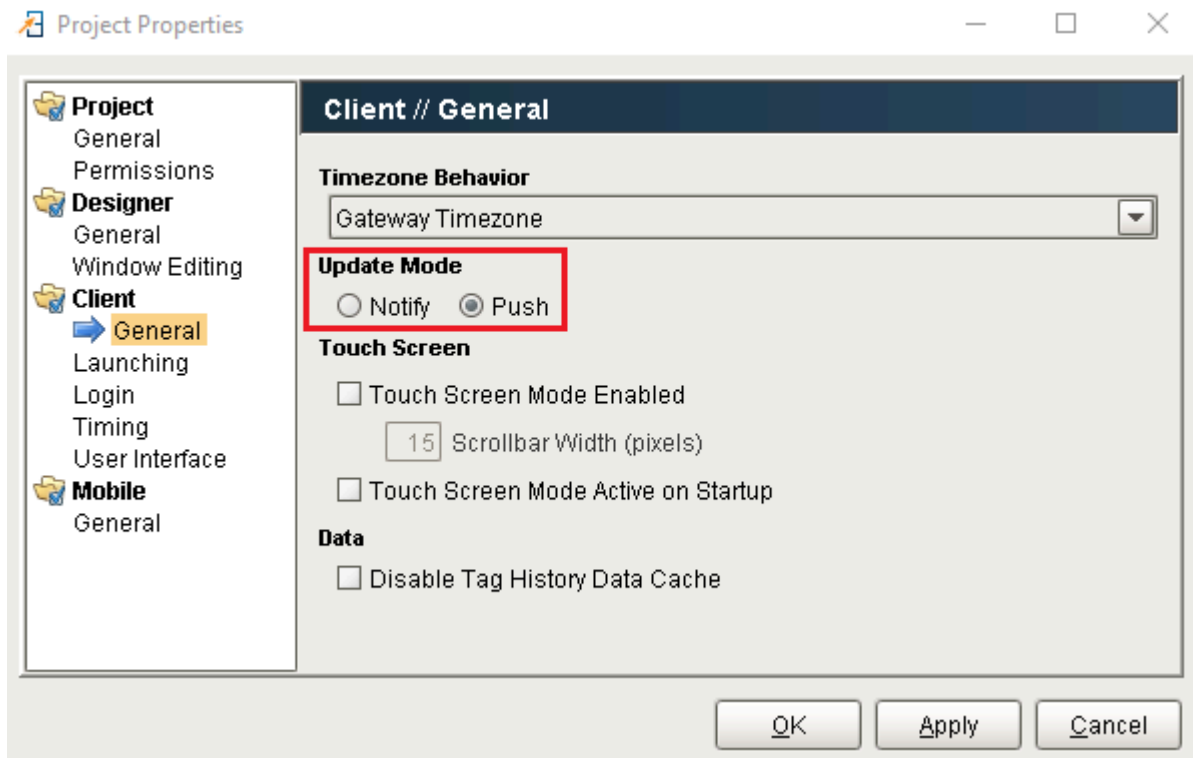
- **Notify** - This is the default mode to automatically **notify** operators when project updates are available. In this mode, every time you update a project in the Designer and save the changes, Clients will display a yellow information bar at the top of their display. This yellow bar notifies the operator that an update is available. By clicking on the notification banner, the new project modifications are downloaded and applied.
- **Push** - When you save your changes in the Designer, this mode automatically pushes all project changes and updates to all running clients with no operator interaction, that is, the new version is downloaded and applied automatically. This is often desirable when a Client is running in a situation where keyboard and mouse access is inconvenient, such as in a large overhead display.

NOTE

This property (and many other project properties) are only read when a client is launched. Changing to the Push mode will not get pushed when you save, your clients need to be re-launched to start using the Push Update Mode.

To Set the Client Update Modes

1. In the Designer, from Project Browser, double-click on **Project > Properties**, or from the top menus go to **Project > Properties**. The **Project Properties** window is displayed.
2. Go to **Client > General** page, set either **Notify** or **Push** as your client update mode.



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

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

Using Touch Screen Mode



INDUCTIVE
UNIVERSITY

Using Touchscreen Mode

[Watch the video](#)

It is very common to deploy Ignition Vision projects on touchscreen computers, such as industrial panel-PCs acting as Human Machine Interface (HMI) or Operator Interface Terminal (OIT). In situations where the PC does not have a keyboard attached, Touch Screen Mode can be used to assist with user-input. For this reason, all of the input components in Vision are touch-screen-enabled.

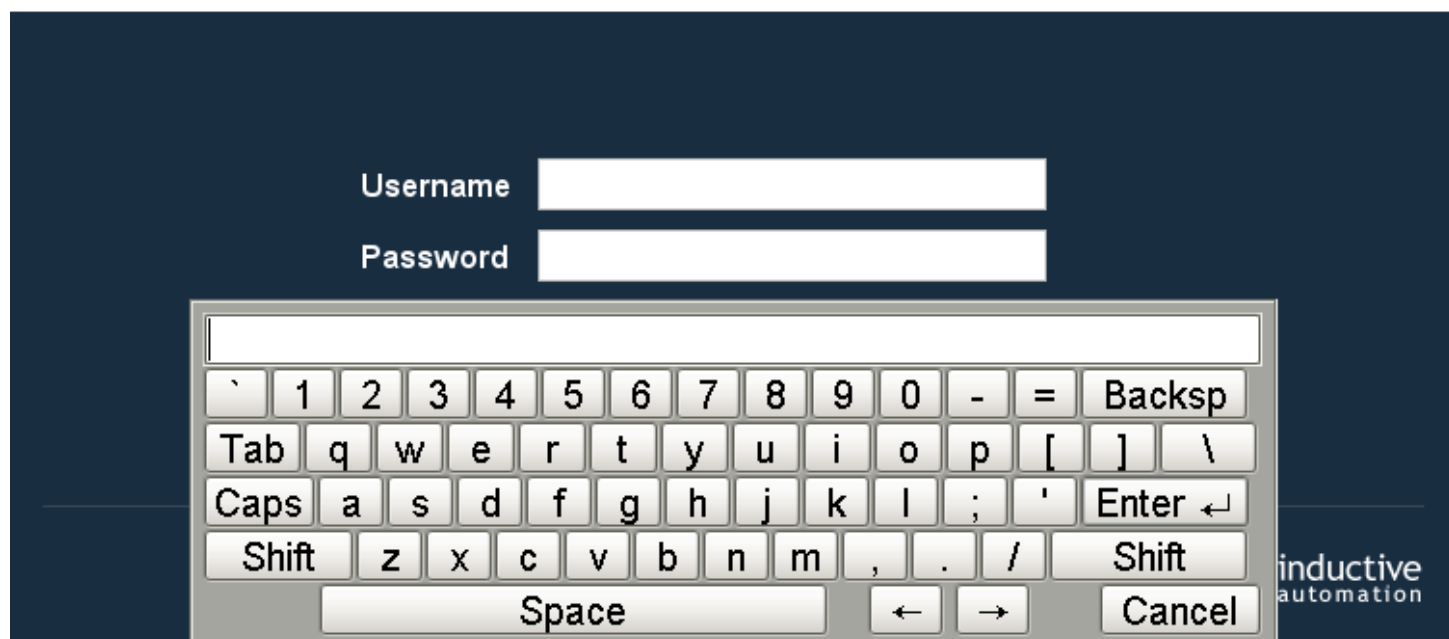
Under normal circumstances, you don't have to do anything special other than enable Touch Screen Mode on your project. This will allow the operator to activate Touch Screen Mode when they log in. You can also enable Touch Screen Mode via scripting.

 Documentation

— □ ×



Documentation



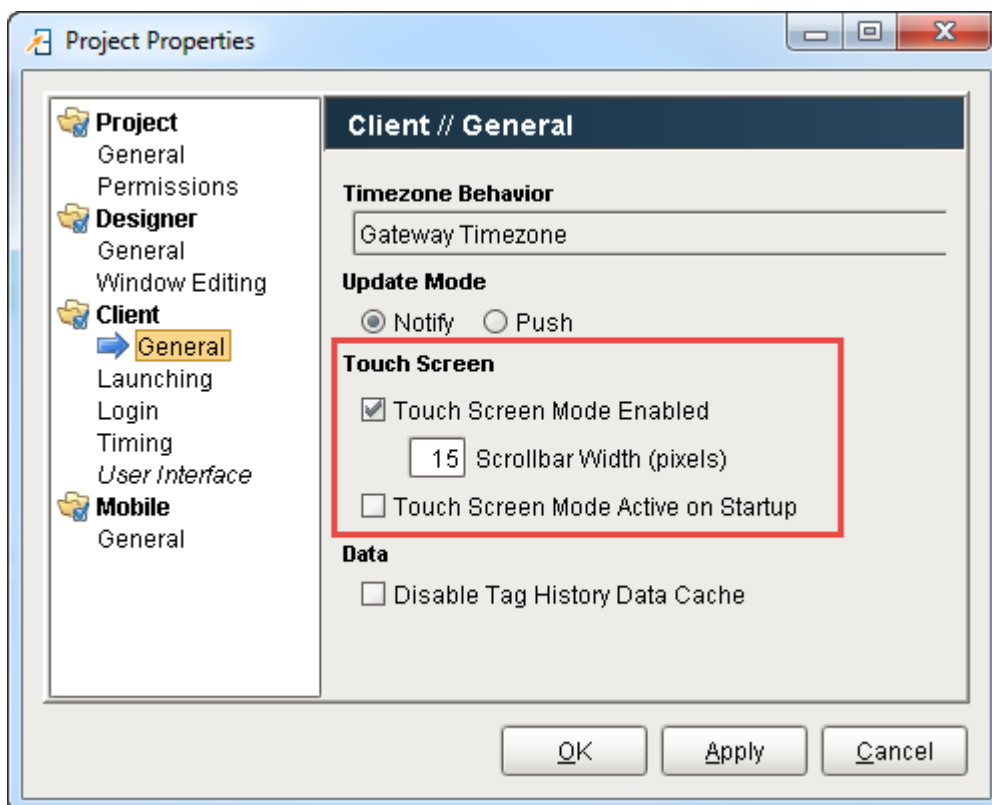
Touch-screen-enabled input components all have an expert level property called **Touch Screen Mode**. This property has three settings:

- **Single-Click:** The keyboard will appear on a single click.
- **Double-Click:** They keyboard will only appear after a double-click.
- **None:** Disable touch screen support on the component. The component will no longer invoke the touch screen keyboard.

Enabling Touch Screen Mode

Touch Screen support is built into Ignition. Turn it on through the Project Properties or scripting.

1. In the Designer, from Project Browser, double-click on **Project > Properties**, or from the top menus go to **Project > Properties**. The **Project Properties** window is displayed.
2. Go to **Client > General** page, to see the **Touch Screen** options.



Property	Description
Touch Screen Mode Enabled	By default, this is enabled, which means an operator can activate the mode on the startup screen. All Clients can operate in touch screen mode. When Touch Screen mode is enabled, clicking on numeric and text entry boxes will pop up on-screen keyboards that can be used for data entry. You can optionally set the width of any scrollbars (number of pixels wide/tall).
Touch Screen Mode Active on Startup	This option sets the Clients to start up with the touch-screen mode active.

These settings are helpful for mixed-use projects, that is, those that are launched on both touch screen devices and traditional computers and laptops.

Invoke the Touch Screen Keyboard with Scripting

To handle touch screen logic via scripting, the general pattern is to respond to a mouse event, popup up a keyboard, and then set the component's value to whatever was entered in the keyboard. For example, for a text field, you would write a script like this:

```
if system.gui.isTouchscreenModeEnabled():  
    currentText = event.source.text  
    newText = system.gui.showTouchscreenKeyboard(currentText)
```

See also: [system.gui.setTouchscreenModeEnabled](#)

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

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

Setting Up Auto Login

Client Auto Login

Clients can log in automatically when launched, once you specify the **Auto Login** settings in the **Client/Login** section of the **Project Properties** window. This is typically used in situations where the client should login with a low access user, such as a 'guest' account.

To Set the Clients to Automatically Login

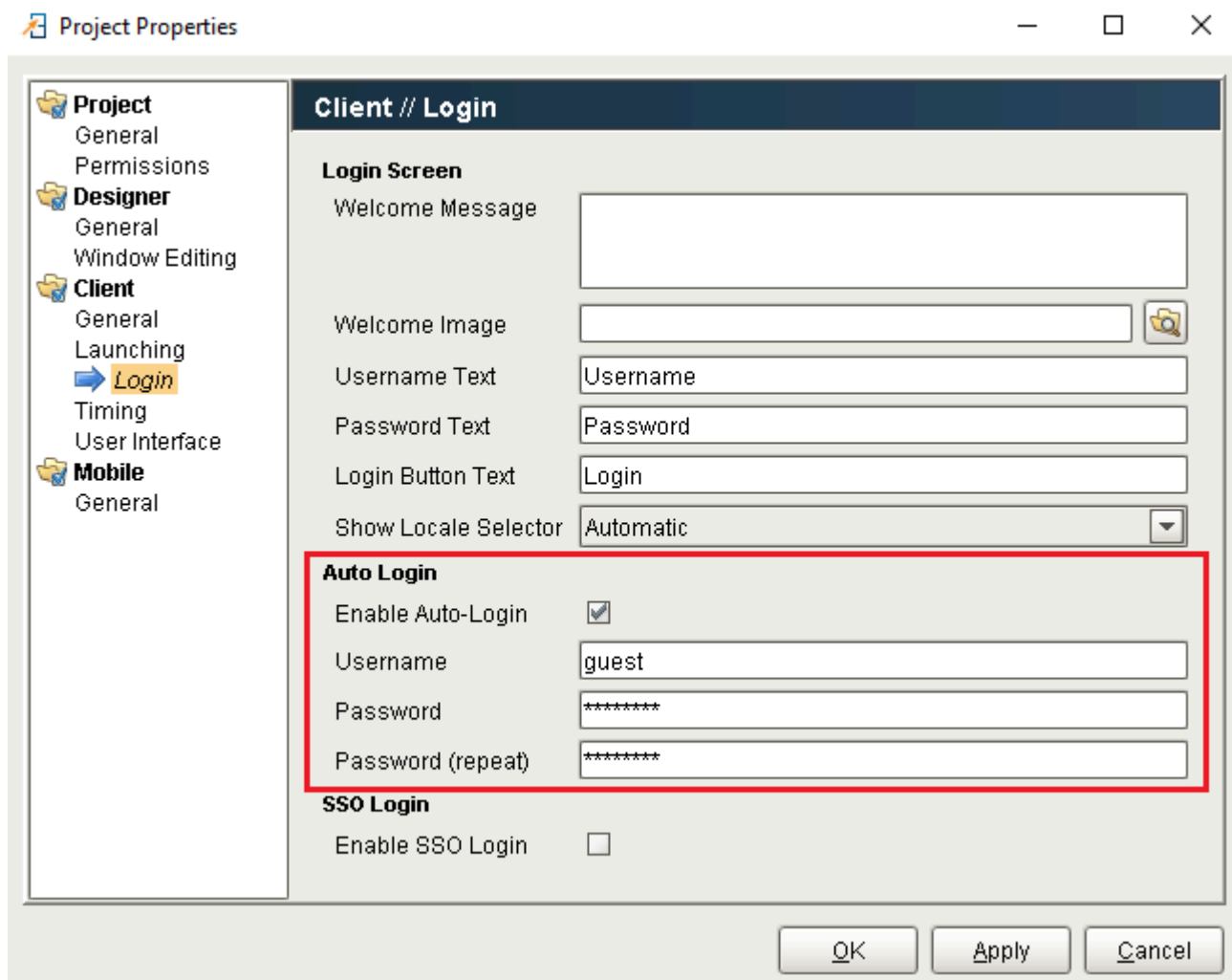


INDUCTIVE
UNIVERSITY

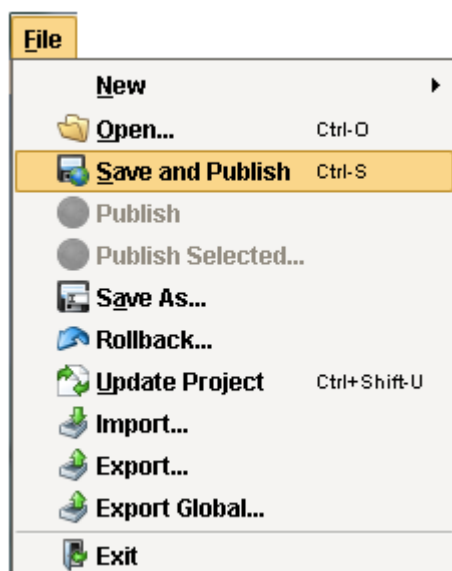
Setting up Client Auto Login

[Watch the video](#)

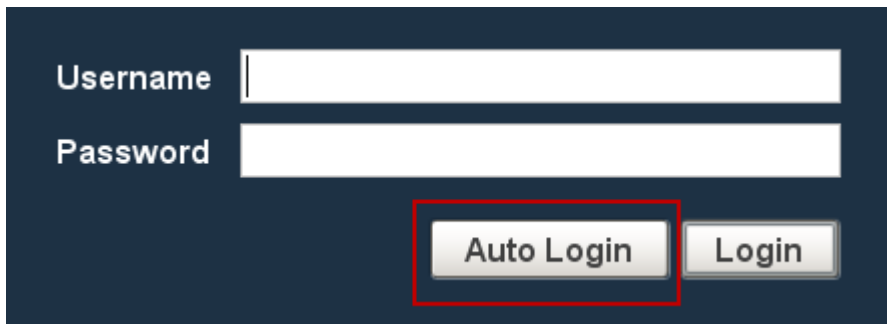
1. In the Designer, go to **Project > Properties**. The **Project Properties** window is displayed.
2. Go to the **Client > Login** page, and select the **Enable Auto-Login** box.
3. Type any Username and Password you want. For example, you can type:
 - Username: **guest**
 - Password: **password**
4. Click **OK**.



5. Select the **File > Save and Publish** to save your changes to the Gateway and publish your project.



New in 7.9.4 When in a client that has auto login setup, if the user accidentally logs out there is a button on the login screen to allow the user to enter in the auto login credentials. That way there is no need for them to know any login credentials.



A login form on a dark blue background. It features two white input fields: 'Username' and 'Password'. Below the 'Password' field, there are two buttons: 'Auto Login' and 'Login'. The 'Auto Login' button is highlighted with a red rectangular border.

Mobile Auto Login

Mobile Clients can log in automatically when launched once you specify the **Auto Login** settings in the **Mobile/General** section of the **Project Properties** window. This behaves the same as the above auto-login, but requires separate credentials for Mobile. This option will only be available when the **Mobile Module** is installed.

To Set the Mobile Clients to Automatically Login

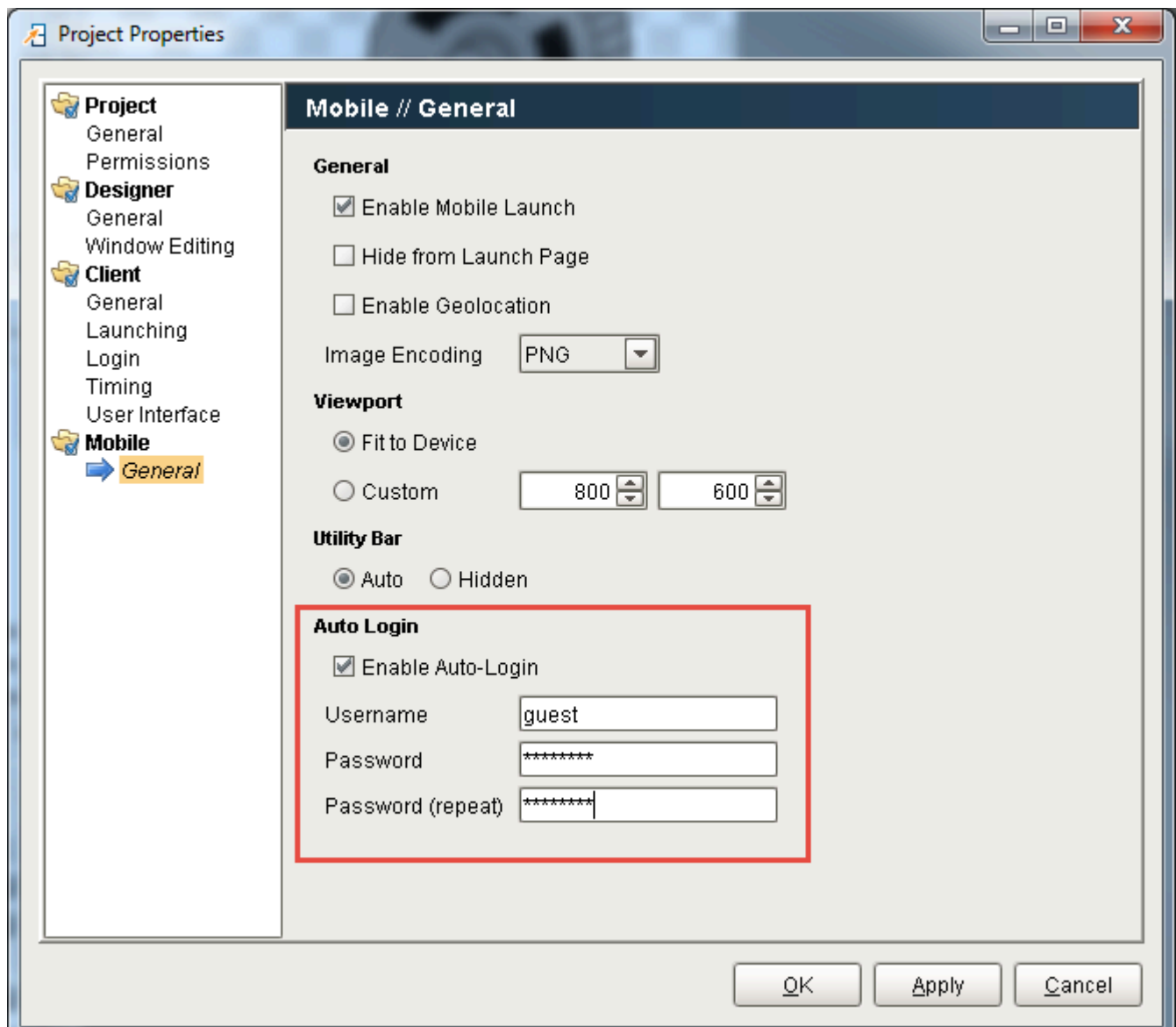


INDUCTIVE
UNIVERSITY

Setting up Mobile Auto Login

[Watch the video](#)

1. In the Designer, go to **Project > Properties**. The **Project Properties** window is displayed.
2. Go to the **Mobile > General** page, and select the **Enable Auto-Login** box.
3. Type any Username and Password you want. For example, you can type:
 - Username: **guest**
 - Password: **password**
4. Click **OK** and select **File > Save and Publish** to save your changes to the Gateway and publish your project.



5. Go to the Gateway **Home** page, find the **Mobile Launch** section, and click on the link. This launches the Client in your mobile device and automatically logs you in the web browser.

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

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

Visualization and Dashboards

User Interfaces

Most people that have Ignition will use it to create displays, controls, dashboards, trends, interactive database forms, etc. to help their operators and managers keep track of their processes. The Vision module allows you to create Clients (Runtimes) that include all of these types of HMI/SCADA screens.

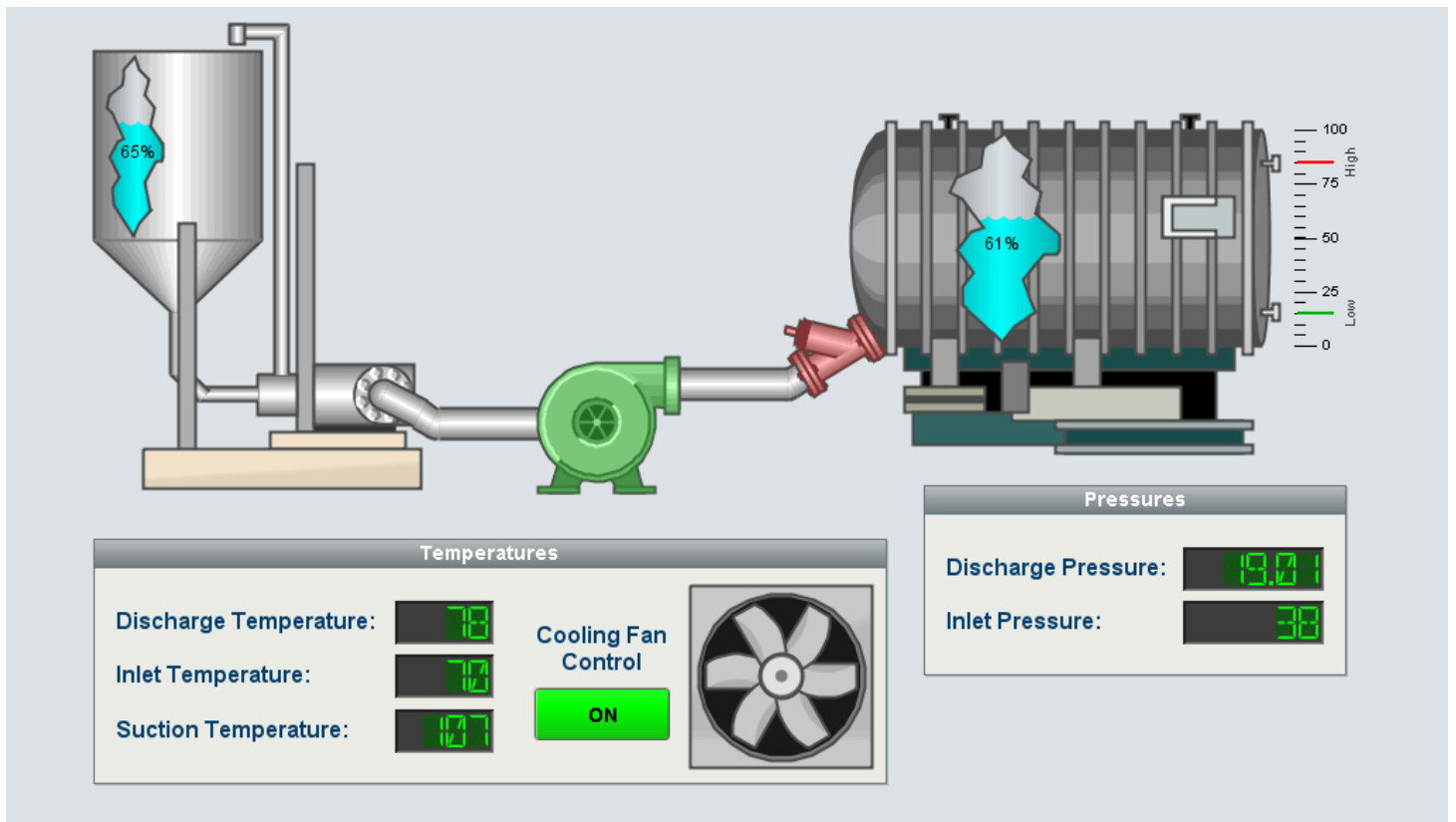
If you are using more modules than just Vision, you can link their tools together to create many types of screens, and set up just about anything you can imagine. Here are just a few examples of screens you can create in Ignition:

- Realtime Status and Control
- Dashboards
- Historical Trending
- Templates
- Alarm Status
- Database Manipulation
- Reporting

Because Ignition is so flexible, you can create exactly what you want with it. No more compromising on features, just build what you need!

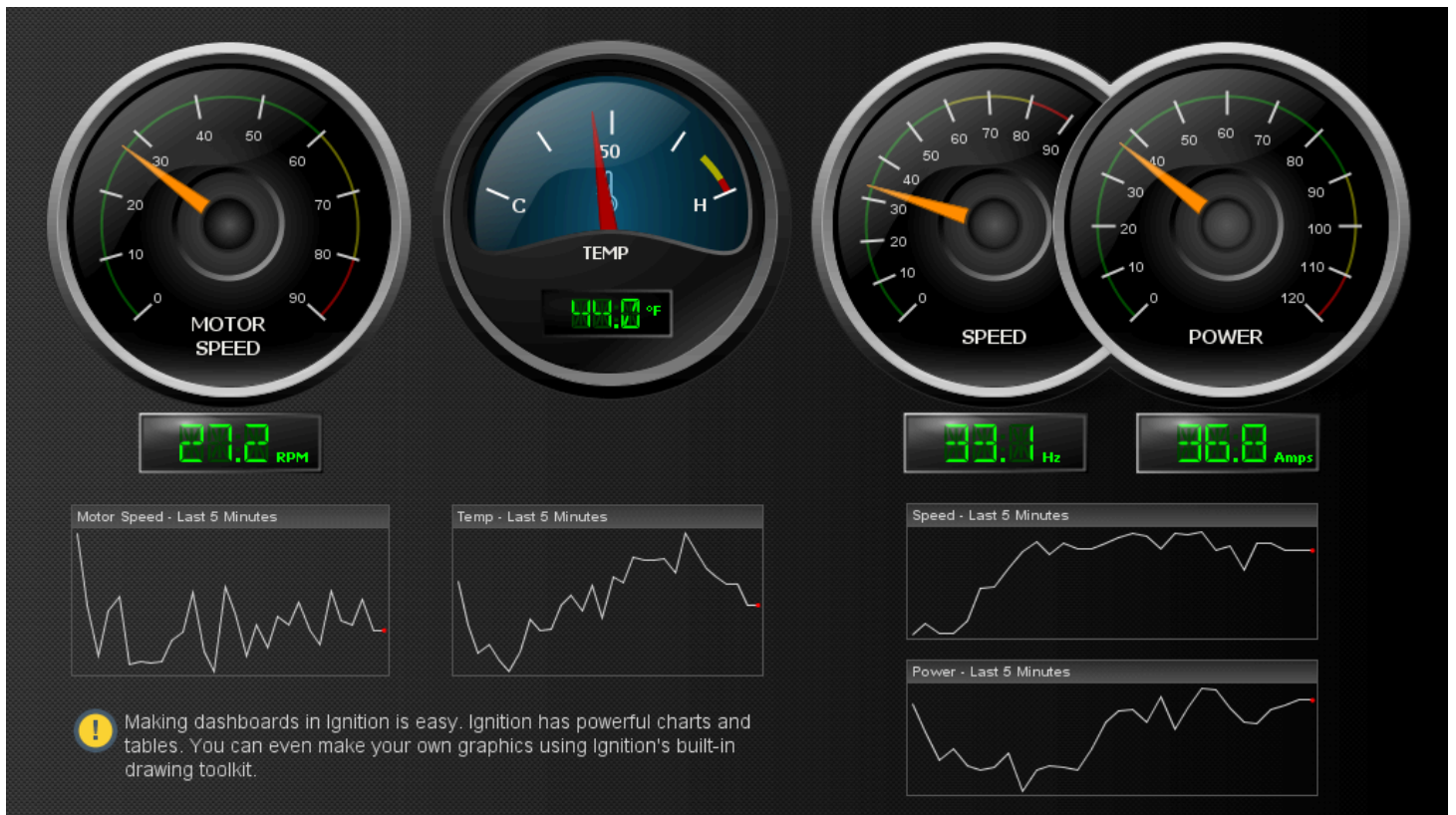
Realtime Status and Control

Create screens to monitor motor status, show digital displays, and control machines.



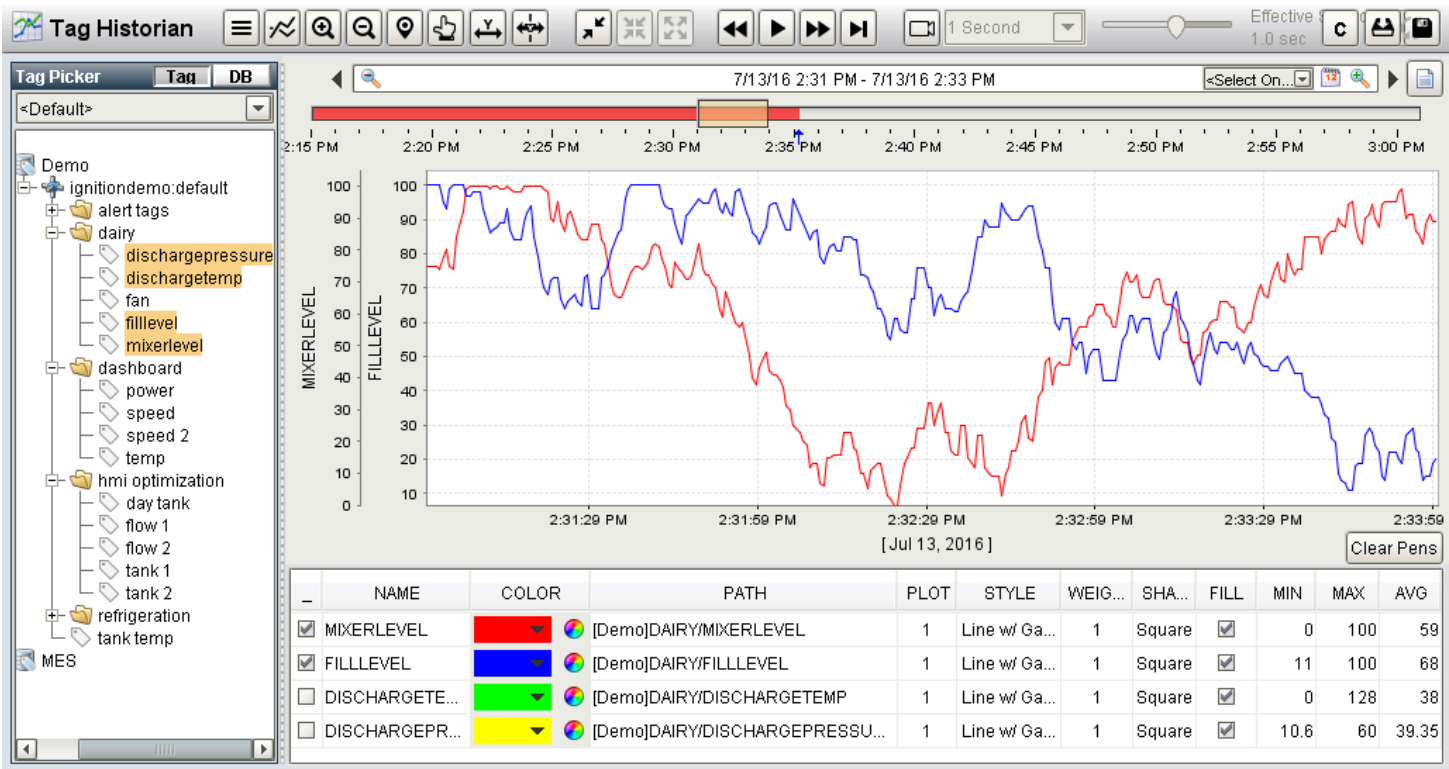
Dashboards

Create dashboards for quick and easy to read data at a glance.



Historical Trending

Create powerful **trending charts** that allow your users to see what they need, compare data, and look back through any history.



Templates

Create **powerful templates** and let Ignition do the work for you. Ask for as many as you want and Ignition will fill in everything for you.



Alarm Status

Quickly **monitor critical processes** at a glance. Change setpoints, acknowledge events, and receive notification when something goes wrong.

Example #1: Tank Level Analog Alarm

100%

Current Severity: Unknown

Last Alarm: 7/12/16 09:36:53

Change Value:

Alarm Conditions:
Lo Lo: 0 - 5
Lo: 5 - 15
Hi: 85 - 95
Hi Hi: 95 - 100

Example #2: Temperature Alarm

212.00

Change Value:

Current Severity: Unknown

Last Alarm: 7/12/16 09:36:57

Alarm Conditions:
Freezing: -∞ - 32
Boiling: 212 - ∞

Alarm Status

☐	Active Time	Display Path	Name	Current State	Priority
☐	7/13/16 2:36 PM	Day Tank	Hi	Active, Unacknowledged	Critical
☐	7/13/16 2:00 PM	Pump 4	Fault	Active, Unacknowledged	Medium
☐	7/13/16 2:00 PM	Pump 3	Fault	Active, Unacknowledged	Medium

Acknowledge

Shelve

Database Manipulation

Interact directly with your **databases**. Connect to existing ERP and inventory systems, read or modify data, and more.

Search:

Location:

<Select One>

X

Name	Description	Location	Quantity
100HP Motor	Some descripton	7654	10
250HP Motor		Warehouse 12	10
500HP Motor		Bin7	19
55 HP Motor	some motor	Warehouse 12	4
66 HP Motor		Warehouse 12	5
750HP Motor			5
88HP Motor	some motor	Warehouse 12	2
AB PLC		California	10
Battery		Bin3	8
Battery		Bin7	100
Compressor		bin5	5
Fortress of Solitude	It's cold	Warehouse 13	1
Harina	Molino de Harina	Cali	1,000
junk		Warehouse 13	1
PHDialiasis CVFRTY	sgdshdt	Tijuana	50
PLC Siemens		DF	41
PLC5	some plc	Warehouse 12	4
Some name	Adding some stuff	Bin3	999
UNitornix PLC	Descripción en español		5
Widget	Test	bin5	9

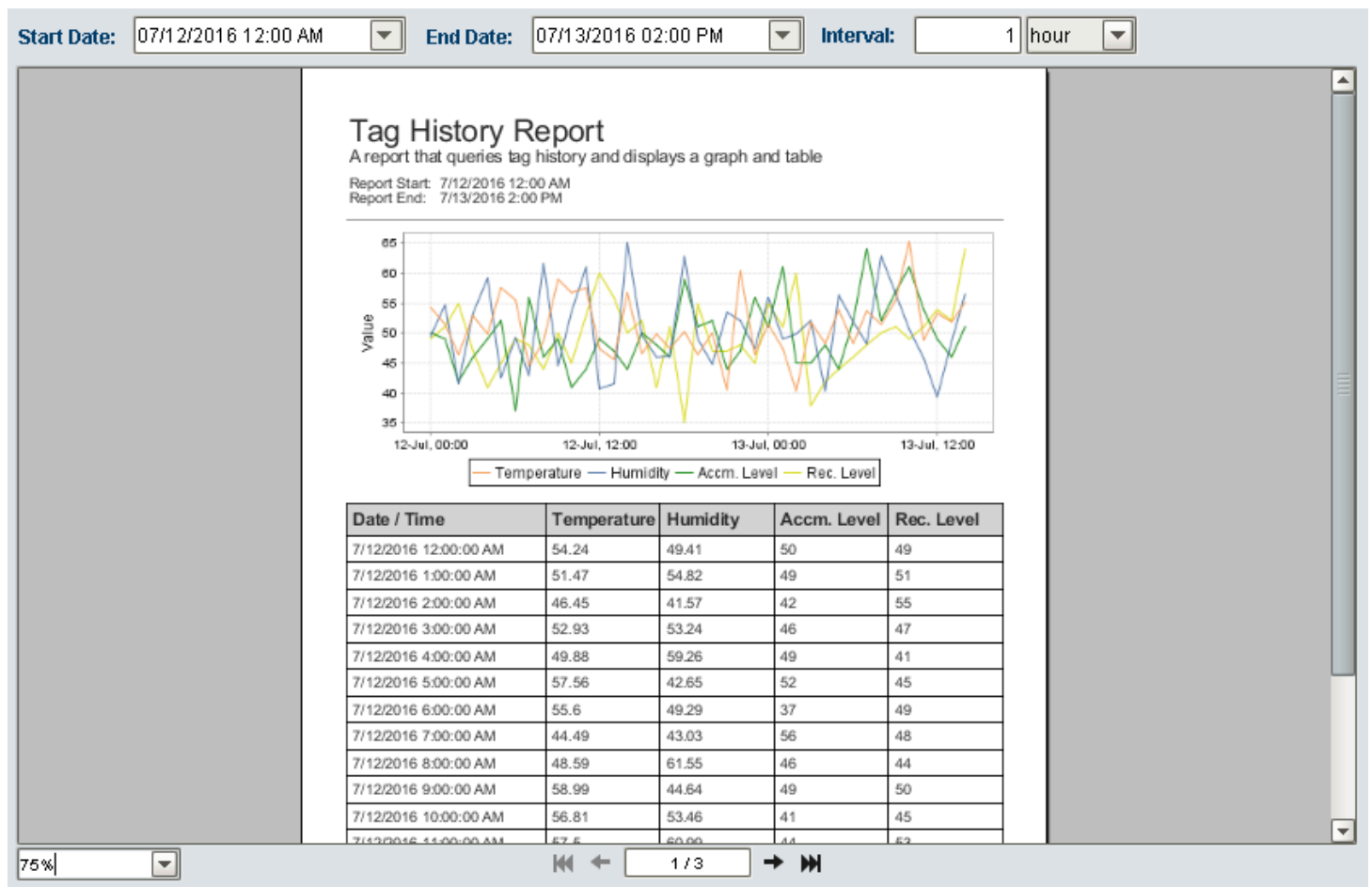
+ Add

Edit

X Delete

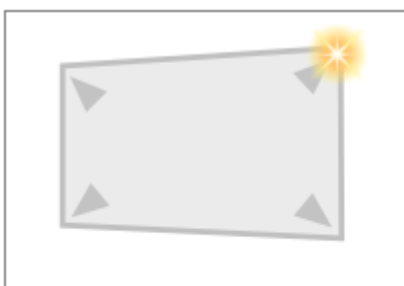
Reporting

Create **dynamic PDF reports** in your clients. Allow users to select data ranges, report types, and more. Print, save, and email your reports on the spot.

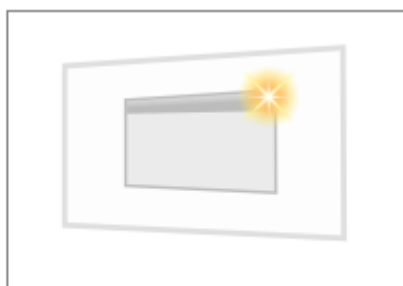


Windows

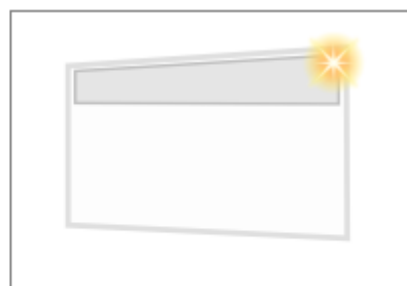
Windows are the basic building blocks for all of your screens. There are several types of windows in Ignition and they can be arranged however you like. You get to choose what windows are available on startup and how your navigation is set up. All components in your screens will live inside your windows. You can create and re-use windows easily inside of Ignition. Passing custom parameters into windows allows you to create once and re-use your screens. You can set windows to open once, or create popup windows with multiple instances so users can compare live values.



+ New Main Window



+ New Popup Window



+ New Docked Window

Navigation

You can set up **navigation** however you like in Ignition. There are several options available in the project templates that you can take advantage of, or you can create your own using a map with clickable zones, or whatever you like.

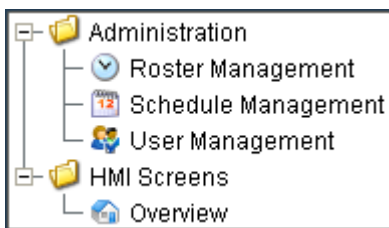
- **Back/forward buttons**



- **Navigation tab strips**



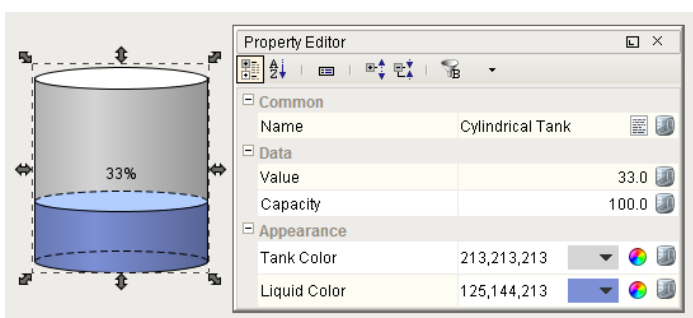
- **Multi-tier navigation**



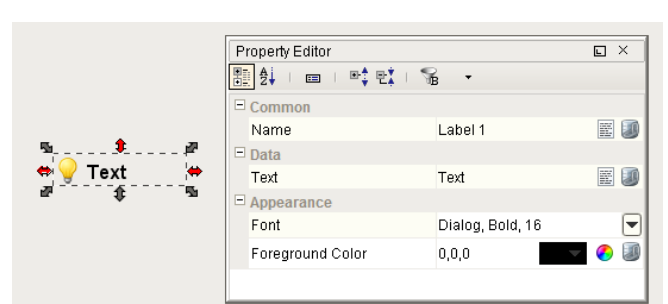
Components and Property Bindings

Components will take up the majority of your screens, and you can put them together in any way you like. Components are made up of properties that control the looks, behavior, and data. For example, a tank has a level, capacity, a liquid color, and more. An input field has text, font, an image, etc. You can even add your own **custom properties** to enhance components and create additional functionality. Components can be combined in simple groupings, **database editor forms**, or anything else you can think of.

Component Binding - Tank



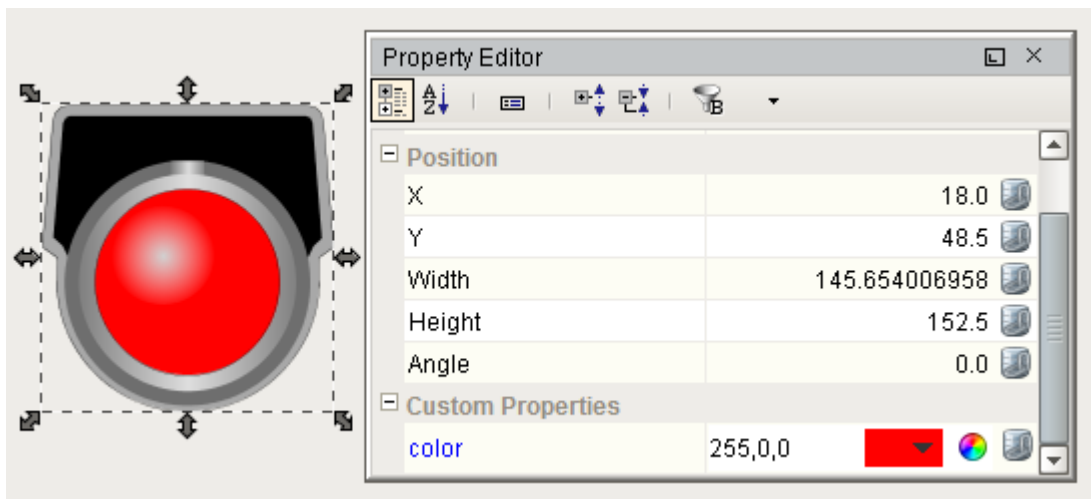
Component Binding - Text Shape



The power of **bindings** comes from the variety of different binding types. Bindings are how you can control component properties dynamically. Instead of setting a label statically, the text might change based on a PLC value or on-screen selection. There many ways and 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:

- **Other properties:** [Property Binding](#), [Cell Update Binding](#), [Component Styles](#)
- **Tags:** [Tag Binding](#), [Indirect Tag Binding](#), [Tag History Binding](#)
- **Expressions:** [Expression Binding](#)
- **Databases:** [DB Browse Binding](#), [SQL Query Binding](#)
- **Functions:** [Function Binding](#)

In addition to standard components, Ignition has a set of **2D Drawing tools** to allow you to create your own graphics or use Symbol Factory to drag and drop SVG images into your windows. Just like any other component, you can use property bindings and set up custom properties to add whatever functionality you need.



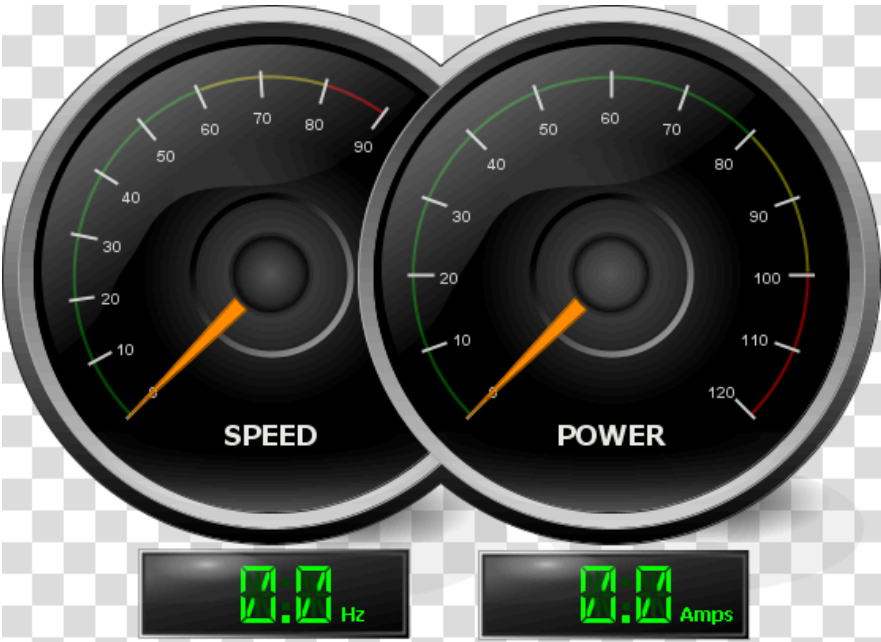
Templates

Components can be combined to create **Templates**. These are re-usable objects that can be set up once and used throughout your project. The real power of templates comes from the ability to modify it in one location while affecting all of the instances used throughout the project. Most templates use one or more custom properties (Template Parameters) to tie data from a window to the internals of the template.

New in 7.9.13

Cloud Templates have been moved to the new **Ignition Exchange**. The Cloud browser itself has been removed as of 7.9.13. All Cloud Templates have been automatically moved to the Ignition Exchange and are still available.

In the Exchange, you can view and download many of templates, such as the one below.



Property Editor

Name	Double Gauge
Appearance	
Width	561
Height	403
Template Parameters	
gauge1interval2StartPercent	67.0
gauge1interval3StartPercent	89.0
gauge1label	SPEED
gauge1max	90.0
gauge1value	0.0
gauge2interval2StartPercent	67.0
gauge2interval3StartPercent	83.0
gauge2label	POWER
gauge2max	120.0
gauge2value	0.0
led1Label	Hz
led2Label	Amps
showLED	☒ true

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

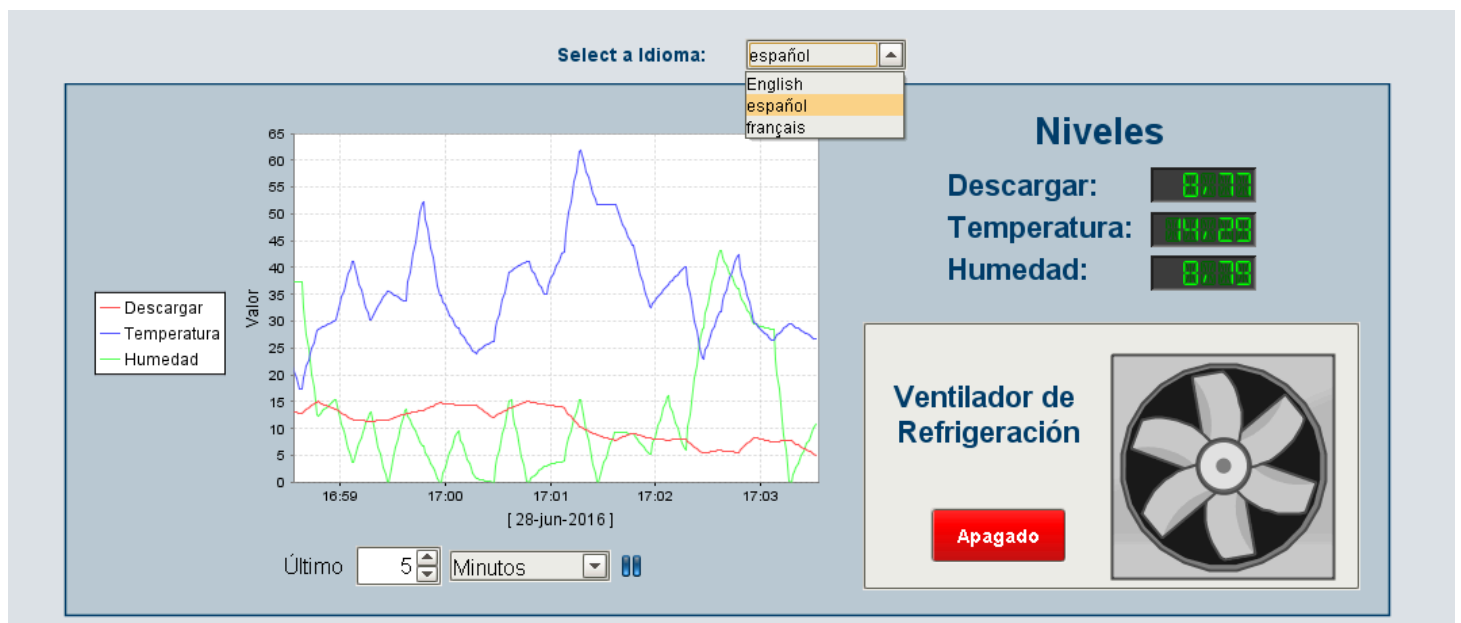
For documentation on the latest stable release, see the [latest version](#) (8.3).

Localization and Languages

Translating Ignition

Localization allows you to translate text into multiple languages in a project for display on client screens. The localization feature allows users located in different countries to set their default language so client screens can be displayed in their native language. The user can easily choose which language their Client displays with a click of a button! Text contained in components, that have their properties correctly mapped with localization, will change to reflect the language the user has selected. There is a special **Language Selector** component that requires no set up at all! It's a simple dropdown list that displays the language options for these translations.

Just pick the language you want and all the text on the window will be displayed in the language the user selects.

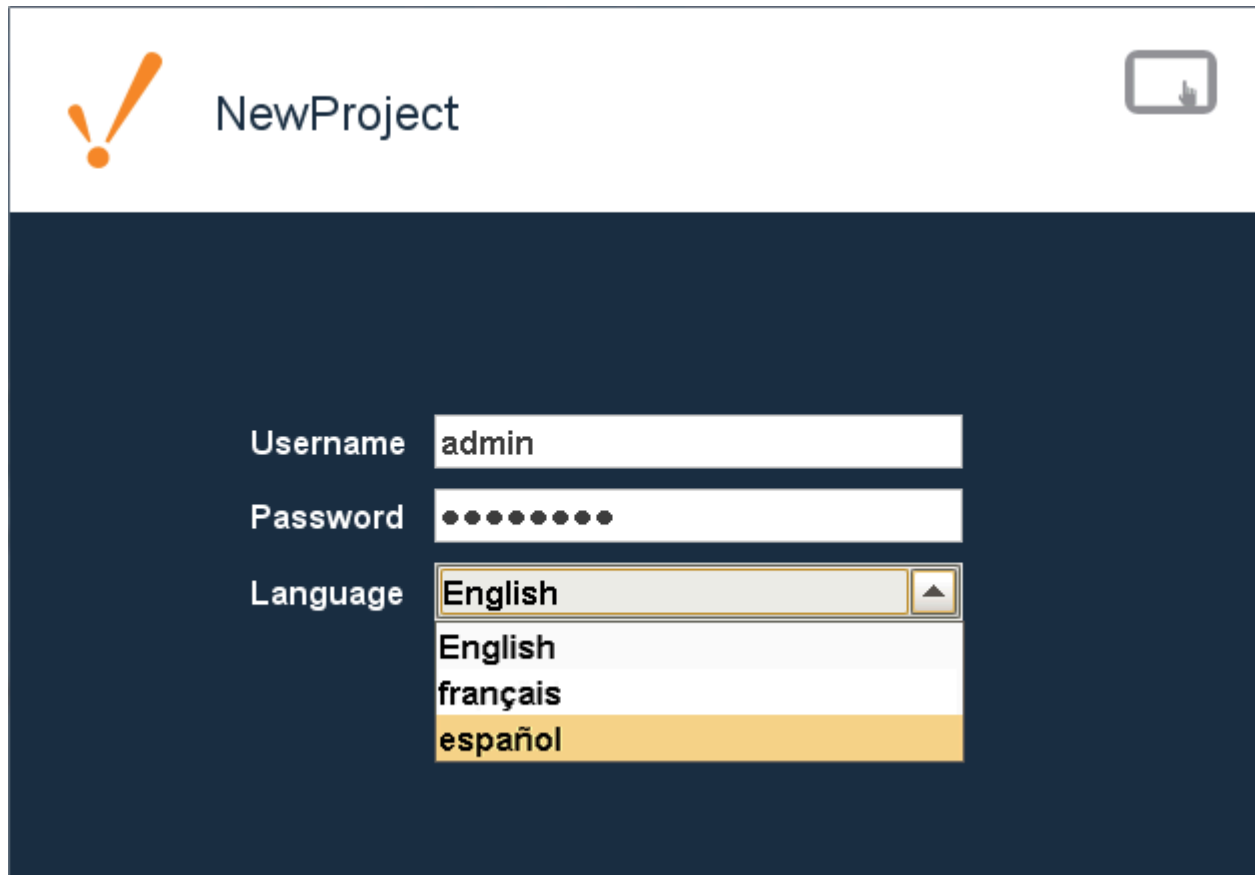


Change all Text just by Selecting the Language

It's simple to **switch the current language** in your project. Ignition has several easy ways to change language settings.

Login Screen

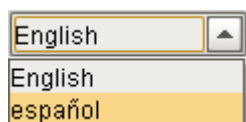
One way you can select your preferred language is on the Login screen. There is a dropdown list that automatically appears as soon as you add a second language to your project.



The screenshot shows the 'NewProject' login interface. At the top left is an orange exclamation mark icon, and at the top right is a small icon of a monitor with a hand cursor. The main area has a dark blue background. It contains three input fields: 'Username' with the text 'admin', 'Password' with ten black dots, and 'Language'. The 'Language' dropdown menu is open, showing three options: 'English', 'français', and 'español'. The 'español' option is highlighted with a yellow background.

Dropdown Language Selector Component on a Window

You can also switch the language in your project by using the Language Selector Component on your project window. In **Preview Mode**, select the dropdown language selector component, and choose your desired language.



This is a close-up of the dropdown language selector component. It shows a small box with 'English' and a dropdown arrow. Below it, the dropdown menu is open, showing 'English' and 'español'. The 'español' option is highlighted with a yellow background.

Using Scripting to Change the Client's Locale

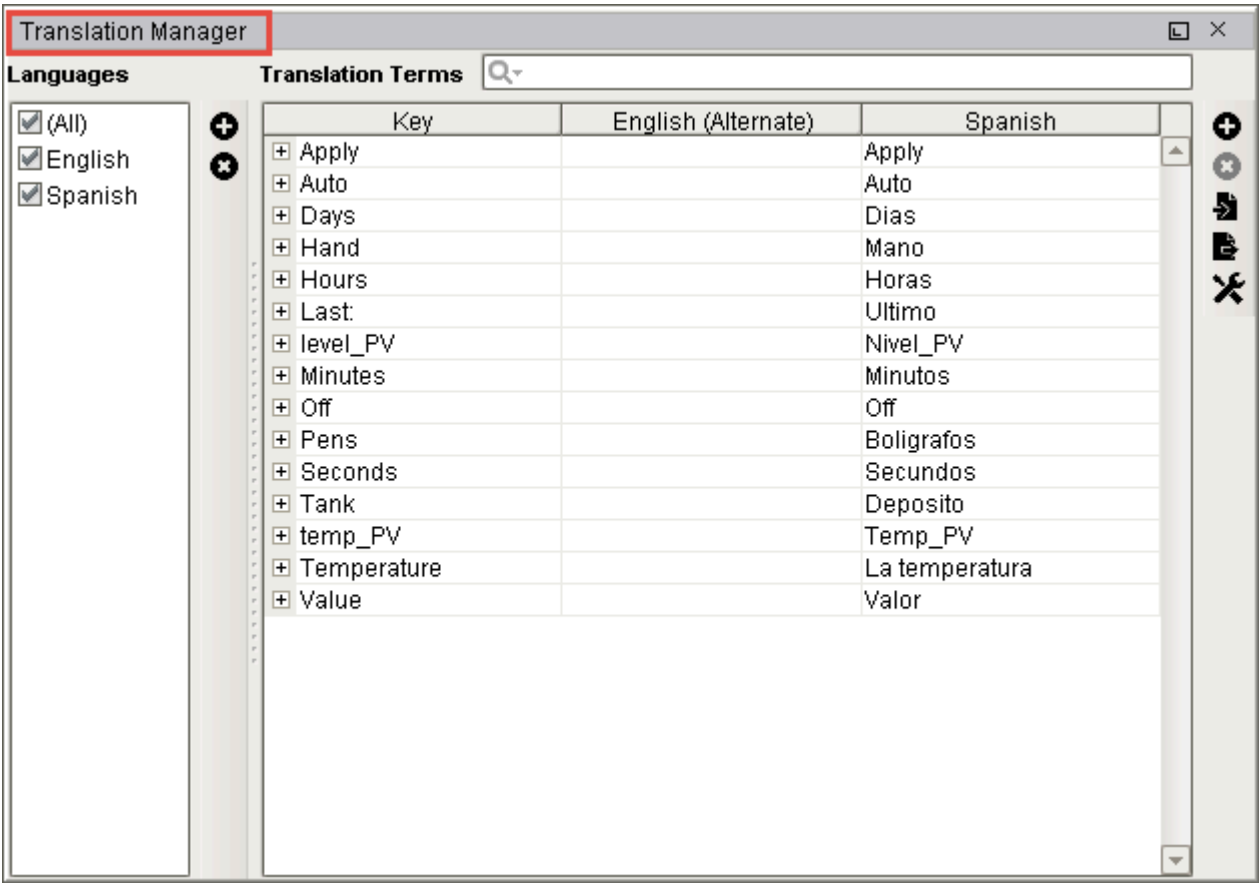
Aside from manually selecting a language, the Client attempts to do this automatically by detecting the locale configured on the host computer. The locale can be specified via scripting with [system.util.setLocale](#). This allows you to utilize scripting events, such as a [Client Startup Script](#) to automatically change the client's locale when a user logs in.

Store All Translations in One Central Location

Translation Manager

All translations are stored centrally in the Gateway and are distributed to each Client and Designer. All projects share the same translations, and those translations can be used in other locations, such as Gateway scripts, and alarm messages. The full translation database can be viewed and edited in the **Translation Manager** panel located in the Designer. You can add new terms and phrases to the Translation Manager. Make your phrases as long or short as you like with single words or whole paragraphs. It allows users to easily **import and export** translatable components and share them across projects and with other users. It also has the ability to define new languages.

A central term database and support is automatically built into all component text properties, as well as other text-based properties, such as alarm messages. Aside from defining translations for terms, there is usually no other work needed to take advantage of the translation system.



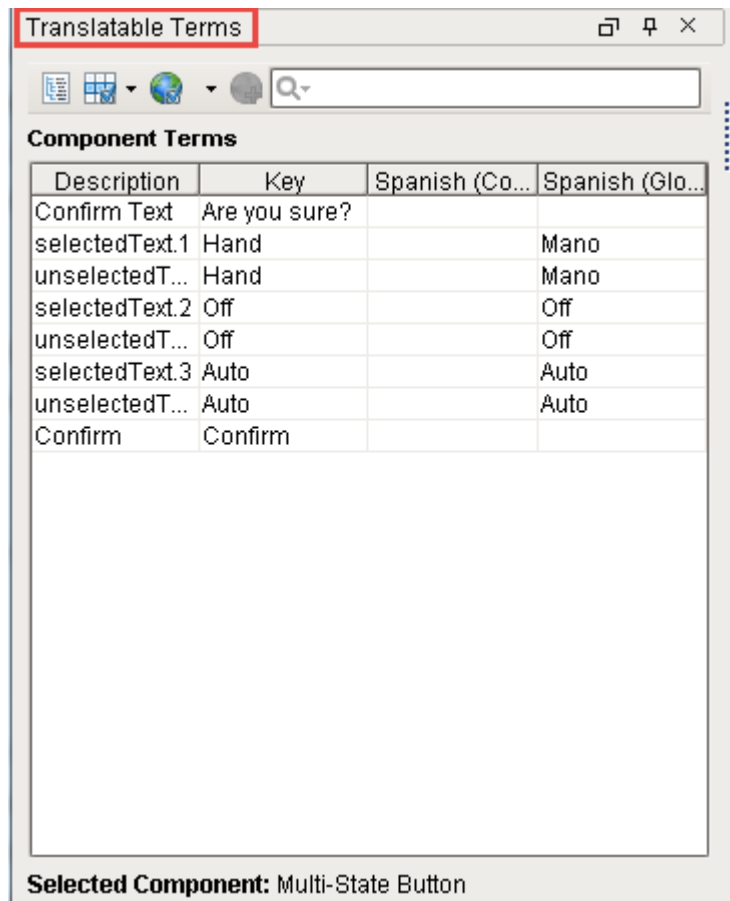
Translation Database and Term Lookup

Terms are referenced in the translation database using direct string comparison. Ignition has special term **Keys** that are used to identify when to translate text. The base term may be a user-readable string such as "Start", or any special code you want such as "START_COMMAND" or "#start", which would have an alternate translation for English defined as "Start." This way you can quickly Identify

which terms will be translated when working in the Designer. To keep things simple, the English key is automatically used if no match is found for the requested language.

Quickly Identify Fields to Translatable Terms

You can quickly and easily identify, and translate all terms by selecting a container and opening the Translatable Terms pane. Here, you can see all terms for each component in the container, and if each term has an associated translation for the requested language.



Previewing Translations

You can easily preview your translations in the Designer! To change the preferred language for Preview Mode, use the **Preview Language** menu item under the Project menu in the Designer. When you put the Designer into Preview Mode, everything on your windows will be automatically translated. Using the Language Selector Component or scripting can still temporarily change the current language, and the language will revert back any time Preview Mode is disabled. The Designer will remember your choice, even when the Designer is restarted.

Project	Component	Alignment
☐	 Comm Off	
☒	 Comm Read-Only	
☐	 Comm Read/Write	
 Properties		
 Gateway Event Scripts		
 Client Event Scripts		
☐	 Preview Mode	F5
 Preview Language ▶		

☐ English
☒ español

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

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

Creating Translation Lists



INDUCTIVE
UNIVERSITY

**Introduction to Project
Translation**

[Watch the video](#)

Project Translation Overview

Project translation is also known as Client localization. This is how the user can choose their language-of-choice for the project they are working with. When the user selects a language, the Client will change its text in accordance with the project's translation settings. Translations work with all components (i.e., Labels, Charts, Multi-State Buttons, etc.,).

ⓘ TRANSLATION NOTES

- It's important to note that there is no translation for text within Table components.
- For Text elements that have bindings, the translation occurs after the binding is evaluated.

The Translation Manager allows you to view all the terms from all your projects in the global translation database. You can add, edit, and remove terms, and provide translations, as well as allow users to easily import and export translatable components. The translation database shares terms across all projects and with other users.

Setting up New Languages, Terms, and Translations

Setting up new languages and adding terms for translation is pretty easy. For every element of text on the screen, you can provide a translation in the desired language. When a user logs into a project, they can choose their preferred language, and all the text will be displayed in their preferred language.

They can also have their preferred language defined in user settings so when a user logs into the Client, their preferred language is automatically enabled.

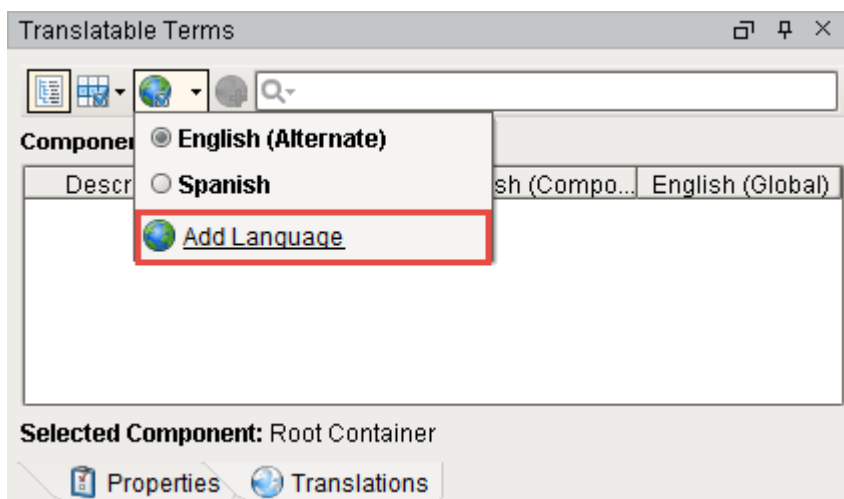
Component vs Global Terms

There are two types of terms used in the translation system: Component and Global. Both behave a little differently, and it's important to know the difference.

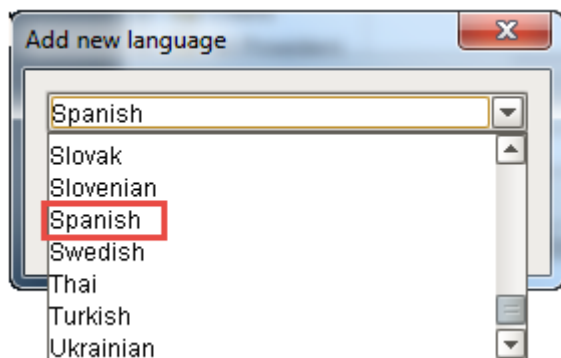
- **Component Terms** - Component level terms are specific to that one component, and translations are added to the translation system using the Translatable Terms Panel. Any component level translation will automatically take precedence in the event there is a matching global term.
- **Global Terms** - Global terms and translations are available in both the Translatable Terms Panel and in the Translation Manager's global database. The global translation database provides language translations on all components, in all windows, and on all projects. You can setup global translations so that anywhere a term is used within your project, it will get replaced with that global translation. Although, there are times, when a component might have a different context than the global translation. For this reason, you may want to create a component level translation to take precedence over the global translation. For example, some users may work in a specific window of a project, and what is typically called a 'Tank,' this group of users refer to it as a 'Barrel.' You can create a component level translation for 'Barrel' to take precedence on the root container of a window. This way the users see the term in the context that is familiar to them.

To Add a New Language

1. In the **Designer**, right click in your workspace and select **Translations** from the dropdown. The **Translatable Terms** window will open.
2. Select the **Earth** icon and click **Add Language**. By default, the English language is defined.



3. Select a new language from the dropdown. That's all there is to it! You can create as many languages as you need for your project.



To Add Global Translation Terms

Creating global translations is always a good first choice over the component level translation because most of your translated terms will be used across all projects unless you want one particular term to be different from the global translated term.

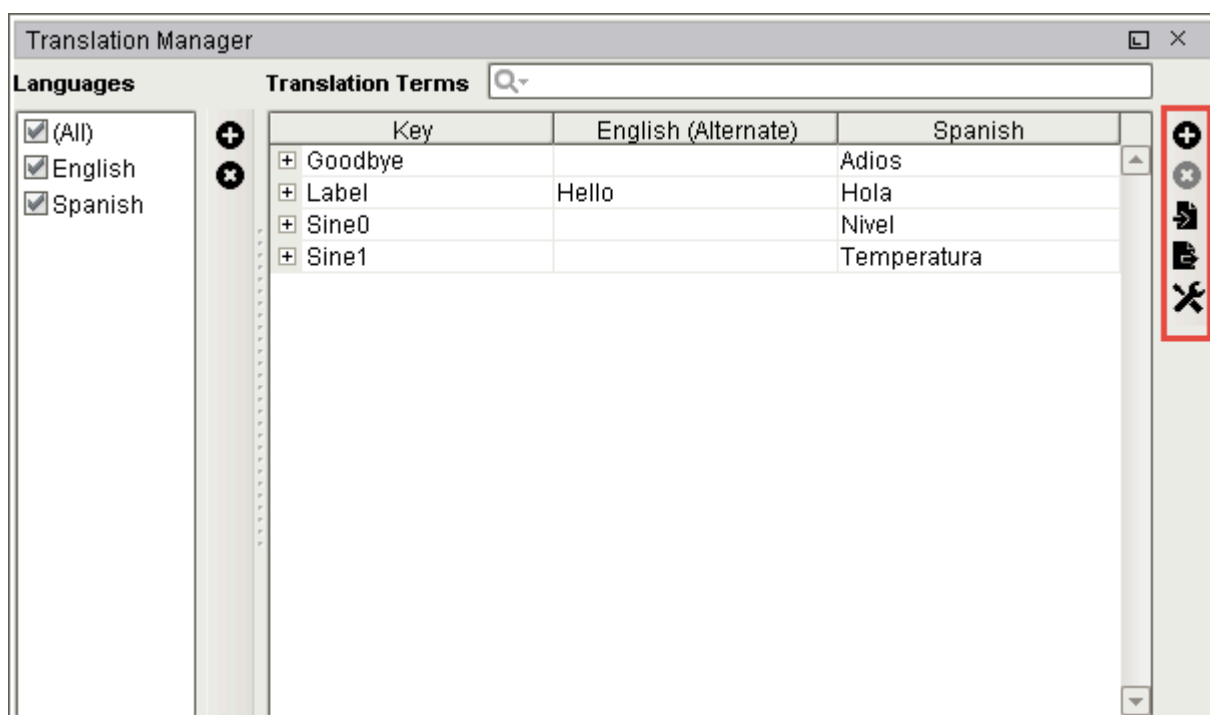
There are a couple of ways to add terms to the Translation Manager. You can add new terms directly to the Translation Manager or by selecting terms from the Translation Terms Panel.

Let's add some terms directly to the Translation Manager.

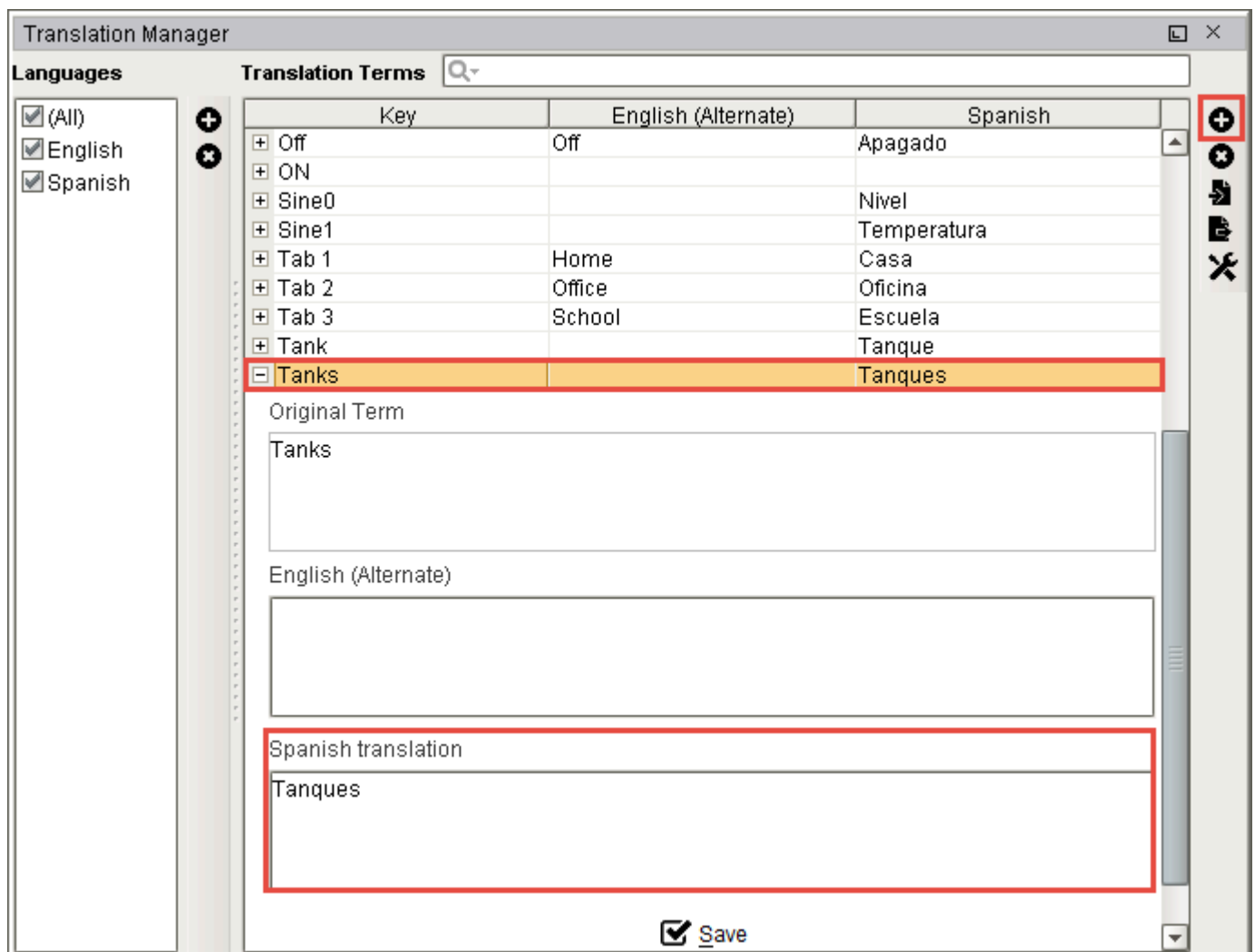
1. From the menu bar at the top, go to **Tools > Translation Manager**.

The Translation Manager window will open, and you can view all the terms that were added. This example has terms that were already added and translated.

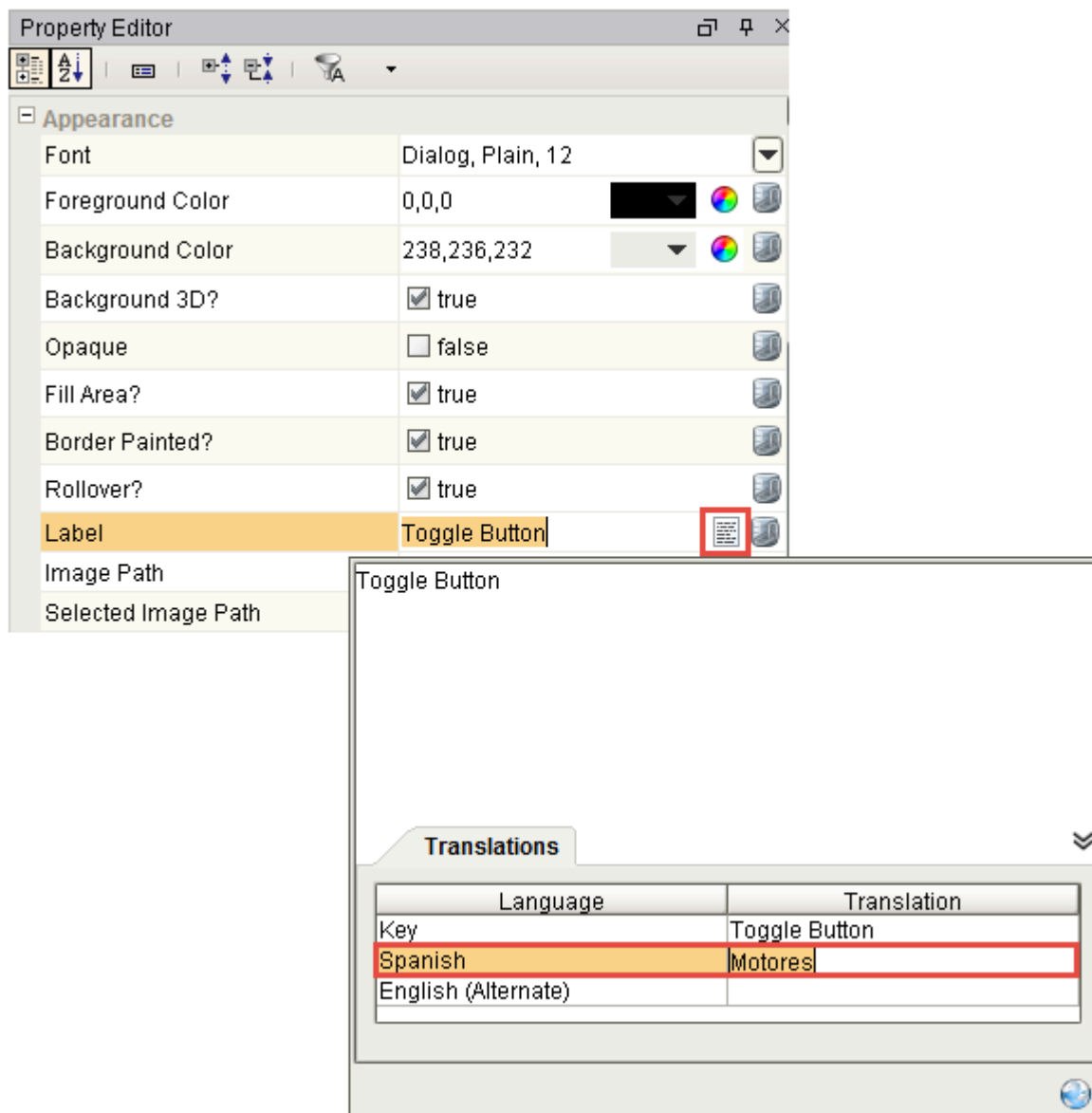
On the right side of the window, there is a list of icons where you can add, delete, import, export, and edit the Translation settings.



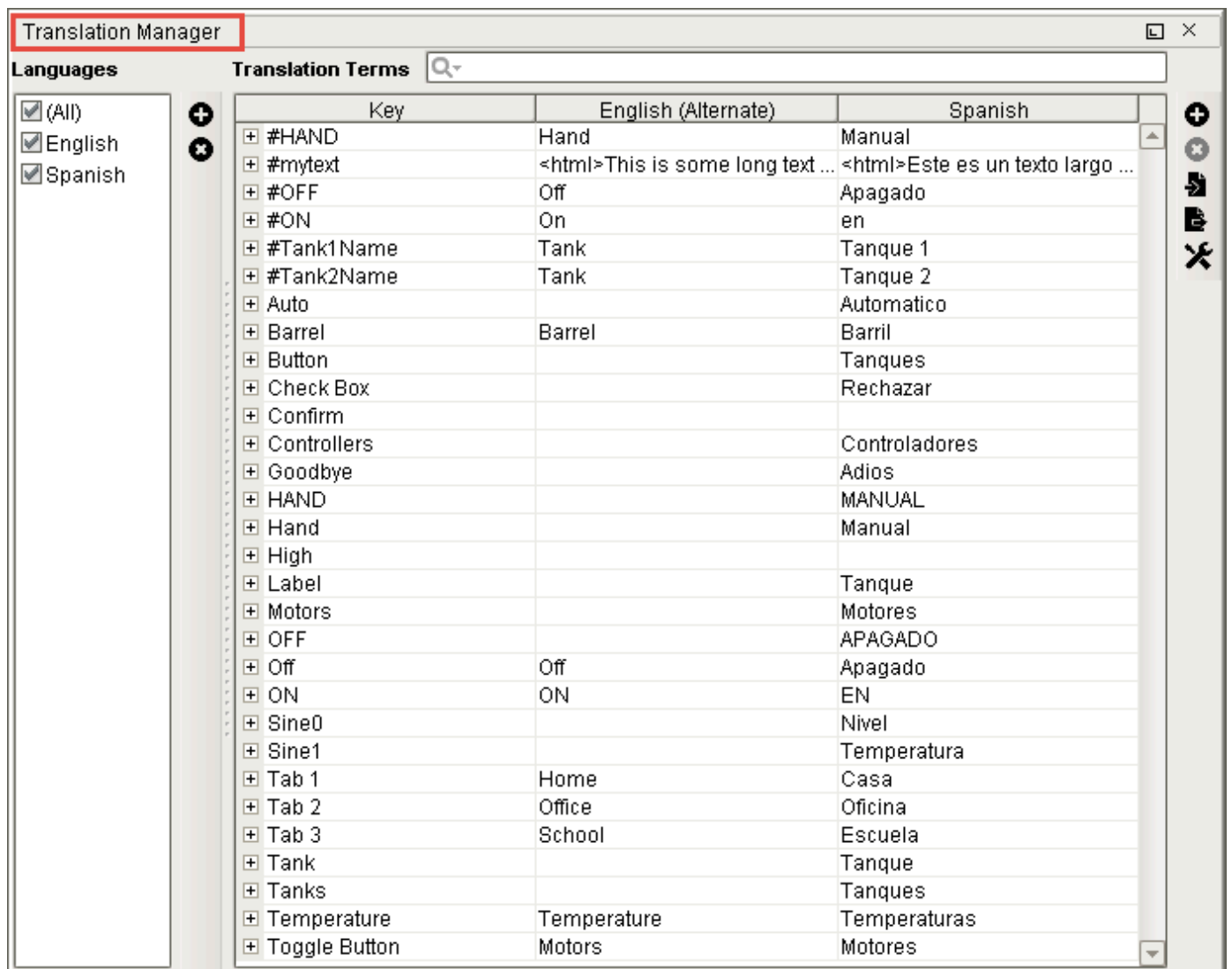
2. To add a new global term, click on the **plus icon**. Enter the term (Tanks), and press **OK**.
3. Double click on the row of the new term, and under **Spanish translation** enter the word 'Tanques.' Click **Save**.



4. For Text components on a screen (i.e., Label, Button, Toggle Button, etc.), you can add translations in the Property Editor. Under **Appearance**, click on the **text box** next to the name of the text component. Click the **Translation Button** at the bottom of the text window. Click the **green plus icon** to add a translation. After you enter the translation, hit enter. The term will be added to the global translation database.



5. To view all the global terms added to a project, go to **Tools > Translation Manager**. If you want to provide a translation to a term that has not been translated, double click on the row of the term, enter the translation, and click **Save**.



6. Now, let's test it out! Drag a **Language Selector** component to your window. Go into **Preview Mode** and toggle the languages. You will see the languages for the text switch between English and Spanish for these two components.



To Add Component Level Translation Terms

What's nice about adding component translation terms is that the Translatable Terms panel provides all the terms for that component on the window. All you have to do is add the translations. The

Translatable Terms Panel is a central place to manage all the translations associated with a specific component. One important reason to use a component level term is if it is going to be different from the global term. Just remember, that because a term is translated at the component level doesn't mean that other components get to use it. Typically, it is used for that component only.

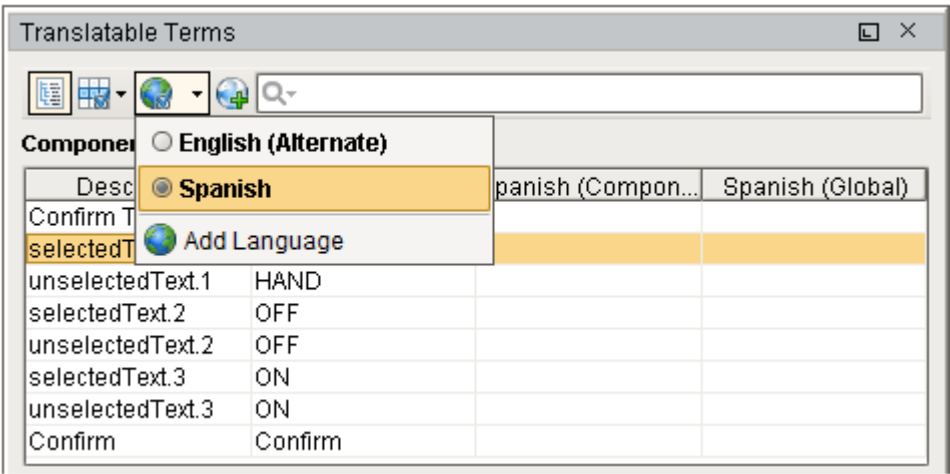
Let's get started!

1. In the **Designer**, drag a **Multi-State Button** from the Component Palette in to your workspace.
2. Right click on the component and scroll down to **Translations**. The Translatable Terms panel will open and display all the terms for the specific component. Alternatively, you can right-click on the Root Container of a window and select **Translations**. The **Translatable Terms Panel** will display all the terms for all the components on the window.

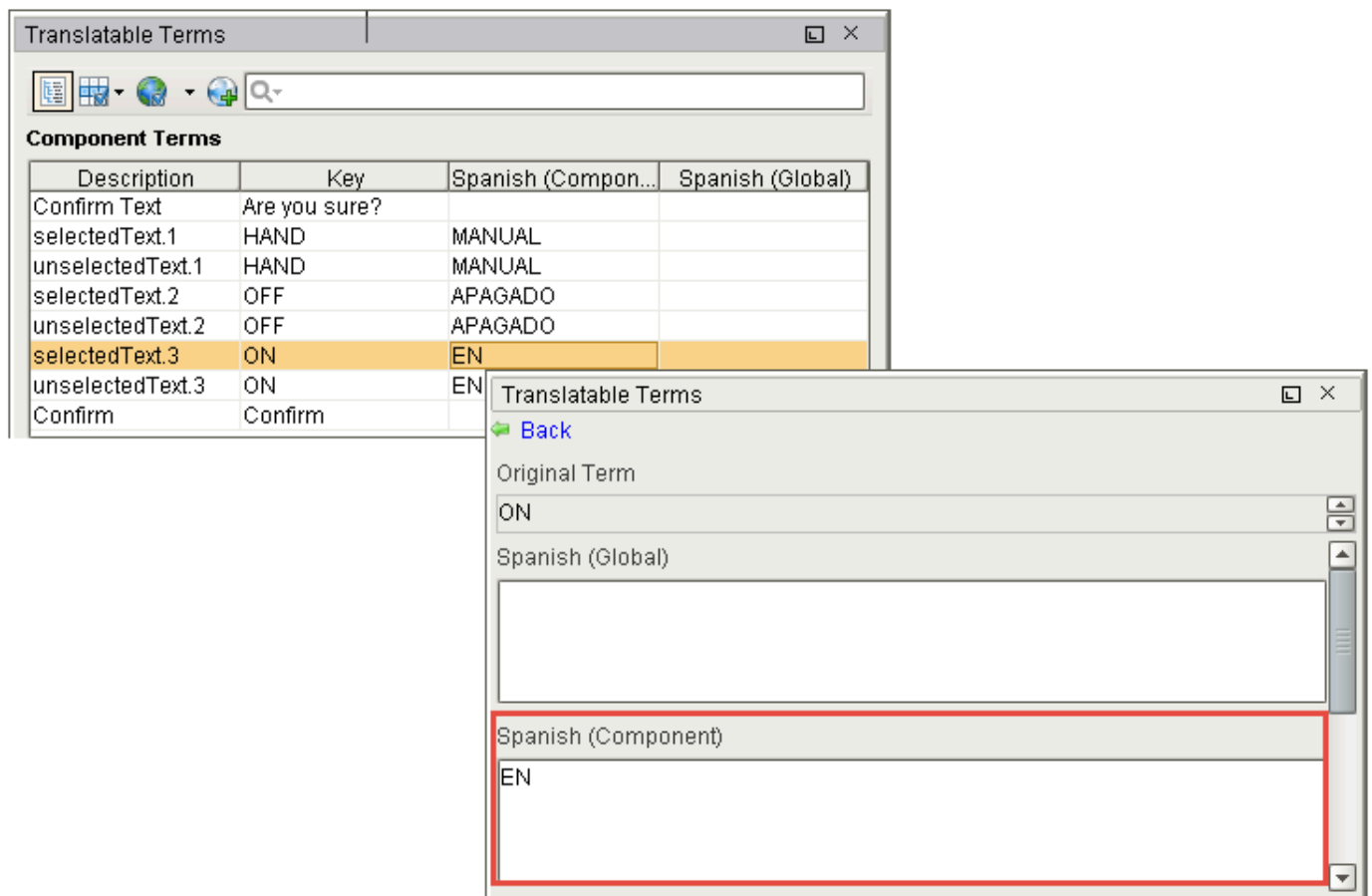
! TRANSLATABLE TERM OPTIONS

In the Translatable Terms Panel, you have the option of adding your translation to the Component, Translation Manager global database, or marking it for translation at a later time. You'll notice under the Component Terms section of the Translatable Terms window that there are four columns; Component Description, Key, Component translation, and Global translation.

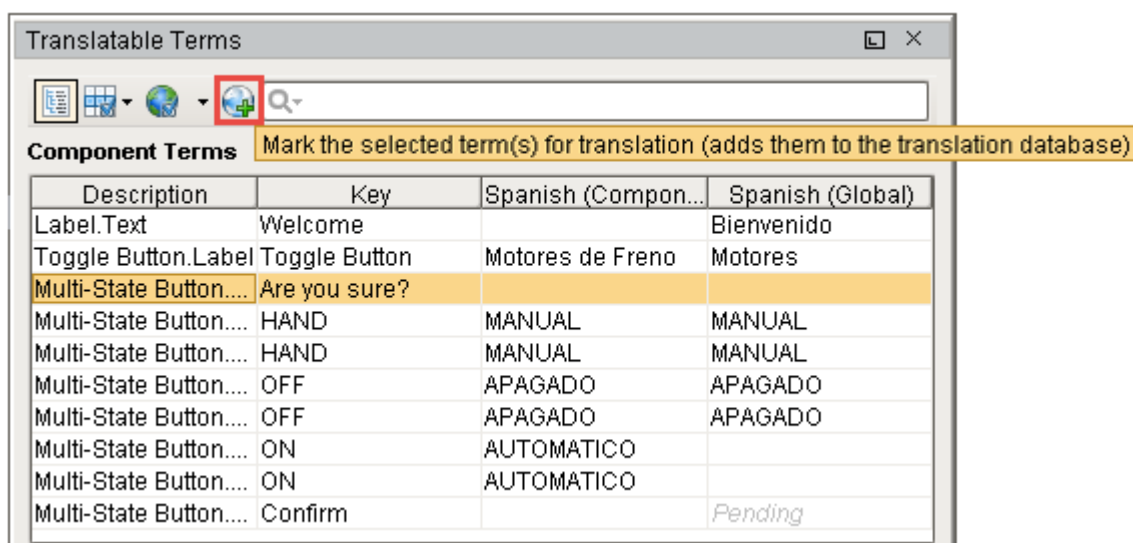
You'll see below all the terms for the Multi-State Button, and none of the terms have translations added. If you don't see the language you want your terms translated to, click the Earth icon  and select the language (i.e., Spanish). You will then see a column for your component translation and global translation.



3. To add translations for your component terms, double click the row of your term (i.e. ON). In the **Spanish Component box**, enter the translation (i.e, EN). You also have the option of creating a global translation on the Translatable Terms Panel as well. Then, click the **Back** link, and repeat this step for all your terms.



4. Once you entered the translations for all your terms, you will see them on the Translatable Terms Panel in the Spanish (Component) column. If a term doesn't have a translation, you can select the term and mark it for translation by clicking the **Earth with a green plus icon**. This adds the term(s) to the translation database for someone else to add the translation. In this example, you'll notice several things about each of the terms so look closely at them. You'll find examples of both component and global level terms, a component level term, a global term, a component term that overrides a global term, and terms marked for translation.



5. Let's test it out! Drag a **Language Selector** component from the Component Palette to your workspace. Go to **Preview Mode** and select Spanish from the dropdown to see the text change from English to Spanish for these two components.



Translation List Import and Export



INDUCTIVE
UNIVERSITY

**Import and Export
Translations**

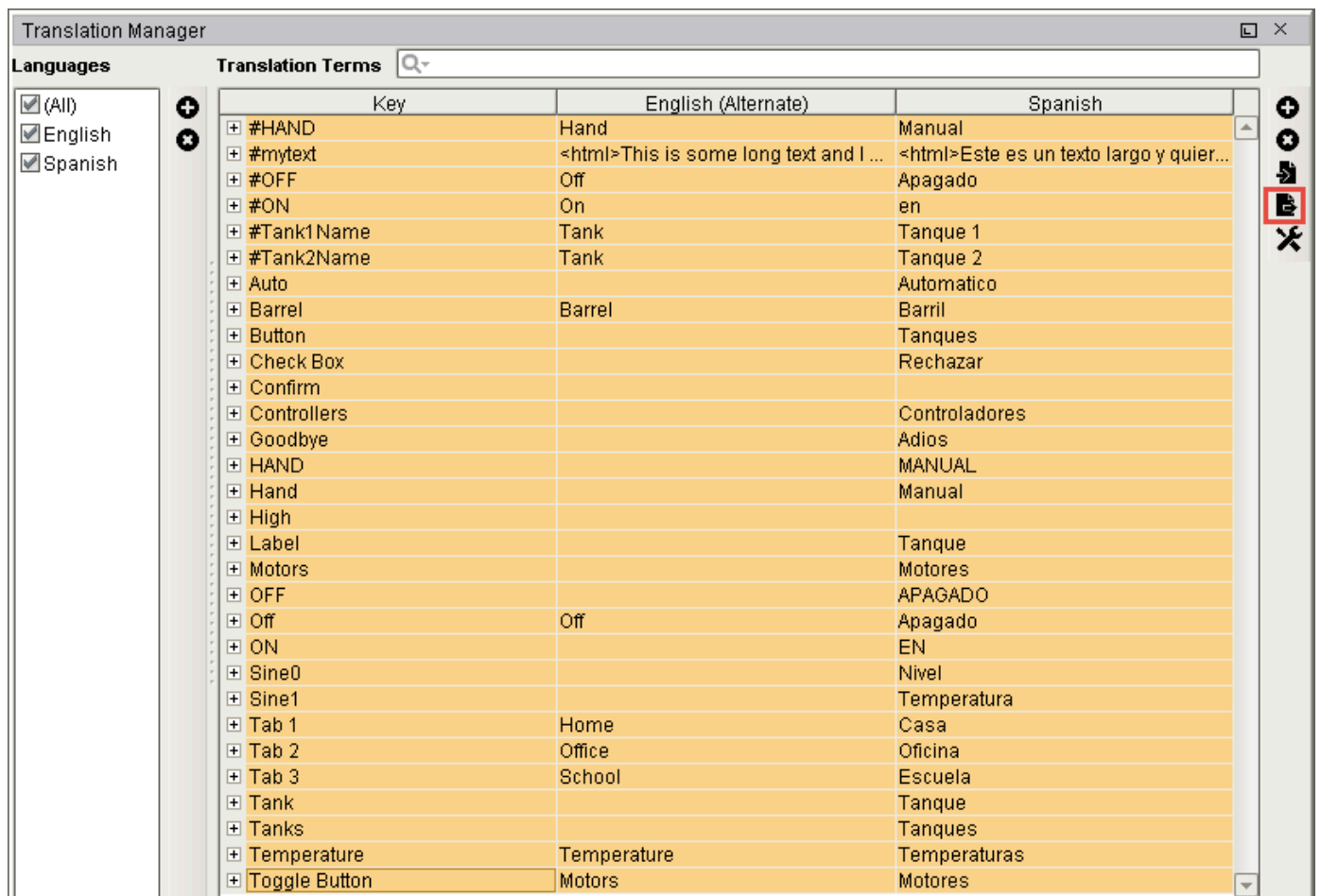
[Watch the video](#)

The ability to import and export terms and translations can be extremely important particularly to a large project. It allows you disseminate terms and translations rapidly, send them to a third party for translation, and keeps all related projects up-to-date.

To Export Terms from a Window

Let's export some terms from the Translation Manager. In this example, you can see all the terms that were added to the Translation Manager. You can export selected terms, or all terms from the Translation Manager.

1. Go to **Tools > Translation Manager**. Select all terms or only specific terms to export. Click on the **Export Terms** icon on the right side of the **Translation Manager** window.



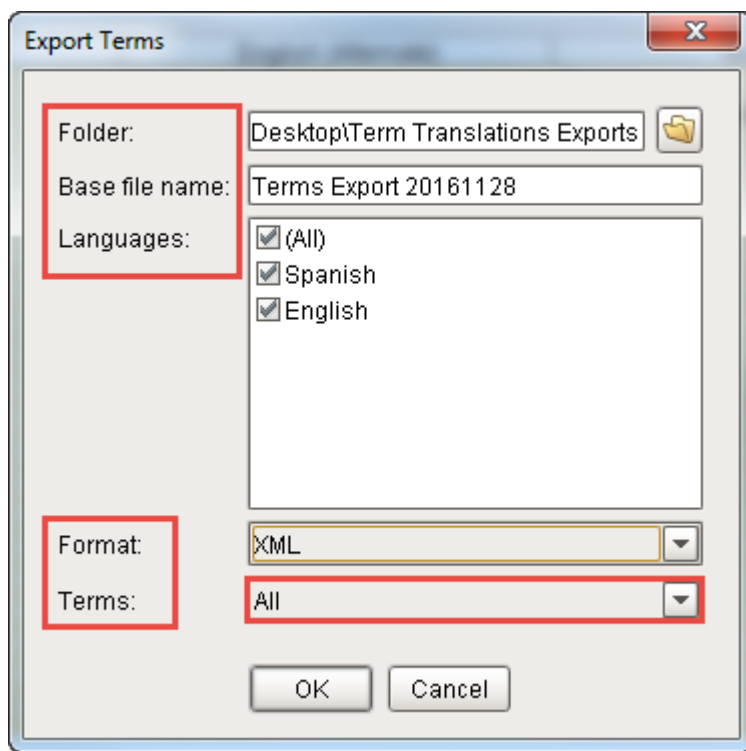
2. The Exported Terms window will open. Enter the following information:

- i. Select the **Folder** location where you want to put your exported terms.
- ii. Enter the **Base file name**. (File name of your exported terms).
- iii. Select the appropriate **Languages**. You can have multiple languages selected depending on the number of language translations you are using (i.e., English, Spanish, and French). This example exports two files: one for the English translation, and one for the Spanish translation.
- iv. From the **Format** dropdown list, select either **PROPERTIES** or **XML**. Both formats behave the same, the only difference is that the XML format supports UTF-8 encoding directly. A person can now translate the XML file directly and import it back into Ignition's Translation Manager.
- v. From the **Terms** dropdown list, choose either **Selected** or **All**.

i FOR SPECIFIC TERMS

If you want to select specific terms to export, choose '**Selected**,' otherwise, '**All**' terms will be exported from the dropdown list at the bottom of the window.

- i. Press **OK**.



3. Go to the folder location and check to see if your exported files are there. You will see **one exported file for each language**. Open the English exported file and verify that all the terms you intended to export are there.

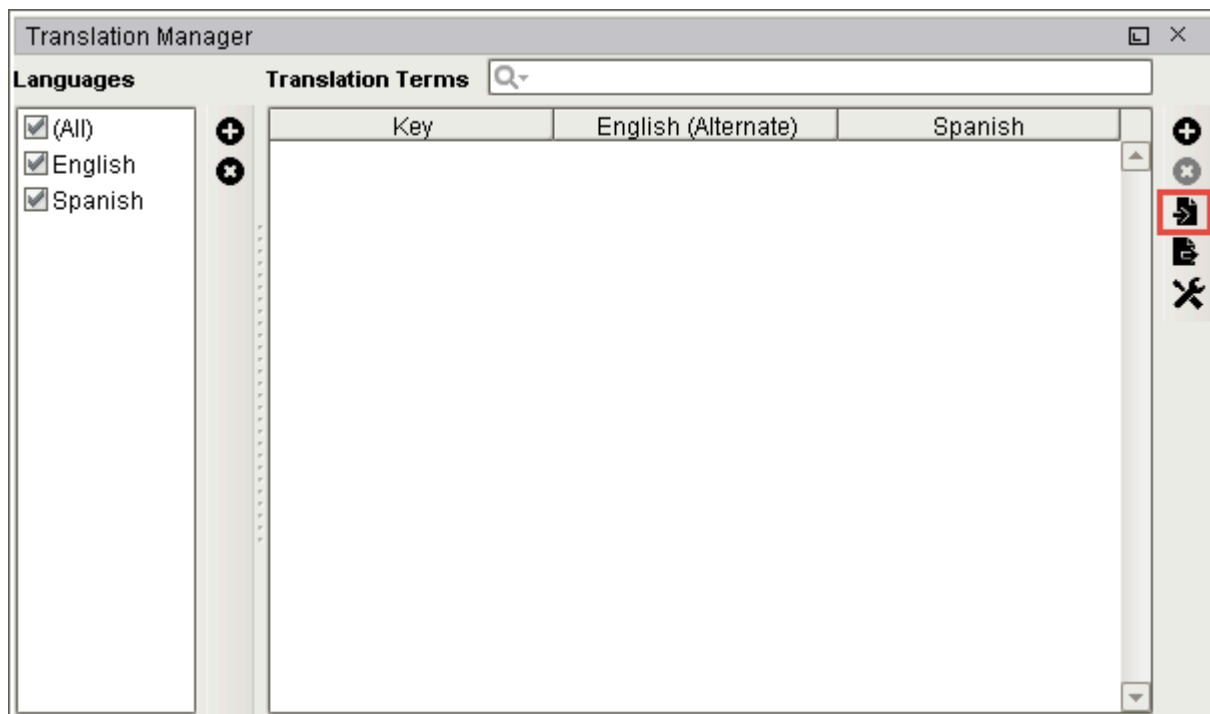
```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
- <properties>
    <comment>Locale: en</comment>
    <entry key="Sine1"/>
    <entry key="Temperature">Temperature</entry>
    <entry key="OFF"/>
    <entry key="Sine0"/>
    <entry key="Tanks"/>
    <entry key="#Tank1Name">Tank</entry>
    <entry key="Goodbye"/>
    <entry key="Toggle Button">Motors</entry>
    <entry key="Controllers"/>
    <entry key="#OFF">Off</entry>
    <entry key="Off">Off</entry>
    <entry key="Tab 3">School</entry>
    <entry key="#mytext"><html>This is some long text and I want it
        to wrap</html></entry>
    <entry key="Tab 2">Office</entry>
    <entry key="Tank"/>
    <entry key="Tab 1">Home</entry>
    <entry key="#HAND">Hand</entry>
    <entry key="Barrel">Barrel</entry>
    <entry key="Welcome"/>
    <entry key="Button"/>
    <entry key="Auto"/>
    <entry key="#ON">On</entry>
    <entry key="Hello"/>
    <entry key="Label"/>
    <entry key="High"/>
    <entry key="Hand"/>
    <entry key="Check Box"/>
    <entry key="#Tank2Name">Tank</entry>
    <entry key="HAND"/>
    <entry key="Motors"/>
    <entry key="Confirm"/>
</properties>

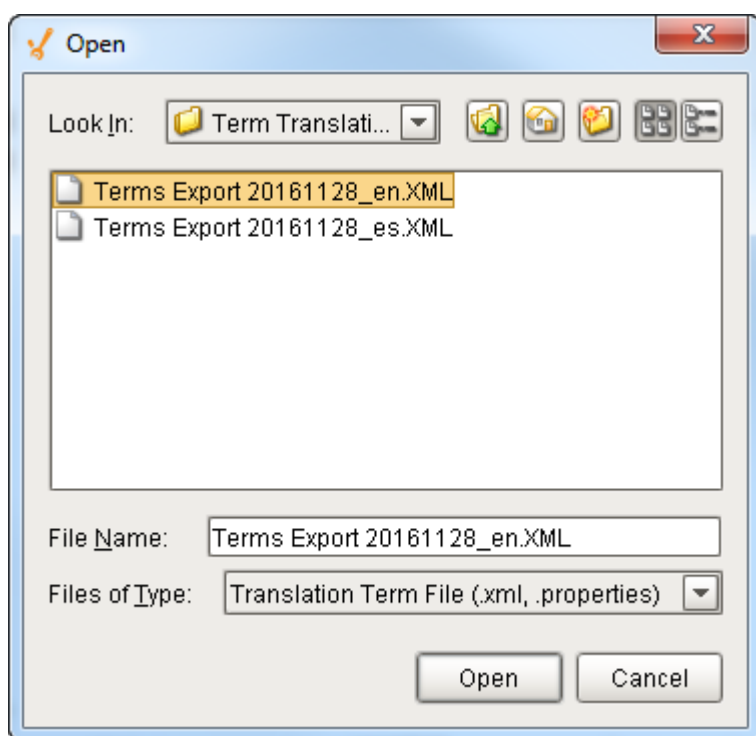
```

To Import Terms into Ignition's Translation Manager

1. Go to **Tools > Translation Manager** and select the **Import Terms** icon from the right side of the Translation Manager window.



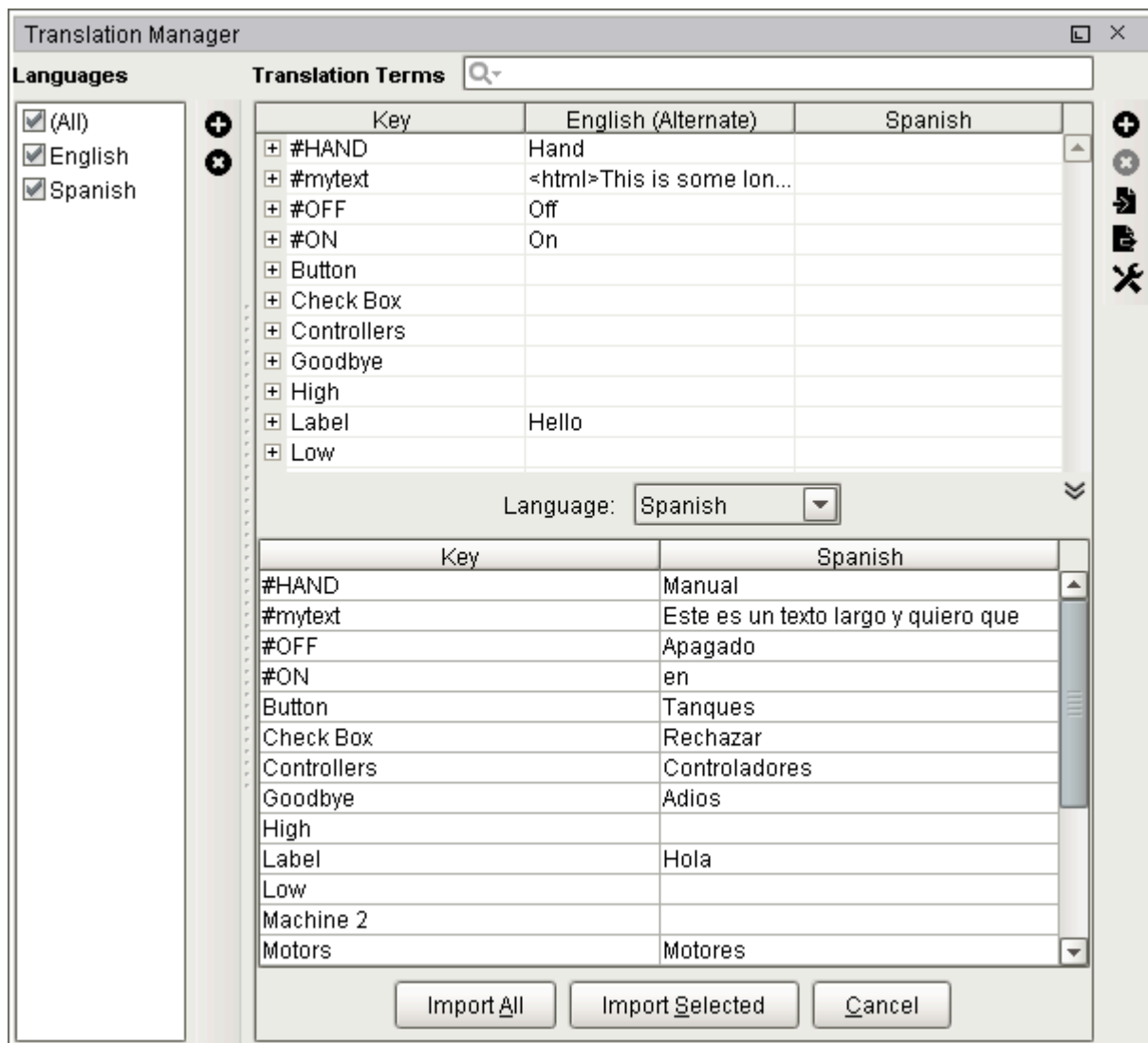
2. Navigate to the folder where your exported files reside, and choose the English file from the **Open** window, and click **Open**.



3. You have the option of importing selected terms or all terms. Click either **Import All** or **Import Selected** depending on what terms you want to import.
Repeat this step to import each translation file. This example imports the Spanish translation file.

IMPORTED TERMS OVERWRITE EXISTING TERMS

Be aware that existing terms will be overwritten by the imported term.



4. Check to see that all your terms were imported into the Translation Manager by going to **Tools > Translation Manager**.

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

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

Switching the Current Language



INDUCTIVE
UNIVERSITY

Switching the Current
Language

[Watch the video](#)

Language Selector

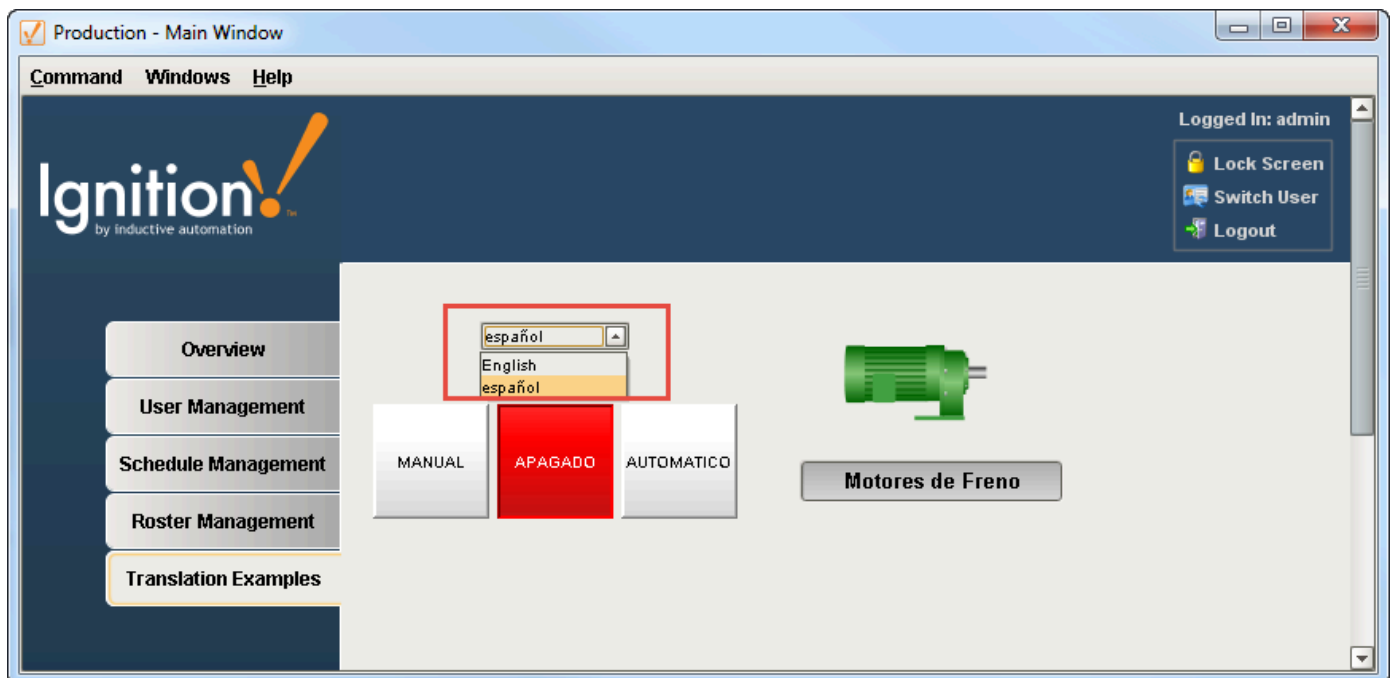
A single Language Selector on a window has the potential to trigger translations on all windows. This way a single Language Selector component can exist on a navigation window and provide language translations for all components on all windows.

Selecting a Language in the Client

Once you have the Language Selector component setup, it's super easy to select and switch between languages in the Client. There is no binding involved in selecting a language because they are compared directly against the Translation Manager database. All matching terms are automatically translated.

There are three simple ways to select a preferred language once the translation terms are created:

1. Add a Language Selector on your window in the Designer. After you Save and Publish your project, open the Client, and the Language Selector will be visible so you can switch between languages on the fly.



2. 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.



3. 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. No Language Selector component is

necessary. This is especially cool if your company is using Active Directory which automatically logs you in and opens the Client in your preferred language.

4. You can also set the client language with the `system.util.setLocale()` function. This function takes an ISO 639-1 two-letter code for the language to use (for example, "en", "fr", "de", "es", and so on).

Expression and Scripting Functions

You can look up translations using the following functions:

- Expression Function

`translate()`

- Scripting Functions

`system.util.translate()`

`system.util.modifyTranslation()`

Translations are matched by looking for the base language value in the translation database. This is especially useful for message boxes and other warnings or errors that you show in your scripts.

System Considerations During Translation

Inherent operating system and Java configurations may affect Ignition's ability to provide a complete translation in certain circumstances.

For example, suppose a user has selected the Spanish option from their Language Selector component. An English to Spanish translation term exist for the word "Information" as well as "Start." The message box script takes two parameters: one for the content while the other parameter is optional. If the second parameter is not included in the function then the resulting title of the message box will include the English word "Information." The following code is executed on a button clicked event handler.

```
message = system.util.translate("Start")
system.gui.messageBox(message)
```

What results is a message box with a translated content while the title remains in English. In addition, the accept button is translated as well even though the English word for "OK" does not exist in the Translation manager. This is due to the system level translations that exist for inherent language support.

A similar occurrence exist with the file open dialog window. The following code is executed on a button clicked event handler.

```
system.file.openFile()
```

The result is a translation for the title as well as the open and cancel buttons while the references to the "File Name" and the "Files of Type" remain in the operating system's language and thus are beyond the scope of Ignition's ability to translate.

Similar occurrences will appear in print and error dialog boxes.

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

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

Localization Best Practices

Best Practices

Before you begin using multiple languages in your projects, we thought we would start you off with a few best practices for using project terms and translations.

As you already know, there are global level translations which are available to all components and text elements in both the Translatable Terms Panel and the global database, and component level translations that are only specific to the component and only available in the Translatable Terms Panel. It's good to understand a little about their behavior before investing a lot of manpower setting up your translations only to find out down the road that you might set them up differently after you become more familiar with them.

Global vs Component

We recommend using global terms over component level terms because this way you will only have to add your term once since global terms are shared across all windows and projects. This will save you a little time from having to add the same term again if you discover that a component level term should extend beyond the specific component and shared across all projects. The only time you would want to use a component level term, is if you want it to be different from the global level term. This ensures that the component level term will override the global term.

WHERE THE COMPONENT LEVEL TERM TAKES PRECEDENCE

The component level term always takes precedence when there is a matching global term.

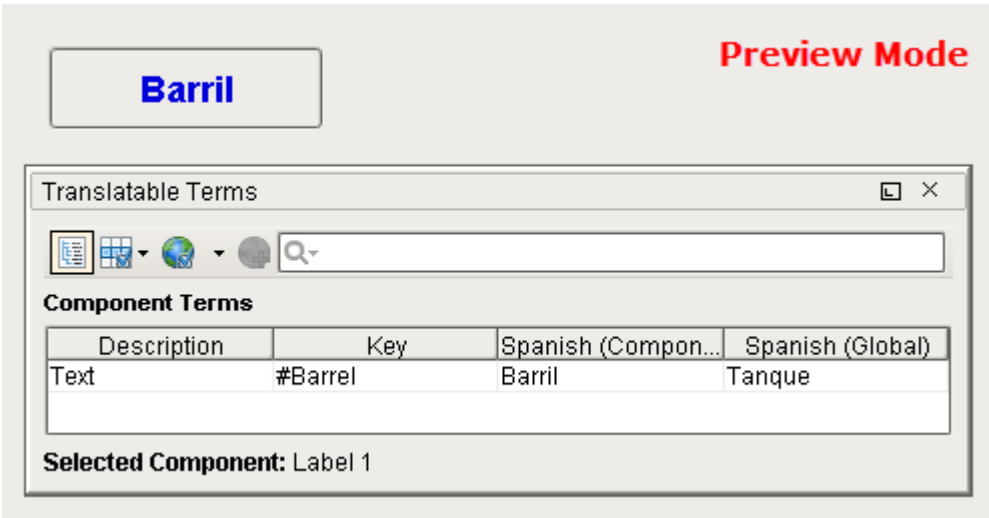
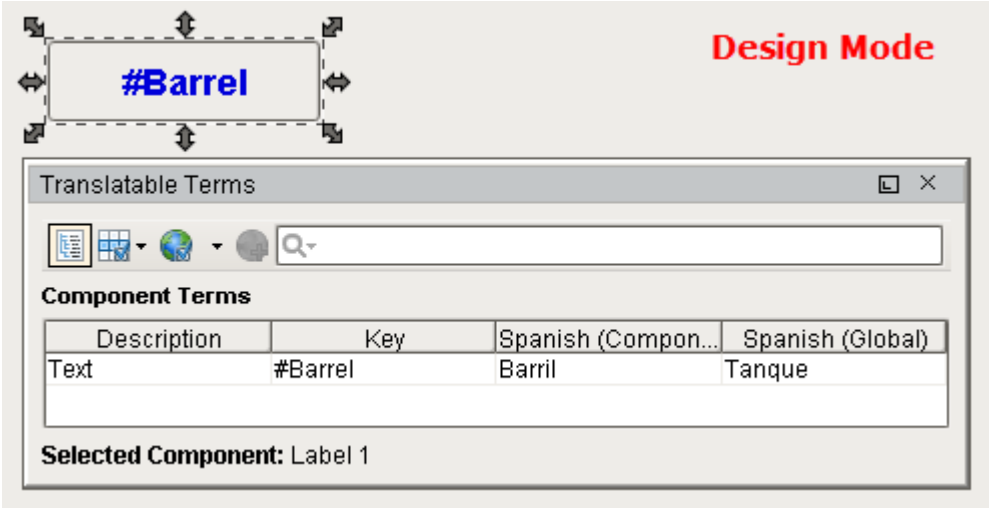
Using Codes

It's a good idea to use codes for the Key field of your Terms for any descriptions that you want to translate.(i.e: #introduction or #welcome_screen_info) so your global term won't ever accidentally translate on another window or component. (The '#' is just for convention, and is not a special character). Using codes works particularly well for long text strings such as introduction paragraphs.

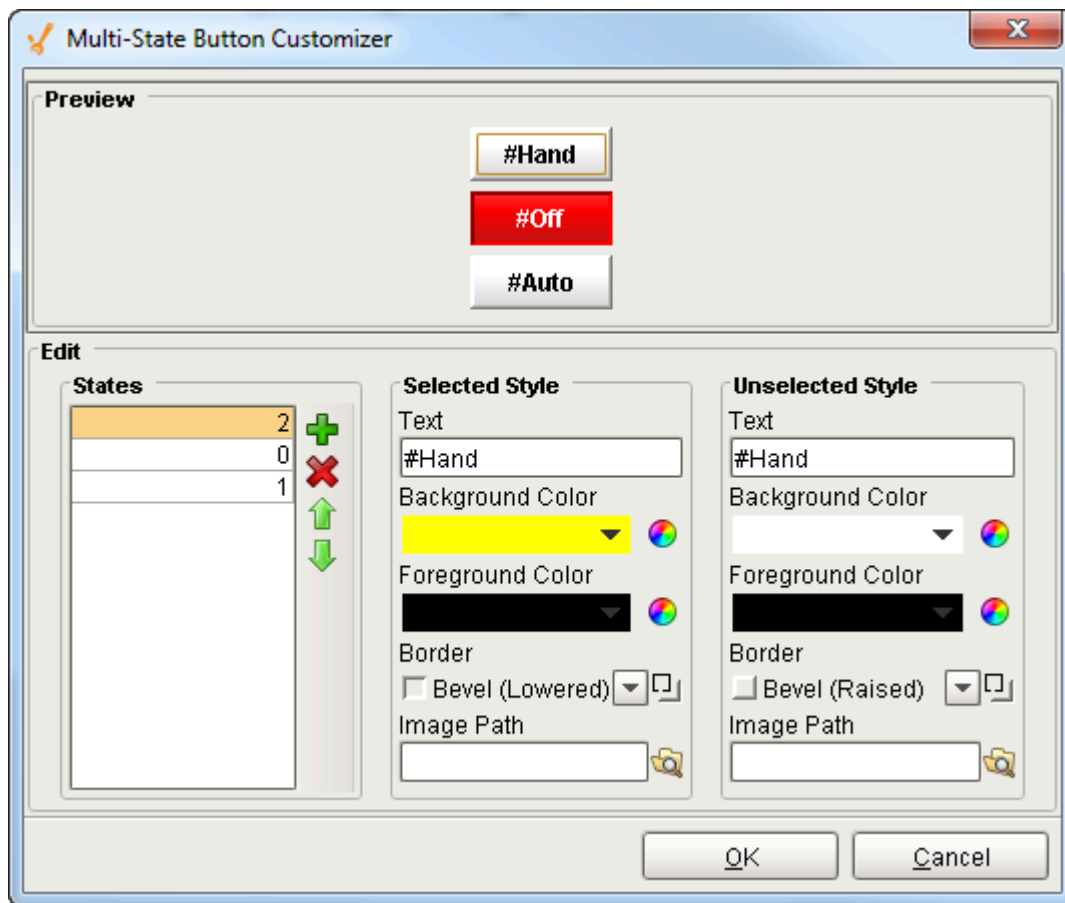
One thing be very careful of is if you edit the codes in a minor way, it could potentially break the translation.

Here are a couple of examples:

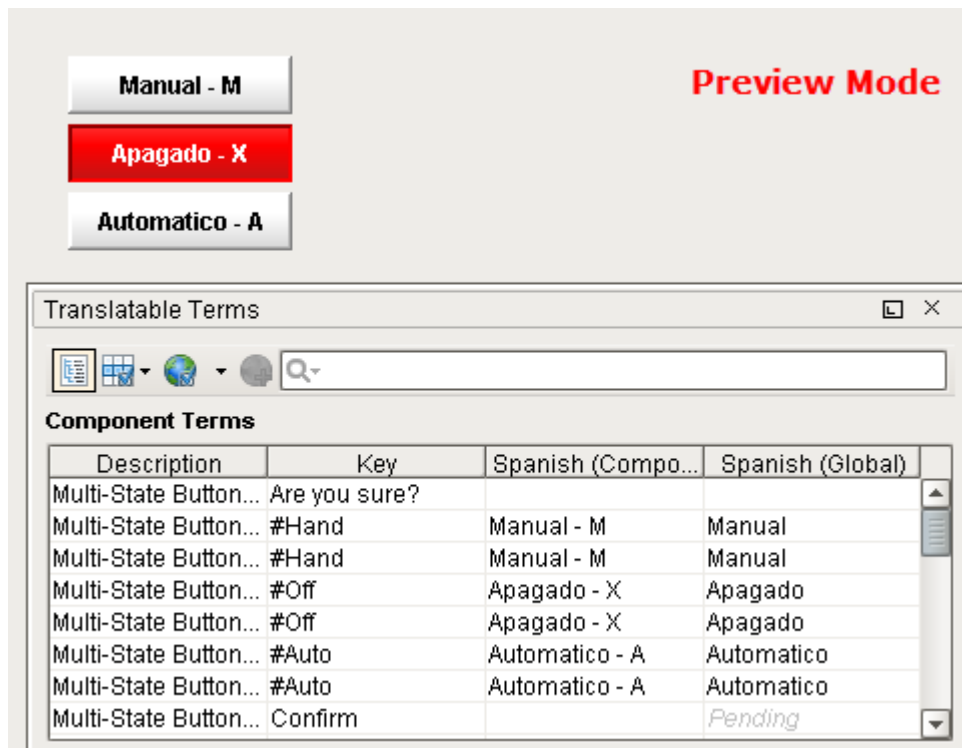
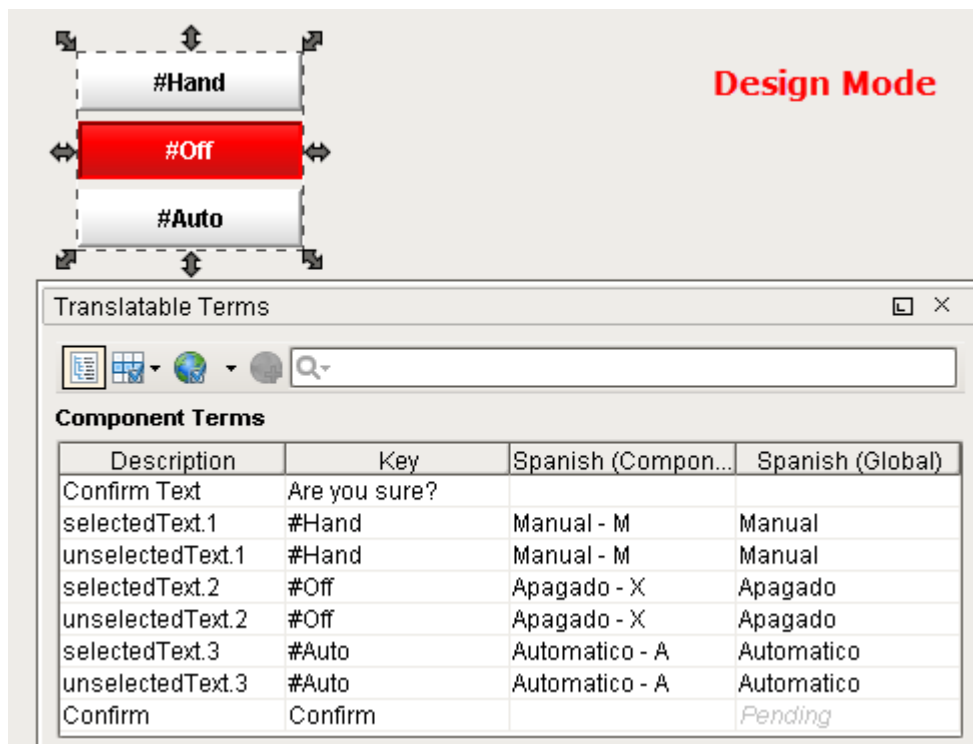
The first example is of a Label component with a Key field of **"#Barrel."** You'll notice the Label has global translation of **"Tanque"** and a component level translation of **"Barril."** By using the **"#,"** the component level translation of **"Barril"** was not overwritten by the global translation of **"Tanque."** Here is what the translation looks like in Design Mode and Preview Mode:



The **"#"** can also be used on components for text that you don't want accidentally translated. This example uses the Multi-State Button component. The only way to change the states / words on the buttons (i.e., Hand, Off, Auto) is to use the Multi-State Button Customizer. Add the **"#"** for each of the three Keys so they don't get translated by global terms that have already been set up.



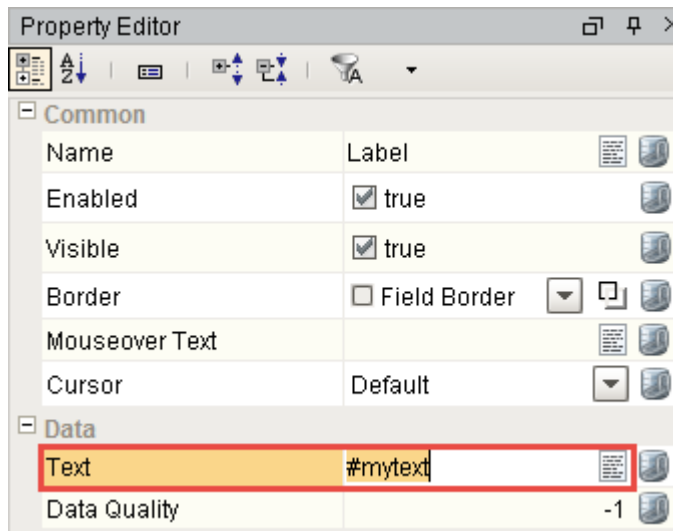
Here's what the translations look like in **Design Mode** and **Preview Mode**:



Using HTML for Text Wrapping

Another good practice is to use HTML to wrap text within your component's buttons, labels, etc.,. You can use it with translating terms that contain lots of text. HTML is just a good way to wrap text to fit within a label and button components.

1. In the **Property Editor** of a Label component, go to the **Text** property and give the Text a unique name (i.e., #Mytext).



2. Right click on the **Label** component and click on **Translations**. Double click on the row for Mytext. Enter the text as shown below for both the **Spanish Global** and **English Global translations**. Make sure you precede your text with `<HTML>`. Click the **Back** link.

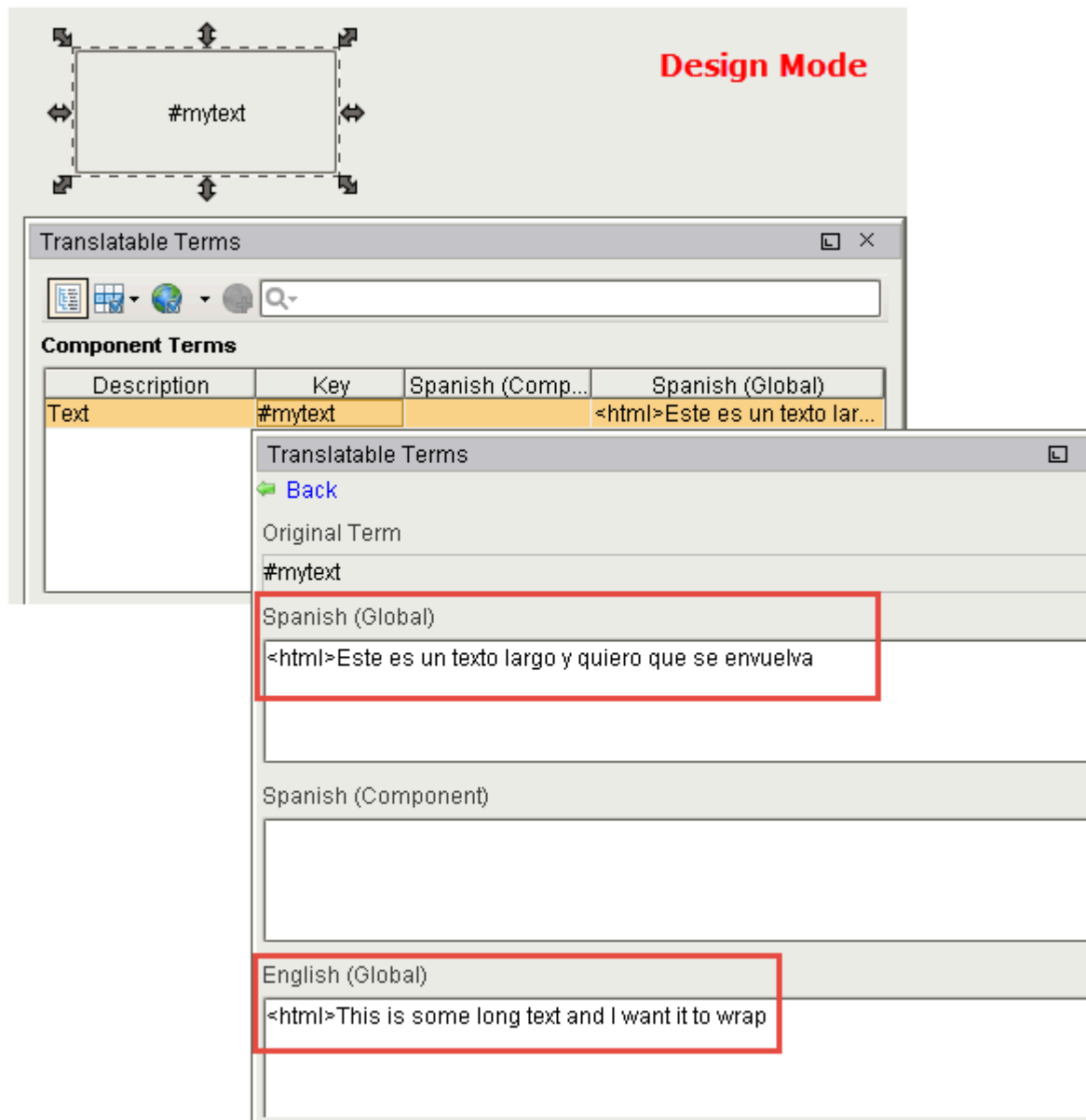
Mytext

#English

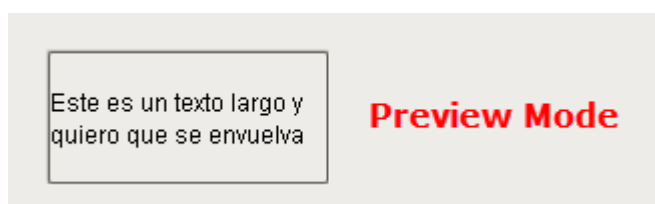
`<html>`This is some long text and I want to wrap it

#Spanish

`<html>`Este es un texto largo y quiero que se envuelva



3. Once in **Preview Mode**, your long text will be wrapped!



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

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

Translating Built-in Terms

Overview

There are several built-in terms used by Ignition that are translatable. This page contains the Keys that can be used to translate these terms.

Built-in Access Denied



When Security Settings are configured on a window, the only restriction is **Do Not Open**, which prevents the window from opening based on User Roles. The text in this message box is not normally configurable, but can be altered via translations.

The line below should be used as the Key.

Access Violation - Key

```
<HTML><P><font color='red'>ACCESS VIOLATION.</font><P>You do not have the  
privileges to open window '%s'
```

Here we have a modified version of the text that should appear. In this example, we're using **English (Alternate)** as the language. The name of the window is passed into the message box via the '%s' substring.

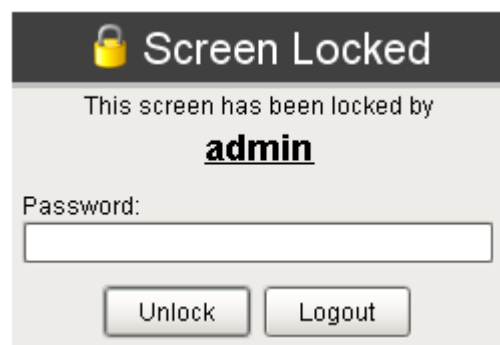
Access Violation - Translated Term

```
<HTML><P><font color='red'>ACCESS VIOLATION.</font><P>Sorry! You're not allowed to  
access '%s' <br> <b>So sorry!<b/>
```

Now, our message box appears as the following:



Screen Locked Message



The built-in `system.security.lockScreen` function can be used to lock the screen. In some cases, you may wish to change the text on the Screen Locked window.

Screen Locked - Key

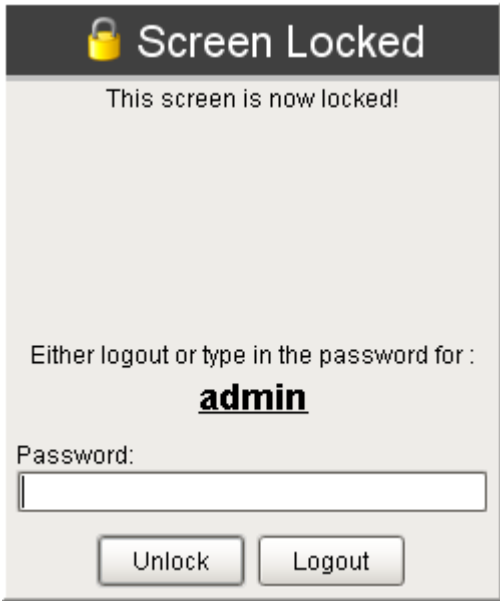
```
<HTML><CENTER>This screen has been locked by<BR><font size=+1><b><u>%s</u></b>
```

We can use the following key to translate the message on the Screen Locked window. The '%s' substring will reference the username of the user currently logged in. In this example, we added a few line breaks, which will make the Screen Locked window appear taller.

```
Screen Locked - Translated Term

<HTML><CENTER>This screen is now locked! <br><br><br><br><br><br><br>Either
logout or type in the password for :<BR><font size=+1><b><u>%s</u></b>
```

Now, the message on the Screen Locked window appears as follows:



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

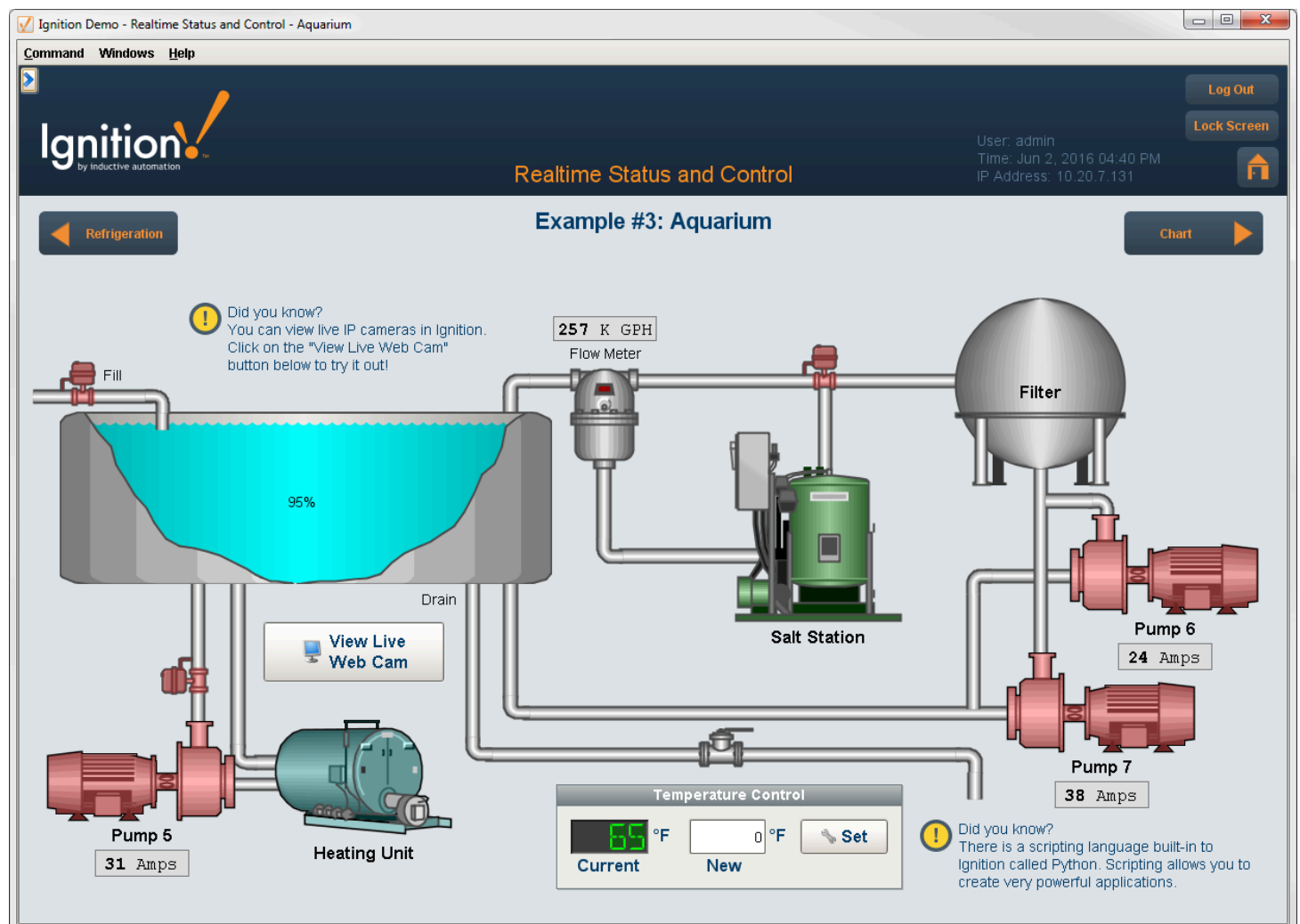
For documentation on the latest stable release, see the [latest version](#) (8.3).

Windows

Using Windows

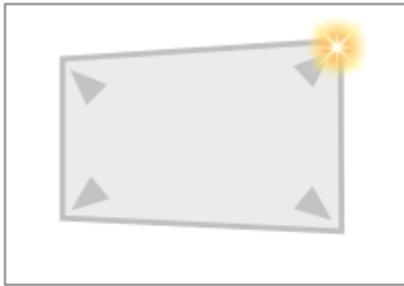
Windows are the gateway to your HMI/SCADA application. A project is a collection of windows which are the basic building blocks for all your screens, and a hierarchy of components that users interact with. When you publish your project, these windows are loaded into the Vision Client where any number of windows can be opened at one time. Your windows are brought to life through the property bindings and event handlers on your components. With the power of windows, they can be designed to be very sophisticated and complex, displaying lots of information, and yet very easy to use. 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**.

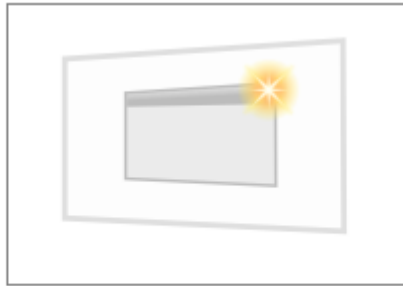


Working with Windows

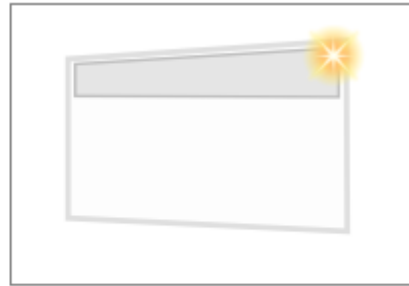
Working with Windows is super easy! Windows are the top-level unit of a design for your project. There are three different window types: **Main**, **Popup**, and **Docked**. Each **window type** behaves differently as you can see here, and can be arranged any way you like.



+ New Main Window



+ New Popup Window

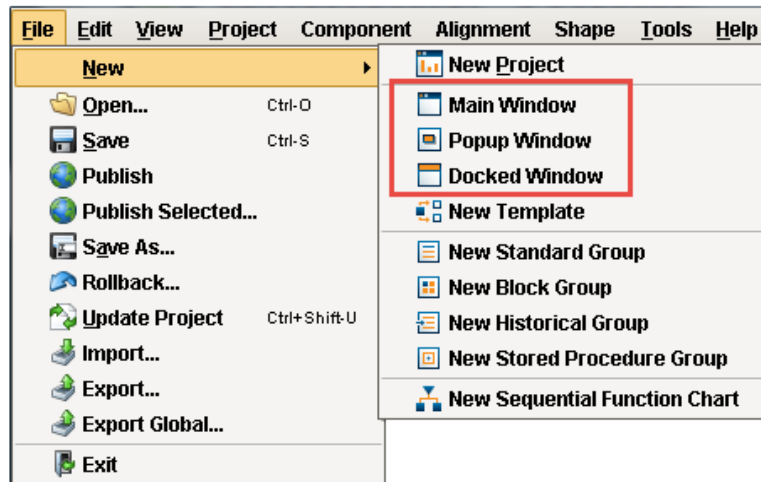


+ New Docked Window

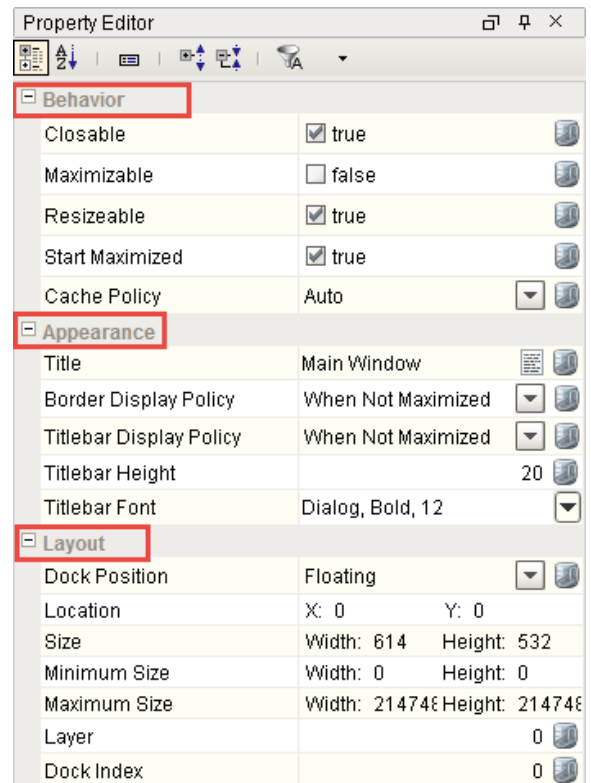
You can change a **window's properties** to behave differently, transforming the window into various configurations. You can create and re-use windows inside of Ignition. **Passing custom parameters** into windows allows you to create it once and re-use your screens. You can set **windows to open once**, or **create popup windows with multiple instances** so users can compare live values.

To **create a window**, all you need to do is select a window type, drag some tags to your window, and select the components you want to use. Next, change the default **window properties** to define the look and feel of the window, and set the behavior of the window. You can change window properties in the Property Editor as shown below or from the Project Browser. You can also organize your window into folders, rename them, add notes about your windows for others to see, import and export windows, and add security on your windows. To learn more, go to **Working with Vision Windows**.

New Window - Menubar



Window Property Editor



Root Container

Inside a window there is always a **Root Container**. The Root Container is where you place all components in the window. This is a normal **Container component** except that it cannot be deleted or resized, and is always set to fill the entire window.

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.

For more information on Titlebars, Borders and Root Container refer to [Window Properties](#).

Navigation Strategy

Setting up a navigation strategy allows you to navigate between different windows in the runtime Client. To help you select the right strategy that fits your project structure, the [Navigation Windows](#)

section provides a description of each navigation strategy and things to consider for determining the best navigation strategy for your facility. You can select from any of the following types of runtime navigational strategies when designing your Ignition project.

- [Tab Strip](#)
- [2 Tier](#)
- [Tree View](#)
- [Back and Forward Buttons](#)
- [Drill Down](#)

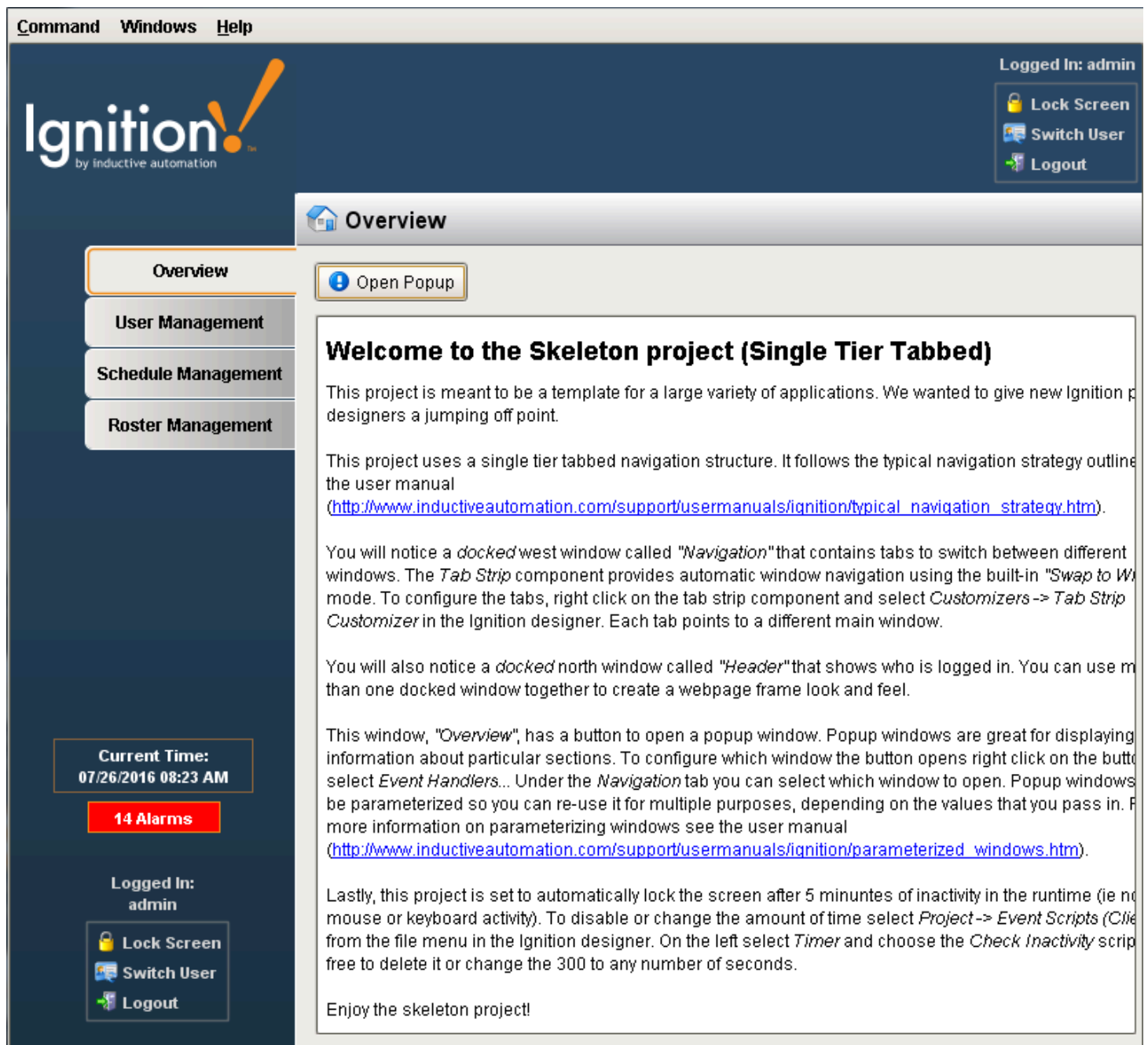
A typical navigation strategy for a Vision project is as follows:

- 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 "screen" window, and will automatically swap from the current screen to the desired screen.

Furthermore, 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 Public Demonstration](#) that you can download from our website.

To learn more, refer to [Navigation Window](#).



Popup Windows

Popup windows are often opened by components in a main window and are meant to be on top of the screen. You can make a popup window as big or small as you want. It is useful to create a popup window for displaying additional information about a selected item on the screen. Windows can act as **popups or drill down** to show more information about anything you want.

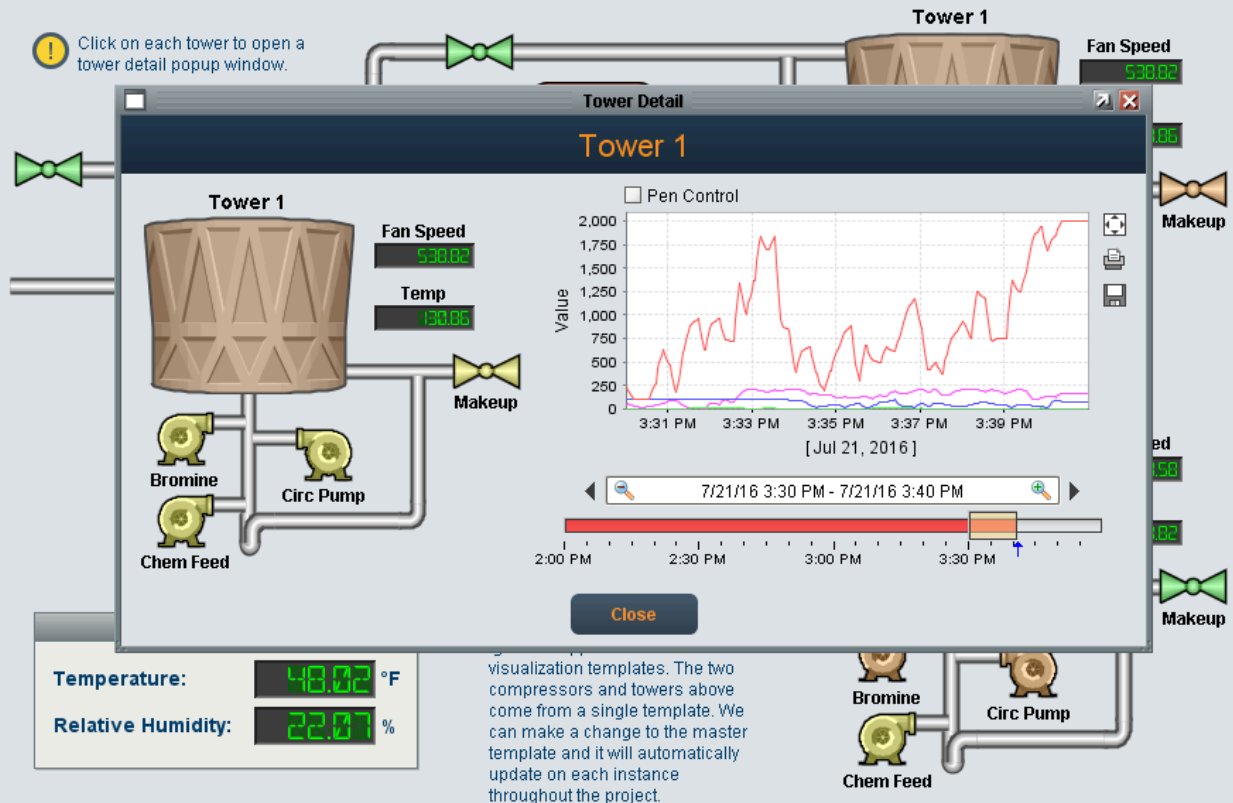


Dairy Plant

Example #2: Refrigeration

Aquarium

Click on each tower to open a tower detail popup window.



The great thing about all windows is that they can be parameterized so they can be re-used. For example, a parameterized popup window can display graphical historical information related to the Tags of your components, like a Tank. One popup window design can be re-used for many tanks as long as the proper information is passed to the popup window. Once you define a Custom Property on your popup window, your window can use it to show values based on whatever is in that property. Refer to [Parameterized Popup Window](#) for more information.

Overview

User Management

Schedule Management

Roster Management

Tank 100

Tank Details

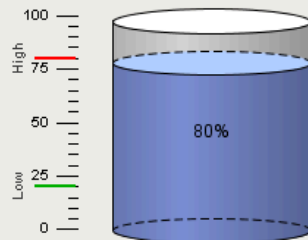
13 Active Alarms

Logged In:
admin

Lock Screen
 Switch User
 Logout

Popup

Tank 100

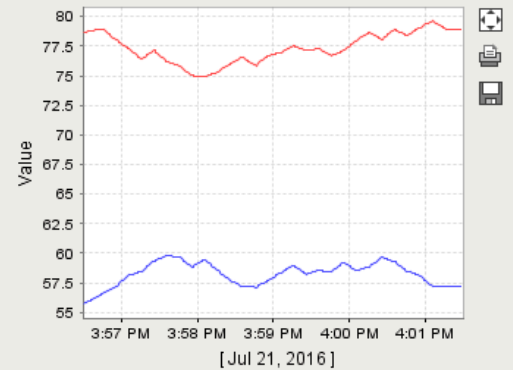


Temperature:

68.43

Pens

☒ level_PV
☒ temp_PV



[Jul 21, 2016]

Apply

Last: 5 Minutes

Overview

User Management

Schedule Management

Roster Management

Motor Detail

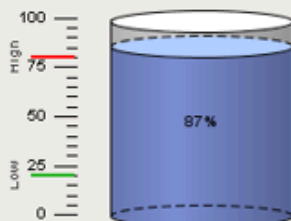
Tank Details

14 Active Alarms

Logged In:
admin

Lock Screen
 Switch User
 Logout

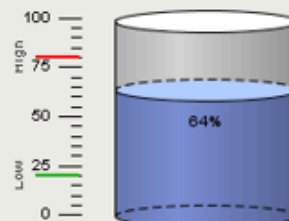
Tank 100



Temperature:

66.22

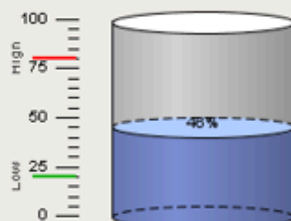
Tank 101



Temperature:

82.58

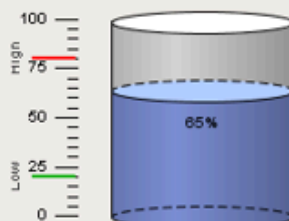
Tank 102



Temperature:

52.14

Tank 103



Temperature:

39.97

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

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

Working with Vision Windows

Working with windows is easy. Windows are the top-level design units for Vision projects and come in different types such as **Main**, **Popup**, and **Docked** windows. All you need to do is select a window from the window types, drag some Tags to your window, and select the components you want to use. If you prefer, you can even drag individual components to a window, and then select your Tags. You can organize your window into folders, rename them, add notes about your windows for others to see, import and export windows, and add security on your windows.

Windows are where your projects get their start.

Window Types

Ignition provides three window types and each behaves a little differently.

1. **Main** window - acts like a typical HMI screen taking up all available space.
2. **Popup** window - are often opened by components in a main window and appear on top of a main window.
3. **Docked** window - sticks to one side of the screen and nothing can overlap it.

These three windows types are described in detail in [Window Types](#).

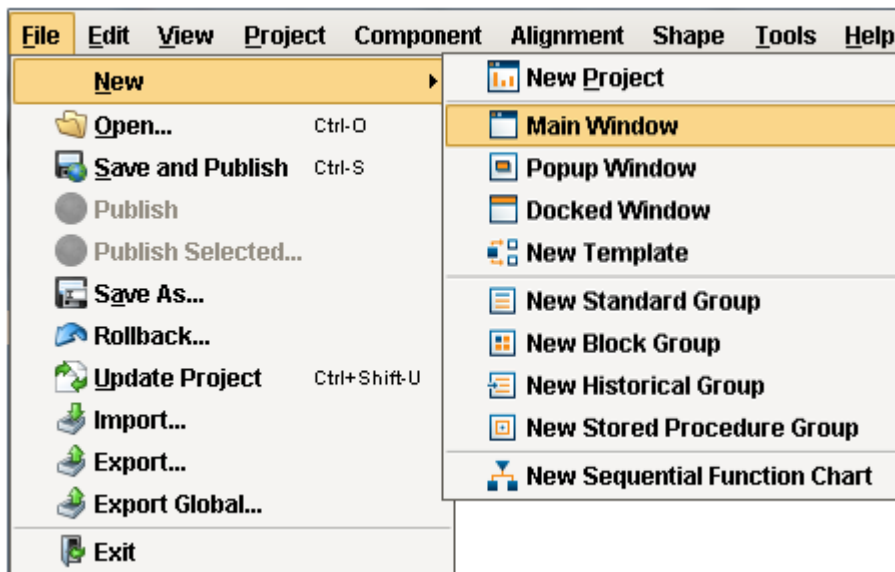
To Create a Window

Creating windows is as easy. There are three ways to create a new window and each way is shown below.

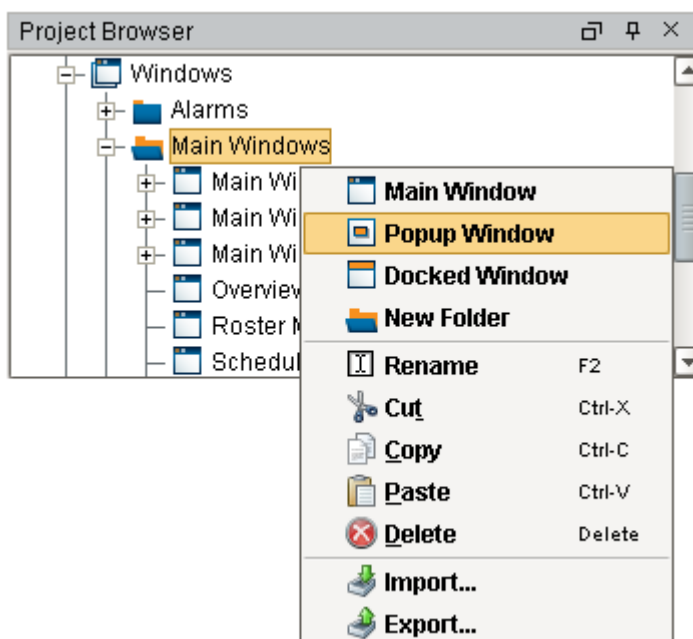
1. When you open your project, the three window types are displayed. Click on the **New Main Window**, or any window type you want to create.



2. Go to your menubar at the top of the Designer and select **File > New > Main Window** or any other of the window types you want to create.



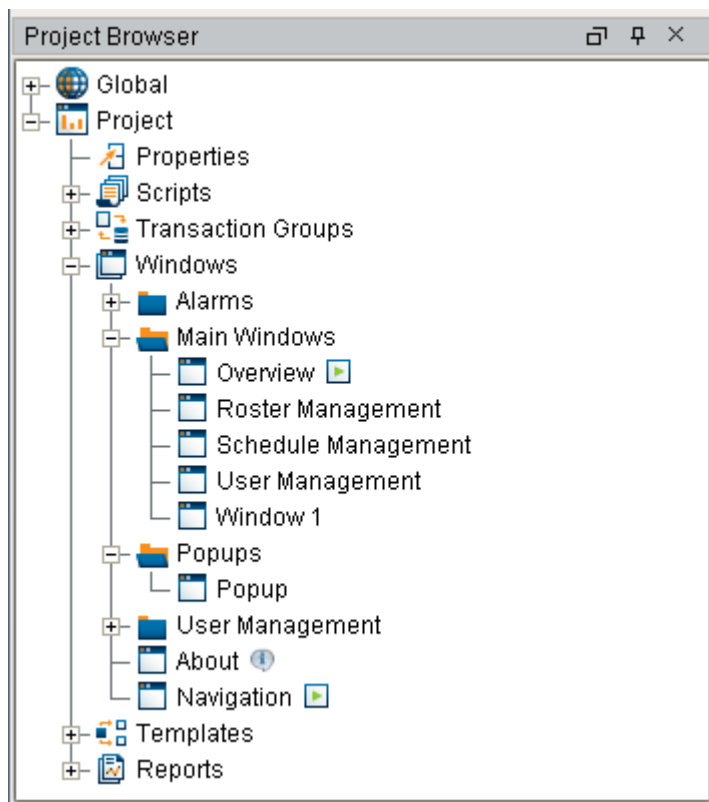
3. In the **Project Browser**, click **Main Windows** and select any of the window types. This example creates a Popup Window.



That's it! You now have a new window. To see it in action, all you have to do is add components, drawing tools, and tags to your window. Once your components and Tags are added, save and publish your project. Open your project in the client to see how your components look in the window.

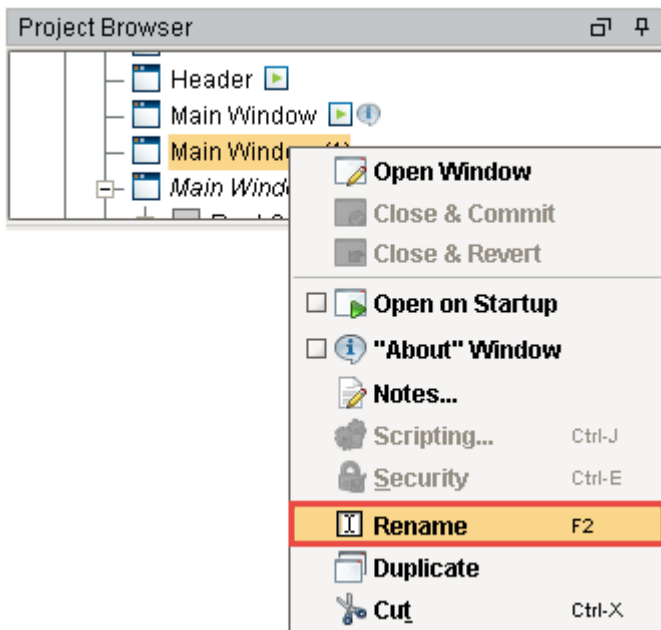
Organizing Windows

You can also 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. A window's name and folder path is very important - it will be how other windows reference it. You can create as many folders as you want, and nest them as deep as you need for your project. The built-in **Project Templates** use this structure:



Naming and Renaming a Window

In the **Project Browser**, right-click on the window and choose **Rename** or press **F2**. Enter a new name for the window and press **<CR>**. The window name must be unique in its folder. A descriptive window names is always recommended.

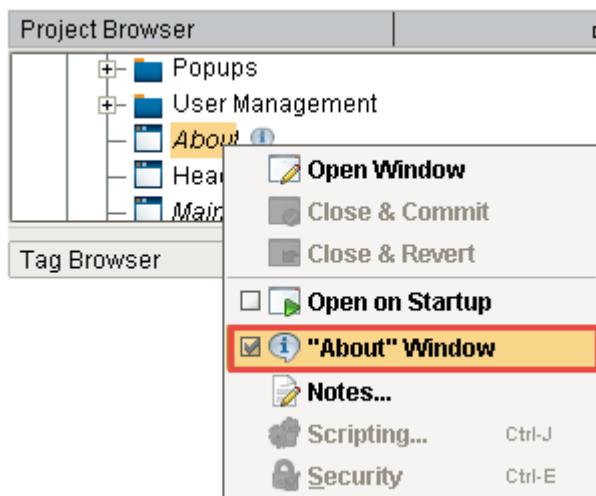


Special Window Properties

About Window

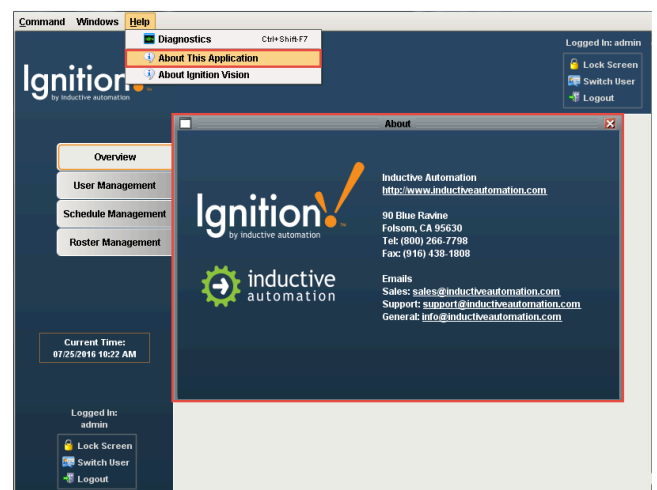
About Window - Designer

In the Project Browser, expand the **Windows** folder. Right click on **About** and select the check box next to **"About" Window** to make the window accessible from the **Help > About This Application** menu in the Client.

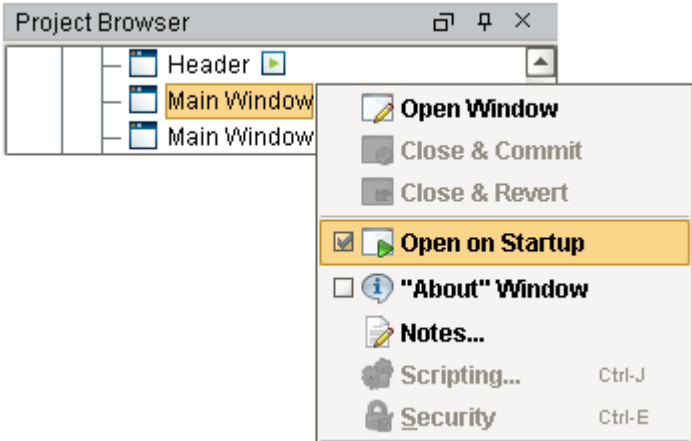
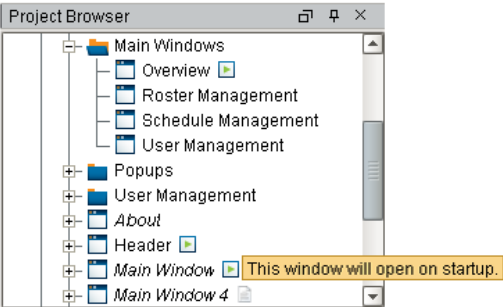


About Window - Vision Client

In this Project Template example, the **About Window** provides some contact information for Inductive Automation.

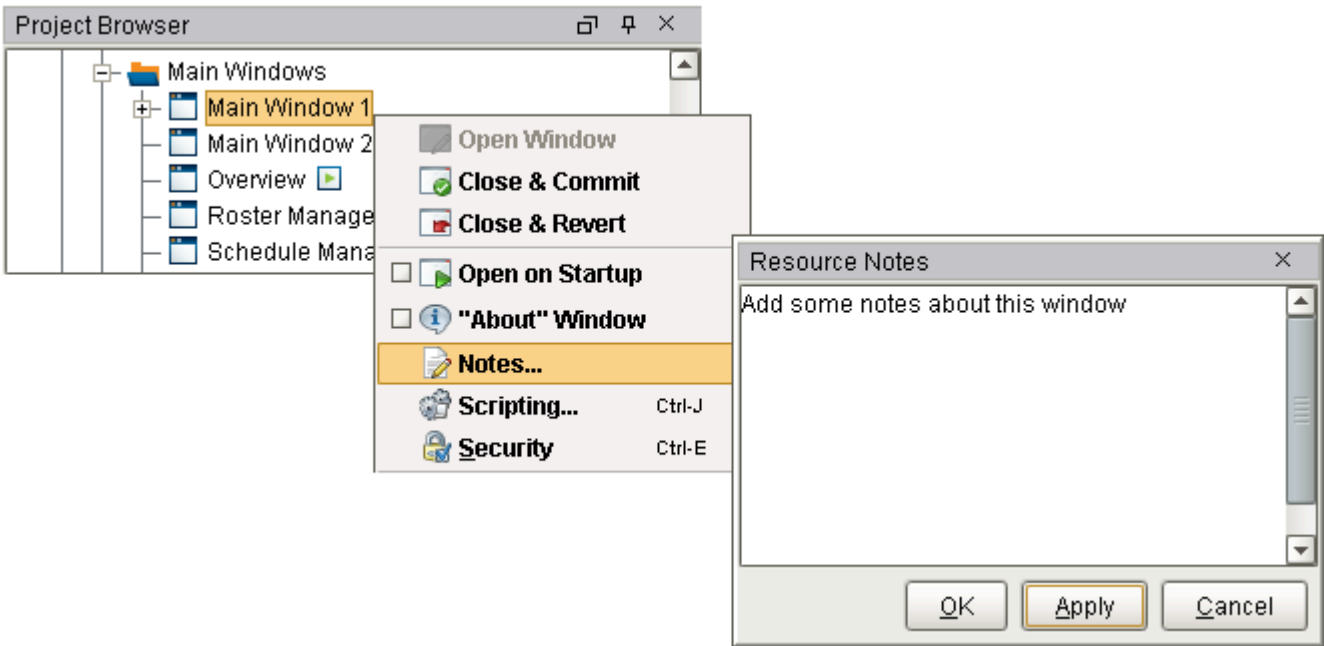


Open on Startup

Open on Startup - Context Menu	Open on Startup - Green Arrow
By default, most projects open a static set of windows automatically when the client is opened. You can select which windows to open on startup by going to the Designer, right click on a window, and choose Open on Startup.  A screenshot of the 'Project Browser' window. A right-click context menu is open over the 'Main Window' item. The menu options are: 'Open Window' (with a pencil icon), 'Close & Commit' (with a floppy disk icon), 'Close & Revert' (with a trash can icon), 'Open on Startup' (checked, with a green arrow icon), '"About" Window' (with an information icon), 'Notes...' (with a notepad icon), 'Scripting...' (with a gear icon and 'Ctrl-J'), and 'Security' (with a lock icon and 'Ctrl-E').	All windows that Open on Startup have a little box with a green right arrow next to the window name.  A screenshot of the 'Project Browser' window showing a tree structure of windows. The 'Main Windows' folder is expanded, showing 'Overview', 'Roster Management', 'Schedule Management', and 'User Management'. Below it are 'Popups', 'User Management', 'About', 'Header', 'Main Window', and 'Main Window 4'. A yellow tooltip points to the 'Main Window' item, stating 'This window will open on startup.' The 'Main Window' and 'Main Window 4' items have a small green arrow icon next to their names.

Adding Notes on Windows

Through the right-click menu on a window in the **Project Browser**, you can access the window's **Notes**. This free-form text field is provided to let you document the purpose of the window and/or any technical information about how the window works.

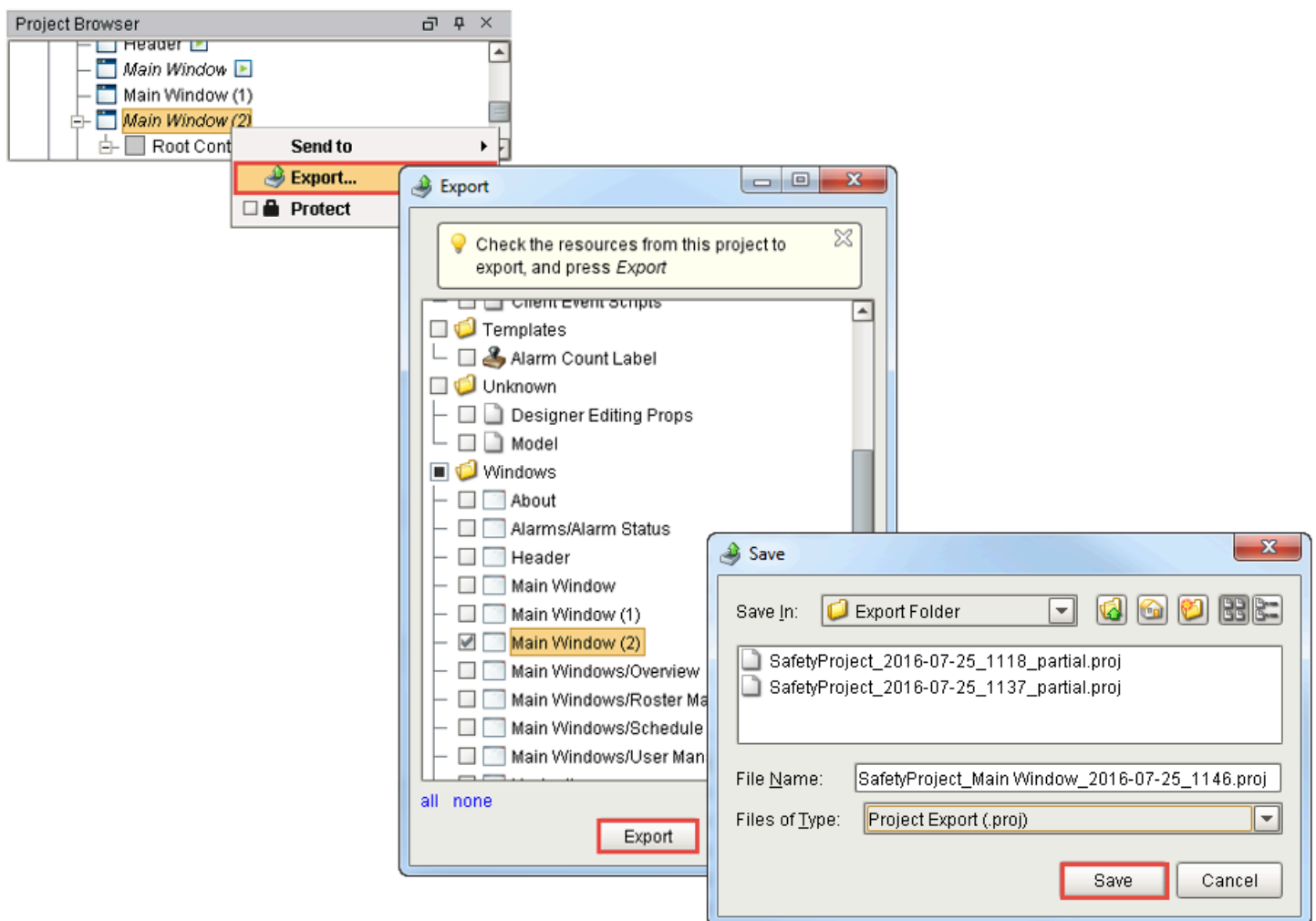


Exporting and Importing Windows

In the Designer, you can export and import windows from one project to another project using external files. You can even use the Export function to do a **backup** of your window(s).

Exporting a Window

1. In the **Project Browser**, select a window folder, right click, and select **Export**. The Export wizard will open.
2. Check the windows and the resources that you would like to export.
3. Click **Export** and choose a folder path for your backup file. Ignition will provide a default filename with the file extension ".proj".

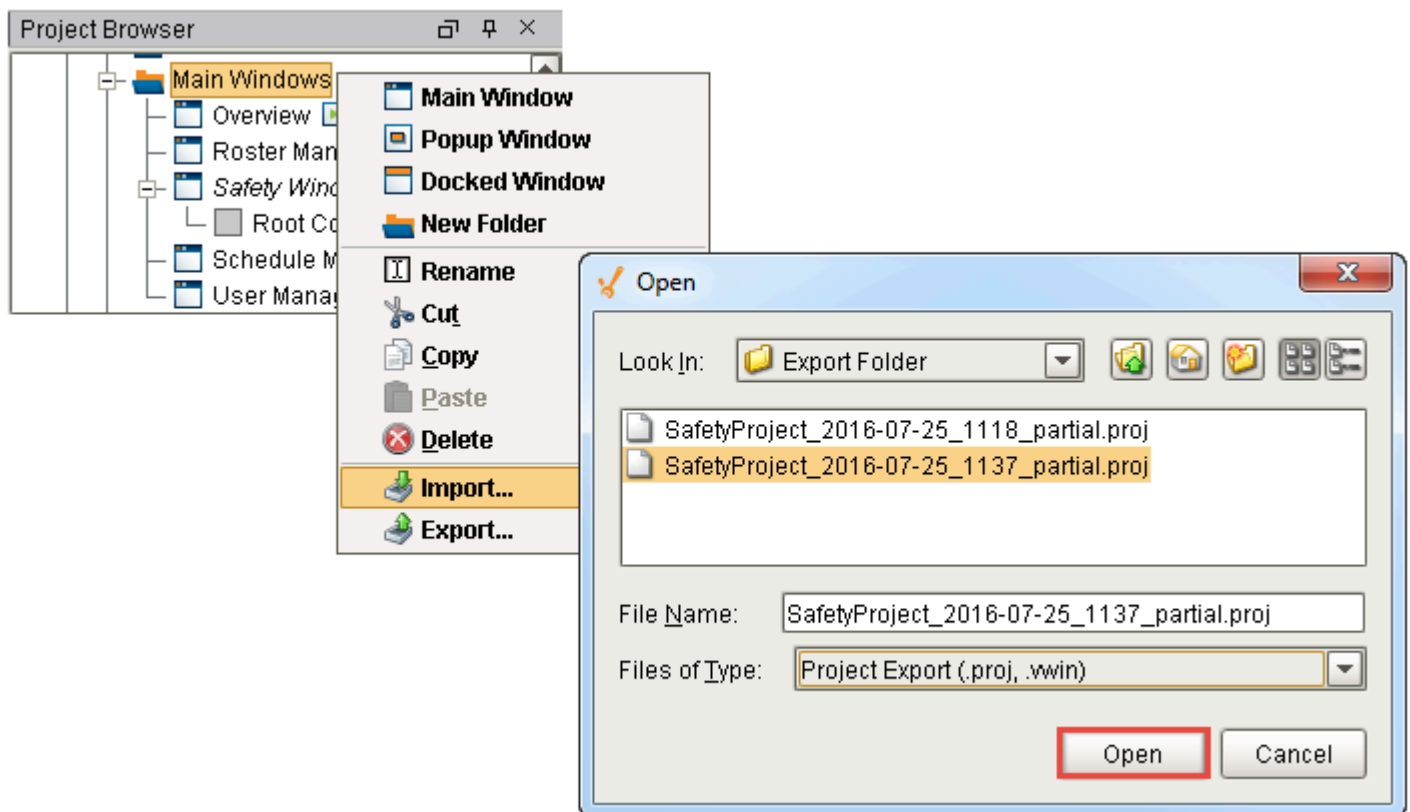


Importing a Window

Importing windows is just as easy as exporting. You can replace or **restore** a window using the exported *.proj file.

1. In the **Project Browser**, select the folder location where you want to add your window. Right click and select **Import**.

2. Select the ***.proj** file from the folder location where you saved your export, and press **Open**. Ignition will create the folder path and window if it doesn't already exist, or replace the existing window. That's it!



💡 DEFAULT EXPORT/IMPORT FILE TYPE ".PROJ, .VWIN"

In previous versions of Ignition, the export file type for windows was **.vwin** instead of **.proj**. Because Ignition is backwards compatible, it will accept both the current **.proj** and older **.vwin** file types when importing.

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

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

Window Types



INDUCTIVE
UNIVERSITY

Window Types

[Watch the video](#)

There are three window types: **Main** windows, **Popup** windows, and **Docked** windows. You can create windows from the **File > New** menu. By changing a window's properties, you can transform any window into various configurations. You can change a window's Dock Position, Border Display Policy, Titlebar Display Policy, and Start Maximized properties to change it into one of the three types listed below. Each window type behaves differently as you'll soon see, but whatever type you decide to use, it's guaranteed to satisfy your business requirements.

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

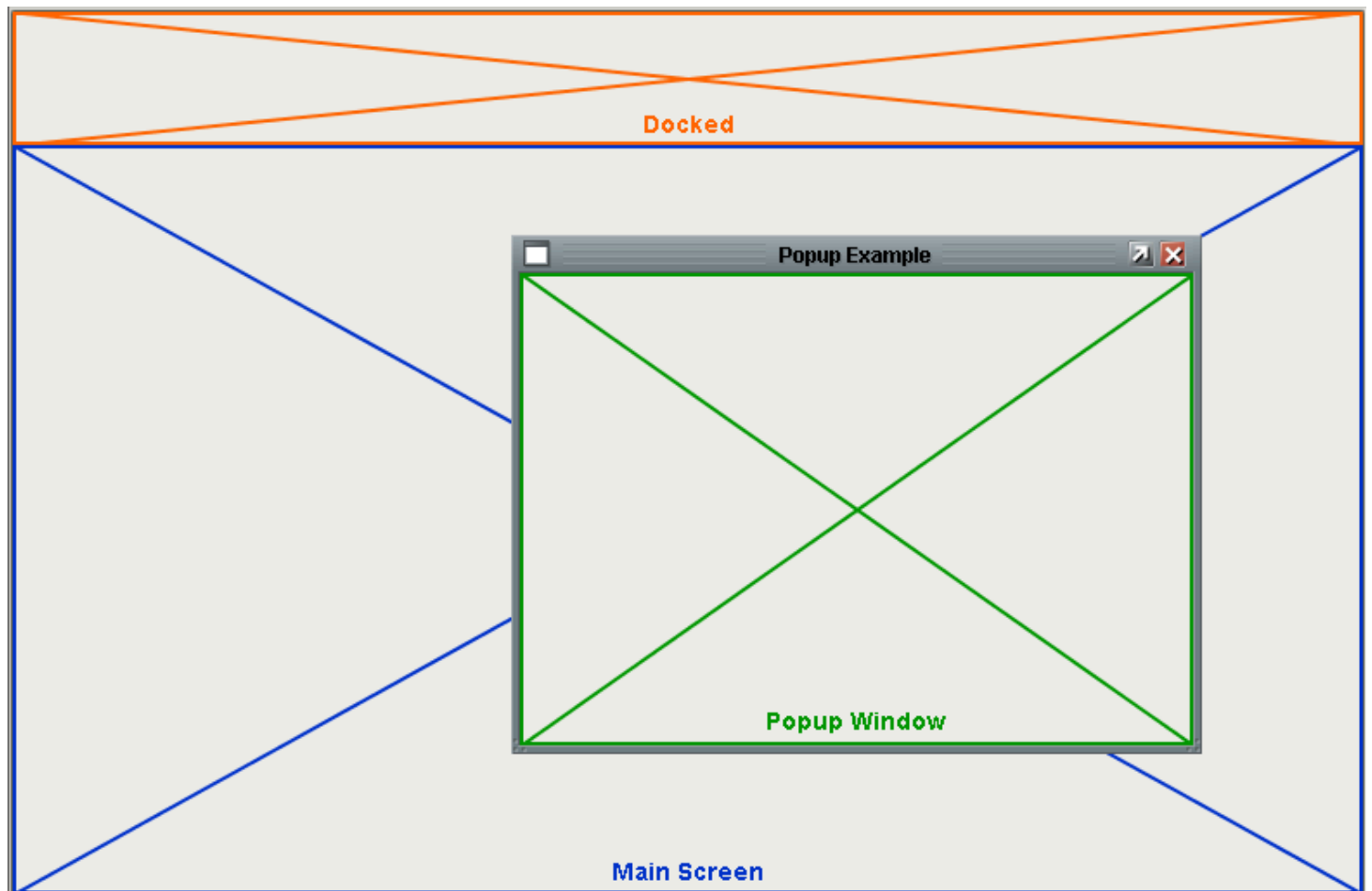
A **Popup** window is a window whose Dock Position is set to '**Floating**' and is not maximized. Its Border and Titlebar display policies are usually set to 'When Not Maximized' or 'Always,' so that they can be manipulated by the end-user. These windows are often opened by components in a main window, 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. To get a window to pop-up at a specific position, edit the window's starting **Location** property.

Popup windows are often [parameterized](#) so they can be re-used.

Docked Windows

A **Docked** window is one whose Dock Position is set to anything but Floating. This will make the window stick to one side of the screen, and nothing can overlap it. 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.



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

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

Window Properties



INDUCTIVE
UNIVERSITY

Anatomy of a Window

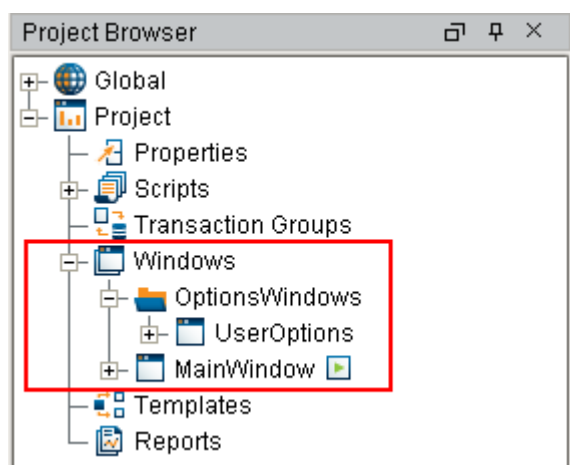
[Watch the video](#)

Anatomy of a Window

Name and Path

Windows are the top-level unit of design for Vision projects. A window is identified by its path, which is the name of all its parent folders plus its name, with forward slashes (/) in between. For example, the path to a window in the top level called **MainWindow** would simply be its name, whereas the path to a window named **UserOptions** under a folder called **OptionsWindows** would be:

OptionsWindows/UserOptions.



Root Container

Inside a window is always the Root Container. This is a normal container component except that it cannot be deleted or resized - its size is always set to fill the entire window. The Root Container is

where you will place all of your components in the window.

Titlebar and Border

A window may display a Titlebar and/or a Border. The titlebar allows the user to drag the window around in the client, and houses the window's close and maximize/restore buttons. The border of a window can be used to resize the window in the client when it is floating or docked. Whether or not the titlebar and border are displayed depends on the values of the window's titlebar and border display policy properties, and its current state. Commonly, a window will display 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 to make main and docked windows join seamlessly.

Window Properties



INDUCTIVE
UNIVERSITY

Window Properties

[Watch the video](#)

There are a number of window properties that change the way the windows look and behave. The commonly changed properties are: Closable, Maximizable, Resizable, Start Maximizable, Title, Border Display Policy, Titlebar Display Policy, Dock Position, and Layer.

You can modify these window properties, just like a component's property. Simply select the window by clicking on the window's name in the **Project Browser**, and while it is open, set the properties in the **Property Editor**.

Behavior

Name	Description	Data Type	Scripting Name	Value
Closable	Closable determines whether or not to draw the close (X) button in the upper right corner.	boolean	closable	
Maximizable	Determines whether or not to draw the maximize button in the upper right corner.	boolean	maximizable	
Resizable	Determines whether or not to let the user resize the window.	boolean	resizable	
Start Maximized	When set to true, the window will become maximized when it is opened.	boolean	startMaximized	
Cache Policy	By default this property is set to <code>Auto</code>, which keeps a window in a memory cache for a while after it is closed, so that if it is opened again it will be quick. The window isn't "active" while it is closed: all of its bindings and scripts are shut down. Setting this property to <code>Never</code> causes a fresh copy of the window to be deserialized every time it is opened. This is a performance hit, but it also is a convenient way to "clear out" the values of the window from the last time it was opened, which can be helpful in data-entry screens. Setting the property to <code>Always</code>	int	cachePolicy	0 Auto 1 Never 2 Always

Name	Description	Data Type	Scripting Name	Value
	will trade memory for higher performance, causing the window to always remain cached after the first time it is opened. This means the window will open very fast, but your Client will need lots of memory if you do this to a large amount of windows.			

Appearance

Name	Description	Data Type	Scripting Name	Value
Title	The title to be displayed in this window's titlebar.	String	title	
Border Display Policy	Determines if window's border is shown in various window states.	int	borderDisplayPolicy	0 Always 1 Never 2 When Not Maximized
Titlebar Display Policy	Determines if window's titlebar is shown in various window states.	int	titlebarDisplayPolicy	0 Always 1 Never 2 When Not Maximized
Titlebar Height	The height of the window's titlebar.	int	titlebarHeight	
Titlebar Font	The font of the window title in the titlebar.	Font	titlebarFont	

Layout

Name	Description	Data Typ	Scripting Name	Value
Dock Position	Determines the position this window is docked to, or if it is floating.	int	dockPosition	0 Floating 3 West 4 South 2 East 1 North
Location	The starting location that this window will open up at. Only applicable to floating windows that are not set to start maximized. This value will be overridden when an open window script specifies where to open.	Point	startingLocation	
Size	The dimensions of the window. This can be manipulated by selecting the window and dragging the resize handles along the windows right and bottom edges.	Dimension	size	
Minimum Size	The minimum size that this window will allow itself to be resized to.	Dimension	minimumSize	
Maximum Size	The maximum size that this window will allow itself to be resized to.	Dimension	maximumSize	
Layer	Sets the layer that this window is in. Default layer is 0, which is the bottom layer. Windows in higher layers will always be shown	int	layer	

Name	Description	Data Typ	Scripting Name	Value
	on top of windows in layers beneath them. A common strategy for using the layer property is to set Main Windows and Docked windows to 0, Popups to 1 and very important popups to 2.			
Dock Index	Determines the order of docked windows if multiple are open on the same edge. Lower numbers are on the outside (closest to the edge the window is docked to), and higher numbers are closer to the center.	int	dockIndex	

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

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

Open Windows on Startup



INDUCTIVE
UNIVERSITY

Open Static Window(s) on
Startup

[Watch the video](#)

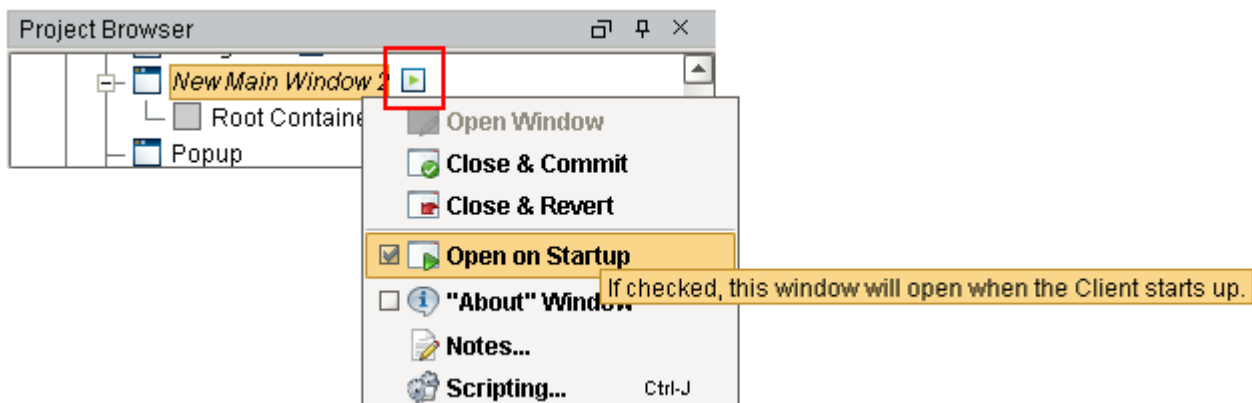
Open Static Windows on Startup

By default, most projects will open a static set of windows automatically when the client is opened. Usually, a navigation window and a main window are selected to open when the client is opened. You can edit which windows are opened on startup by selecting or deselecting the **Open on Startup** option for each window.

To Set a Window to Open on Startup

Lets say you want your Main Window and Navigation Window to open in the Client on Startup.

1. In the Designer, right-click on a Main Window and choose **Open on Startup**. An icon is displayed next to the window name so you can see which ones will be opened on start up.



2. Do the same for your Navigation Window.

3. **Save** and **Publish** your project.

4. Go to your **Client**, logout and log back in. The "on Startup" event only happens when a user logs in, switching users does not trigger this event. Now you can see the Main Window and the Navigation Window open automatically.

If you are using one of the project templates, you will notice that multiple windows are already set to Open on Startup. You can remove those in the same way.

Open Dynamic Window on Startup



INDUCTIVE
UNIVERSITY

**Open Dynamic Window(s) on
Startup**

[Watch the video](#)

Sometimes a project needs to change its startup windows depending on who logged in, what security roles they have, or what computer the Client is launched on. In these cases, rather than setting a static startup window, you can write a Client Startup Script that uses the `system.nav` library to open a dynamic set of windows based on hostname, IP address, user who logged in, and more.

This means you will remove the Open on Startup option from some or all of your windows and use a **Client Startup Script** to determine which windows will be opened. Typically, you will set your navigation window to Open on Startup, but decide on a main window in the startup script.

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

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

Docked Windows - Axis Precedence and Dock Index



INDUCTIVE
UNIVERSITY

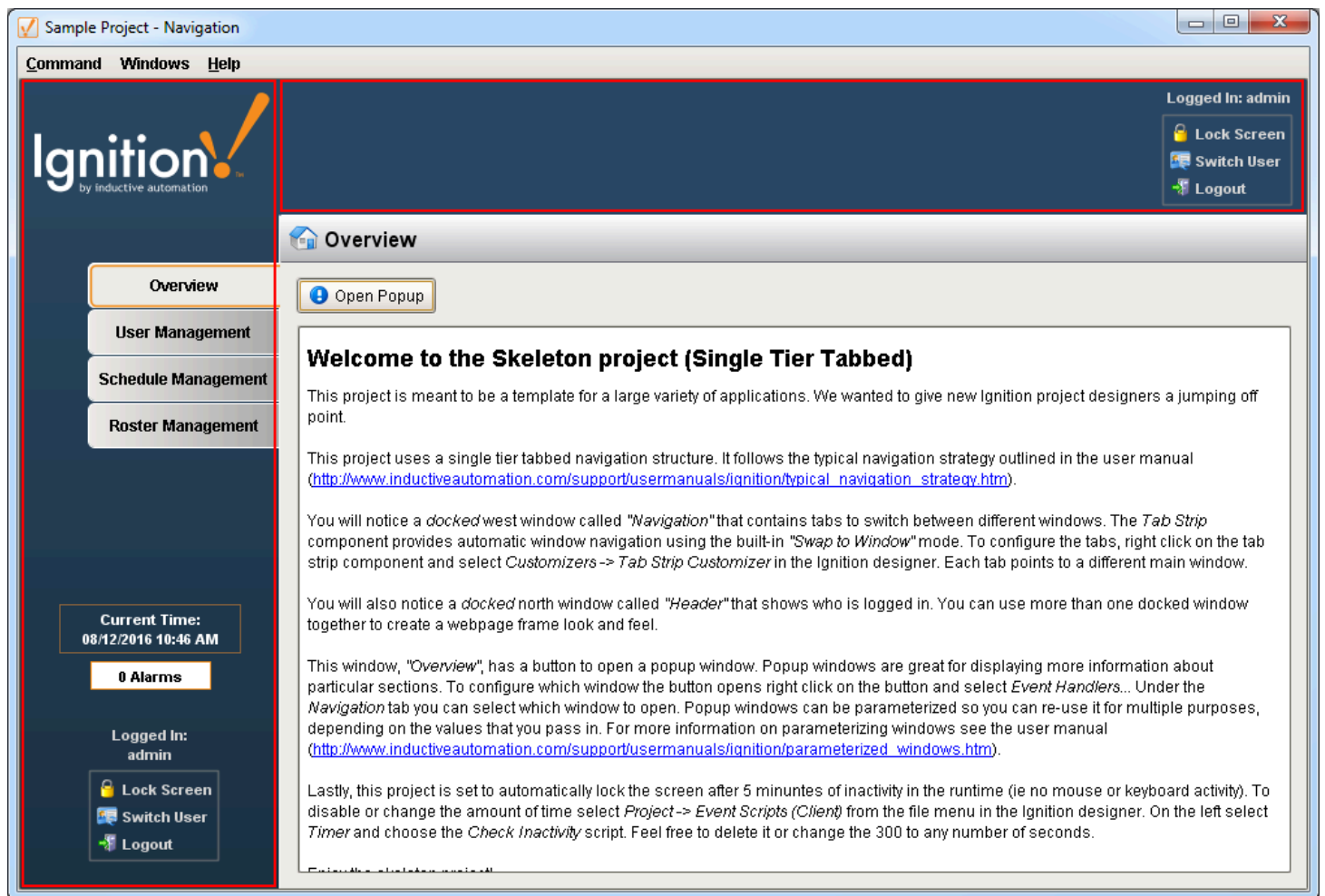
**Docked Windows - Order
Precedence**

[Watch the video](#)

Docked Windows

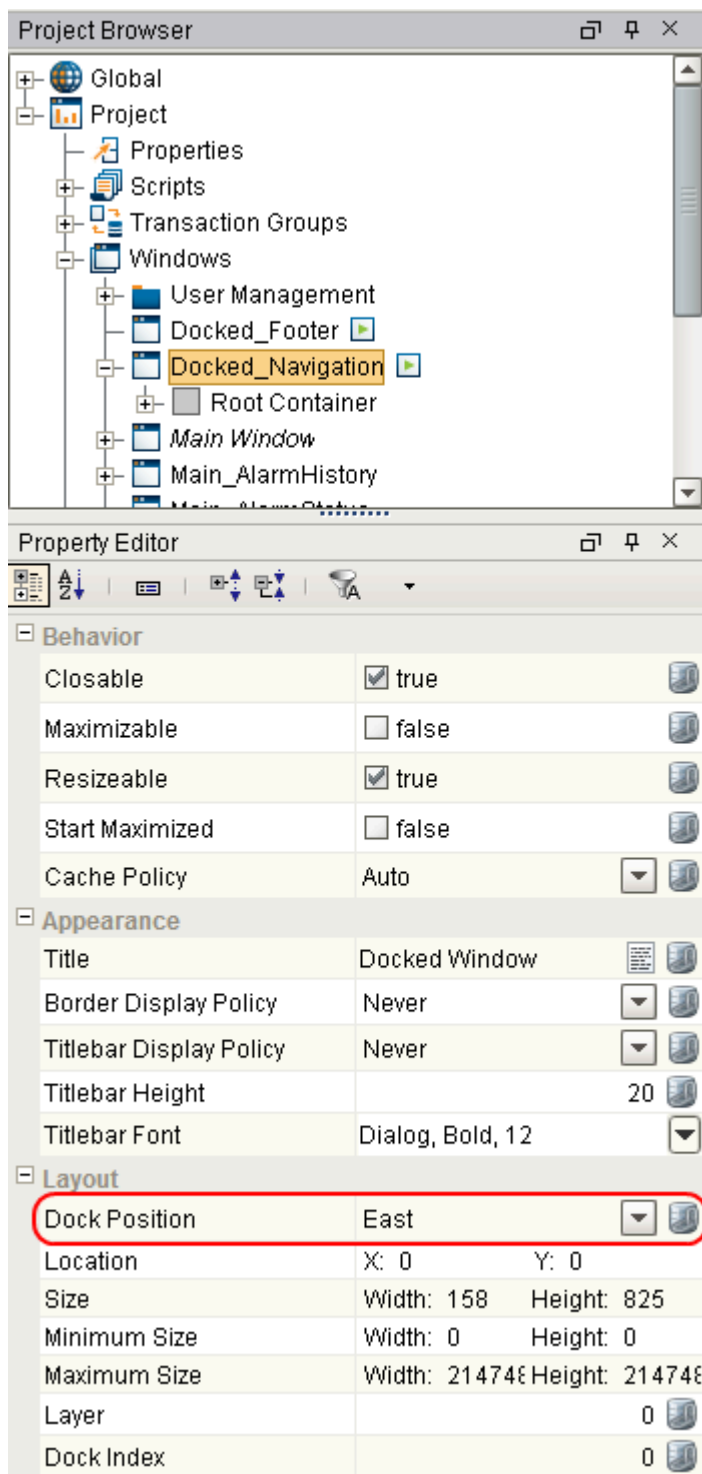
Docked windows are locked to the edges of the Client and fill all the space on that edge. IE: West Docked fills the left side of the Client. Often you see clients with a West (left) and/or North (top) docked window bordering a single main window. This common navigation strategy gives you the flexibility to easily select windows without worrying about re-creating your navigation many times.

Having multiple docked windows means you need to decide how you want them to appear in relation to each other. For that, Ignition has 2 settings: **Axis Precedence** and **Dock Index**. Here is an example from the **Single-Tier Nav, West & North** [Project Template](#).



To Set a Window's Dock Position

1. Select the window you want to dock in the **Project Browser**.
2. Select the correct **Dock Position** value (North/South/East/West) .



- Set the **Open on Startup** option using the open on startup icon  (next to the window name) for each docked window that you want automatically opened.

NOTE

Logging in is the only way to activate the **Open on Startup** settings.

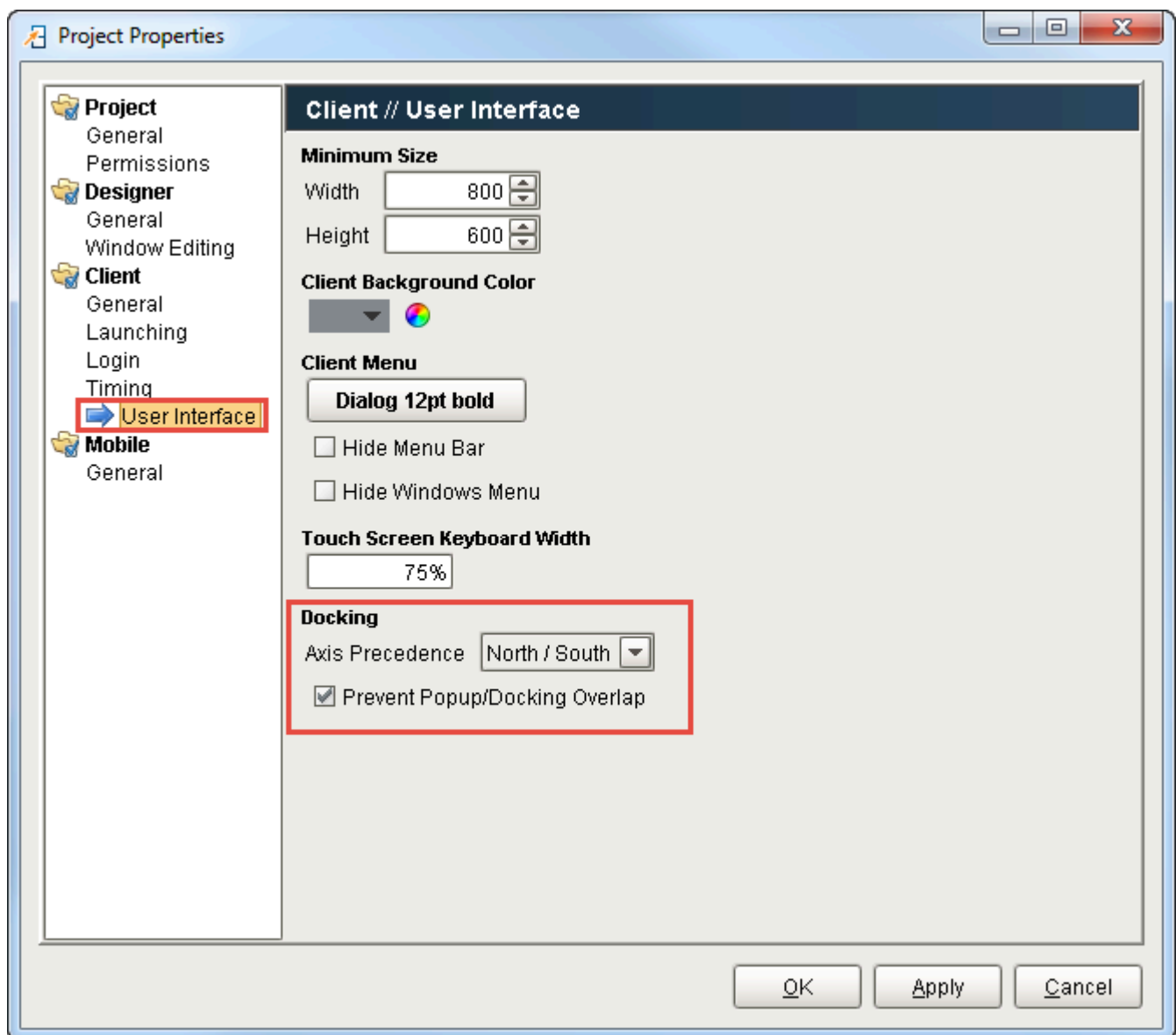
- Don't forget to **save** your project.
- Open your **Client** or log out and back in to see the changes.

Docked Windows - Axis Precedence

When you have more than one docked window that share a corner, you can specify which window will take control over that corner.

To Set the Docked Window Axis Precedence

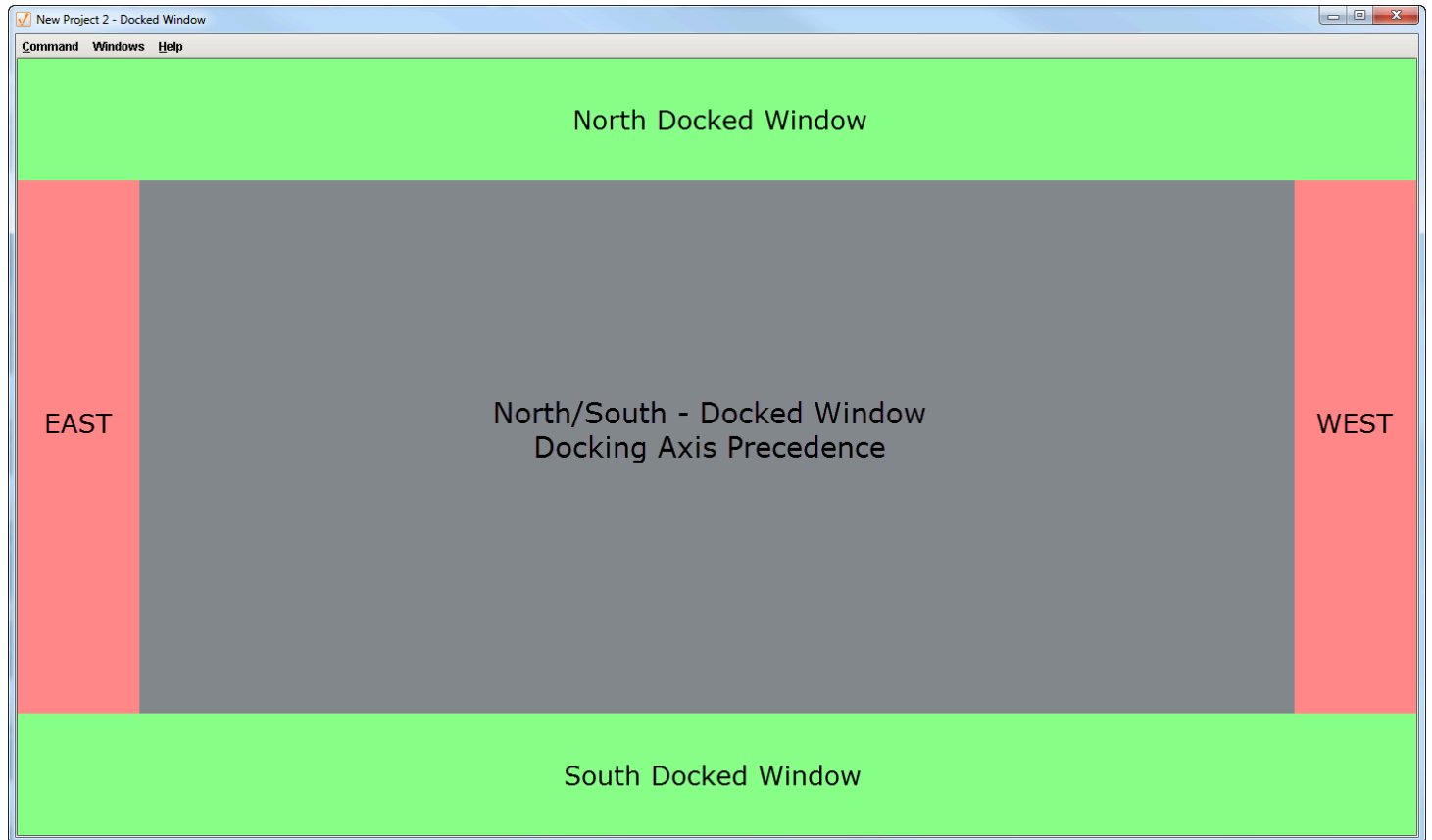
1. Go to **Project > Properties**.
2. Select the **Client > User Interface** page.
3. Select a value for the **Docking > Axis Precedence** drop-down. For example, if you choose **North/South** then North and South docked windows will take precedence or over the East and West windows.



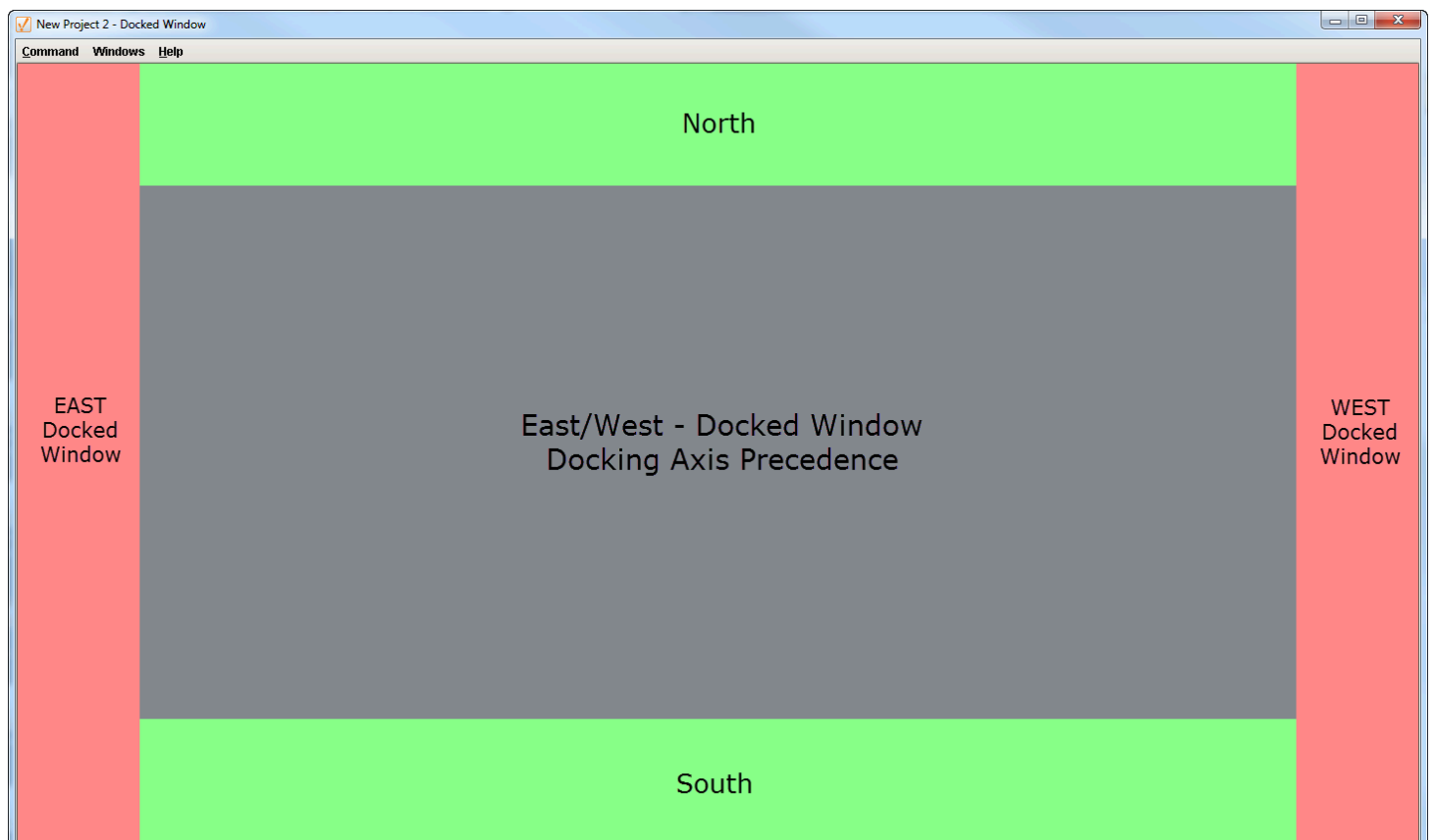
4. **Save and Publish** your project.
5. Open your project in the **Client**.

Examples of Docked Windows

North/South Docking Axis Precedence



East/West Docking Axis Precedence



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

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

Locate All Opened Windows In Client



INDUCTIVE
UNIVERSITY

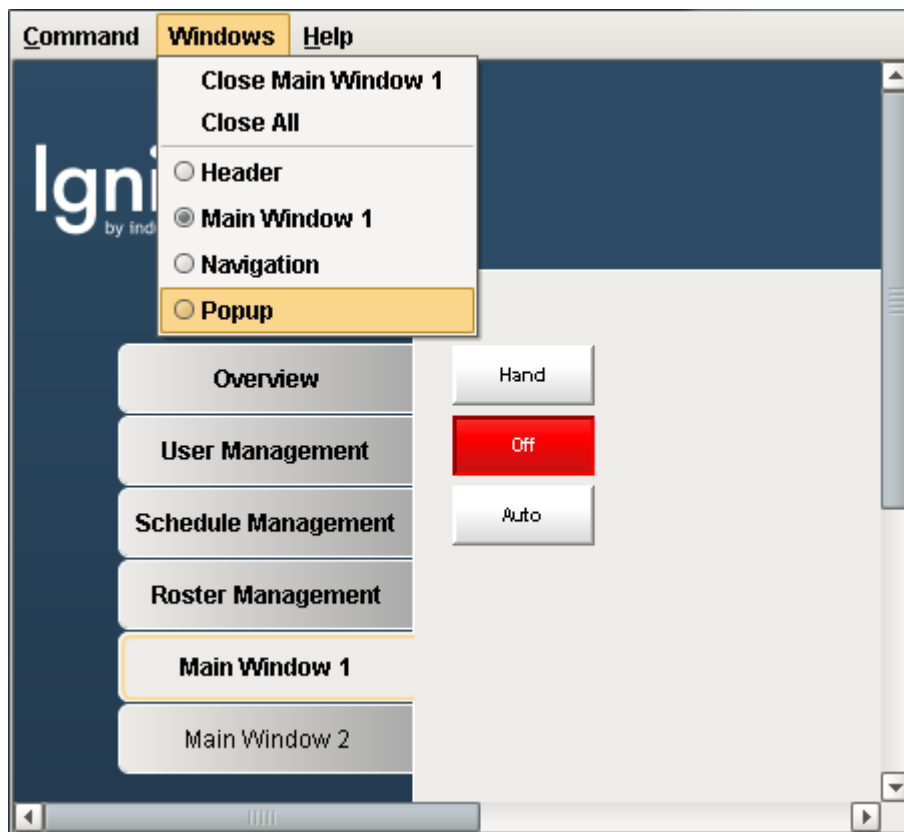
**Locate All Opened Windows
In Client**

[Watch the video](#)

Open Windows and Performance

While a window is open, its query bindings are running, its Tag bindings are keeping Tags subscribed, and its event scripts are being executed. This means that an open window is actively using system resources, both on the Client's host machine, and on the Gateway's server machine as its queries and Tag subscriptions must be handled. For these reasons, it is important that you properly implement a navigation strategy that prevents windows that are no longer being used from being held open. Fortunately, Ignition can tell you all the windows you currently have open in a client.

You can locate all of the currently opened windows in a Client from the **Windows** menu command. It is important to note, this list shows you the Title of each of the windows, not the name or path of the window.



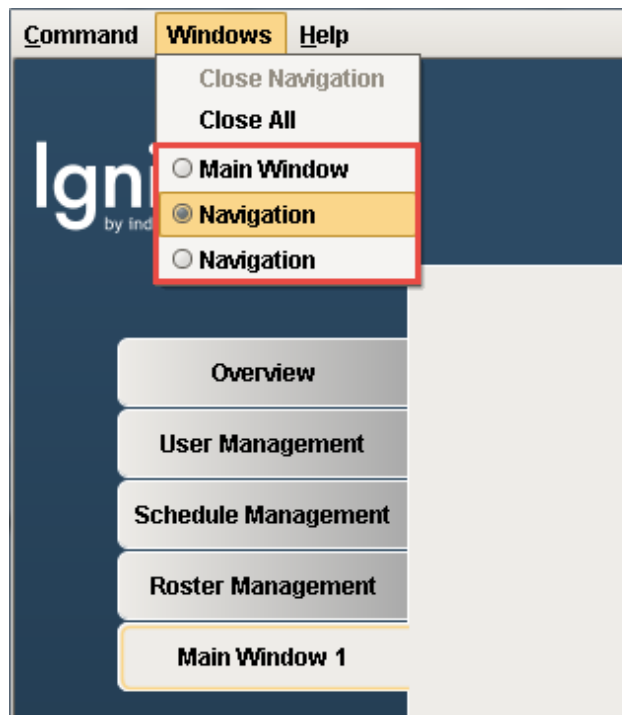
Window Titles

By default, Ignition assigns the Title property the same name as the window type you're creating (i.e., Main Window, Docked Window, and Popup Window). If you notice that you have two windows with the same name in the Window Menu dropdown list, go to Property Editor and change the Title from the window type to whatever you want to call your window (i.e., Main Window 1, Main Window 2, etc.), making sure the Titles are unique.

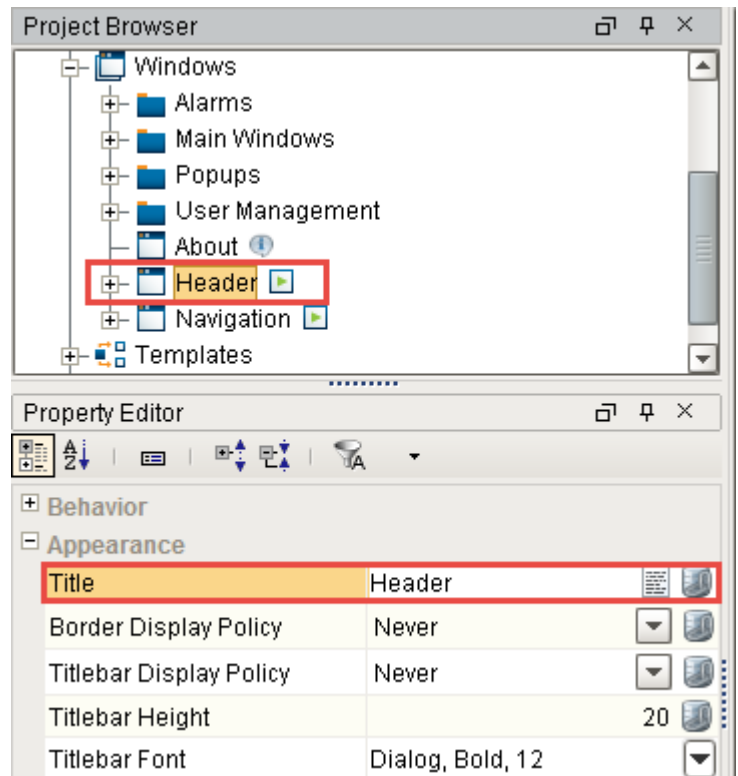
When you use one of the [Project Templates](#), by default, all of the existing Main Window Titles are bound to the text in the header label. This is a very easy way to keep track of which windows are open.

If your project has multiple windows with the same Title (i.e., both docked windows are called 'Navigation'), you can correct the duplicate Title in the Designer. Open one of the windows and go to the Property Editor, and change the Title property. Once again make sure the Title names are unique.

Duplicate Navigation Windows



Rename North "Navigation Window" to "Header"



Find Open Windows with Scripting

You can use the `system.gui.getOpenedWindows()` or `system.gui.getOpenedWindowNames()` functions in a script to get a list of the currently open windows in a client. Once you have the list, you can step through it and do whatever you want (i.e., check properties for a match, use the list to decide whether to open another window, etc.). Refer to the function descriptions in the Appendix for examples of how they are used.

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

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

Navigation Windows



INDUCTIVE
UNIVERSITY

Navigation Strategies

[Watch the video](#)

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. Before selecting the proper navigation strategy 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.

- 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

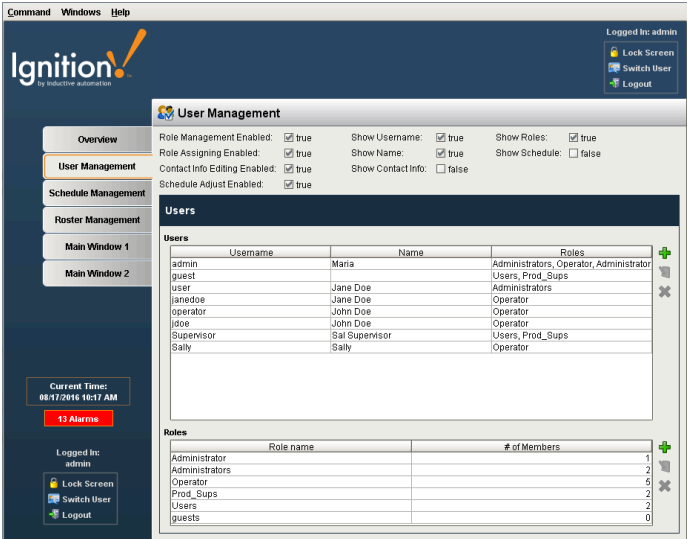
To help you select the right strategy that fits your project structure, here is a brief description of each navigation strategy that Ignition provides. Keep in mind your project structure, size, organization, and types of things you are doing while you are reviewing these strategies so you can select the best runtime strategy for your project.

- **Tab Strip** - 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

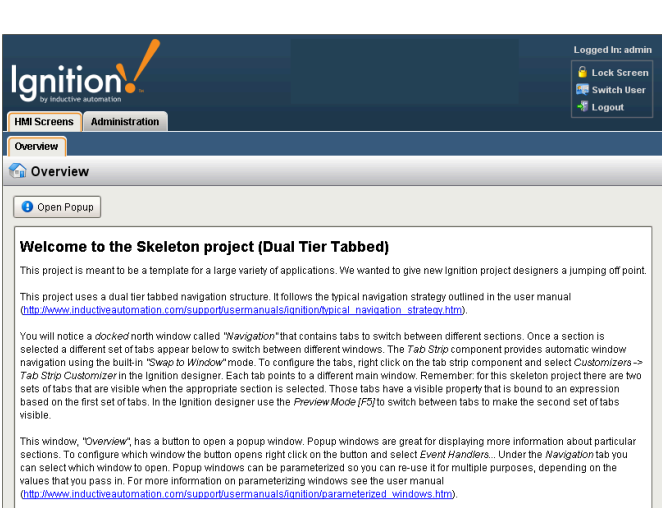
navigation window. Having too many tabs does not work well with the Tab Strip because of size limitations. You want your users to see all the navigation tabs immediately on the first screen. The Tab Strip works by clicking to swap one main window for another.

- **2 Tier** - 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** - 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** - this strategy is 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** - 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 on the 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.

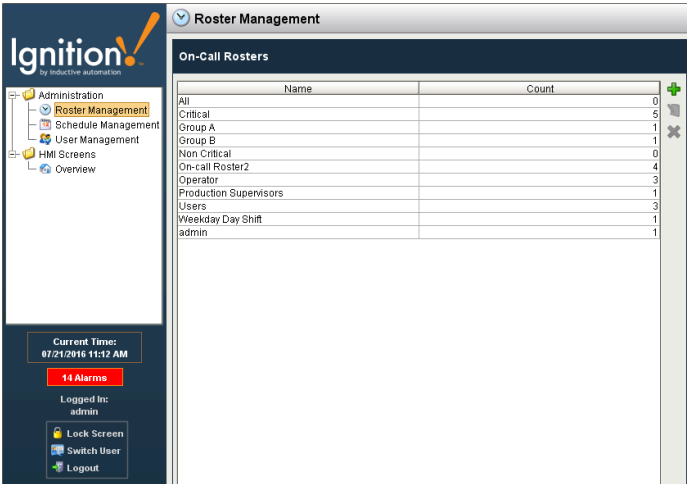
Tab Strip Navigation



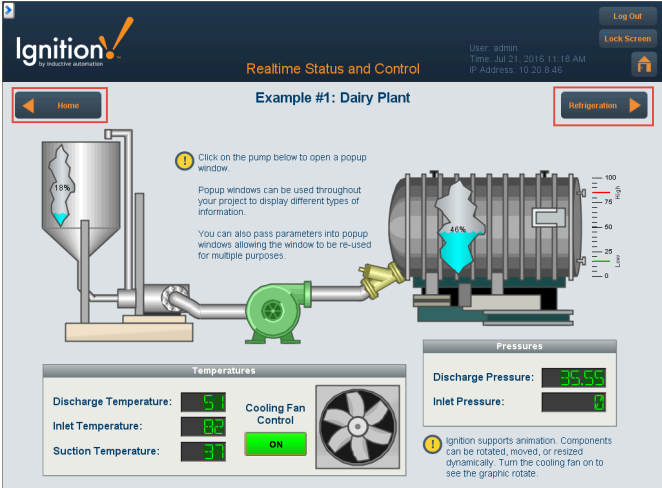
2-Tier Navigation



Tree View Navigation



Back and Forward Button Navigation



Drill Down Navigation

Tab Strip Navigation

2-Tier Navigation



Navigation Operations

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.

- **Swapping** - involves closing one window and opening another window in its place. This operation is usually reserved for maximized windows since they take up the entire screen.
- **Opening** - involves opening a window as it was saved in the Designer. This operation is usually reserved for popups since they are opened in addition to other windows.

To learn how they work, refer to the section on **Swapping vs. Opening**

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.

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

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

Swapping vs. Opening



INDUCTIVE
UNIVERSITY

Swapping vs. Opening

[Watch the video](#)

The two primary window navigation operations are: swapping and opening.

Swapping

When Ignition swaps a window, it closes the window and then opens another window in its place. This operation can be performed on windows in any state: docked or floating, maximized or not. The **Start Maximized** and **Dock Position** properties of the window that is being swapped *in* will be ignored - it will take the dock and maximized state of the window that it is replacing.

This operation is so common in the typical navigation strategy that there is even a version of the swapping function dedicated to it, the **swapTo** function. This function eliminates the need to specify the window to swap from - you only need to specify the window to swap to. It will take the current window - that is, the current maximized window - as the window to swap from.

See the following functions for more information:

- [system.nav.swapWindow](#)
- [system.nav.swapTo](#)

Opening

Opening and 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*.

To make a floating popup window open at a specific location, make sure to set the **Location** property of the window in the Designer. If the window was recently open, it will open in its last state due to window caching. See the [Window's Cache Policy](#) property for more information.

See the following function for more information:

- [system.nav.openWindow](#)

Open or Swap a Window with Scripting

1. In the Designer, right-click on a component and select **Scripting**.



TIP

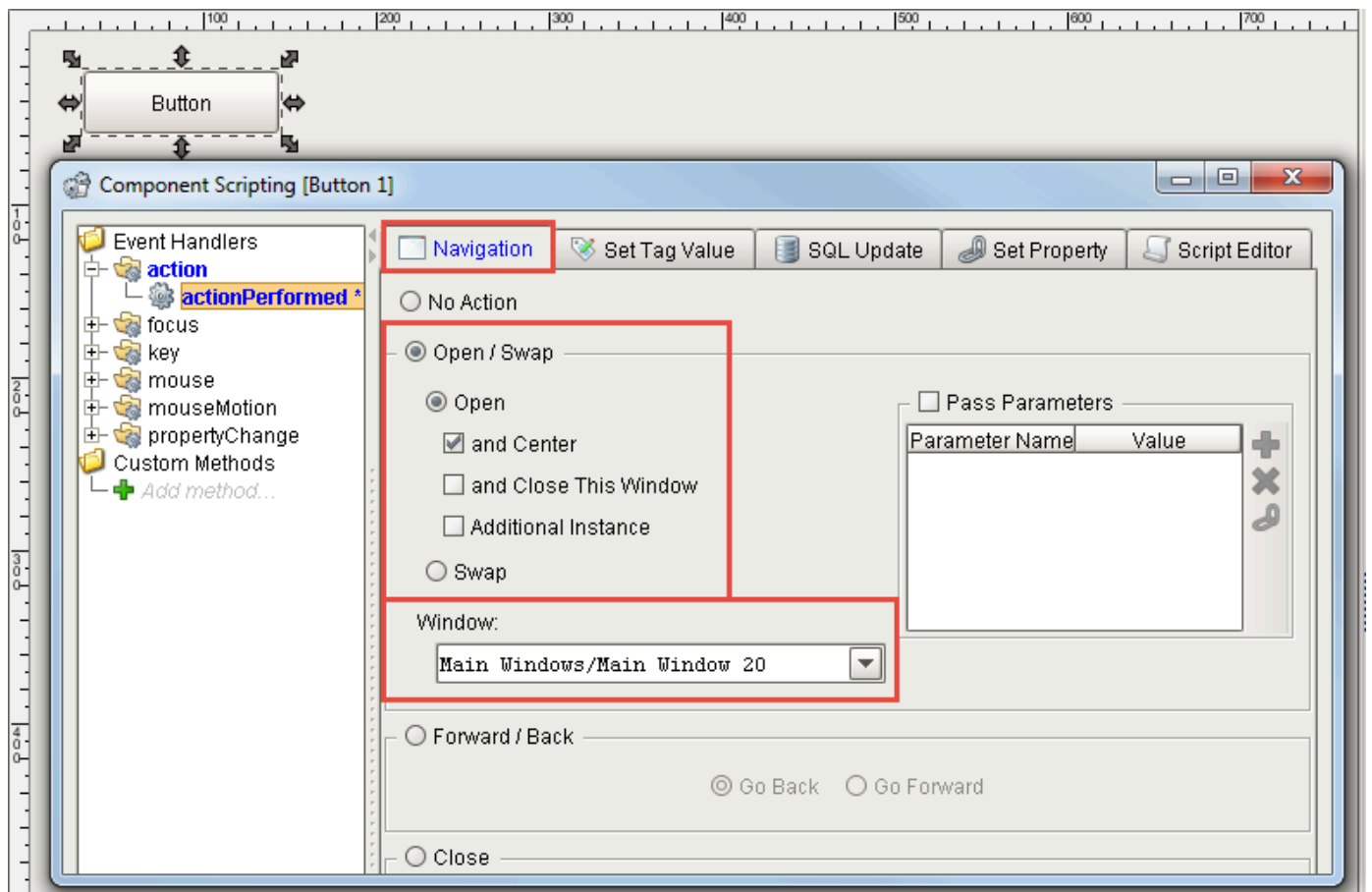
Usually a button is used to open a window.

2. Select the appropriate [Event Handler](#) for the component.

NOTE

Event Handlers are different for different components. There are a number of different options to choose from based on how and when you want your window to open or swap. If you are using a button, select the **actionPerformed** Event Handler.

3. Select the **Navigation** tab.
4. Click the **Open/Swap** radio button.
5. Select either the **Open/Swap** radio button as your Navigation strategy. Typically, you want to use the **Open** option for Popup and Docked windows, and the **Swap** for the Main windows that will be in the background as you can only view one at a time.
6. Select the Window you want to open/swap from of the **Window** dropdown.



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

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

Navigation - Tab Strip



INDUCTIVE
UNIVERSITY

Navigation - Tab Strip

[Watch the video](#)

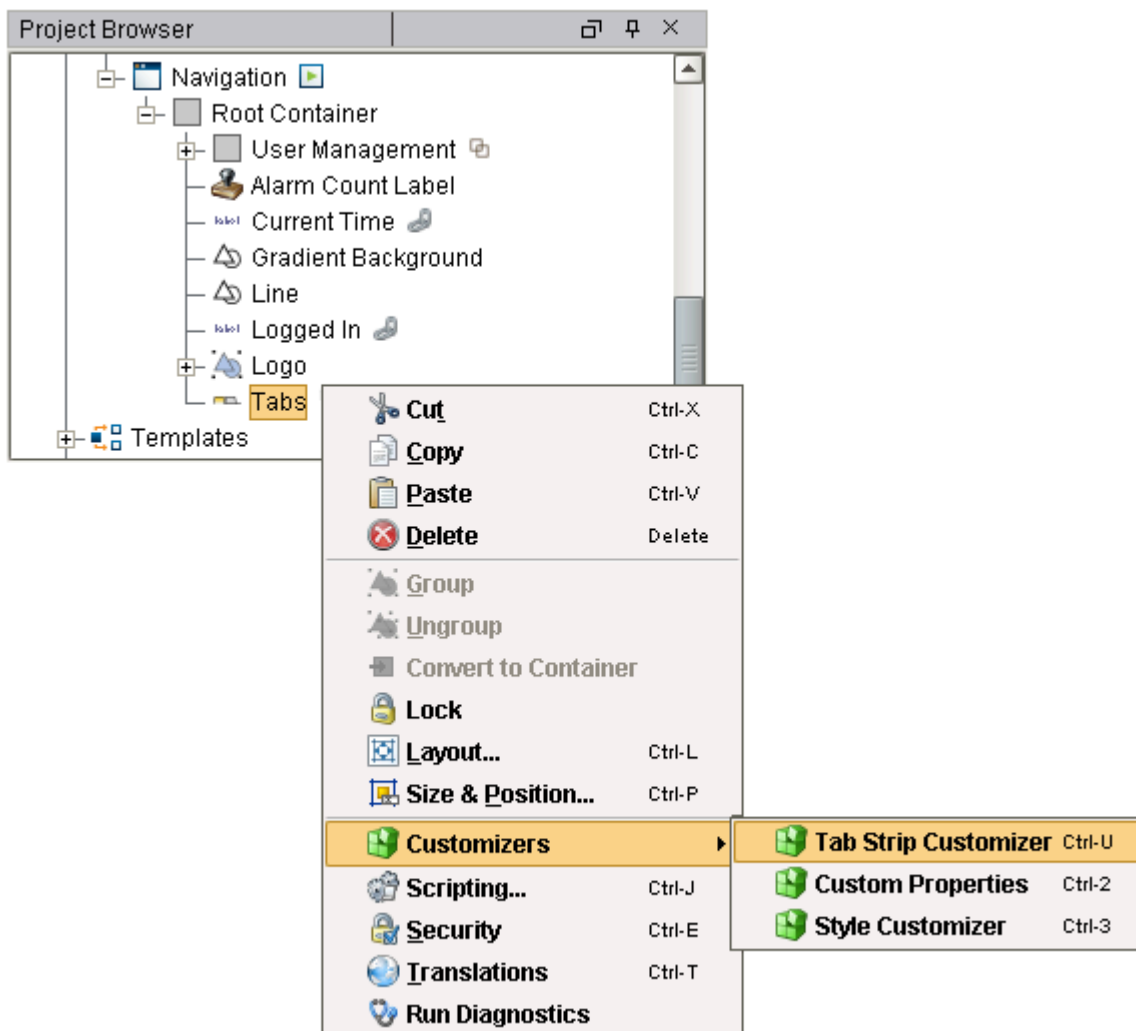
The **Tab Strip** is a simple navigation strategy used for small project structures having only a few windows. It allows users to see all the navigation tabs on the first screen of the runtime. It is most commonly used in a docked window to provide automatic window navigation. The Tab Strip works by 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.

For a simple product structure, it's common to use the Tab Strip to navigate a project in Ignition.

To Configure the Tab Strip in the Designer

1. Go to the **Project Browser**, expand the **Navigation** window, to see the **Tabs** component inside of it.
2. Right-click on the tab component, choose **Customizers > Tab Strip Customizer**.

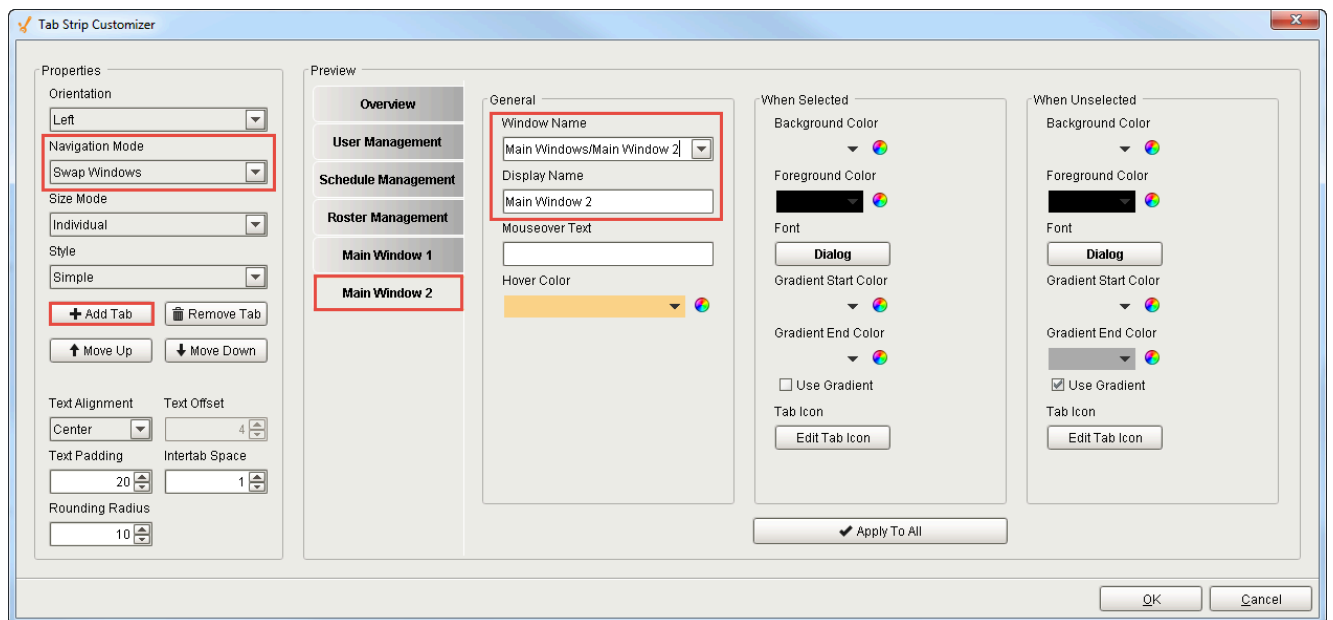


3. In the Tab Strip Customizer you can specify which window to open with each tab. Notice the **Tabs** component has a **Navigation Mode** property which is set to **Swap Windows** as shown in the screenshot below.
4. To create a new tab, click **Add Tab** (this example creates a Main Window 2 tab). If you have a tab already selected, clicking the **Add Tab** button creates a duplicate of the selected tab.

i MAIN WINDOWS ALREADY CREATED

This step assumes you already have your main windows created in your Project Browser.

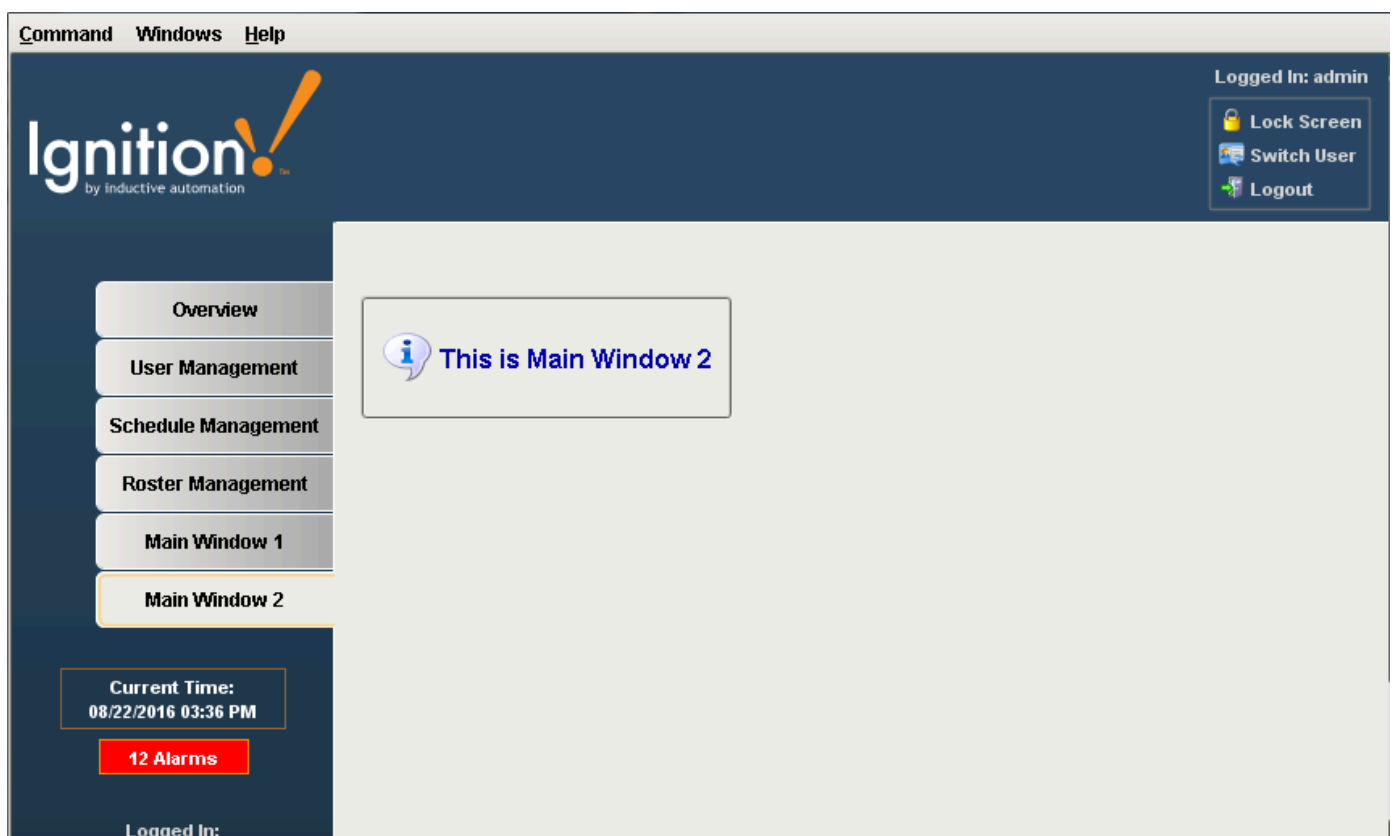
- i. Under **Window Name** dropdown list, select the window you want to open. Note it is the full path from the window and not just the name (i.e., **Main Windows/Main Window 2**).
- ii. Enter the **Display Name** for your new tab (i.e., **Main Window 2**).
- iii. With the **Move Up**, **Move Down**, 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.
- iv. When you're finished, press **OK**.



5. **Save** and **Publish** your project.

6. Open your project in the **Client**, and see that each tab navigates to a different window. You can also test your window navigation in the Designer, but only one window can be open at a time. To do so, use the **Preview Mode** to switch between windows.

7. As you can see, the **Tab Strip** works by clicking on a tab to swap one main window for another. If one of the windows happens to be a Popup Window, it will be treated as a Main Window by the **Swap Windows** mode.



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

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

Navigation - 2 Tier



INDUCTIVE
UNIVERSITY

Navigation - 2 Tier

[Watch the video](#)

2 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 level to switch between different windows. The 2 Tier navigation [Project Template](#) has a docked north window that contains tabs that are always open and a main window which fills the rest of the space.

The screenshot displays the Ignition HMI software interface. At the top, there is a menu bar with 'Command', 'Windows', and 'Help'. Below this, a dark blue header contains the 'Ignition' logo on the left, the 'Current Time: 08/15/2016 03:06 PM' in the center, and '13 Alarms' in a yellow box on the right. On the far right of the header, it says 'Logged In: admin' and provides buttons for 'Lock Screen', 'Switch User', and 'Logout'. Below the header, there is a navigation bar with 'HMI Screens' and 'Administration' tabs. The 'Overview' tab is selected and highlighted with a red border. Below the navigation bar, there is a button labeled 'Open Popup'. The main content area is titled 'Welcome to the Skeleton project (Dual Tier Tabbed)' and contains several paragraphs of text explaining the project's purpose and configuration. The text mentions a 'docked north window called "Navigation"' and provides links to user manuals for navigation strategy and parameterized windows. It also mentions a 5-minute inactivity timeout for the screen lock.

Command Windows Help

Current Time: 08/15/2016 03:06 PM

13 Alarms

Logged In: admin

Lock Screen

Switch User

Logout

HMI Screens Administration

Overview

Open Popup

Welcome to the Skeleton project (Dual Tier Tabbed)

This project is meant to be a template for a large variety of applications. We wanted to give new Ignition project designers a jumping off point.

This project uses a dual tier tabbed navigation structure. It follows the typical navigation strategy outlined in the user manual (http://www.inductiveautomation.com/support/usermanuals/ignition/typical_navigation_strategy.htm).

You will notice a *docked* north window called "*Navigation*" that contains tabs to switch between different sections. Once a section is selected a different set of tabs appear below to switch between different windows. The *Tab Strip* component provides automatic window navigation using the built-in "*Swap to Window*" mode. To configure the tabs, right click on the tab strip component and select *Customizers -> Tab Strip Customizer* in the Ignition designer. Each tab points to a different main window. Remember: for this skeleton project there are two sets of tabs that are visible when the appropriate section is selected. Those tabs have a visible property that is bound to an expression based on the first set of tabs. In the Ignition designer use the *Preview Mode [F5]* to switch between tabs to make the second set of tabs visible.

This window, "*Overview*", has a button to open a popup window. Popup windows are great for displaying more information about particular sections. To configure which window the button opens right click on the button and select *Event Handlers...* Under the *Navigation* tab you can select which window to open. Popup windows can be parameterized so you can re-use it for multiple purposes, depending on the values that you pass in. For more information on parameterizing windows see the user manual (http://www.inductiveautomation.com/support/usermanuals/ignition/parameterized_windows.htm).

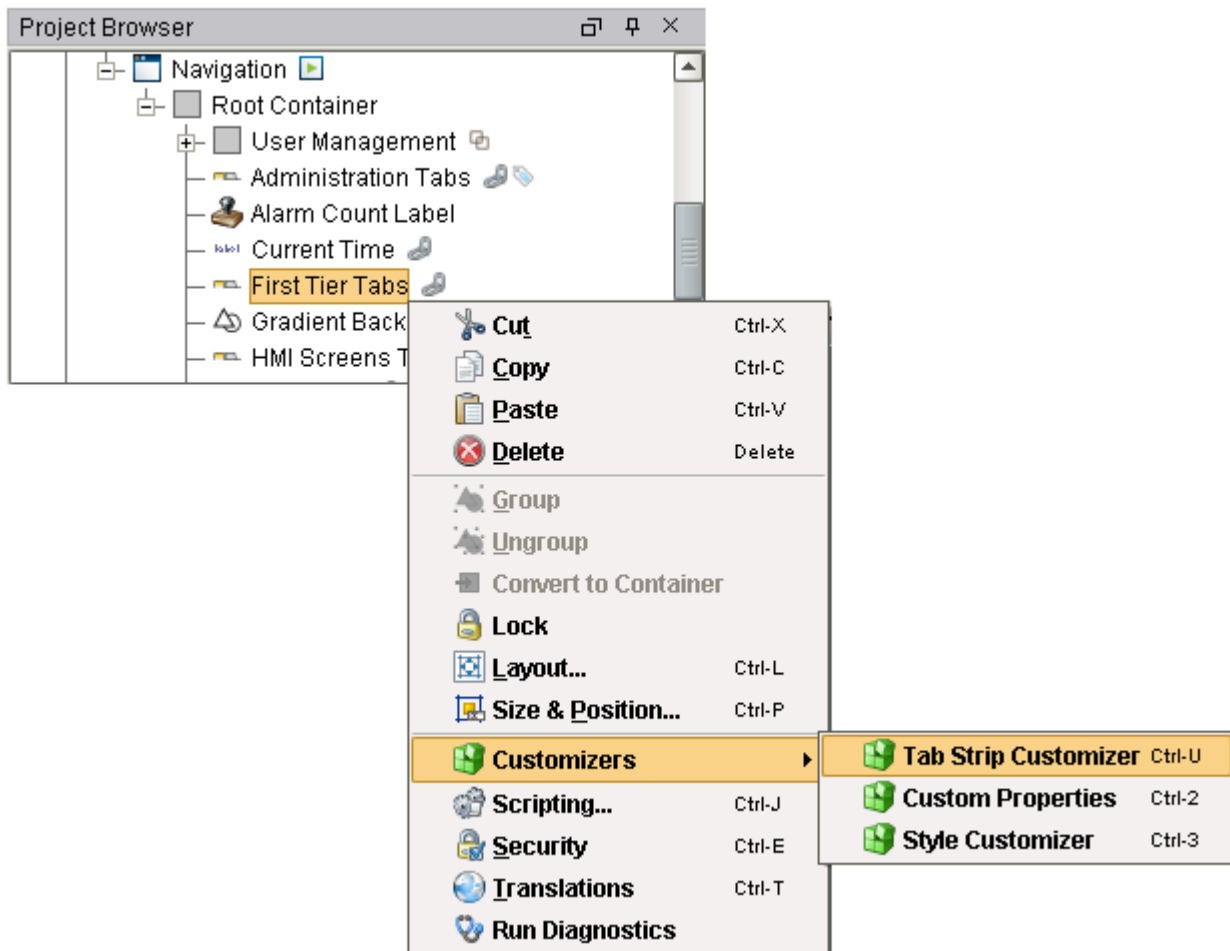
Lastly, this project is set to automatically lock the screen after 5 minutes of inactivity in the runtime (ie no mouse or keyboard activity). To disable or change the amount of time select *Project -> Event Scripts (Client)* from the file menu in the Ignition designer. On the left select *Timer* and choose the *Check Inactivity* script. Feel free to delete it or change the 300 to any number of seconds.

Enjoy the skeleton project!

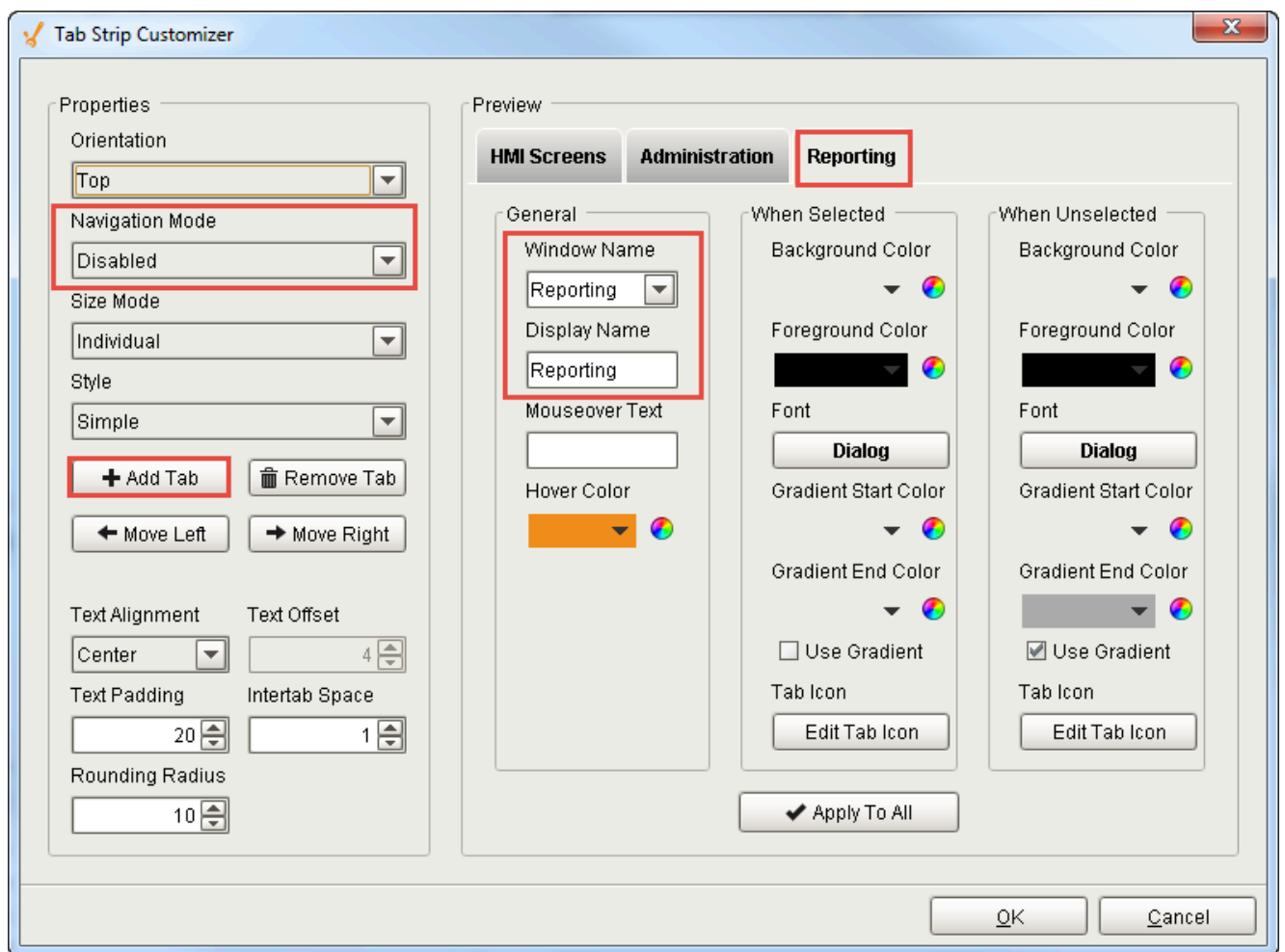
To Configure the 2 Tier in the Designer

A docked north window called Navigation contains tabs that allow you to switch between different sections in your project.

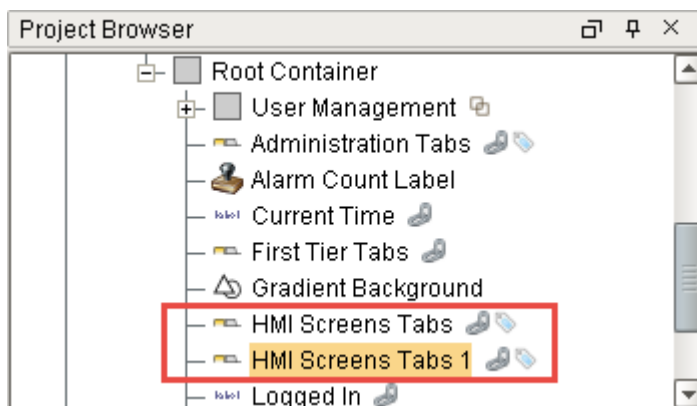
1. To create first level tabs, go to the **Project Browser**, right click on the **Tab Strip** component (i.e., **First Tier Tabs**) and select the **Customizers > Tab Strip Customizer**.



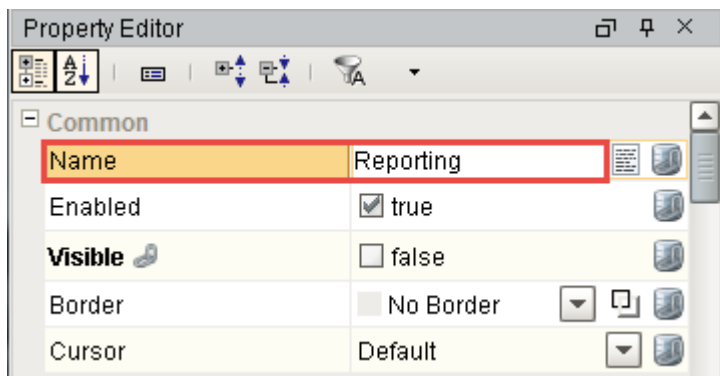
2. You'll notice that the Navigation Mode is disabled for the First Tier Tabs, and there are HMI Screens and Administration tabs that already exist as part of the skeleton project as shown in the screenshot below.
 - i. Click **Add Tab** to create a new First Tier Tab. Enter a keyword for your Tab name (i.e., **Reporting**) in the **Window Name** and **Display Name fields**. The keyword can be the same name for both fields. (**Note**: a folder path is not required for First Tier Tabs).
 - ii. Press **OK**.



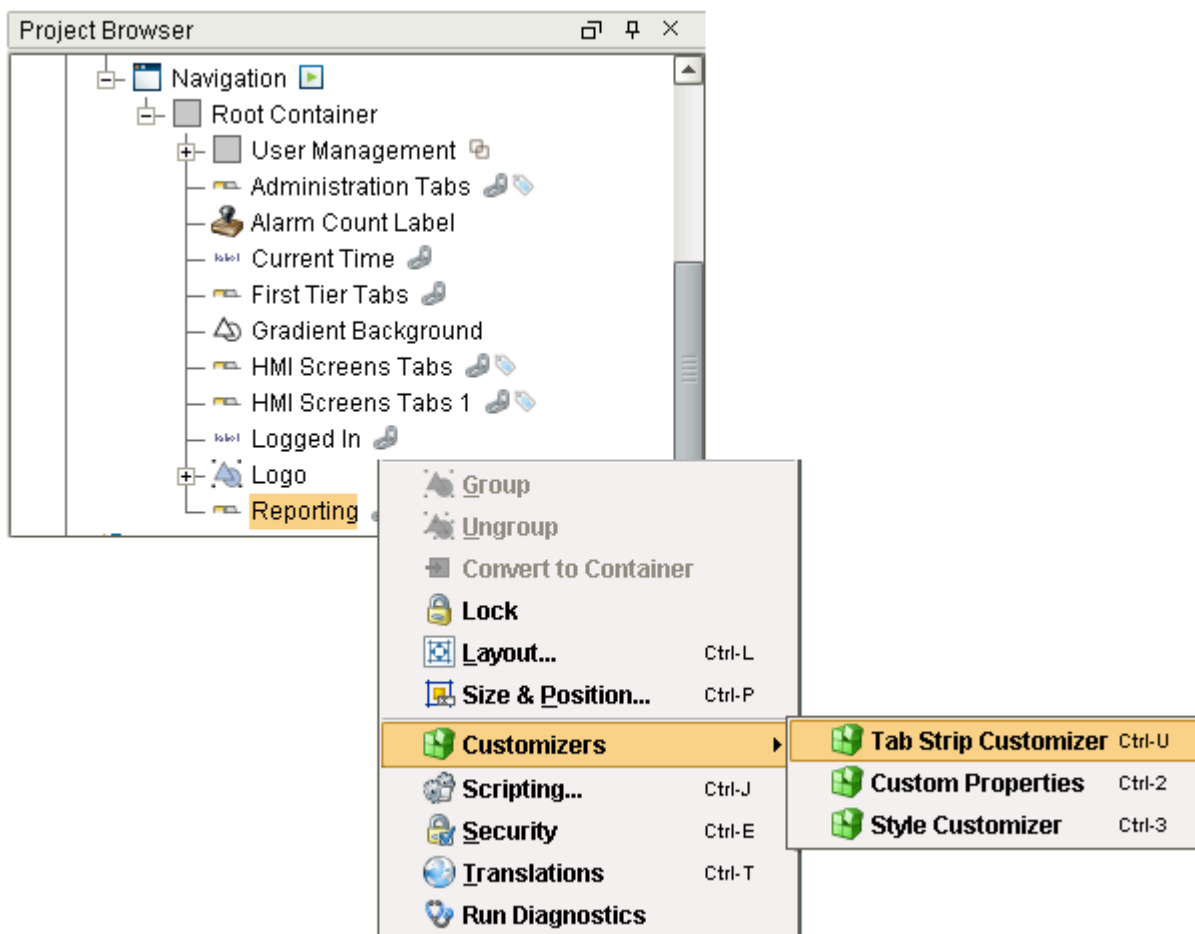
3. To create a second level component, go to the **Project Browser**, select an existing second level tab (i.e., **HMI Screens Tab**), and duplicate it by pressing **CTRL D**.



In the **Property Editor**, change the duplicate **Name** (i.e., **HMI Screens Tab 1**) to reflect the second tier tab name (i.e. **Reporting**).



4. To create your second level tabs, go to the **Project Browser**, right click on the **Tab Strip component** for **Reporting**, and select the **Tab Strip Customizer**.



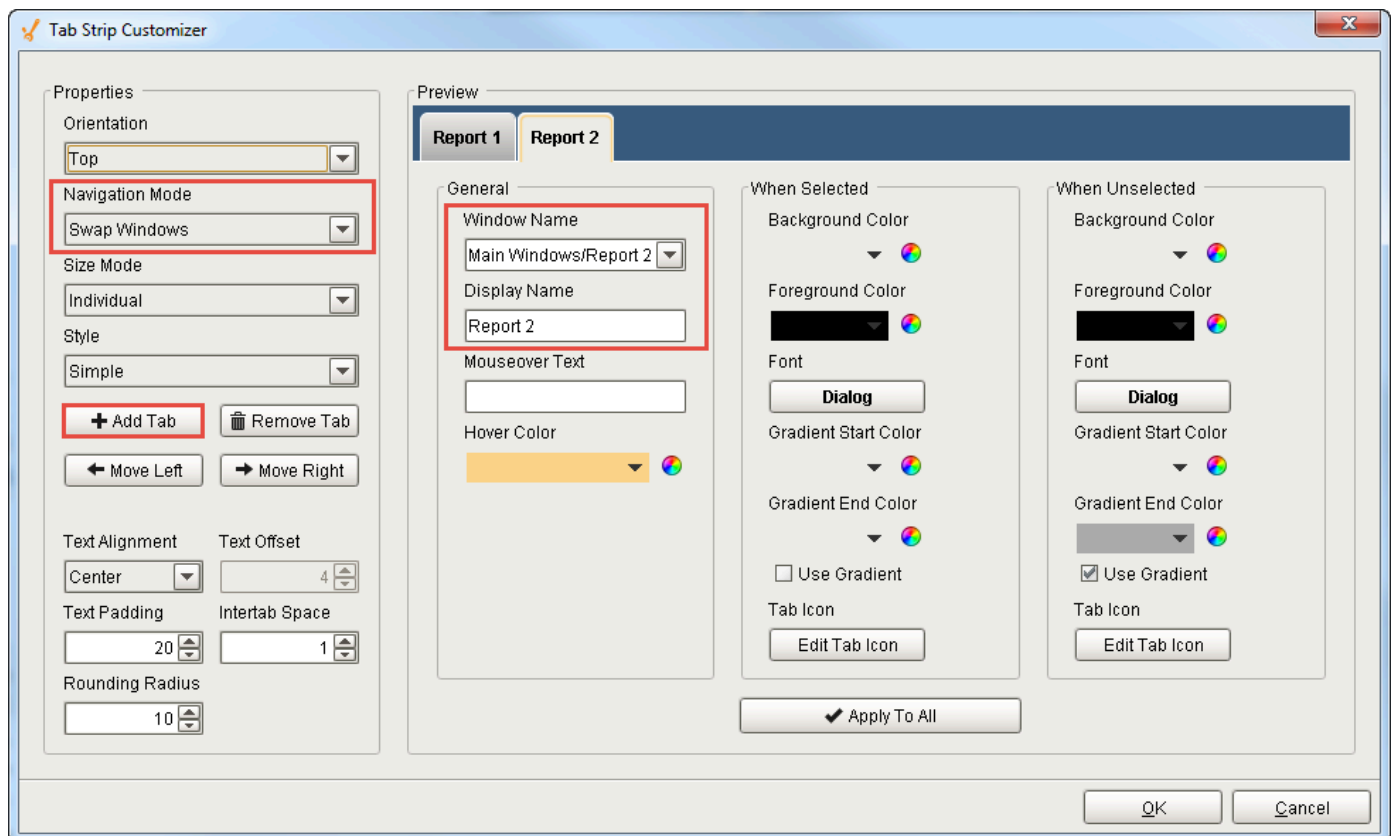
5. Notice that the Tab Strip Component provides automatic window navigation using the built-in **'Swap Windows'** mode for second level tabs. Each second level tab that you create will point to a different main window.

① MAIN WINDOWS ALREADY CREATED

This step assumes you already have your second tier main windows created in your Project Browser.

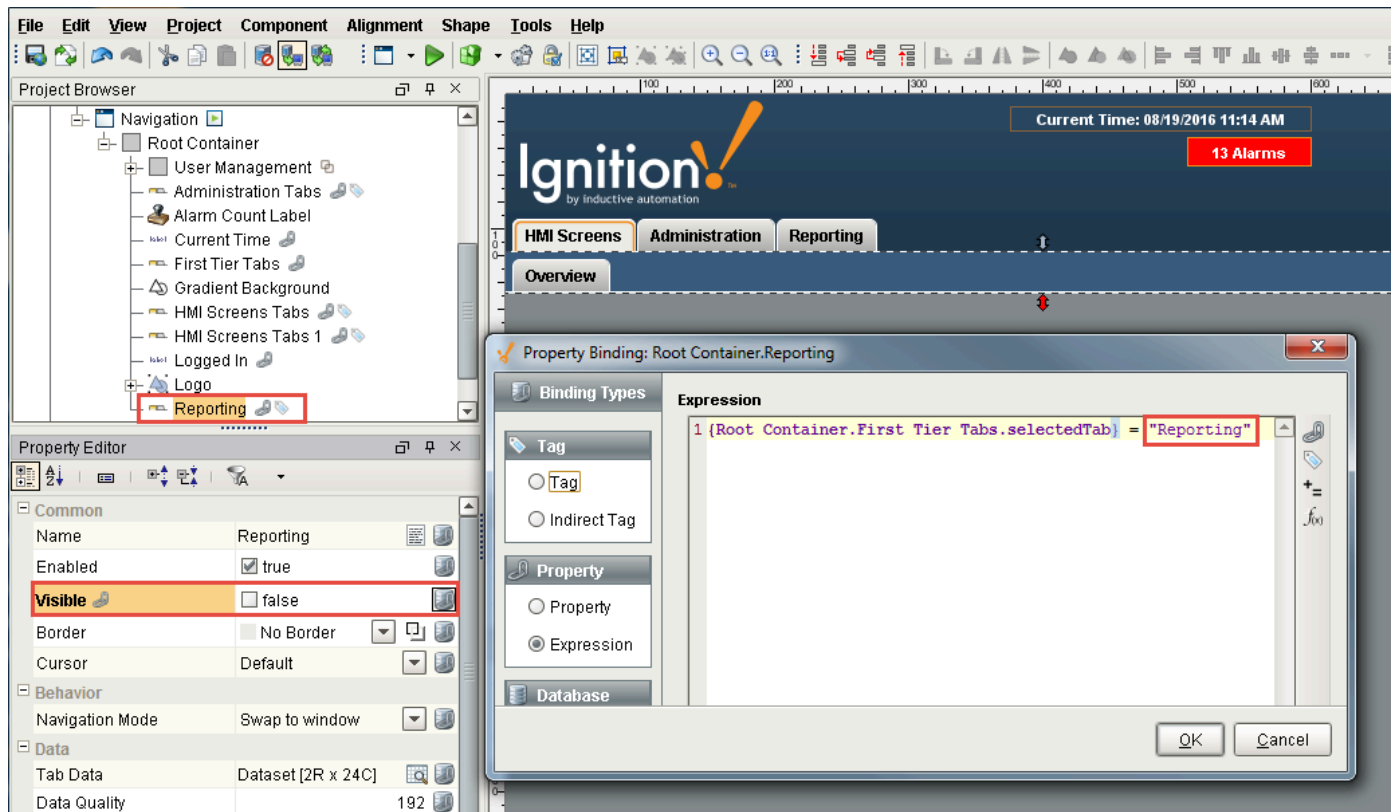
- i. Click **Add Tab** and select the **Window Name** from the dropdown list (i.e., **Main Windows/Report 2**).

- ii. Enter the **Display Name/Tab Name** (i.e., **Report 2**).
- iii. Repeat this step for as many tabs that you want to create.
- iv. Press **OK**.

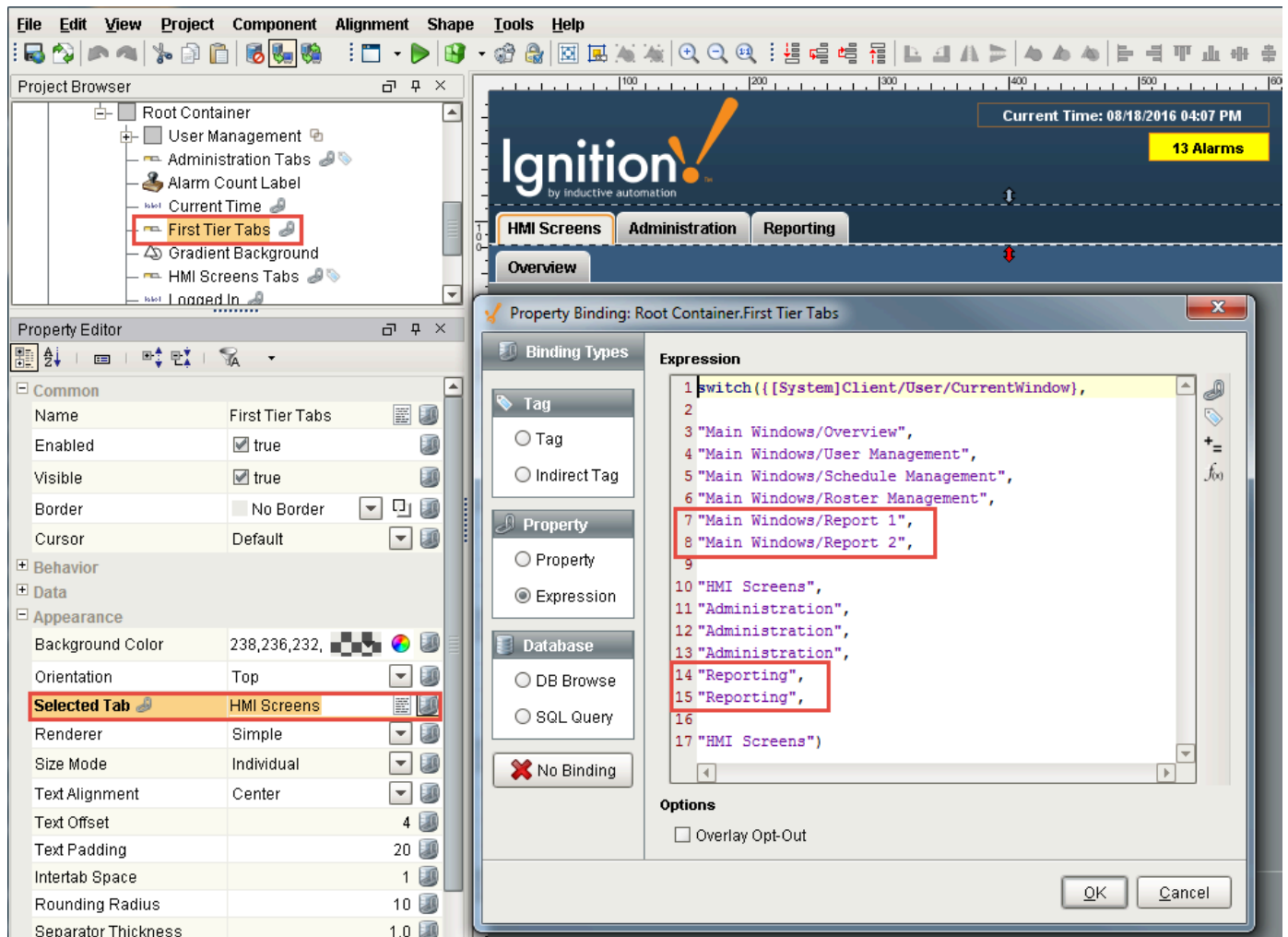


6. Second Level Tabs have a **Visible** property that is bound to an expression based on the First Level Tabs. This expression needs to be updated for the new second level tabs.

- i. Select the **Tab Strip component** for the second tier (i.e., **Reporting**).
- ii. Go to the **Property Editor**, select the **Visible** binding icon, and enter the **First Level Tab** keyword that you entered in step 2, (i.e., **Reporting**).
- iii. Press **OK**



7. The 2 Tier skeleton project includes a default expression for navigating between windows in a project. If you're adding and/or modifying existing tabs, then you need to update the expression binding to fit your windows.
 - i. Select the **First Tier Tab** component, and click on the binding icon for the **Selected Tab** property,
 - ii. Add the **window path** (i.e., **Main Windows/Report 1** and **Main Windows/Report 2**) for the new tabs, and the second tier **Tab Name** (i.e., **Reporting**) for each window as show in the following example.



8. In the Designer, use the **Preview Mode** to switch between tabs to make sure the Second Level Tabs are visible.
9. **Save and Publish** your project.
10. Open your project in the **Client**, click on the various tabs to see your first and second tier tabs switch between the different windows.

2nd Tier Tab - Report 1

CommandWindowsHelp

Current Time: 08/16/2016 03:27 PM

14 Alarms

Logged In: admin

Lock Screen

Switch User

Logout

Ignition

by Inductive Automation

HMI Screens

Administration

Reporting

Report 1

Report 2

Active Time	Display Path	Current State	Priority
☐ 8/1/16 8:15 AM	Writable/WriteableInteger1/...	Active, Unacknowledged	Critical
☐ 8/1/16 8:15 AM	Machine C	Active, Unacknowledged	Critical
☐ 8/1/16 8:15 AM	gensim/Writeable/WriteableIn...	Active, Unacknowledged	Critical
☐ 8/8/16 2:28 PM	Machine B	Active, Unacknowledged	Critical
☐ 8/1/16 8:15 AM	Alarms.ProdSupAttention/Ala...	Active, Unacknowledged	High
☐ 8/16/16 3:27 PM	Ramp	Active, Unacknowledged	High
☐ 8/1/16 8:15 AM	Machine B	Active, Unacknowledged	Medium
☐ 8/1/16 8:15 AM	Machine B	Active, Unacknowledged	Medium
☐ 8/8/16 2:28 PM	Machine B	Active, Unacknowledged	Medium
☐ 8/16/16 3:27 PM	Ramp	Active, Unacknowledged	Medium
☐ 8/1/16 8:15 AM	Alarms.ProdSupAttention/Alarm 1	Active, Unacknowledged	Low
☐ 8/1/16 8:15 AM	Writable/WriteableFloat1/Alarm	Active, Unacknowledged	Low
☐ 8/15/16 12:16 PM	Writable/WriteableInteger2/Alarm	Active, Unacknowledged	Low
☐ 8/15/16 12:16 PM	Writable/WriteableInteger2/Alarm 1	Active, Unacknowledged	Low
☐ 8/1/16 8:15 AM	Machine B	Cleared, Unacknowledged	Critical
☐ 8/16/16 3:26 PM	Ramp	Cleared, Unacknowledged	High
☐ 8/16/16 3:26 PM	Ramp	Cleared, Unacknowledged	High
☐ 8/16/16 3:26 PM	Ramp	Cleared, Unacknowledged	High
☐ 8/16/16 3:26 PM	Ramp	Cleared, Unacknowledged	High
☐ 8/16/16 3:26 PM	Ramp	Cleared, Unacknowledged	High
☐ 8/16/16 3:26 PM	Ramp	Cleared, Unacknowledged	High
☐ 8/16/16 3:27 PM	Ramp	Cleared, Unacknowledged	High

Acknowledge

Shelve

2nd Tier Tab - Report 2

CommandWindowsHelp

Current Time: 08/16/2016 02:40 PM

13 Alarms

Logged In: admin

Lock Screen

Switch User

Logout

Ignition

by Inductive Automation

HMI Screens

Administration

Reporting

Report 1

Report 2

Pens

☒ Sine0

☒ Sine1

☒ Sine2

Value

100

75

50

25

0

-25

-50

-75

-100

2:20 PM

2:31 PM

2:33 PM

2:35 PM

2:37 PM

[Aug 16, 2016]

8/16/16 2:28 PM - 8/16/16 2:38 PM

Apply

1:00 PM

1:15 PM

1:30 PM

1:45 PM

2:00 PM

2:15 PM

2:30 PM

2:45 PM

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

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

Navigation - Tree View

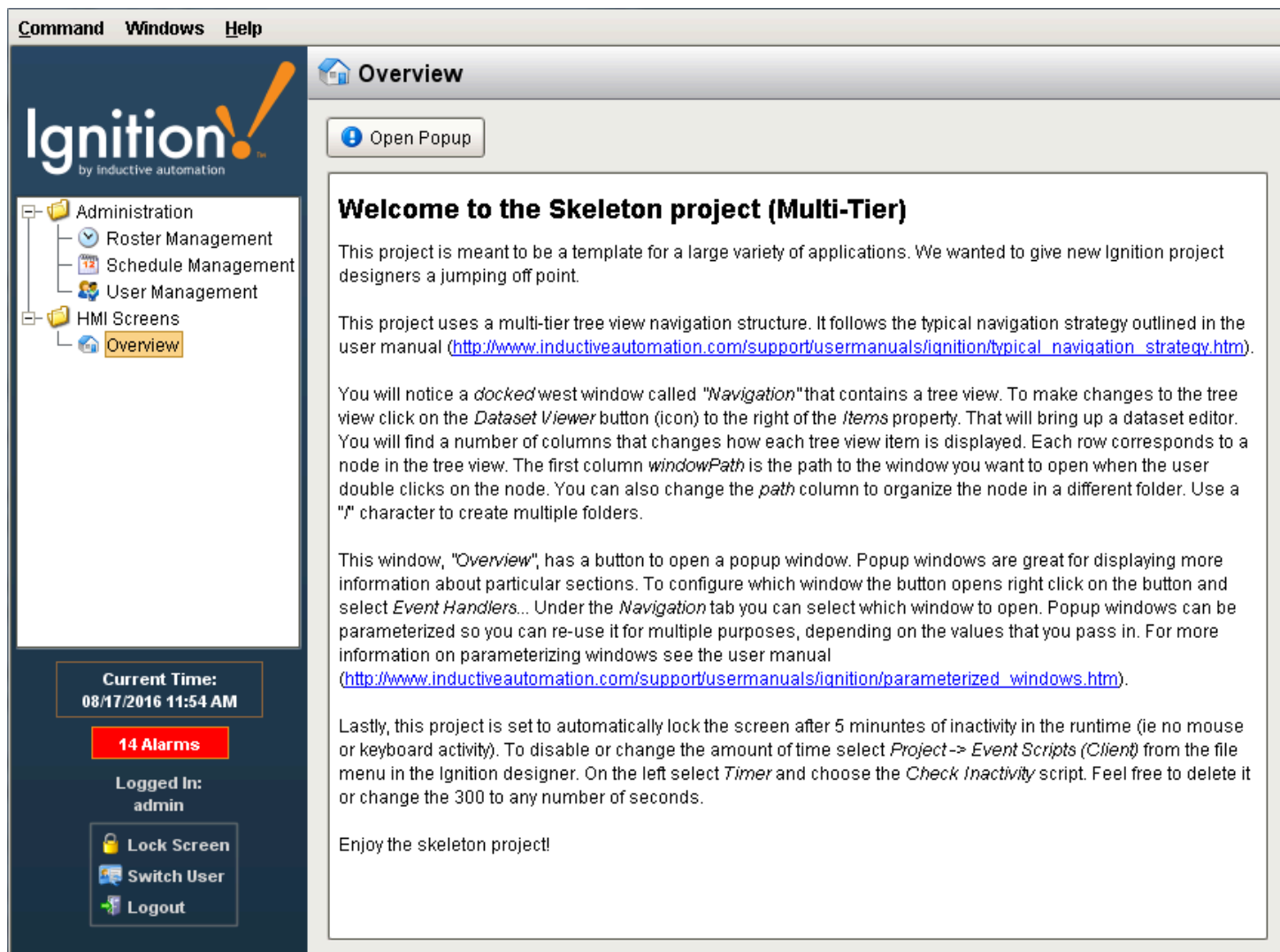


INDUCTIVE
UNIVERSITY

Navigation - Tree View

[Watch the video](#)

The **Tree View** navigation strategy is excellent for large project structures. It uses a typical navigation strategy that has a docked west window that contains a **Tree View component** to navigate around to various areas of your project. The Tree View window is always open and the main screen fills the rest of the space. Users can go to the Tree View, double click on an item and it will swap out one main window for that. The Tree View **Project Template** contains a built-in skeleton project that you can use to help you get started designing your navigation structure. You can also drag a basic Tree View component to a window for navigation, but that requires a little set up in scripting.

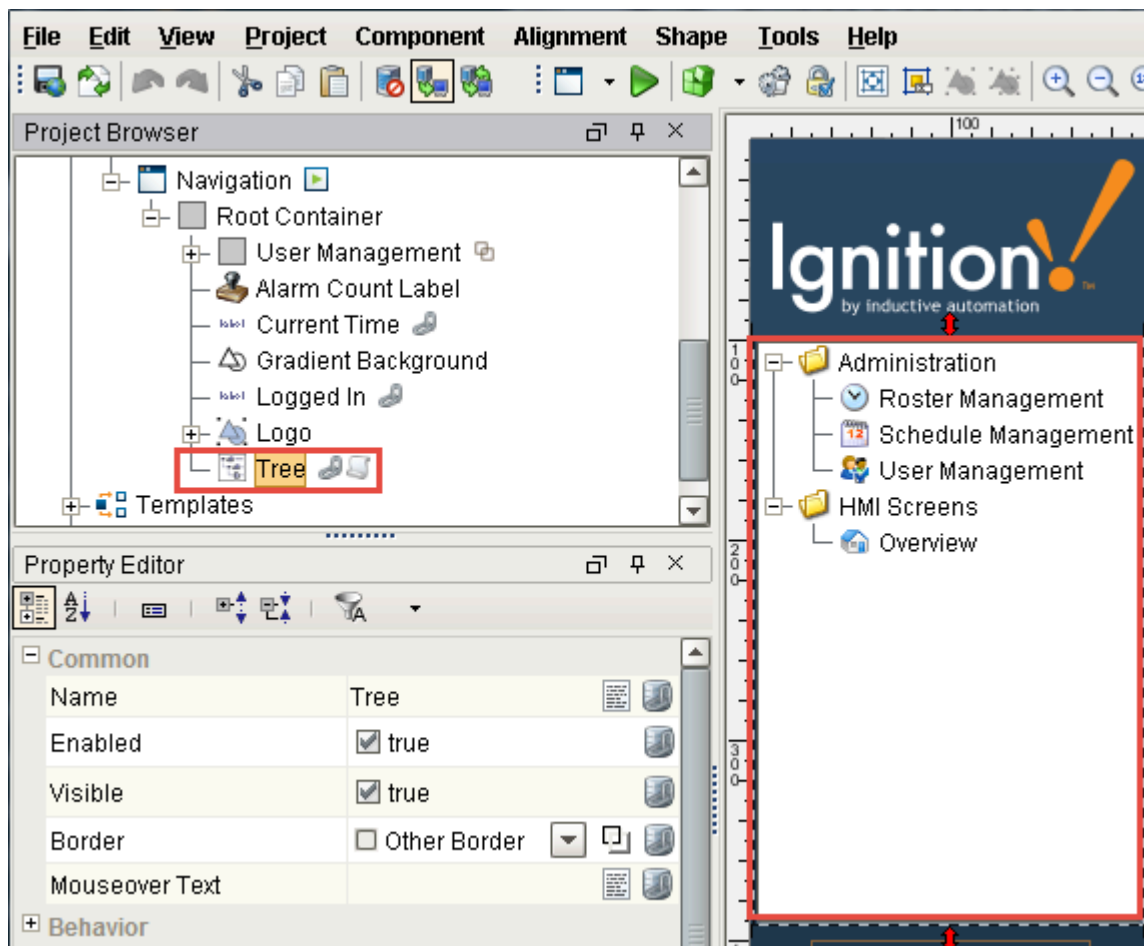


To Configure a Tree View in the Designer

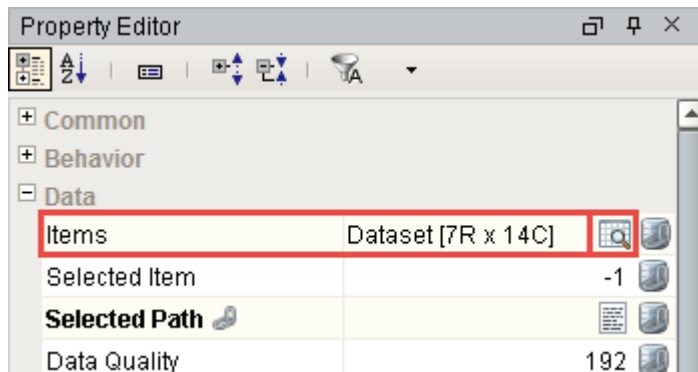
A docked west window called Navigation contains a tree view that allows you to switch between different sections in your project. You can create a Tree View for navigation by either using the skeleton template project or dragging a Tree View component to a docked window.

The following example uses the default Tree View skeleton project.

1. Select the **Tree View** component from the Navigation window.



2. From **Property Editor**, find the **Items** property and click on the **Dataset Viewer** icon.



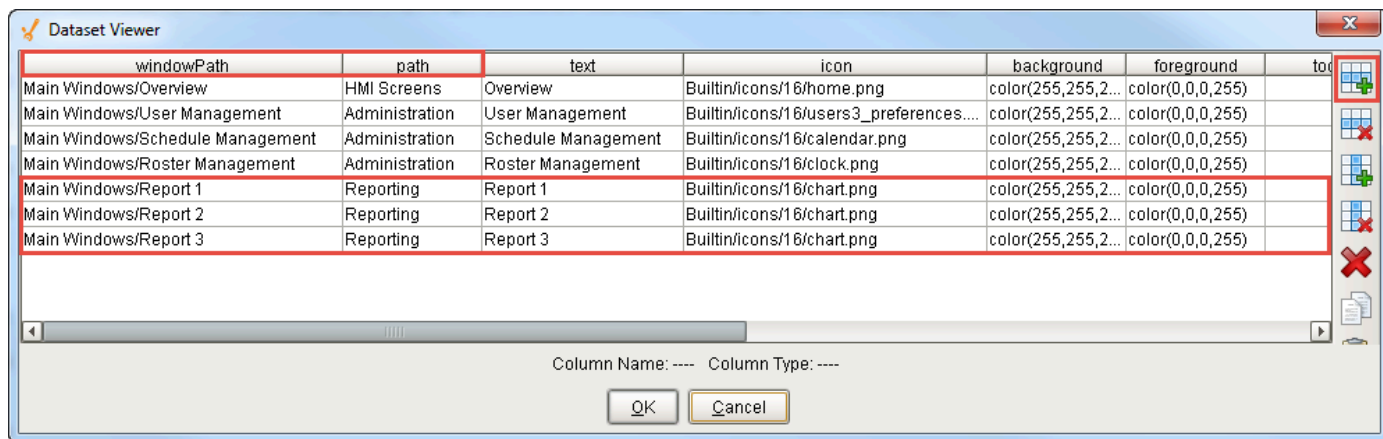
This brings up a **dataset editor** as shown in the screenshot below. You will see a number of columns that identify how each tree view item is displayed. Each row corresponds to a node in the tree view. The **windowPath** column is the path to the window you want to open when a user double clicks on the node. You can also change the **path** column to organize the node in a different folder.

i MAIN WINDOWS ALREADY CREATED

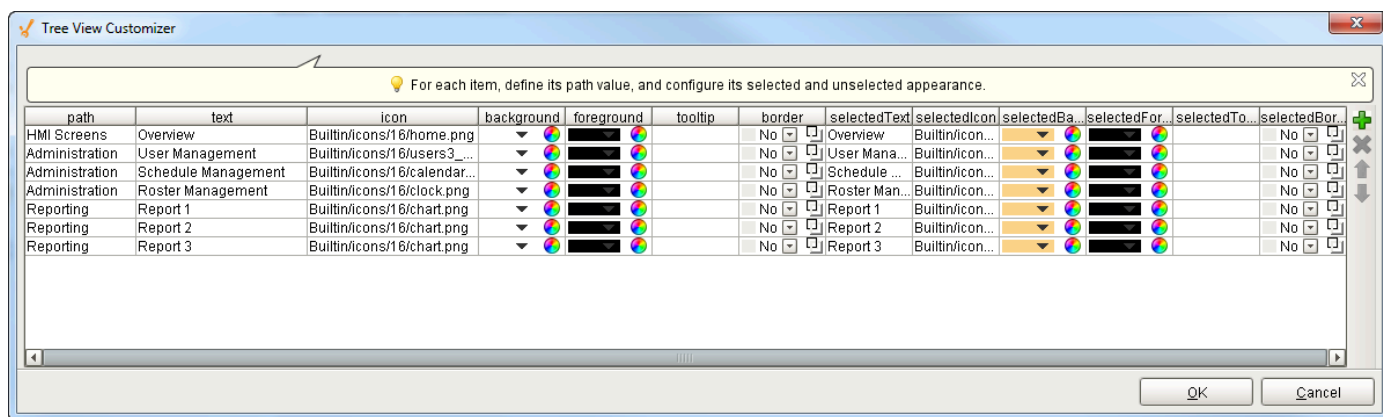
This step assumes you already have your main windows created in your Project Browser.

To add an item to the tree view window, click the add row icon  and enter the appropriate information for your new item in each column of the Dataset Viewer. When you're finished editing,

press **OK**.



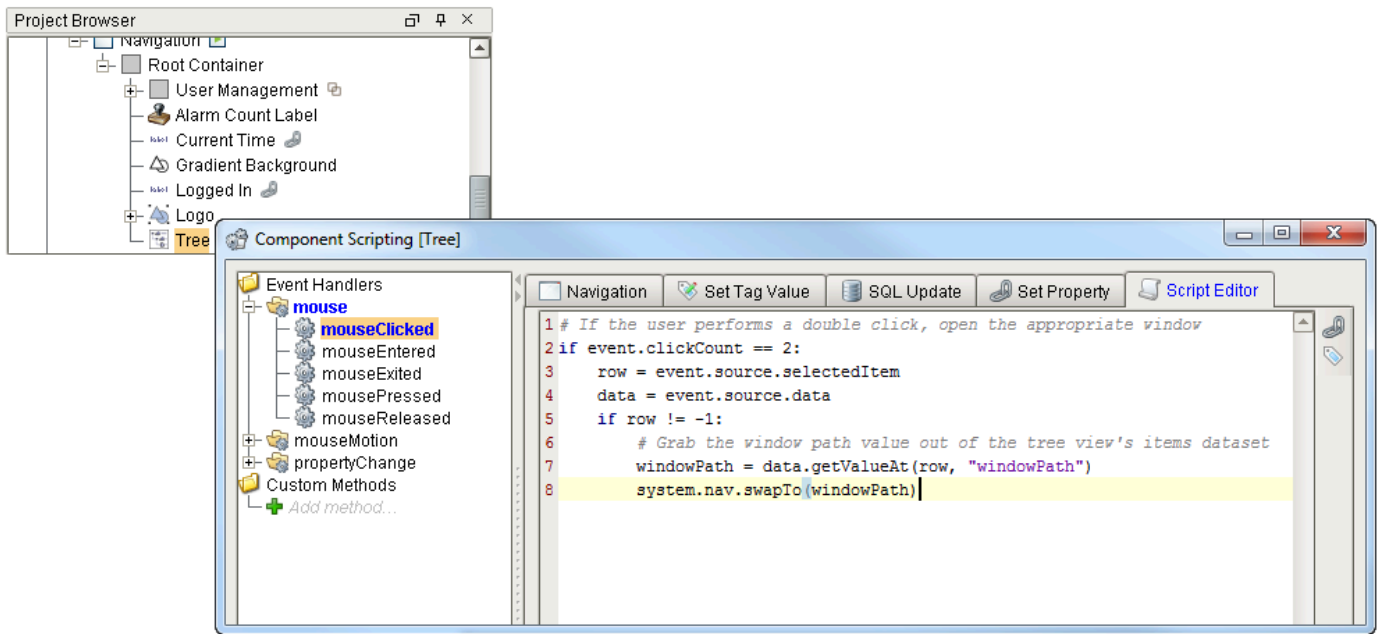
You can also edit the dataset by right-clicking on the Tree View component and choosing **Customizers > Tree View Customizer**. You'll notice that when you open the Tree View Customizer it doesn't have the windowPath column. If you use the Customizer to add or edit your windows, you will also have to go to the Dataset Viewer to edit the windowPath column. It's quicker to use the Dataset Viewer to manage your windows where all columns are visible.



3. The **Tree View** component uses a script that executes when the user double-clicks on one of the nodes in the **Tree View** component to navigate to various windows. Right click on the Tree View component, scroll down to **Scripting**, and select the **Script Editor** tab to view the script. The script is executed on the **Tree View's mouseClicked** event handler.

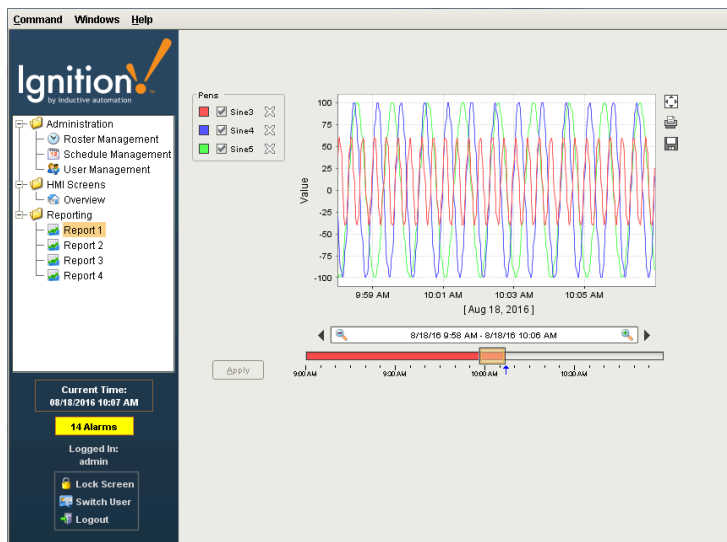
mouseClicked code

```
# If the user performs a double-click, open the appropriate window
if event.clickCount == 2:
    row = 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)
```



4. To test your window navigation, go to **Preview Mode** and switch between windows.
5. **Save and Publish** your project.
6. Give it a try! Open your project in the **Client**, and double click on the various Tree View items to swap between windows.

Tree View - Report 1



Tree View - Report 2

Active Time	Display Path	Current State	Priority
8/18/16 8:15 AM	WriteableWritableInteger1...	Active, Unacknowledged	Critical
8/18/16 8:15 AM	Machine C	Active, Unacknowledged	Critical
8/18/16 8:15 AM	gensimWritableWritable...	Active, Unacknowledged	Critical
8/18/16 2:20 PM	Machine B	Active, Unacknowledged	Critical
8/18/16 8:15 AM	AlarmsProdSupAttention/AL...	Active, Unacknowledged	High
8/18/16 10:08 AM	Ramp	Active, Unacknowledged	High
8/18/16 9:15 AM	Machine B	Active, Unacknowledged	Medium
8/18/16 9:15 AM	Machine B	Active, Unacknowledged	Medium
8/18/16 2:23 PM	Machine B	Active, Unacknowledged	Medium
8/18/16 10:08 AM	Ramp	Active, Unacknowledged	Medium
8/18/16 9:15 AM	AlarmProdSupAttention/Alarm 1	Active, Unacknowledged	Low
8/18/16 8:15 AM	WriteableWritableInteger10/Alarm	Active, Unacknowledged	Low
8/18/16 12:10 PM	WriteableWritableInteger2/Alarm	Active, Unacknowledged	Low
8/18/16 12:10 PM	WriteableWritableInteger2/Alarm	Active, Unacknowledged	Low

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

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

Navigation - Back and Forward Buttons



INDUCTIVE
UNIVERSITY

**Navigation - Back and
Forward Buttons**

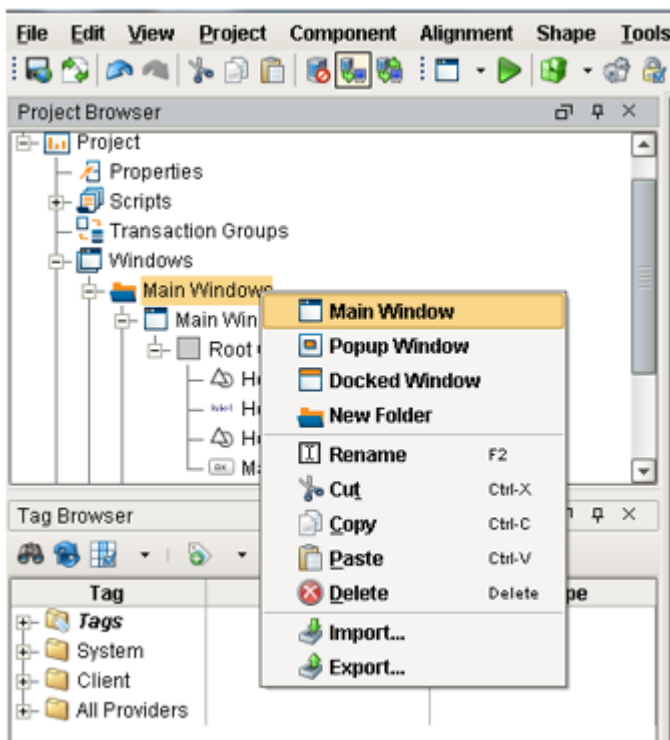
[Watch the video](#)

Another navigation strategy in Ignition is to have Forward and Back Buttons on your main screen to navigate between different windows. It does not have a docked window, tree view, or tabs to navigate around. This strategy is perfect for a small business process with ordered steps. It is one big main window and has buttons to navigate forward and back from one Main Window to the next in the list.

Setting Up the Back and Forward Buttons

This navigation strategy is very simple to setup in the Designer. You can start by creating your project using a [blank template](#).

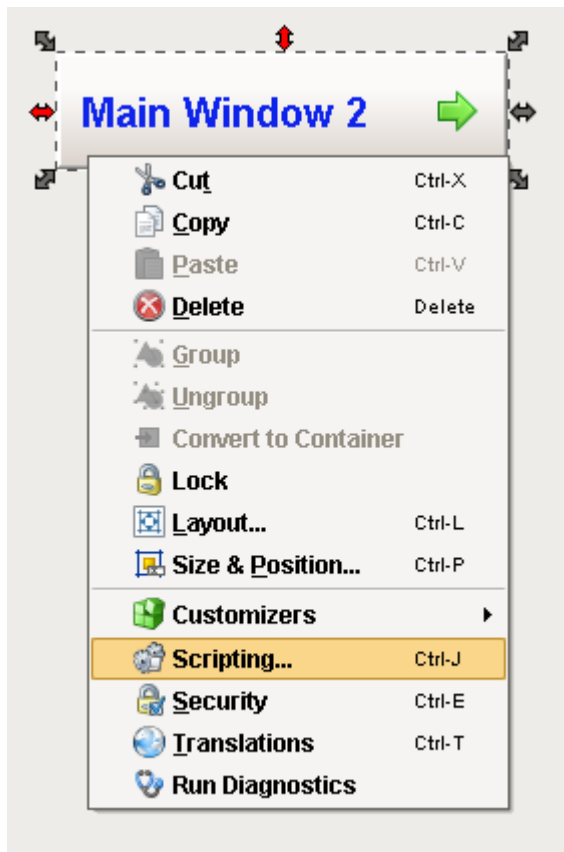
1. In the **Project Browser**, right click on the Main Windows folder and select **Main Window**. You have the option of renaming your Main Window to whatever best describes your window. In this example, it's called **Main Window 1**. You don't need to have a header on each window, but for this example, a label was added for clarity and called Main Window 1. Repeat this step to create Main Window 2 and Main Window 3.



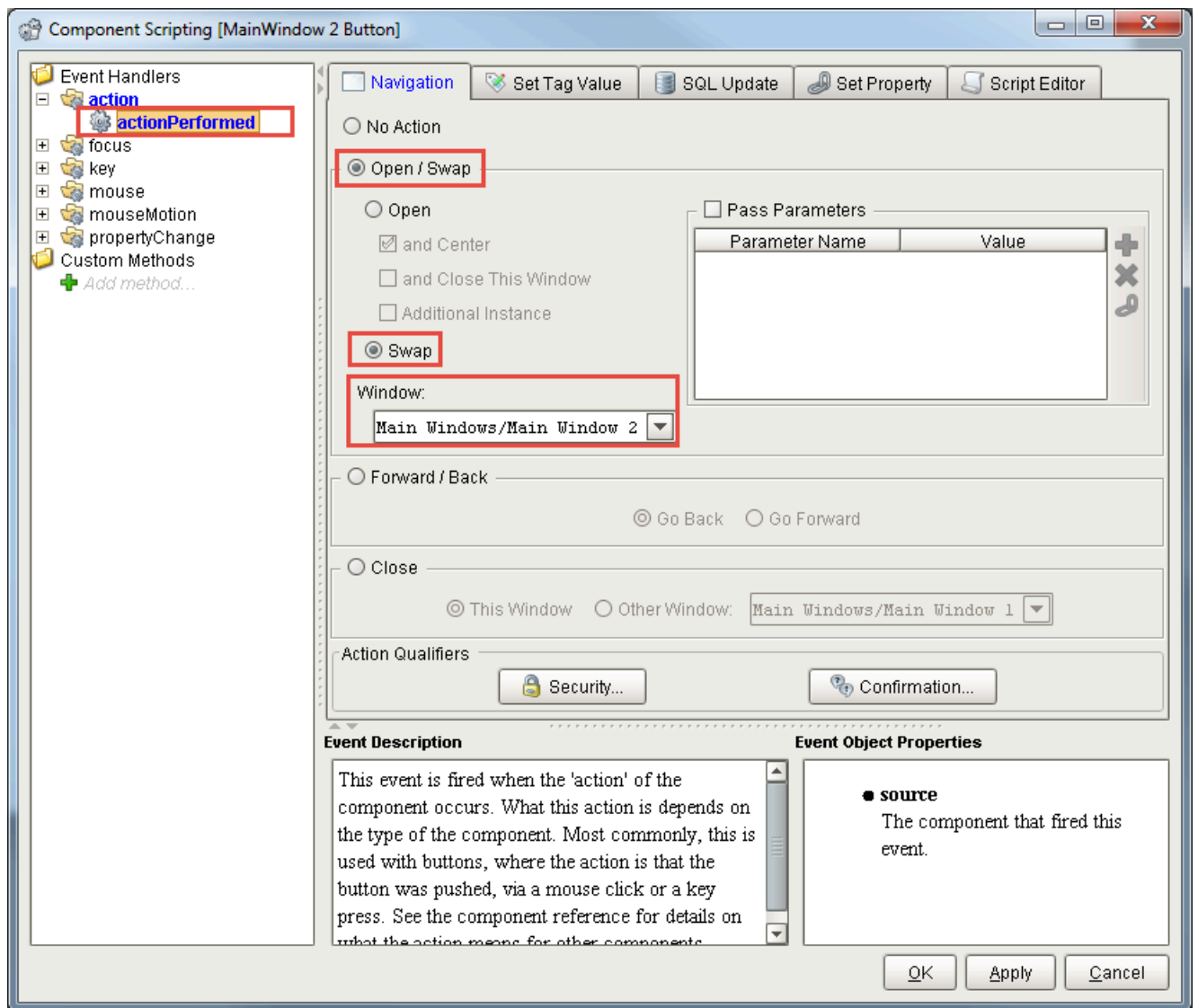
- From the **Component Palette** in Designer, select and drag a **Button** component to your Main Window 1. Label the button, **Main Window 2**. In the **Property Editor**, add a right green arrow to your button component from the **Image Path** property. To get the arrow to move to the right side of the button, set the **Horizontal Text Position** property to **Left**.

Layout	
Margin	Top: 0 Left: 0 Bottom: 0 Right: 0
Horizontal Alignment	Center
Horizontal Text Position	Left
Vertical Alignment	Center
Vertical Text Position	Center

- Now you need to tell the Main Window 2 button what to do when someone clicks on it. Right click on the Main Window 2 button and select **Scripting** and the component scripting dialogue box will open.

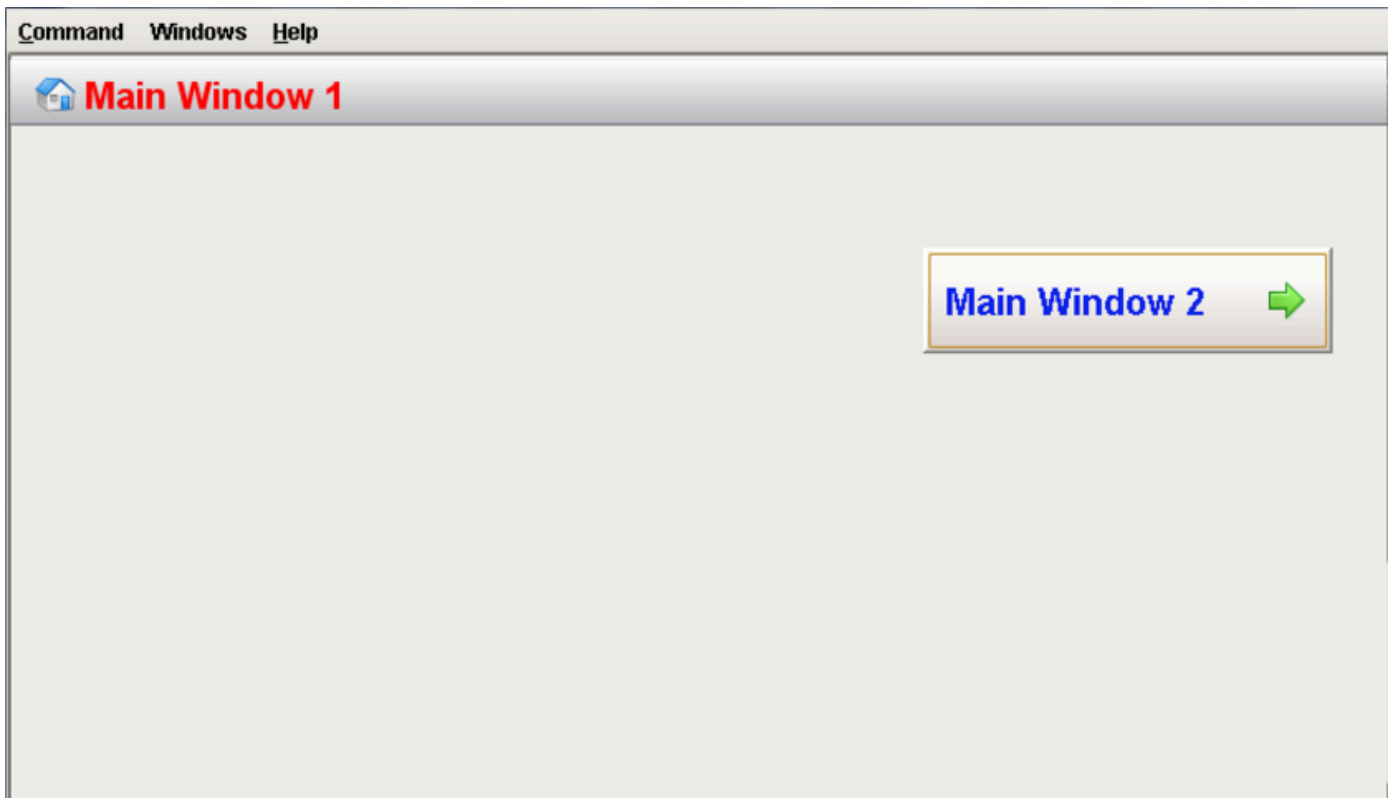


4. Under the Event Handlers folder, open the action folder and select **actionPerformed**.
 - i. Click the **Open/Swap** and **Swap** radio buttons. The **Swapping** function is used to put buttons on windows for easily building simple scripts to go back and forth between different main windows. The Swap function ensures only one main window is open at a time.
 - ii. From the **Window dropdown box**, select **Main Windows/Main Window 2**. Click **OK**.



5. **Save and Publish** your project.

6. Open your **Client**. Click on the **Main Window 2** button and Ignition swaps the Main Window 1 with Main Window 2.



7. Repeat Steps 2 and 3 to create Main Windows 2 and 3 respectively. Make sure for Main Window 2, you have a button for both Main Windows 1 and 3. Your window path for the Main Window 1 button is set to Main Windows/Main Window 1. For the Main Window 3 button, set the window path to Main Windows/Main Window 3.
8. In the following example, each button navigates you to the next main window. Click on the **Main Window 1** button and Ignition navigates you to Main Window 1. It swaps out Main Window 2 with Main Window 1 so you always have one main window open. Click on the Main Window 3 button and Ignition swaps out Main Window 2 with Main Window 3.



Main Window 1

Main Window 3



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

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

Navigation - Drill Down



INDUCTIVE
UNIVERSITY

Navigation - Drill Down

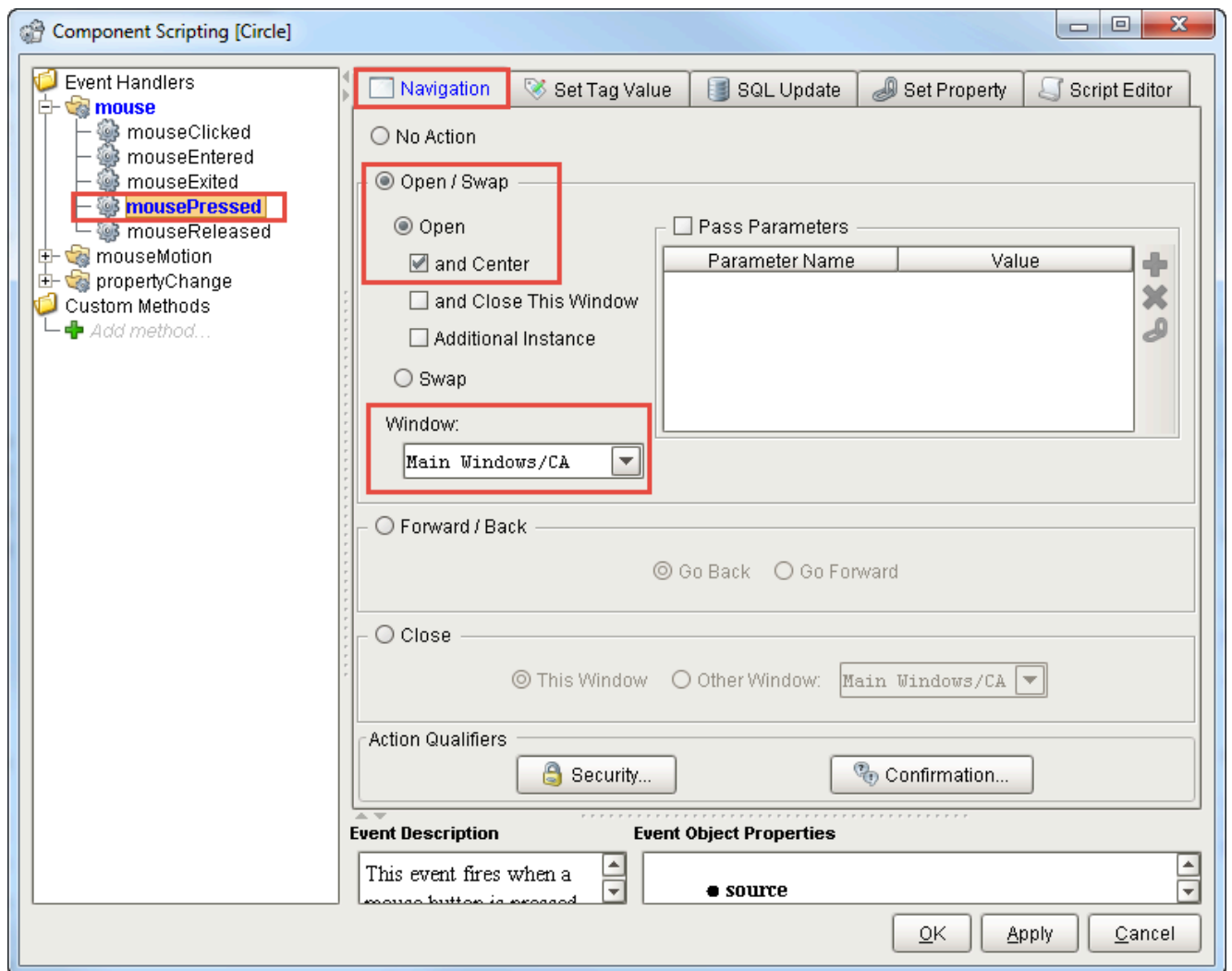
[Watch the video](#)

Another popular navigation strategy in Ignition 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 like a map or a background image representing a plant. The image has various components such as boxes, circles, etc., that overlay 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 map or plant.

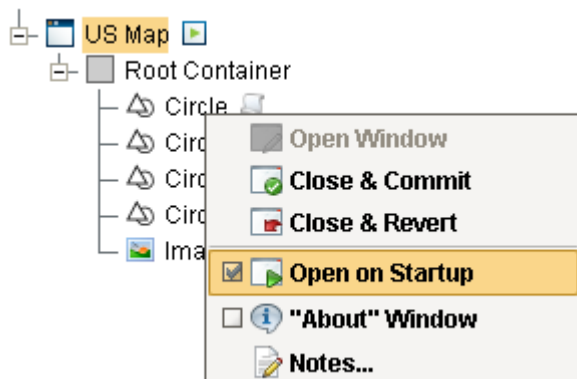
It is a very simple navigation strategy to setup and is popular because it lets people orient themselves on where they are located in the factory, and they can drill down into all these different sections very easily from one place.

To Create a Navigation Drill Down in Designer

1. Drag an **image** on to a Main Window (this example uses a US Map).
2. Create a component such as a box, circle, star, etc., over an area on your image.
3. Right click the component (box, circle, etc.,) and select **Scripting**.
4. Select the **mousePressed** event handler, and with the **Navigation** tab selected:
 - i. Select the **Open/Swap** radio button.
 - ii. From the **Window** drop-down box, select the window that relates to the selected area.
 - iii. Click **OK**.



5. Select your US Map window and set it to **Open on Startup**.



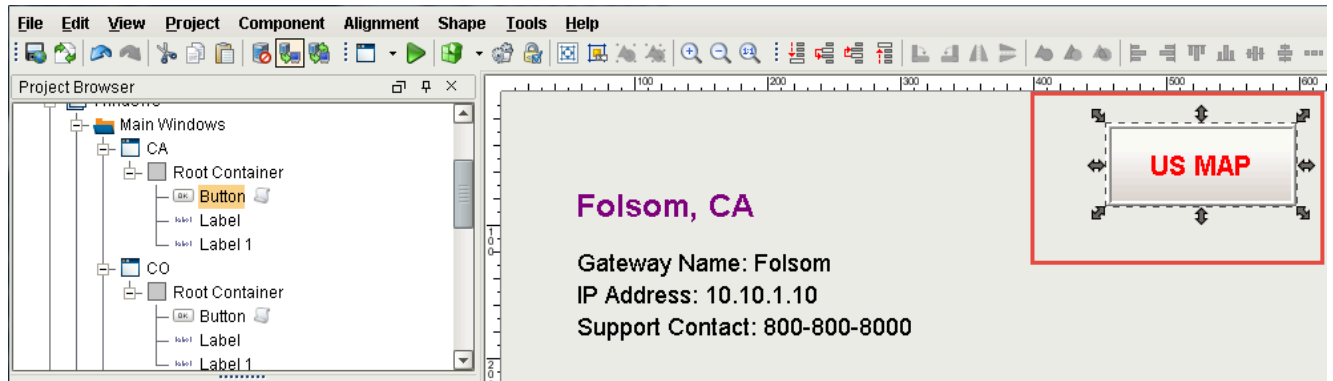
AREA WINDOWS ALREADY CREATED

This example assumes you already have your area windows created in your Project Browser.

6. On your area window, add a **Button** component to navigate you back to the main window (if you have multiple area windows, copy and paste this button to each window).

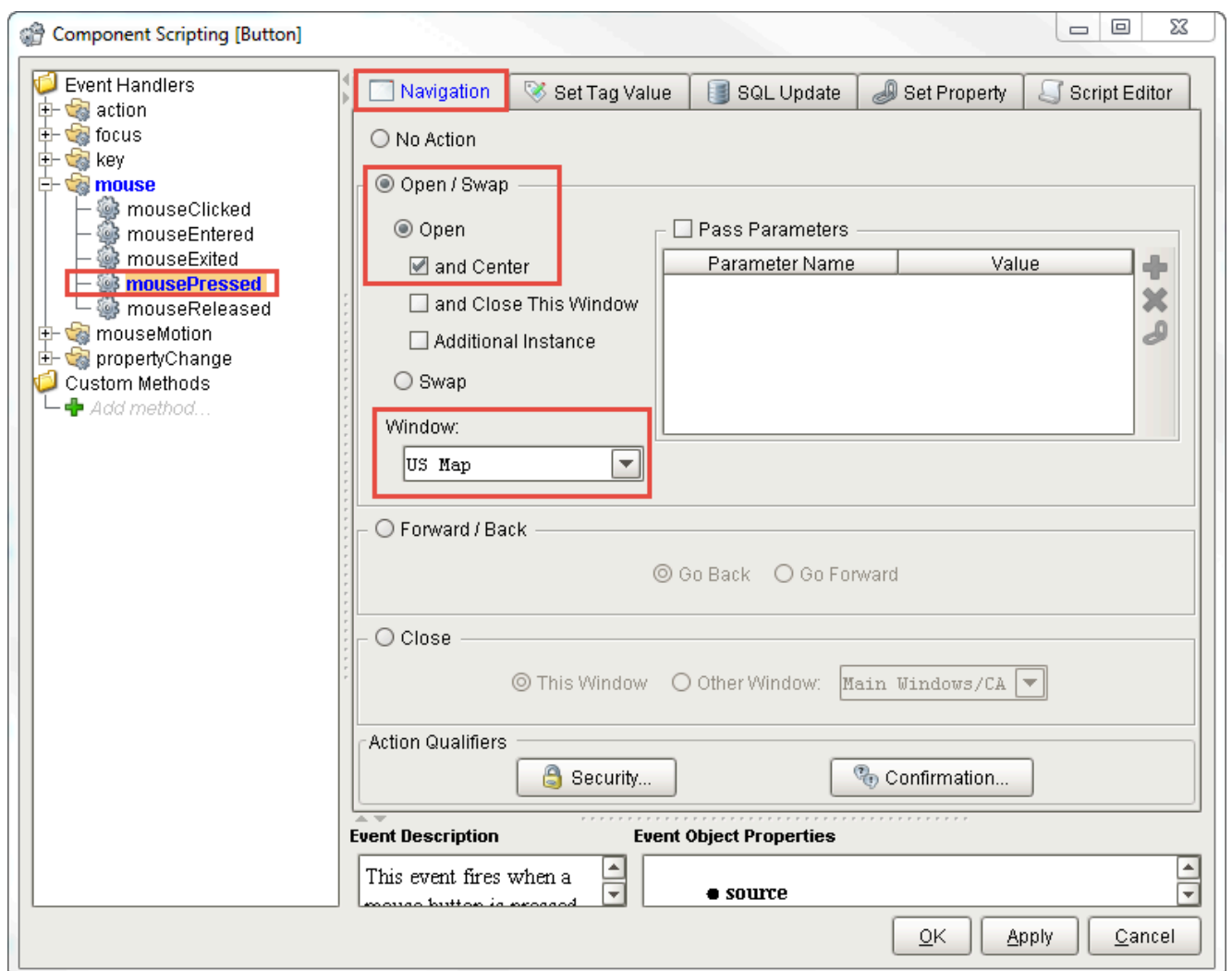
i. **Open** your area window.

- ii. Add a **Button** component.
- iii. Right click on the **Button** component and select **Scripting**.



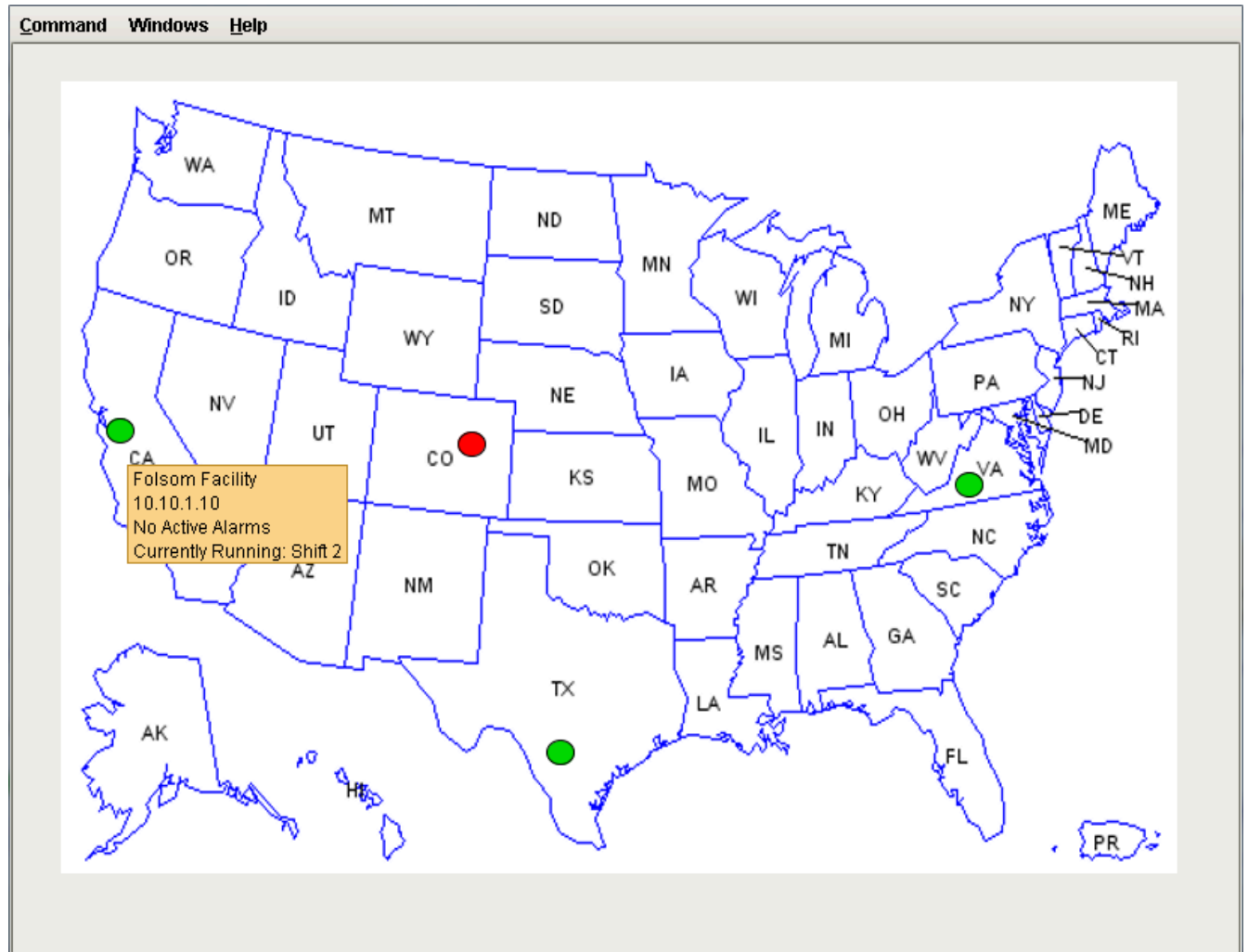
7. Select the **mousePressed** event handler, and with the **Navigation** tab selected:

- i. Select the **Open/Swap** radio button.
- ii. From the **Window** drop-down box, select the window that is your overview map.
- iii. Click **OK**.



8. To test your window navigation in the Designer, use the **Preview Mode** to switch between windows.
9. **Save and Publish** your project.
10. Now, try it out! Open your **Client**, and click on your special shape to navigate to the area selected. You'll notice that in the following example the Mouseover Text property was used to display the location information when you hover over one of the circles.

Drill Down Navigation with Mouseover Text



11. Click on the **US Map button** to take you back to the US Map.

Area Window with US Map Back Button

Command Windows Help

Folsom, CA

Gateway Name: Folsom

IP Address: 10.10.1.10

Support Contact: 800-800-8000

US MAP

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

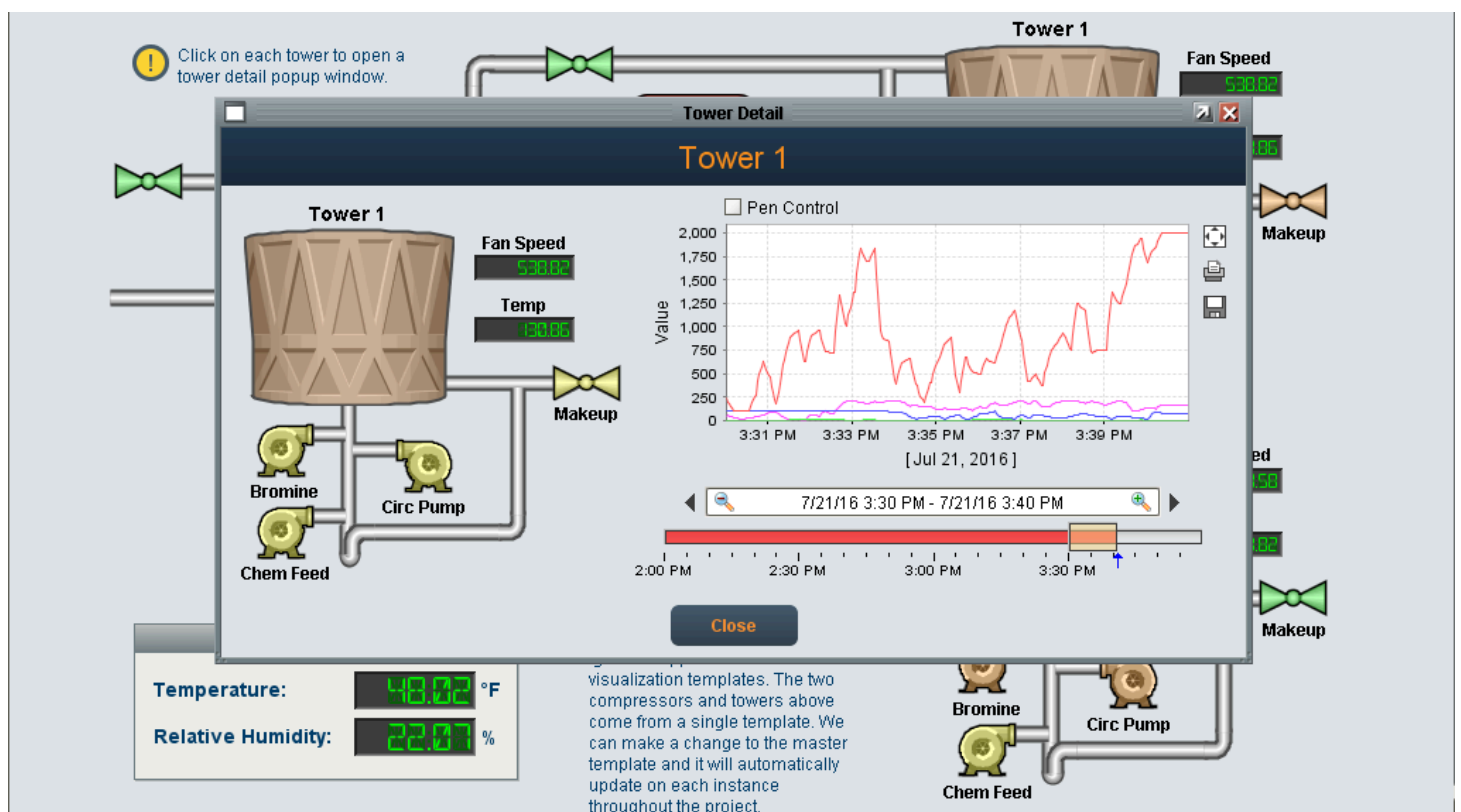
For documentation on the latest stable release, see the [latest version](#) (8.3).

Popup Windows

About Popup Windows

Popup windows are great for displaying additional information about a selected item on the screen. 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 or zoom into a specific area, and more.

The great thing about Popup windows is they can also be parameterized so they can be re-used. For example, a parameterized popup window can display graphical historical information related to the tags of your components. One popup window design can be re-used for many components as long as the proper information is passed to the popup window. Just define custom parameters for your popup window, and when you open an instance of it, you simply modify its parameter values.



Popup Window Designs

Ignition provides several different Popup Window designs you can use depending on your requirements. It can range from a simple popup window to a more sophisticated popup window

passing parameters, and even entire UDTs into a window. What's nice about parameterized popup windows is they are designed to be re-used, thus, reducing development time. So when you create or update your components in your popup window, the changes will extend to all your all your instances.

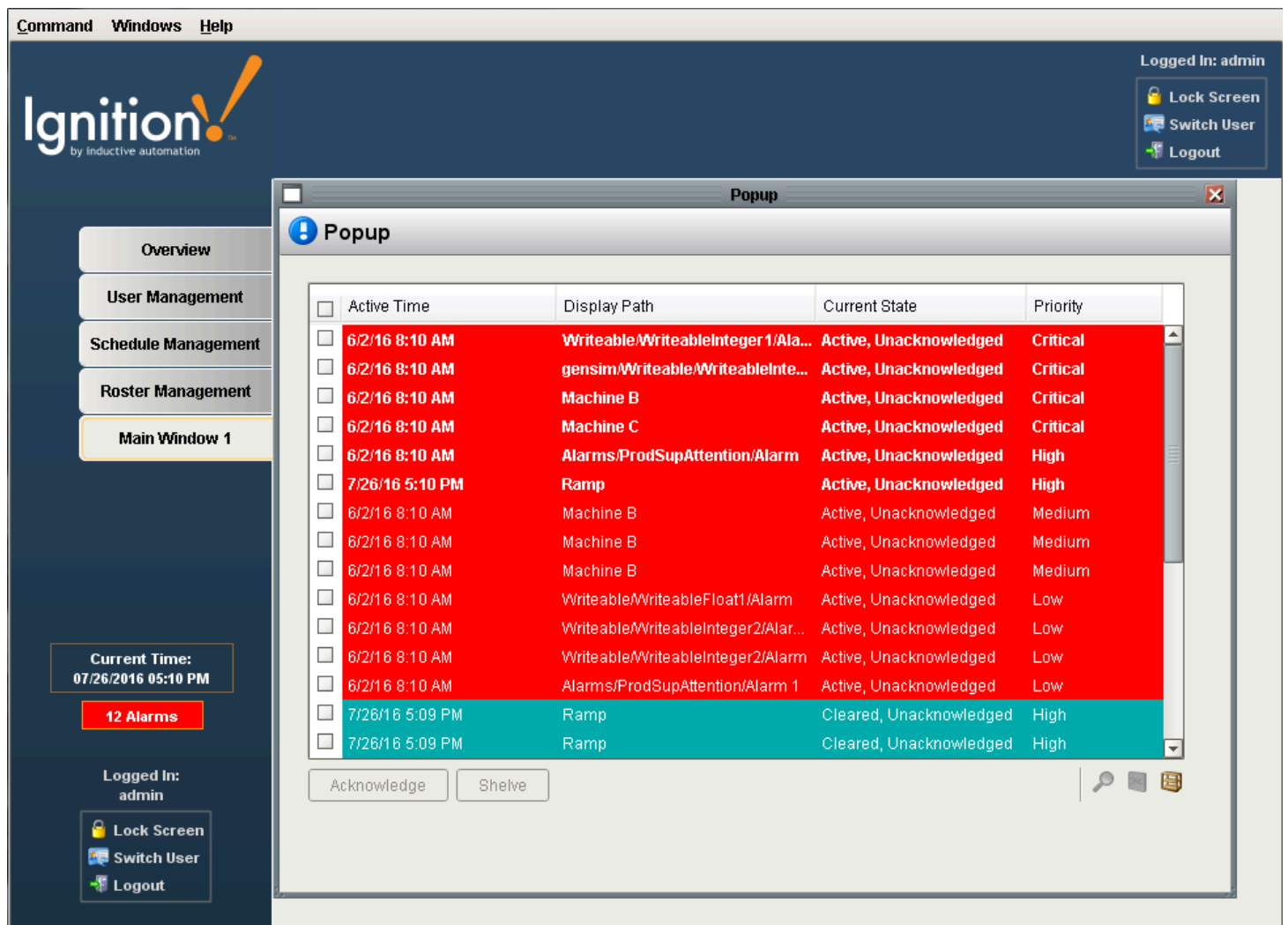
Here's a brief description of the available Popup Window designs:

- **Popup Windows** - displays additional information about a selected component or item on top of the main screen.
- **Parameterized Popup Window** and **Passing UDTs** - displays graphical historical information related to tags of a component passing parameters from one window to another window using Custom properties.
- **Multiple Instances of a Popup Window** - used to open multiple instances of the popup window for the component or item selected.

Opening a Popup Window

Before you can open a popup window, you have to create it. In the Designer, under the Window section, you can **add a popup window** to any folder. 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 titlebar and border in the window. You can open a popup window from another window such as a Main Window, or a component such as a button, label, image or whatever you like.

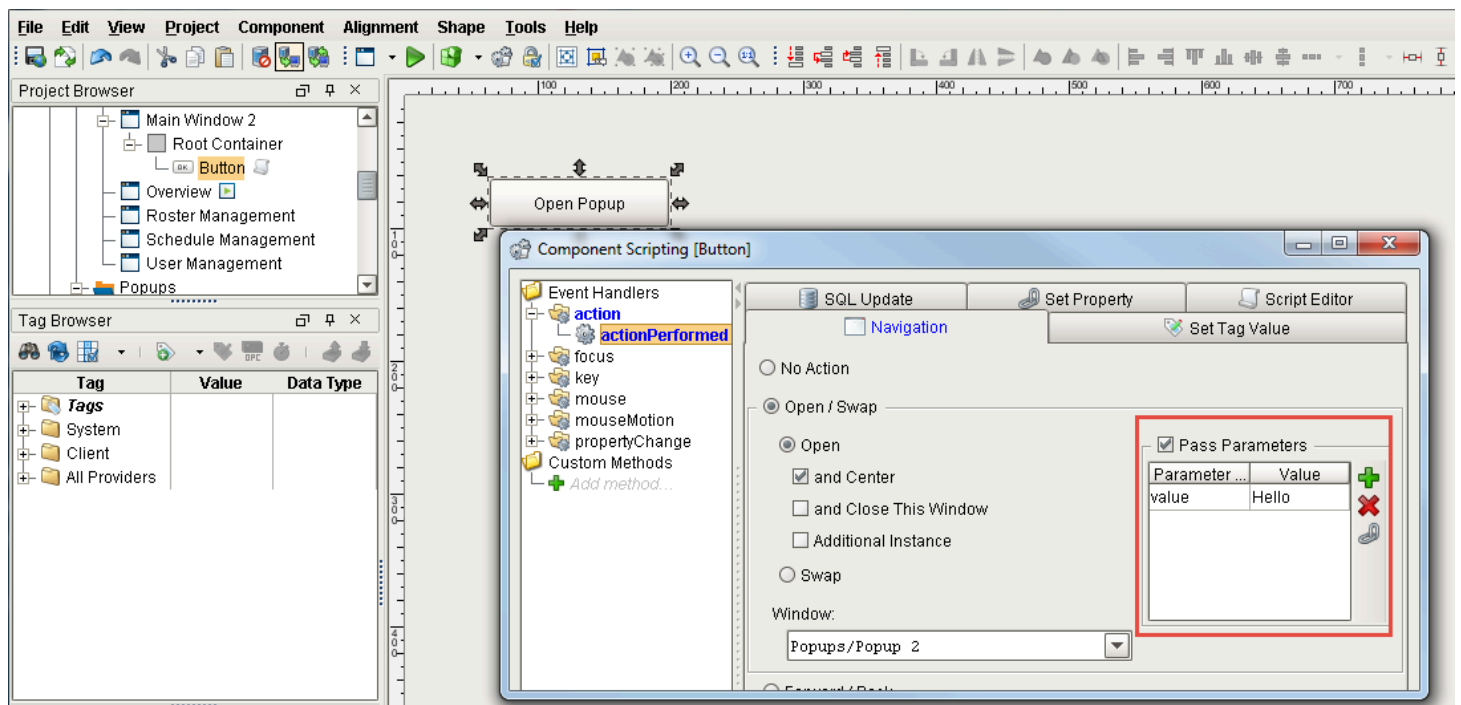
In any Main Window, you can add a script to any component to open your popup. Just tell the component what you want it to do when you click on it. In this example, a popup window was opened containing an alarm table using a Button component in Main Window 1. For all the details on creating and opening popup windows, refer to the section on **Open Popup Windows**.



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. Parameterized popup windows can use basic Tag types and as well as UDTs. Parameterized popup windows can be re-used as long as the proper information is passed into the popup window.

Once you create a custom property on the popup window that you can pass a value into, go to your Main Window and create a script to open up your popup window. In the Component Scripting window, check the **Pass Parameter** box, enter your parameter name from the dropdown, and the value you want to pass.



That's it! **Save and Publish** your project. Open your **Client**, click the button, and your Popup Window will open. To learn more, see [Parameterized Popup Windows](#) and [Parameterized Popup Windows and UDTs](#).

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**'. Then, click on each Tank in your window, and a popup window containing additional information for each tank will be displayed.

To learn more, go to [Popup Window - Multiple Instances](#).

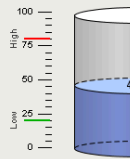
- Overview
- User Management
- Schedule Management
- Roster Management
- Tank 100
- Tank Details
- Main Window 11

13 Active Alarms

Logged In:
admin

Lock Screen
Switch User
Logout

Tank 100

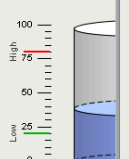


Temperature:



Temperature:

Tank 101



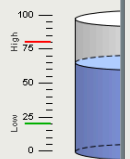
Temperature:



Temperature:

Tank 101

Tank 102

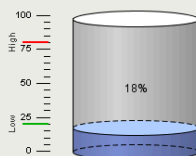


Temperature:



Temperature:

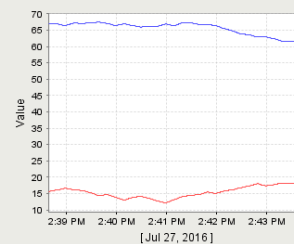
Tank 103



Temperature:

62.36

- Pens
- ☒ level_PV
 - ☒ temp_PV



Apply

Last: 5 Minutes

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

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

Open Popup Window



INDUCTIVE
UNIVERSITY

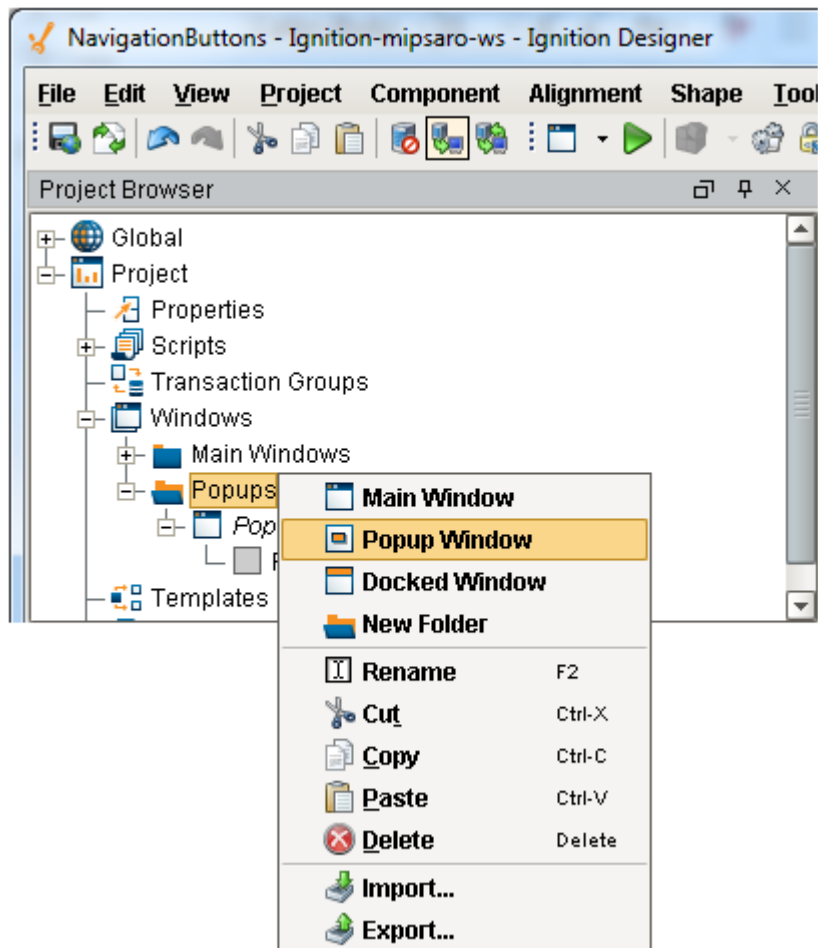
Open Popup Window

[Watch the video](#)

Creating a Popup Window

In order to open a popup window, you first have to create a **Popup Window**. You can add a popup window to any folder in the Designer.

1. In this example, a Popup folder already exists so we will right click on the Popup folder to create a new '**Popup**' window. Right click on the **Popup window** then rename it to '**Popup**' or give it an appropriate name.

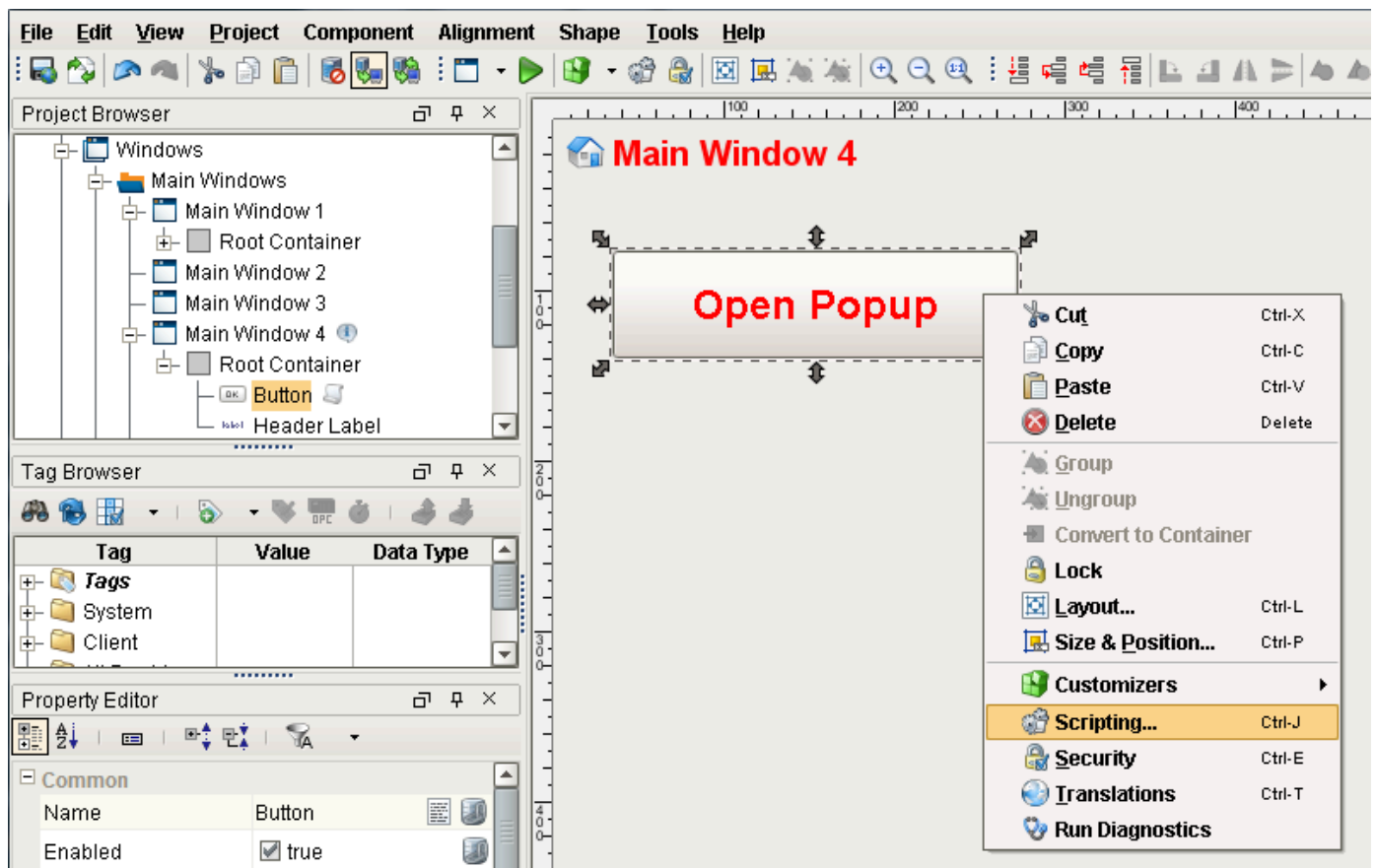


You can make the popup window as big or small as you want. You can also set properties in the Property Editor by making the window Closable, Resizable, and giving it a Title. You can even setup the Border or Title Bar to show-up on the window.

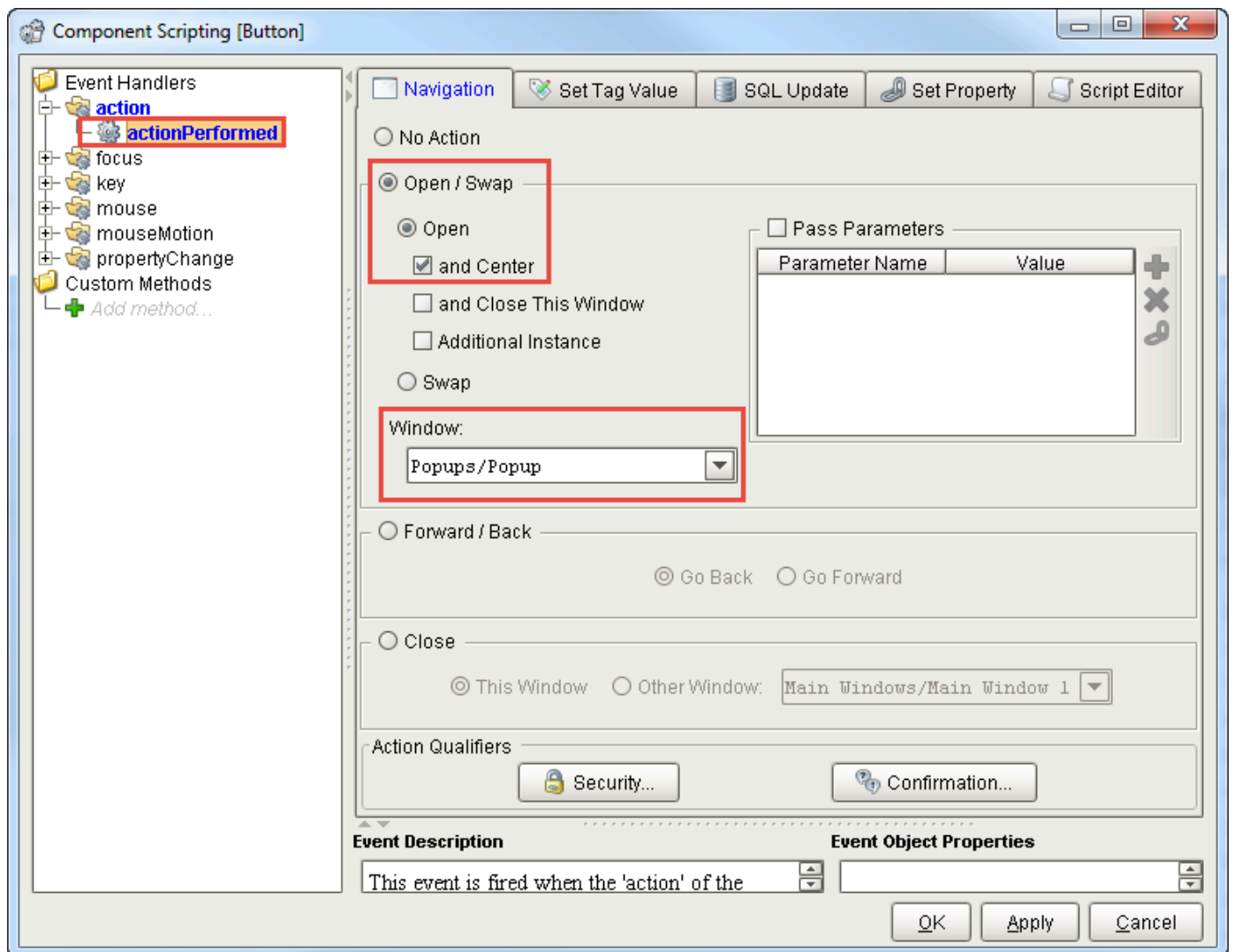
Once the window is created, you can open up the popup window from any other window. You can use a button, a label, an image or any component you like to open up a popup window.

2. This example opens a popup window from a Main Window using a Button component. In any Main Window, **drag a Button component** from the **Component Palette** to your window. Do a slow double click on the word '**Button**' or select the **Text** property and change the text to '**Open Popup**'.

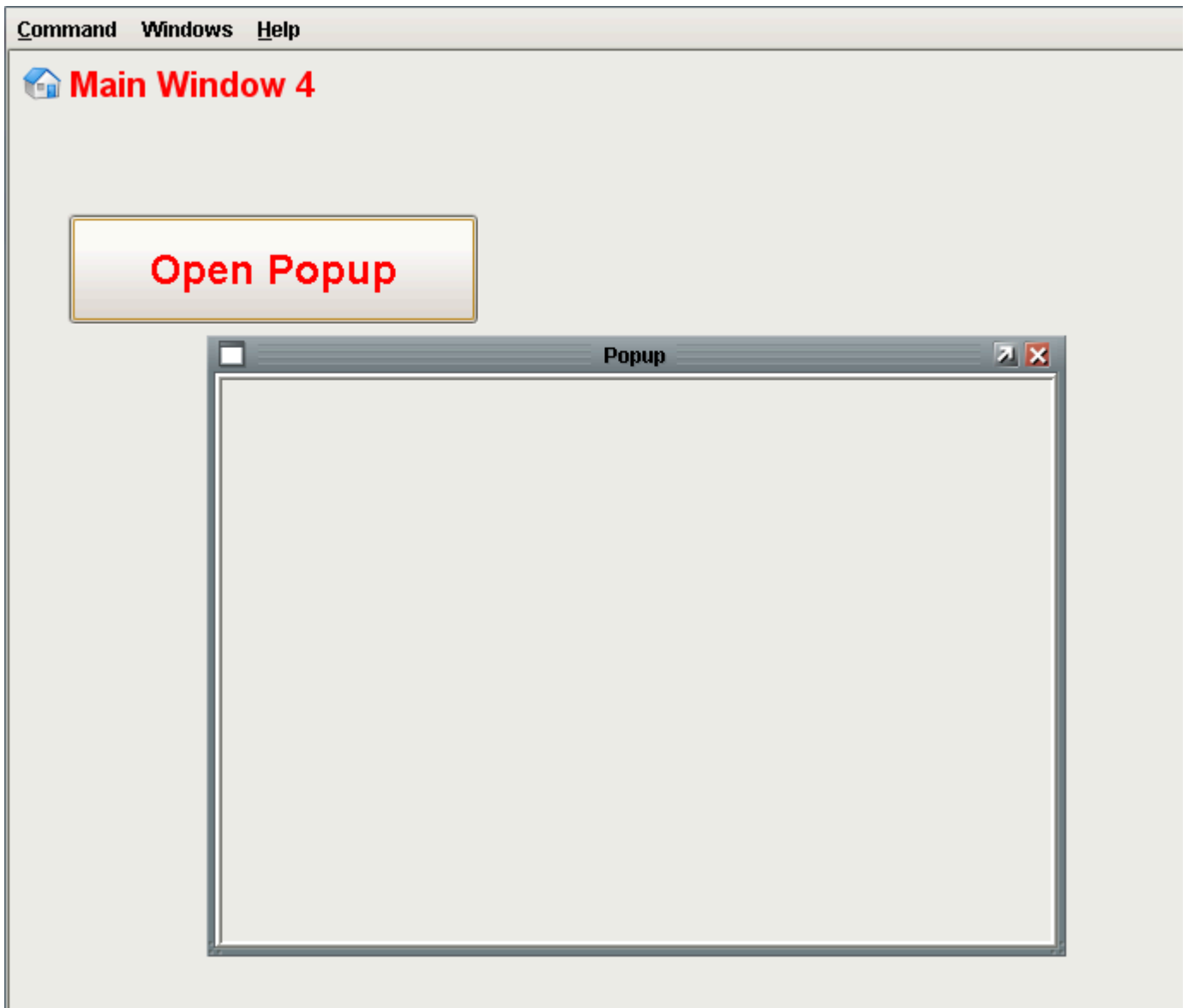
Now you need to tell the Open Popup button what you want it to do when you click it. Select the **Open Popup Button**, right click and select **Scripting** from the dropdown.



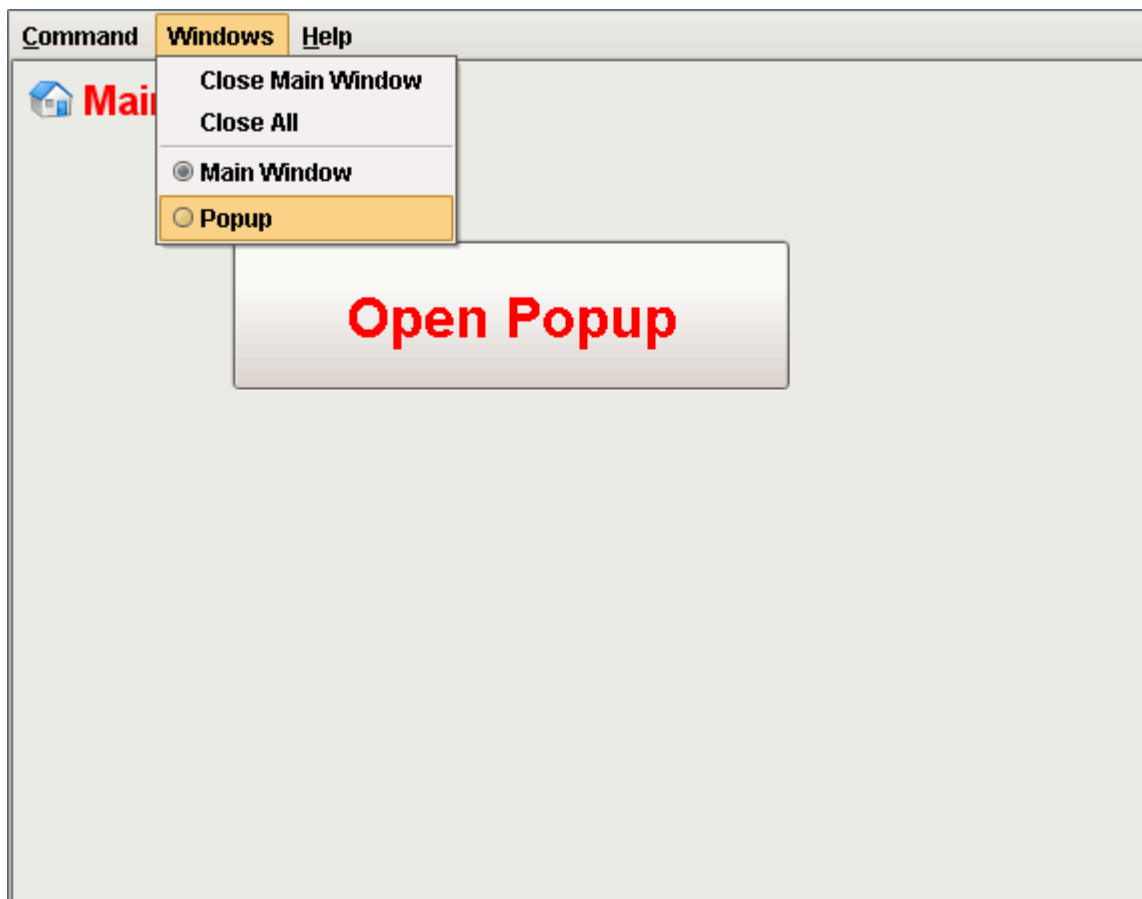
3. The Component Scripting dialogue box will open. Select **actionPerformed**, and click the **Open/Swap** then **Open and Center** options on the Navigation tab. From the **Window** dropdown box, set the window path to **Popups/Popup**. Click **OK**. **Save and Publish** your project.



4. Open your **Client** and click the **Open Popup** button on your Main Window. It opens the Popup window which you can move around, make it bigger, and close it.

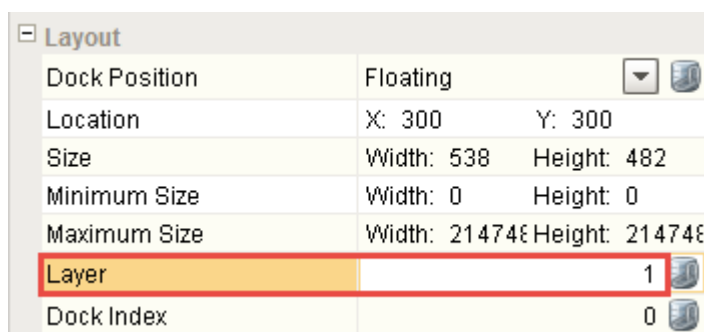


5. Another behavior of the Popup window occurs when you open up the popup window and click on your main window (in this example, Main Window 4). It looks like the popup window disappears, but it's actually behind your main window. Go to **Windows** on the menu bar and click **Popup** to bring the popup window back to focus. The **Windows Menu** will show you all the open windows in your Client.

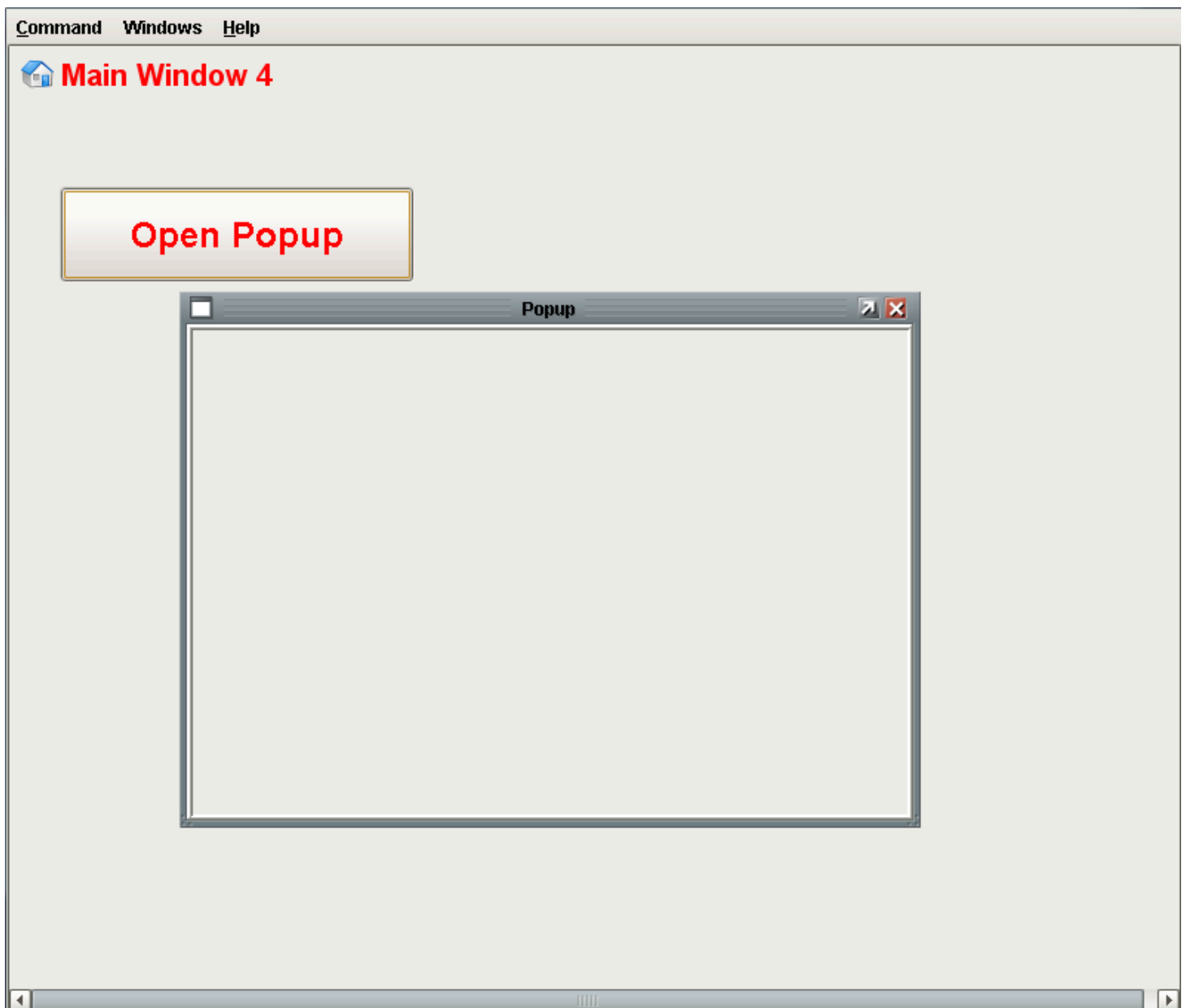


Popup Window Properties

There is a property on the popup window that will force the popup window to stay on top of the Main Window. In the **Designer**, click on your popup window in the Project Browser (not the Root Container). Go to the **Property Editor** and click the **Layer** property. The higher the Layer number, the more on top it will be. If your 'windows' Layer properties are all set to 0, they are all going to share the same layer. They can go back and forth over each other. If you set the Layer number to **1**, the popup window will always be on top. **Save and Publish** your project.



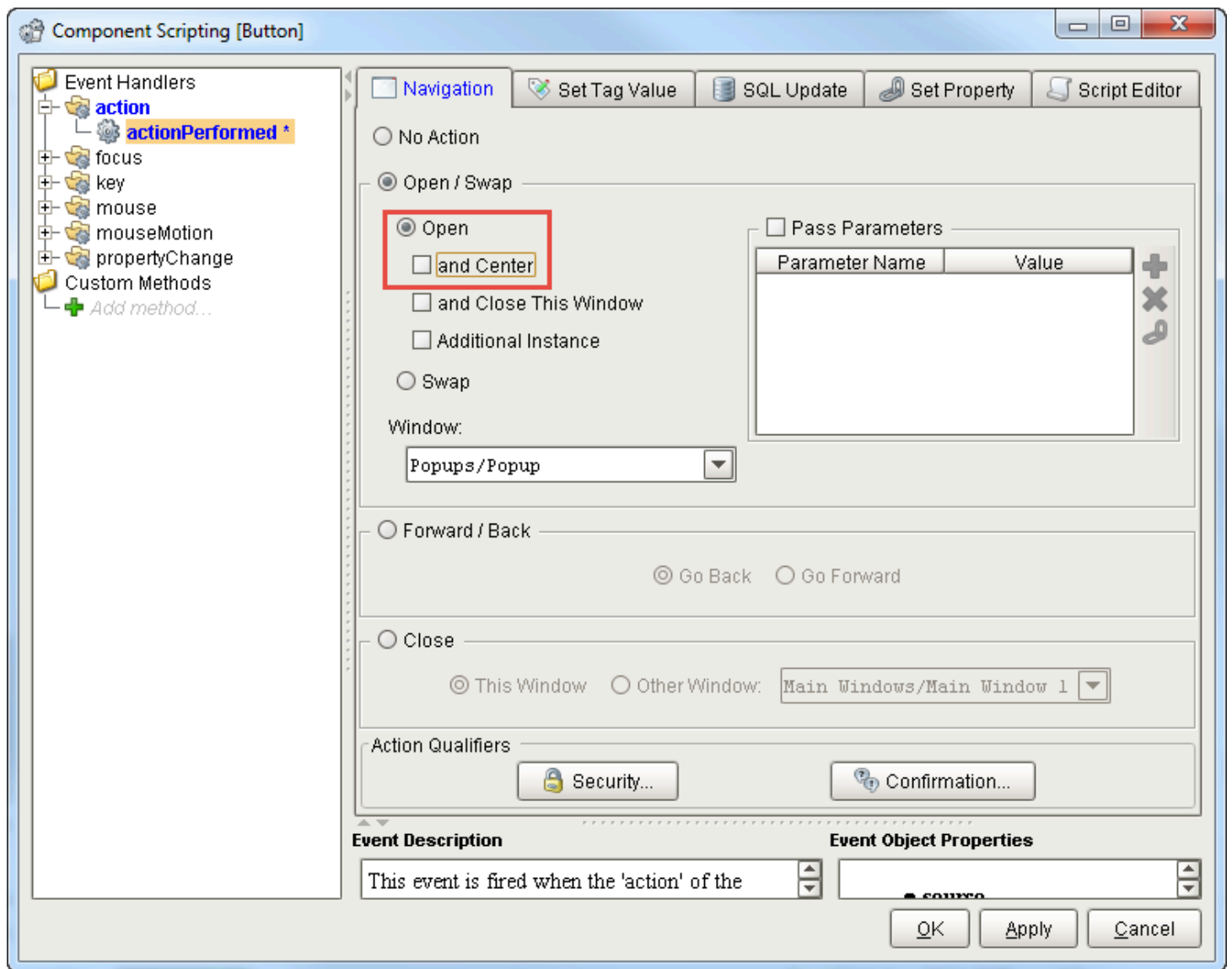
After you update the **Client**, you will see your popup window on top.



It is also possible for you to open and set a specific location where you want the Popup window to be located. In **Designer**, click on your **Popup** window and go to the **Property Editor**. Click on the **Location** property and set to **X to 300** and **Y to 300** pixels.

Layout		
Dock Position	Floating	
Location	X: 300	Y: 300
Size	Width: 538	Height: 482
Minimum Size	Width: 0	Height: 0
Maximum Size	Width: 214748	Height: 214748
Layer	1	
Dock Index	0	

Go to your Main Window, double click **Open Popup Button** to open the Component Scripting dialog box. Uncheck '**and Center**' so your popup window will automatically open up in that specific X and Y location setting.



Save and Publish your project. In your **Client**, click the **update** bar, and then click on the **Open Popup** button. The popup window will open at the X 300 and Y 300 location setting instead of opening in the center of the screen.

Main Window 4



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

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

Parameterized Popup Window



INDUCTIVE
UNIVERSITY

Parameterized Popup Window

[Watch the video](#)

Overview

A parameterized popup window lets you pass parameters from one window into a Popup Window, where the receiving window would display relevant information. Also, this allows you to maintain a single window that can be used to display similar information. For example, suppose you have 2 compressors: Compressor 1 and Compressor 2. Imagine clicking a compressor on your main window and a popup displays the diagnostic information about the compressor. What is passed is simply the compressor number, therefore a single diagnostic window can represent many different compressors.

Passing Parameters to a Popup Window

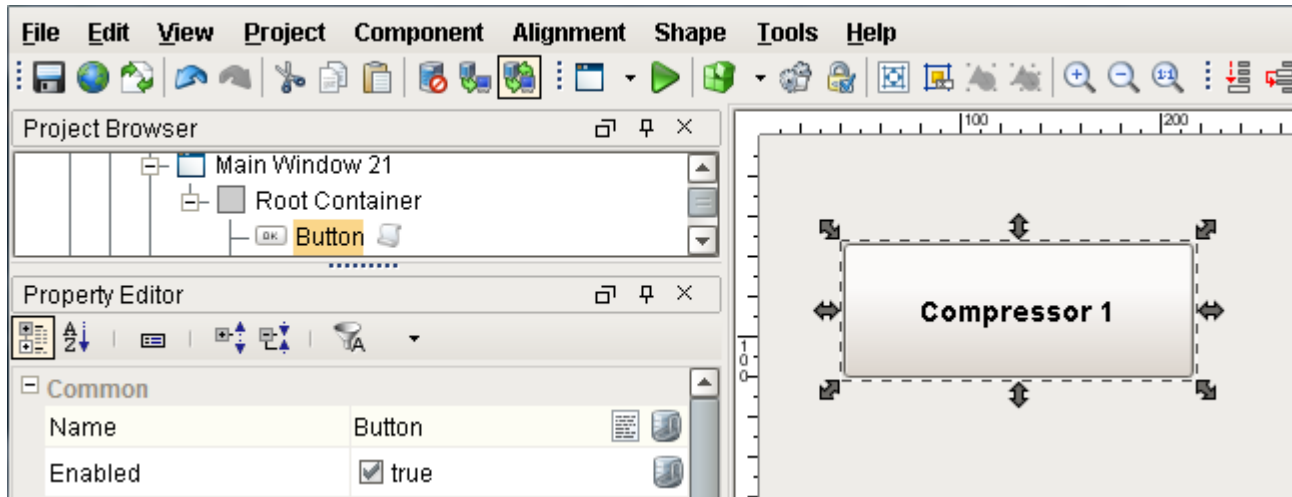
To pass parameters from one window to a Popup Window, the receiving Popup Window must have Custom properties that receive the passed parameters. When the event on the parent window is called, the parameters are passed to the receiving Popup Window's custom properties on its Root Container. The component's properties on the receiving window can use the Root Container's Custom properties to address their bindings.

This example shows 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.

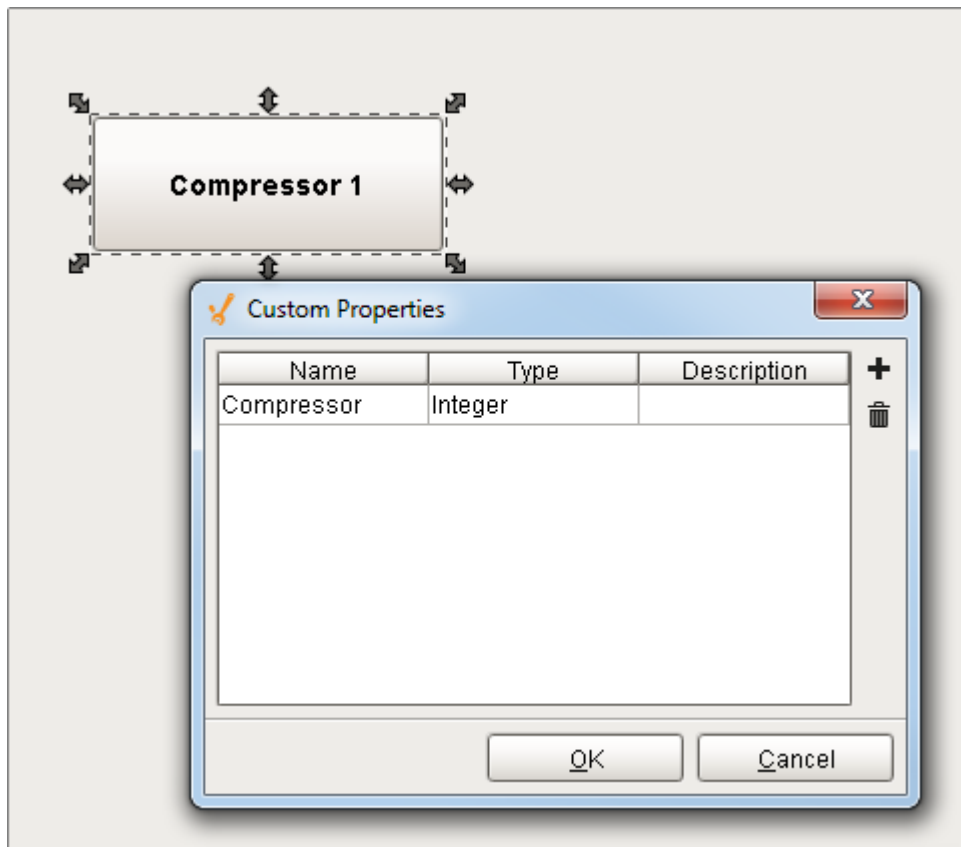
Setting up the Main Window

The Main Window uses buttons to switch between compressors.

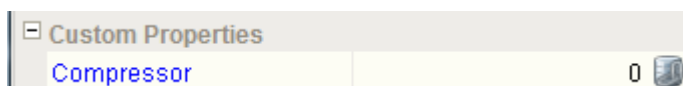
1. In a Main Window (parent window) drag a **Button** from the component palette to your window and call it '**Compressor 1**'.



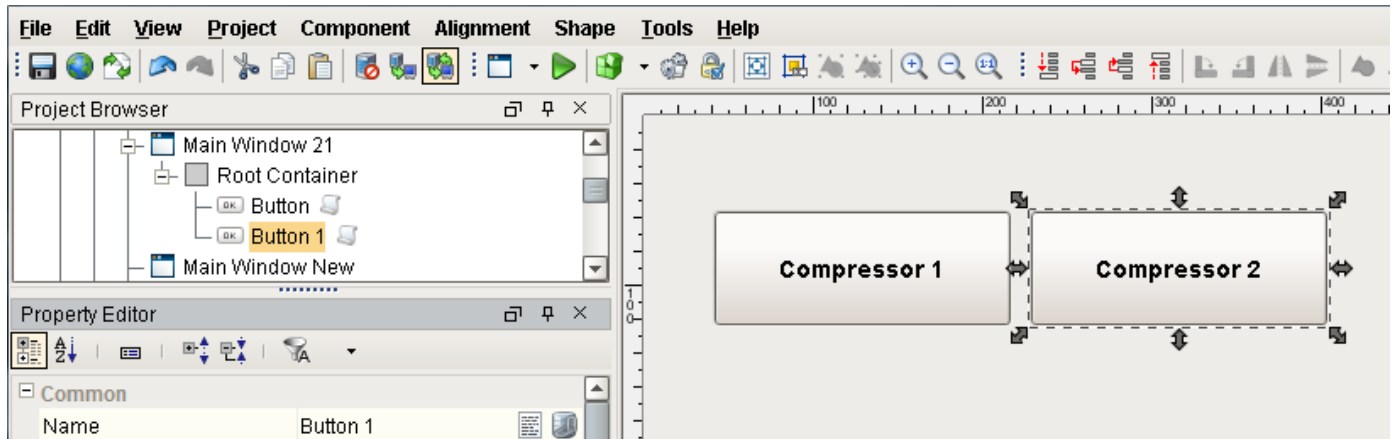
2. Add a custom property which will pass the compressor number to the popup window. Right click on **Compressor 1**, select **Customizers > Custom Properties**. Specify the property named '**Compressor**' with a datatype of '**Integer**' and click **OK**.



The **Compressor** property is now created and displayed in blue in the **Property Editor** under **Custom Properties**.

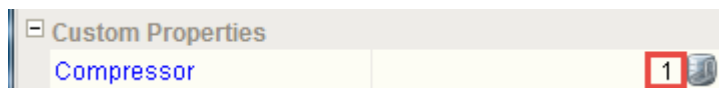


- Now, create a second compressor button. A quick way to do this is duplicate (**Ctrl+D**) the Compressor 1 button so it inherits the custom property, and label it '**Compressor 2**'.

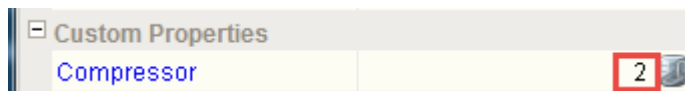


- From the **Property Editor**, change the values for each of the compressors to represent Compressor 1 and Compressor 2, respectively:

- For the Compressor 1 Button, change the value to '1'.



- For the Compressor 2 Button, change the value to '2'.



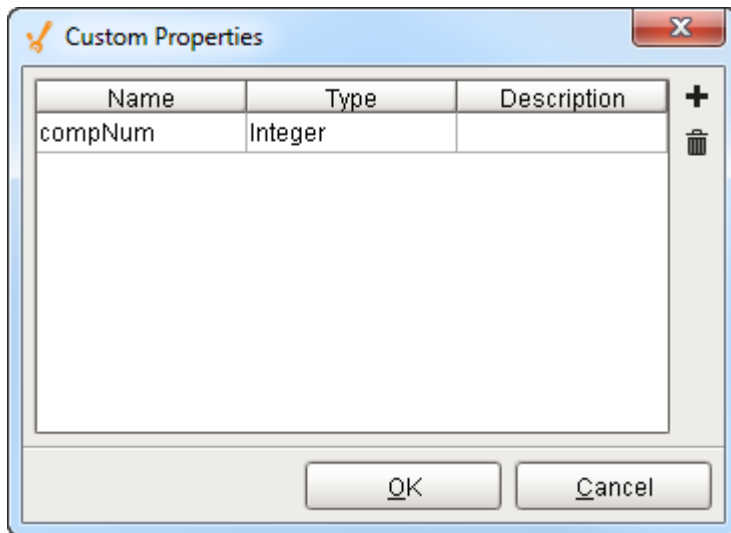
Setting up the Popup Window

- Drag **Compressor** and **Label** components from the component palette to your window. Rename the Label component to '**Compressor**'.



- Now, 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 will be displayed. Specify a **Name** for the **Custom Property**, for example, name it `compNum`, and click **OK**.



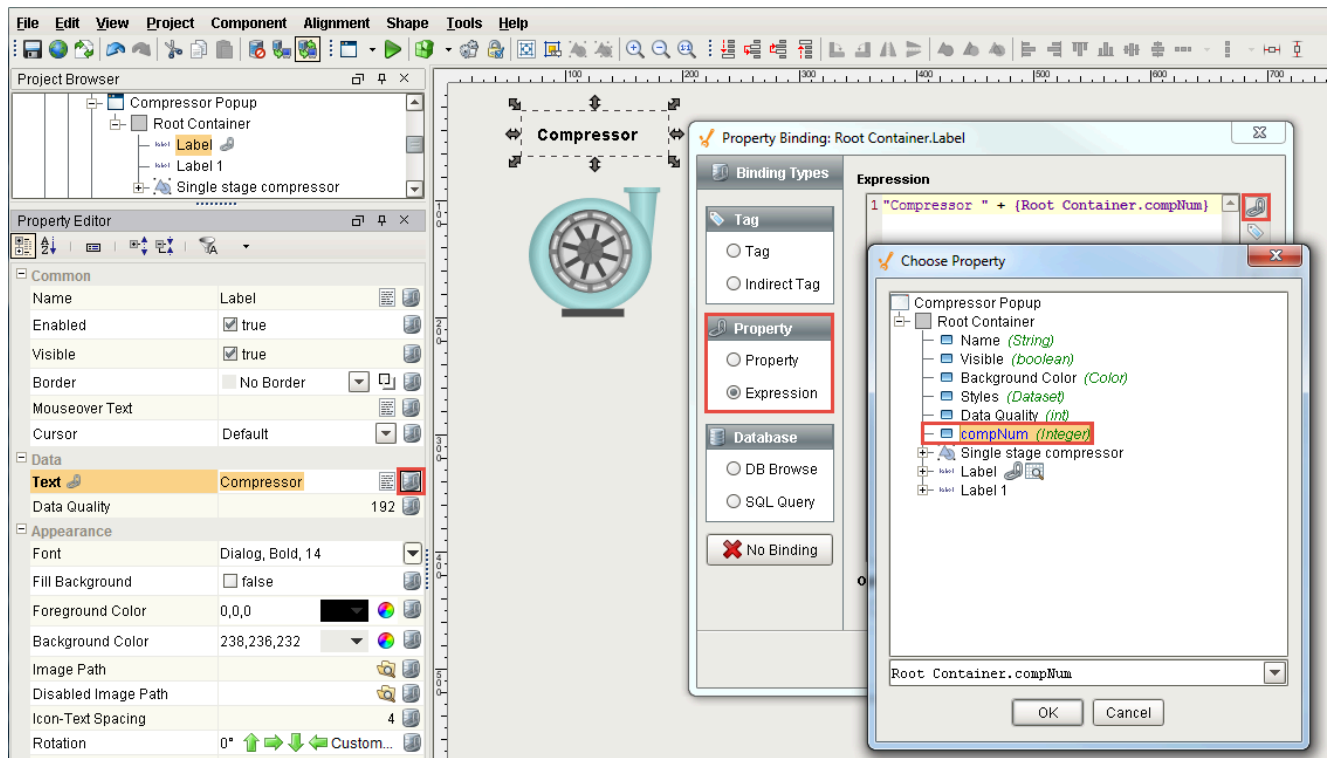
The **Custom** property is created and displayed in blue at the bottom of the **Property Editor**.

NOTE

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.

3. Let's use an expression on the Label to change the compressor number.

- i. Select the **Label** and click the binding icon for the **Text** property.
- ii. Select **Expression** for the binding type.
- iii. Click the **Link icon**  in the Expression window and choose the '**compNum**' custom property on the root container for the compressor popup window as shown in the screenshot below. You can also copy and paste the expression from the Code Block as shown below.

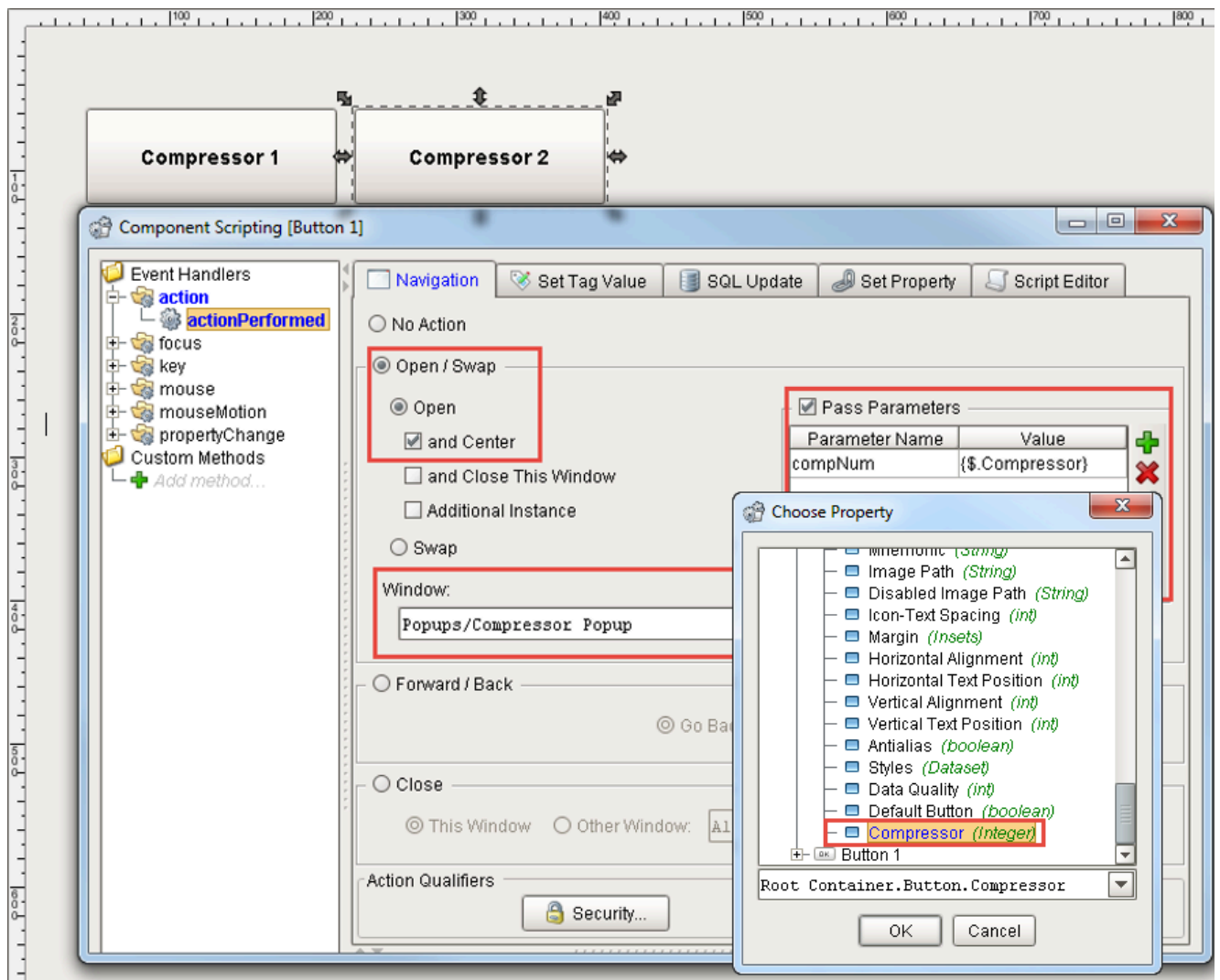


Script to change the compressor number

```
"Compressor " + {Root Container.compNum}
```

4. One last thing to do, add the script to the buttons which read the custom property. This changes the value on the custom property from Compressor 1 to Compressor 2 or visa versa when you press the compressor buttons.
 - i. Go to your **Main Window**, right click on the **Compressor 1 button** and select **Customizers > Scripting**.
 - ii. Select **Open/Swap** navigation.
 - iii. Enter your **window path** for your Popup Window (i.e., Popups/Compressor Popup).
 - iv. Check the **Pass Parameters** box. Click the **green plus**  to add a parameter, which will add a row to the table.
 - v. Click the new row under **Parameter Name** and a dropdown list will appear. 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. Select the custom property '**compNum**'.

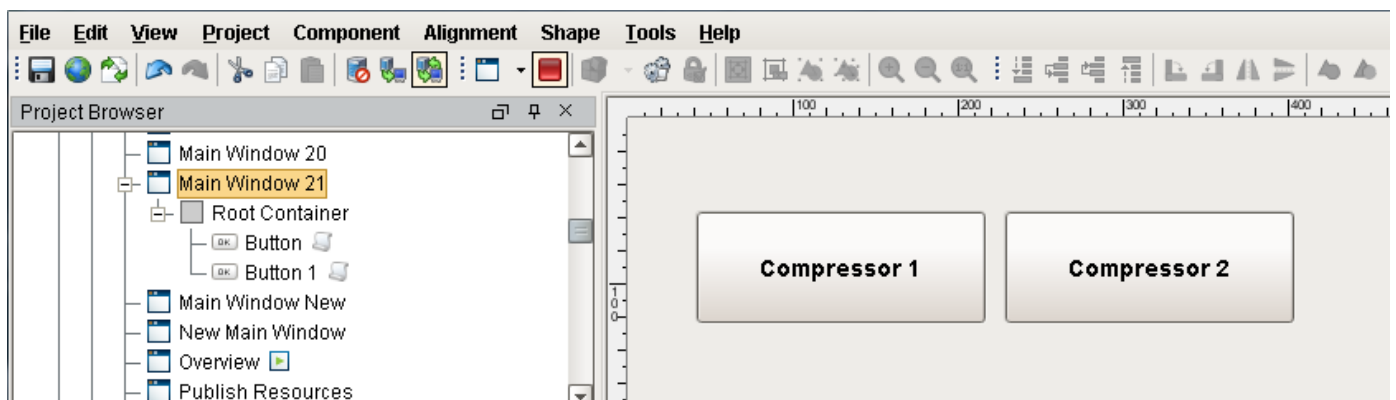
- vi. With the row highlighted, click the **Link icon**  and navigate to the folder where the compressor buttons are located and select the '**Compressor**' custom property. Press **OK** twice. Repeat this step for the Compressor 2 button.

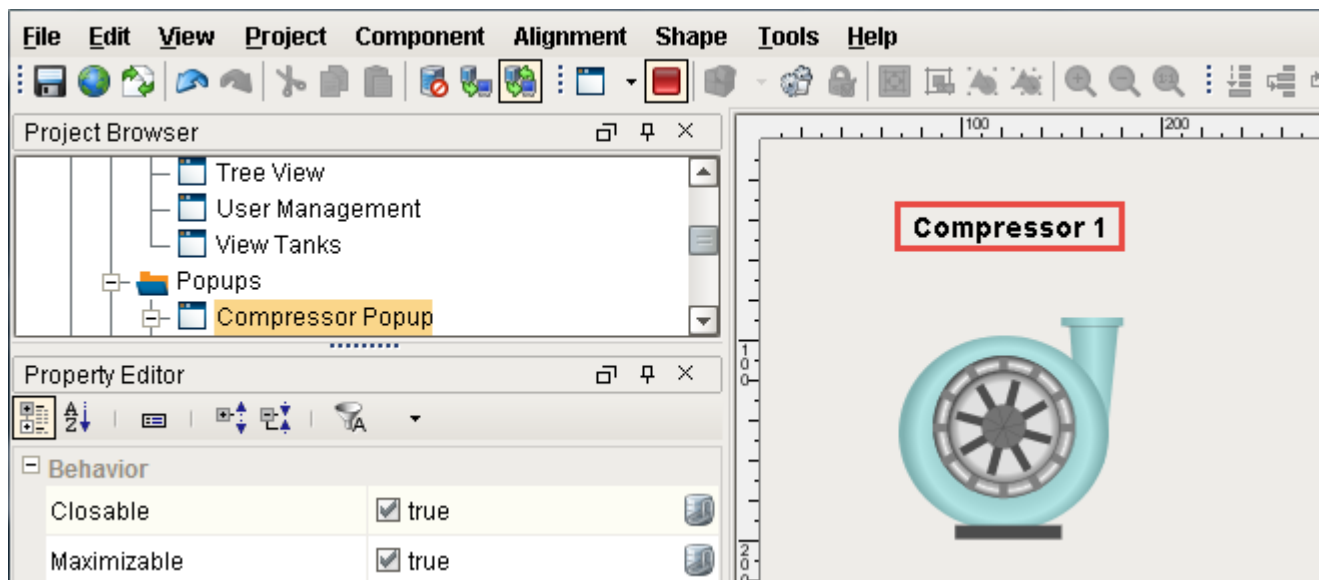


5. **Save** your project.

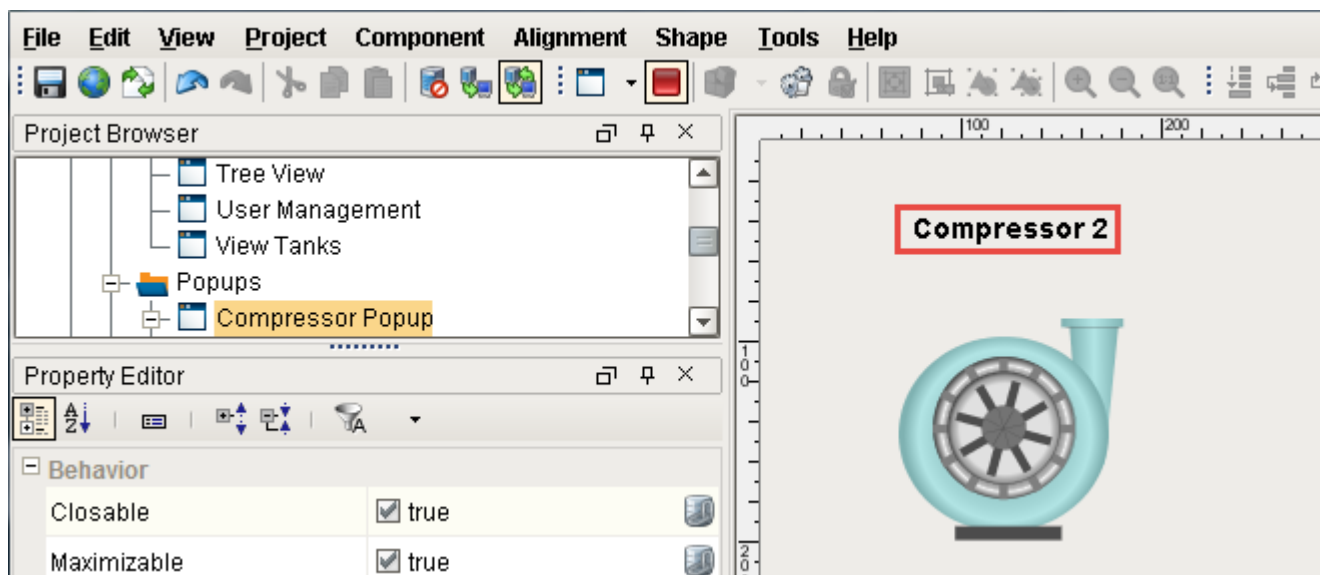
Test the Project

1. Go to **Preview Mode**, and from your Main Window, click on **Compressor 1**. Your Popup Window will open and display Compressor 1.





2. Go back to your Main Window and click on **Compressor 2**. This time, your Popup Window will open and display Compressor 2.



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

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

Parameterized Popup Window and UDTs



INDUCTIVE
UNIVERSITY

Parameterized Popup Window and UDTs

[Watch the video](#)

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. Parameterized popup windows can use basic Tag types and UDTs. Parameterized popup windows can be re-used as long as the proper information is passed into the popup window.

Before we dive into Parameterized Popup Window and UDTs, a small Glossary of terms is provided to help you understand the terminology on this page.

Glossary

Name	Description
Custom Property	A property you add to a component or container. A custom property is a variable that you customize for your particular use.
User Defined Type (UDT)	Is a data structure made up of Tags, attributes and settings. UDTs can be used in any place a standard Tag is used, and are used to design re-usable, parameterized, and extendable data types.
Element/Parameter of a UDT	An individual value inside of a Custom Property, assuming the Custom Property is a UDT type.

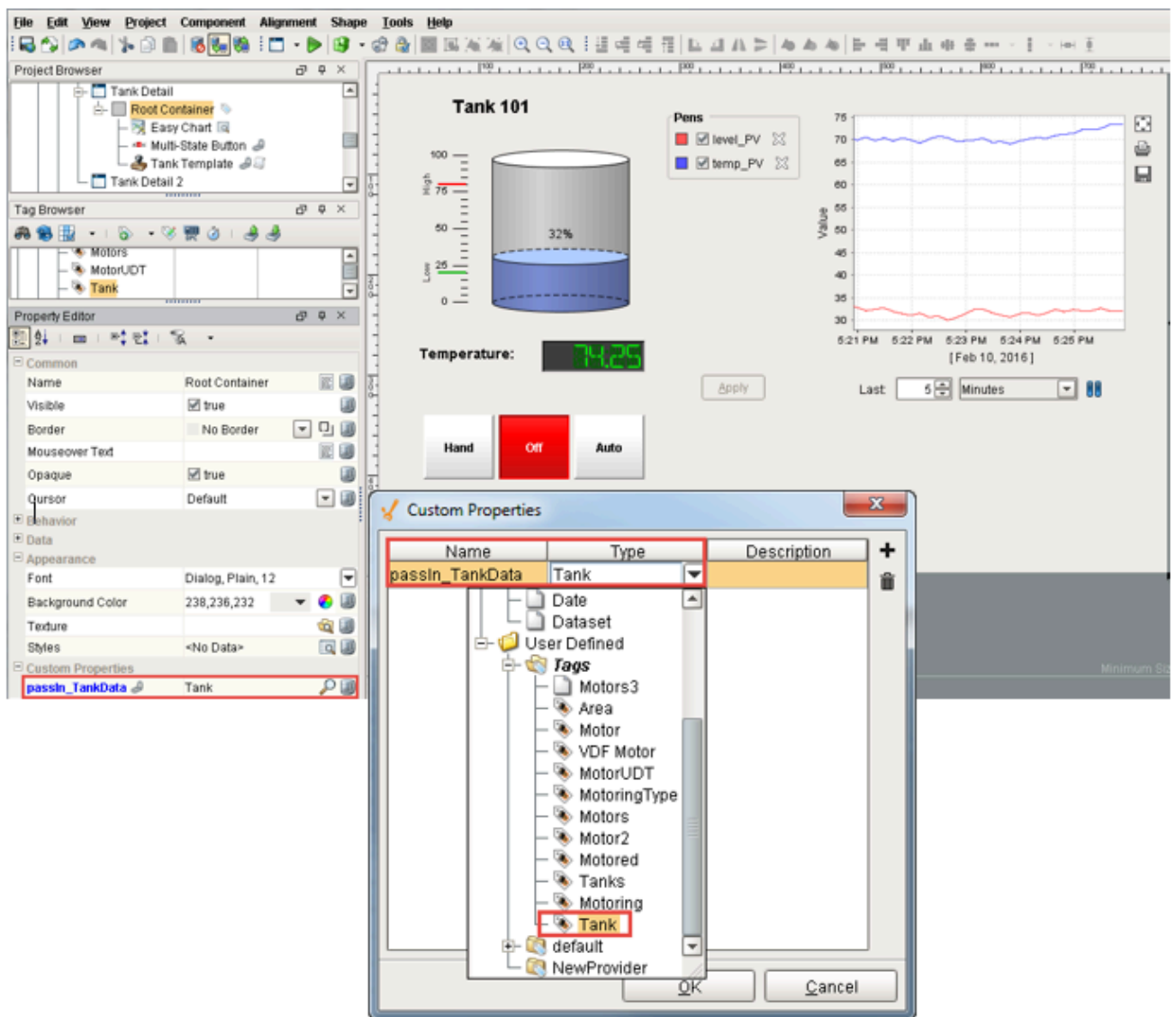
i UDT AND MATCHING TANK TEMPLATE MUST BE CREATED TO DO THIS EXERCISE

The examples in this section assume you have a UDT and matching Tank template (that uses the UDT) created. To learn more about creating UDTs and Templates go to [UDT Parameter](#) and [Template Basics](#) sections of this user manual.

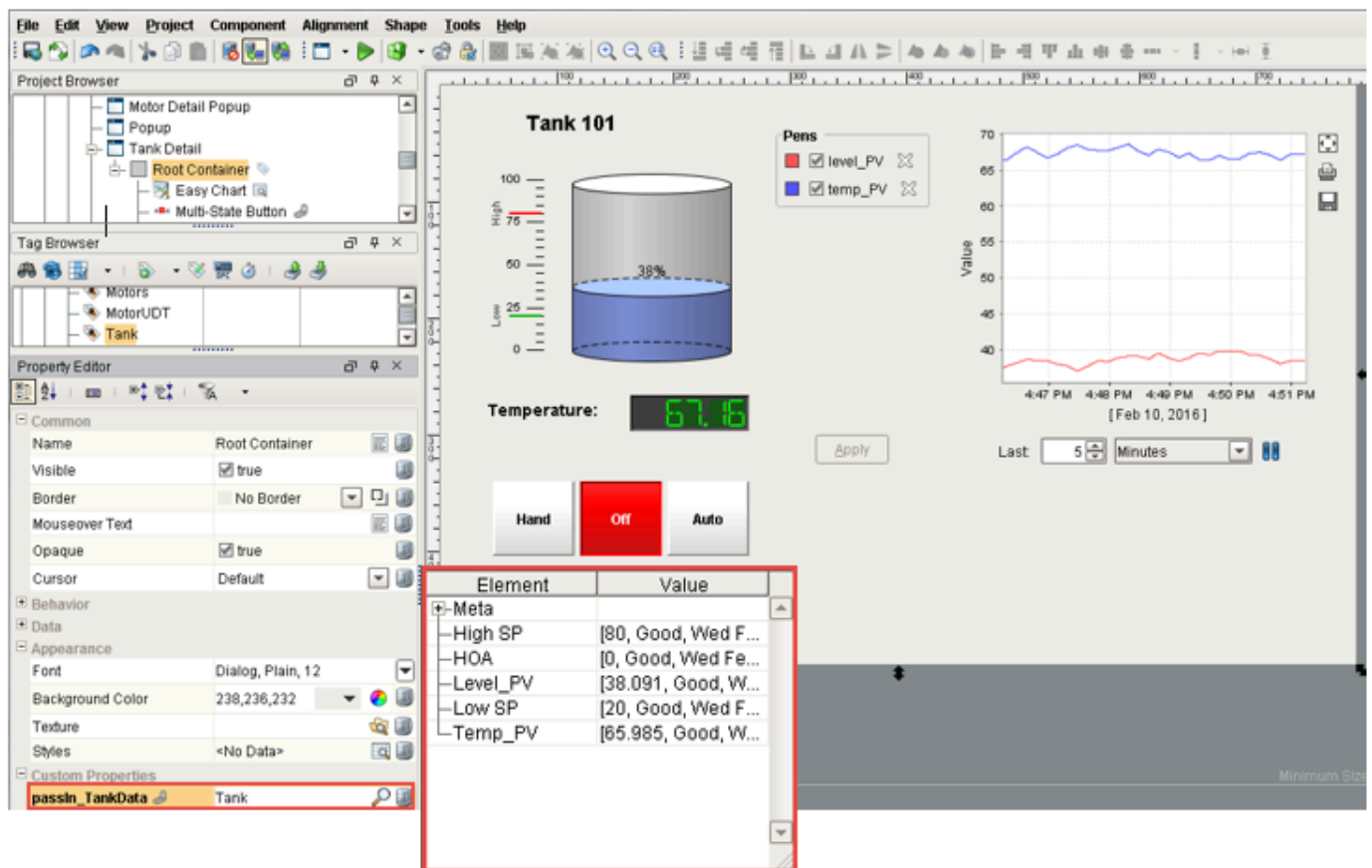
To Pass a UDT to a Popup Window

When you pass an entire UDT into a popup window, you can take advantage of passing all the parameters inside of that UDT at once. By passing the entire UDT, you have every single value available to you that is inside the UDT.

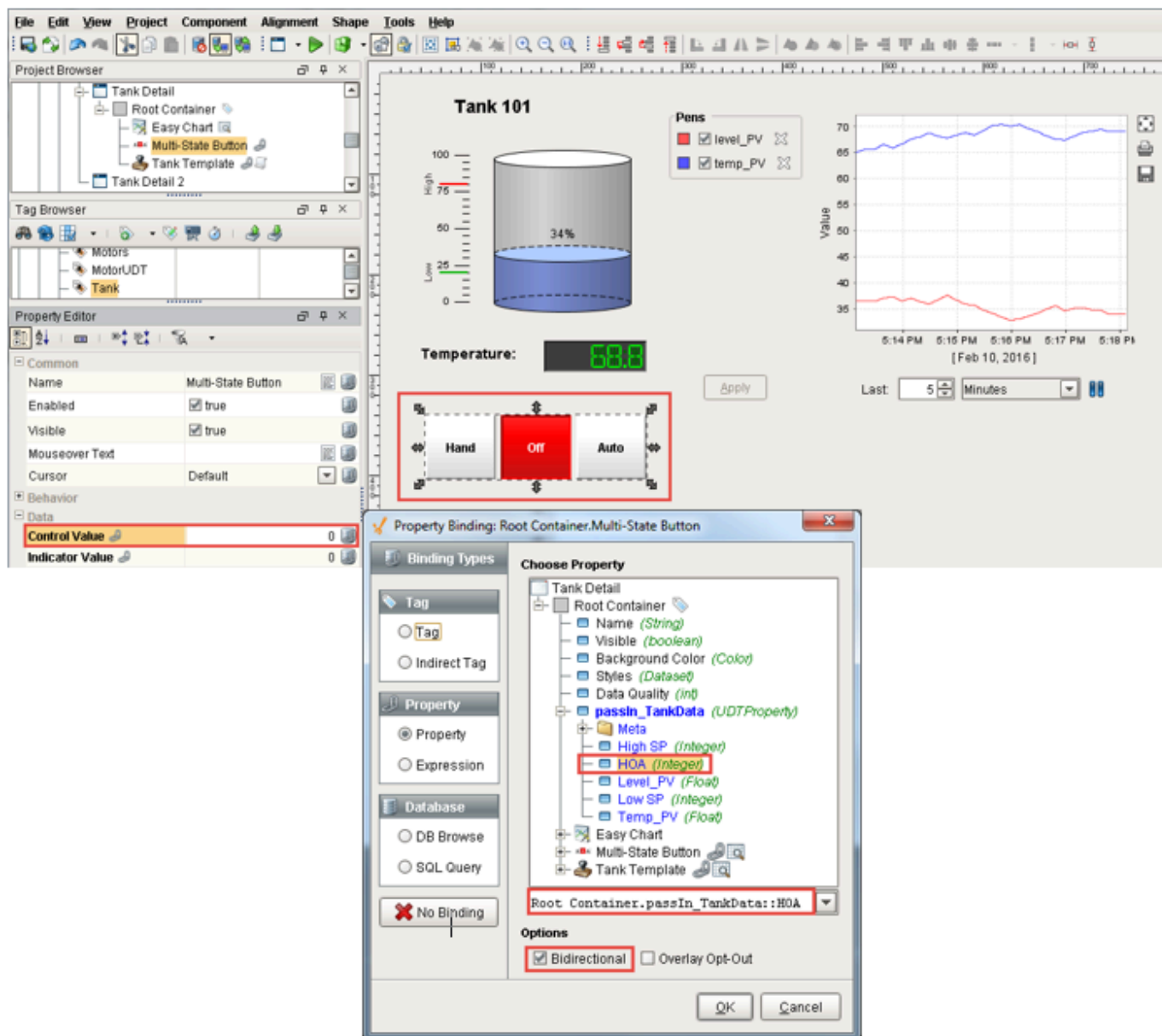
1. In this example, we have a '**Tank Detail**' popup window with a data type called '**Tank**'.
 - i. On the root container, right click and select **Customizers > Custom Properties**.
 - ii. Click the **plus icon** and add a property called '**passIn_TankData**' and select the '**Tank**' data type from the dropdown list.
 - iii. Click **OK**. Note that parameters that are passed into a window can only be passed into properties that are on the **Root Container**.



2. In the Property Editor you will see the Custom Property called '**passIn_TankData**'. It is assigned a unique name so it is clear exactly what the Tank property is doing. It will hold the UDT that is passed in from another window. Click on the **magnifying glass** and see all the elements that are inside of the Tank UDT; (i.e., HOA, level_PV, temp_PV, etc.). All of the values will be NULL if you just added the property. They will get filled in later when tested in Preview Mode.



3. Now, you can bind components inside the Tank Detail window to elements of the Tank UDT.
 - i. For example, let's bind the HOA Control. Click on the **Multi-state Button** in your Tank Detail window.
 - ii. In the **Property Editor**, click the **binding icon** next to the Control Value to open the Property Binding window.
 - iii. Select the **HOA property** of the Tank UDT and **check the Bidirectional box** to make it bidirectional. Click **OK**.
 - iv. By using the entire Tank UDT and all of its parameters, you have all the properties available to you making it very easy to configure the components in this popup window. You can now bind properties on the window without having to use indirect tag bindings. Bind the rest of your components to the appropriate Tags in the Root Container UDT.

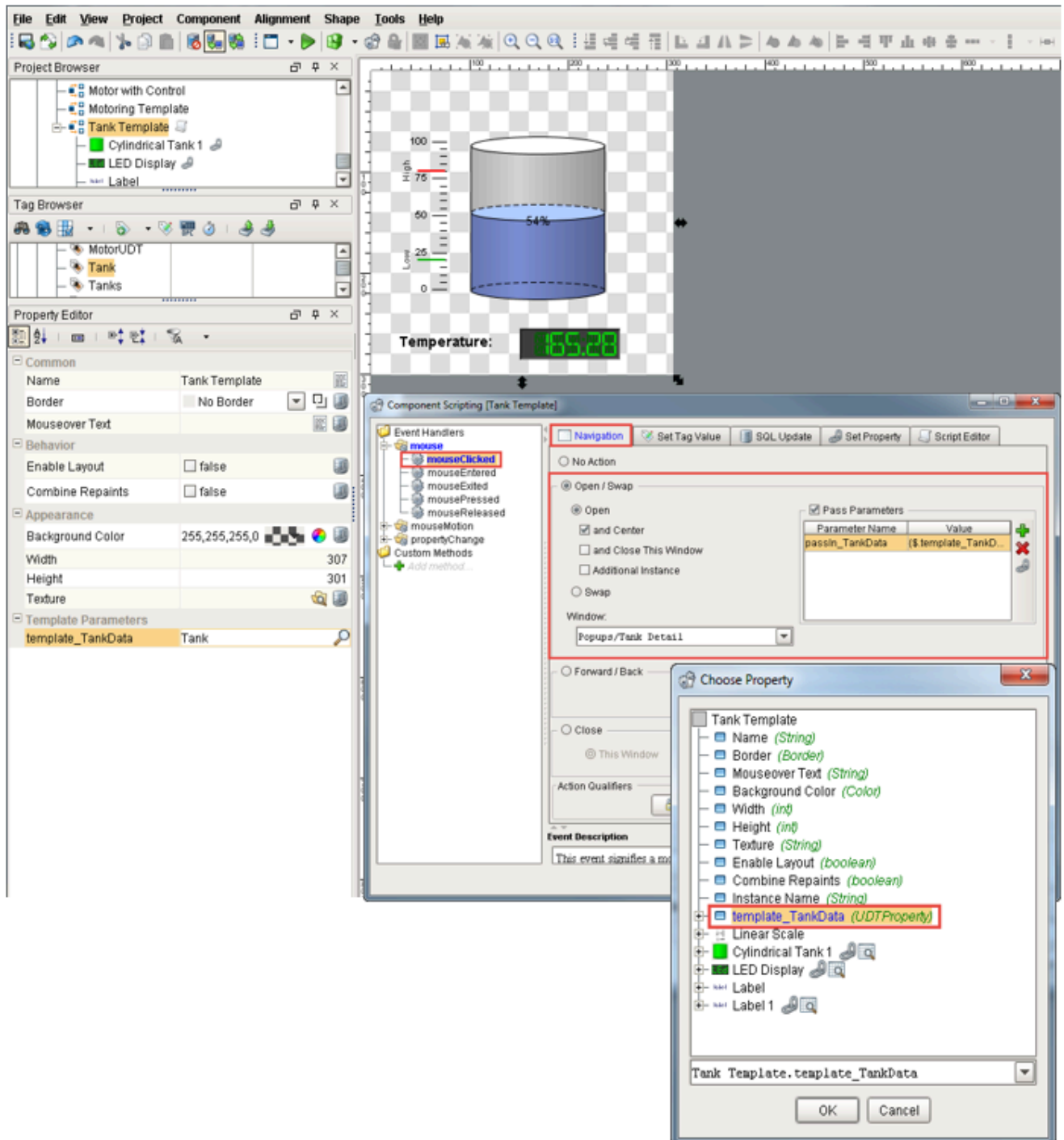


4. Now that there is a property to pass into, we need to set up the script to pass a Tank UDT into the Tank Detail window. A great place to do this is from the Tank template definition. The Tank template is using the same parameter as the Tank Detail window, but for clarity, it was assigned a unique name called '**template_TankData**'.

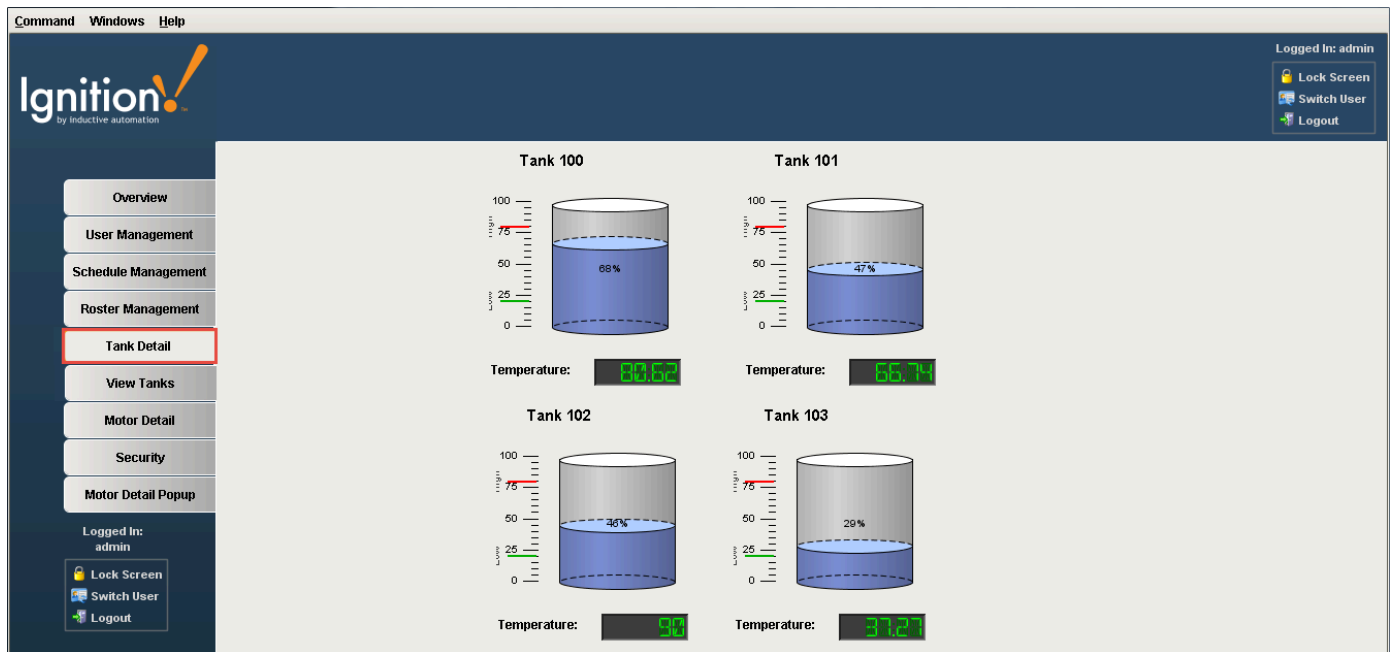
- i. Navigate to your **Tank Template** and right click on the **gray area of the canvas** and scroll down to **Scripting**. Under **Event Handler**, select '**mouseClicked**'.
- ii. In the **Navigation tab**, click **Open/Swap**, **Open and Center** and under the **Window** dropdown, select your '**Tank Detail**' window.
- iii. Check the **Pass Parameters** box. Press the green plus icon to add a row and click in the '**Parameter Name**' cell. This is a dropdown list that shows all of the custom properties on the Root Container of the selected window. Select the '**passIn_TankData**' parameter to pass in to.

If you don't see this parameter, you either have the wrong window selected, or you did not add the property to the Root Container.

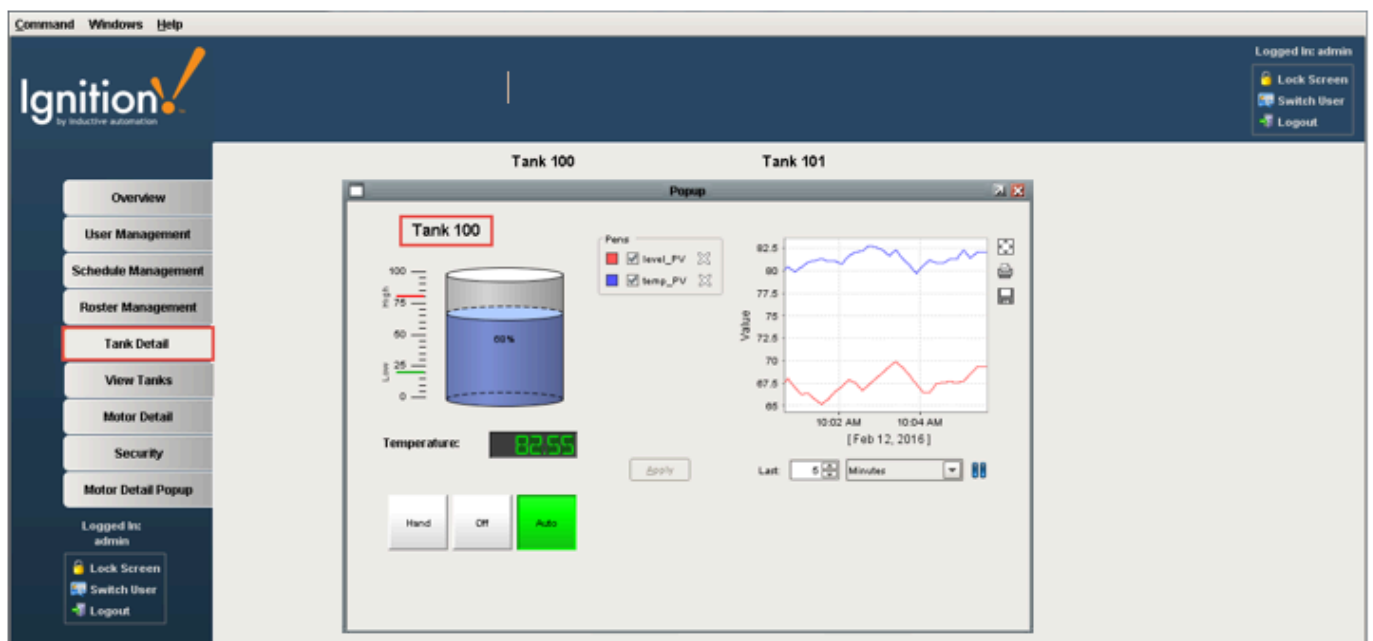
- iv. Select the whole row and click the choose property **link icon** and select the '**template_TankData**' property. The is the value we are passing to the window.
- v. Click **OK. Save and Publish** your project.



- 5. Open the **Runtime** and go to any window that uses your tank template. We used the exact same UDT that is in the Tank template and passed it to the Tank Detail Popup window.



Click on one of the Tanks (ie: Tank 100) and it will open your Tank Details window and display all the details about Tank 100. You can do this with any Tank in your runtime.



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

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

Popup Window - Multiple Instances



INDUCTIVE
UNIVERSITY

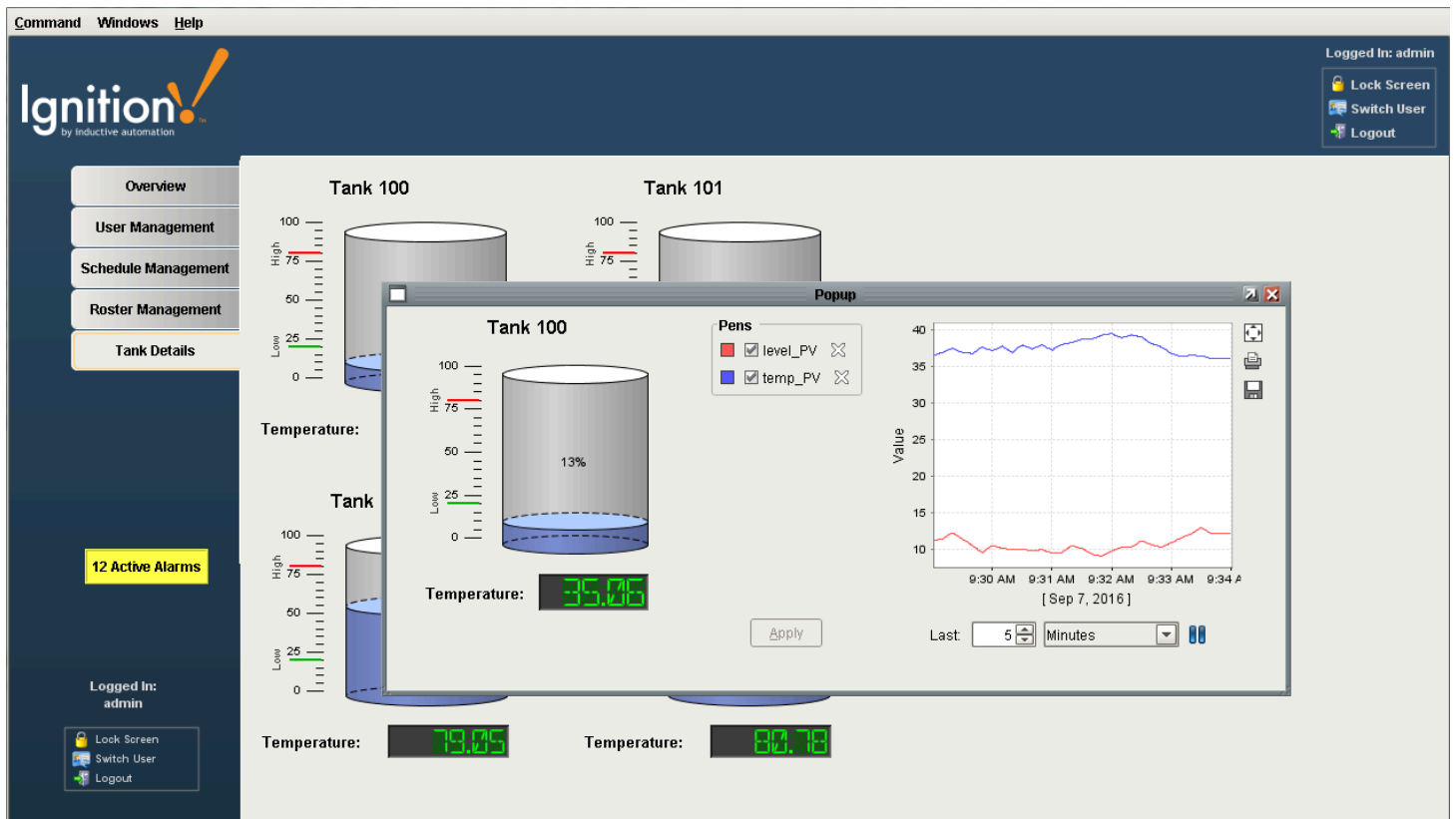
**Popup Window - Multiple
Instances**

[Watch the video](#)

By default, Ignition only opens a single instance or copy of a popup window in the client at a time. However, you can change this behavior and open multiple instances of a popup window in the client.

In this example, suppose you have four tanks in your client. If you click on any one of the four tanks, you will get a popup window that gives you some additional information about that tank.

If you click on another tank, for example Tank 101, the popup window switches to Tank 101, but there are still three windows open: Navigation window, Main window, and a Popup window (use the Windows Menubar at the top of the screen to view all open windows in the Client). What happens is when clicking on a different tank reopens the popup window and passes in new parameters. This is the default behavior because the number of open windows you can have at one time is limited in the client. You can change this behavior and have multiple instances of the same window open at once using a script.



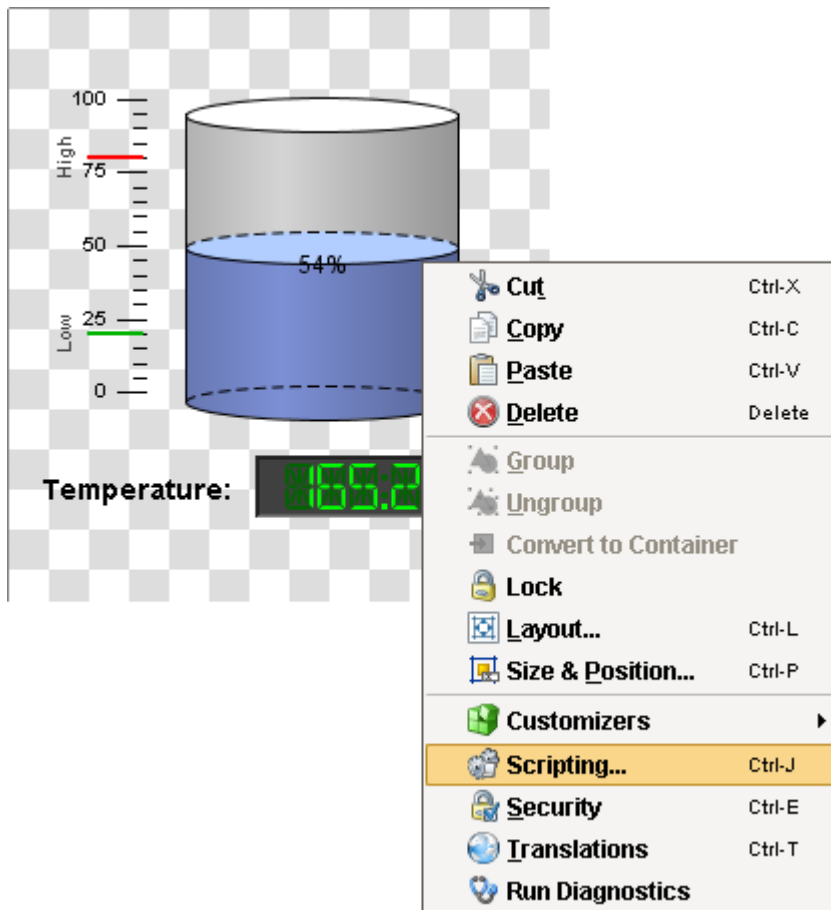
Opening Multiple Instances of a Popup Window

NOTE

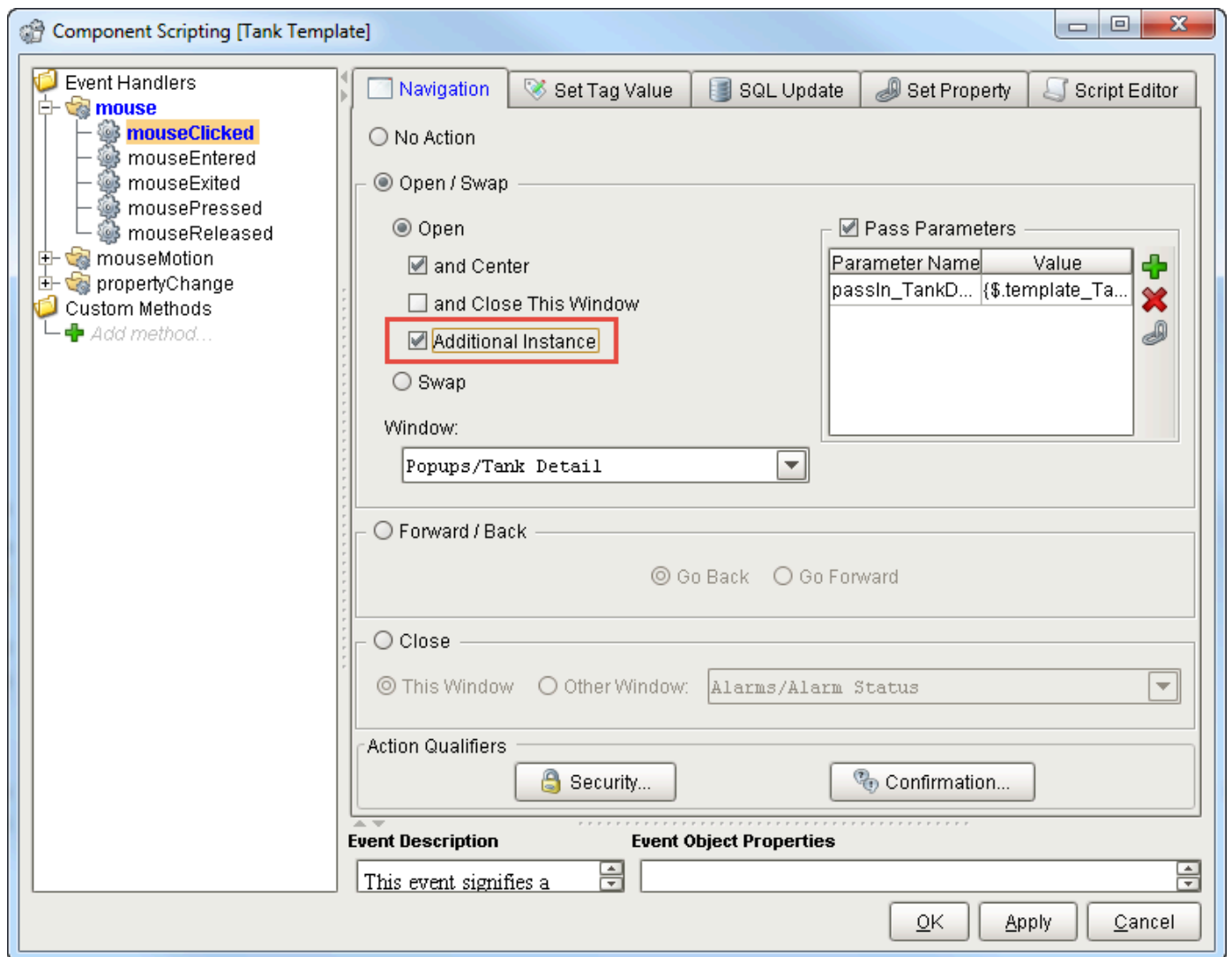
This example expands on the tank example from the [Parameterized Popup Window and UDTs](#) page.

Opening a window will only ever open one copy of any given window. If a popup window is already open, it simply brings it to the front. Sometimes you may want to open a separate popup, one for Tank 100, and one for Tank 101, both at the same time. If this is the case, you can use the [system.nav.openWindowInstance](#) function call to open your window.

1. In **Designer**, go to your Tank template. Right click on your **Template** and scroll down to **Scripting**.



2. You'll notice that the Navigation tab settings are set to Open/Swap and are passing some Tank parameters over to the popup window. Check the **Additional Instance** box to make the script open up additional instances of your popup windows. When you click on the Additional Instance box, Ignition automatically enables the navigation script builder. It also tells the script instead of using the same popup window, to open up a new instance if there is another popup window already open instead of switching it out (to view the script, click on the Script Editor tab. You'll notice the `system.nav.openWindowInstance` function is used). Click **OK**.



3. **Save and Publish** your project.
4. Open your **Client**, or if you already have it open, update the client by clicking the yellow bar. Click on each of your Tanks, and you'll see multiple instances of the same popup for each different tank.

Overview

User Management

Schedule Management

Roster Management

Tank Details

14 Active Alarms

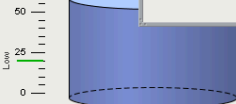
Logged In:
admin

Lock Screen
Switch User
Logout

Tank 100



Temperature:



Temperature: 55.74

Tank 101

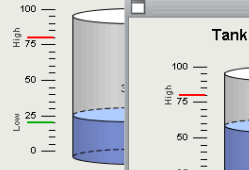


Temperature:



Temperature: 55.74

Tank 101

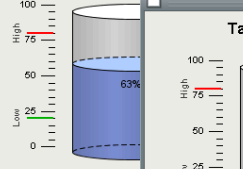


Temperature:



Temperature: 55.74

Tank 102

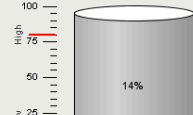


Temperature:



Temperature: 55.74

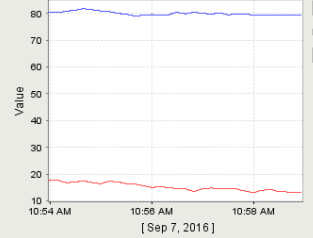
Tank 103



Temperature:

79.85

Pens
☒ level_PV
☒ temp_PV



Apply

Last 5 Minutes

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

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

Understanding Components



INDUCTIVE
UNIVERSITY

Component Overview

[Watch the video](#)

About Components

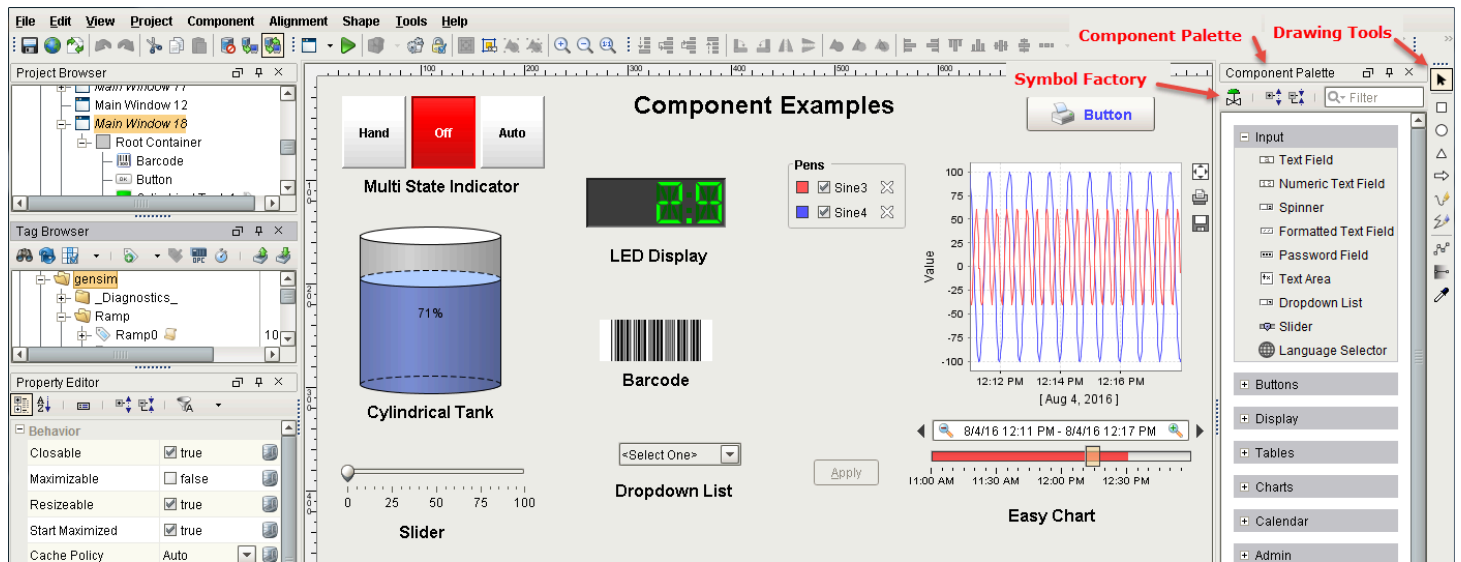
Components are the visual elements that what fill up your windows with useful content. Anyone familiar with computers should already understand the basic concept of a component. They are the widgets that you deal with every day, such as: buttons, text areas, dropdowns, charts, and so on. The Vision module comes with a host of useful components out of the box, many of which are specialized for industrial controls use. Other modules, like the Reporting module, add more components for specialty purposes.

To make components do something useful, like display dynamic information or control a device register, you configure **property bindings** for the component. To add scripting actions to a component, you configure **event handlers** for it.

Configuring components is the bulk of the designer's work when designing a Vision project. The basic workflow is to take a component from the 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.

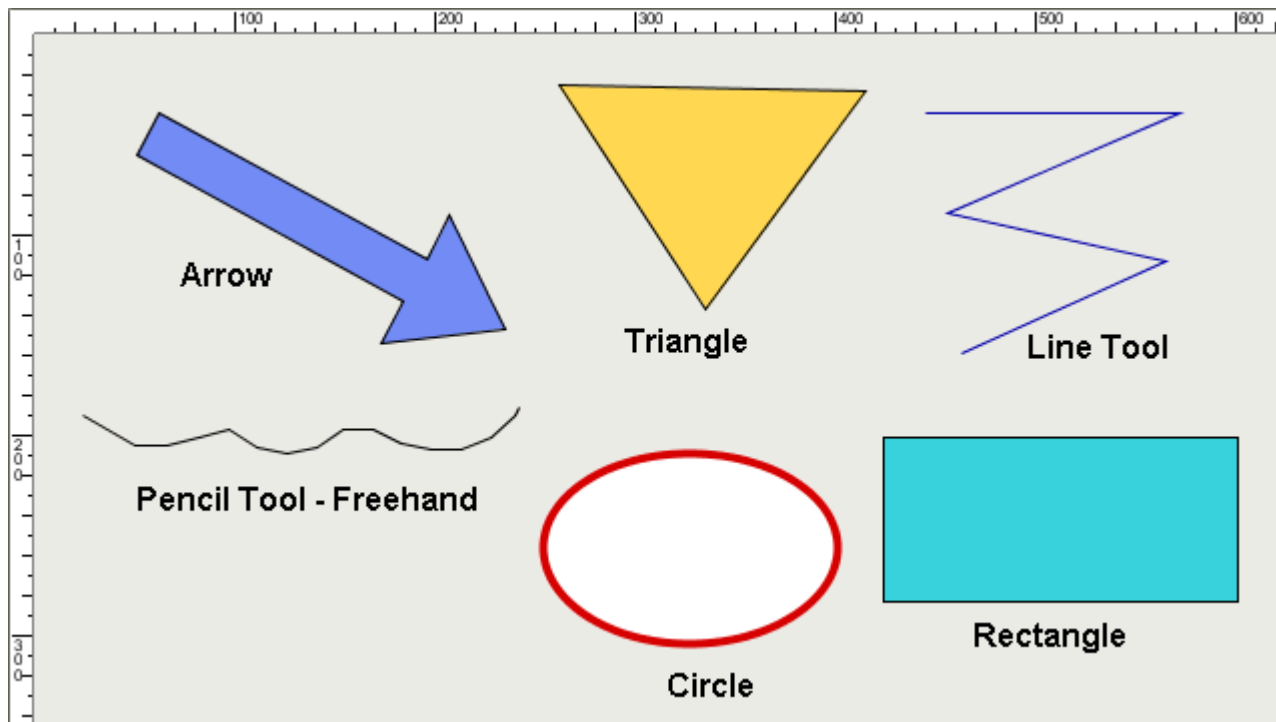
Components

In Designer, there is a **component palette** filled with different types of components that you can drag and drop into your workspace. There are several component types that include Input, Display, Table, Chart, Calendar, Alarming, Reporting components, and more.



Shapes

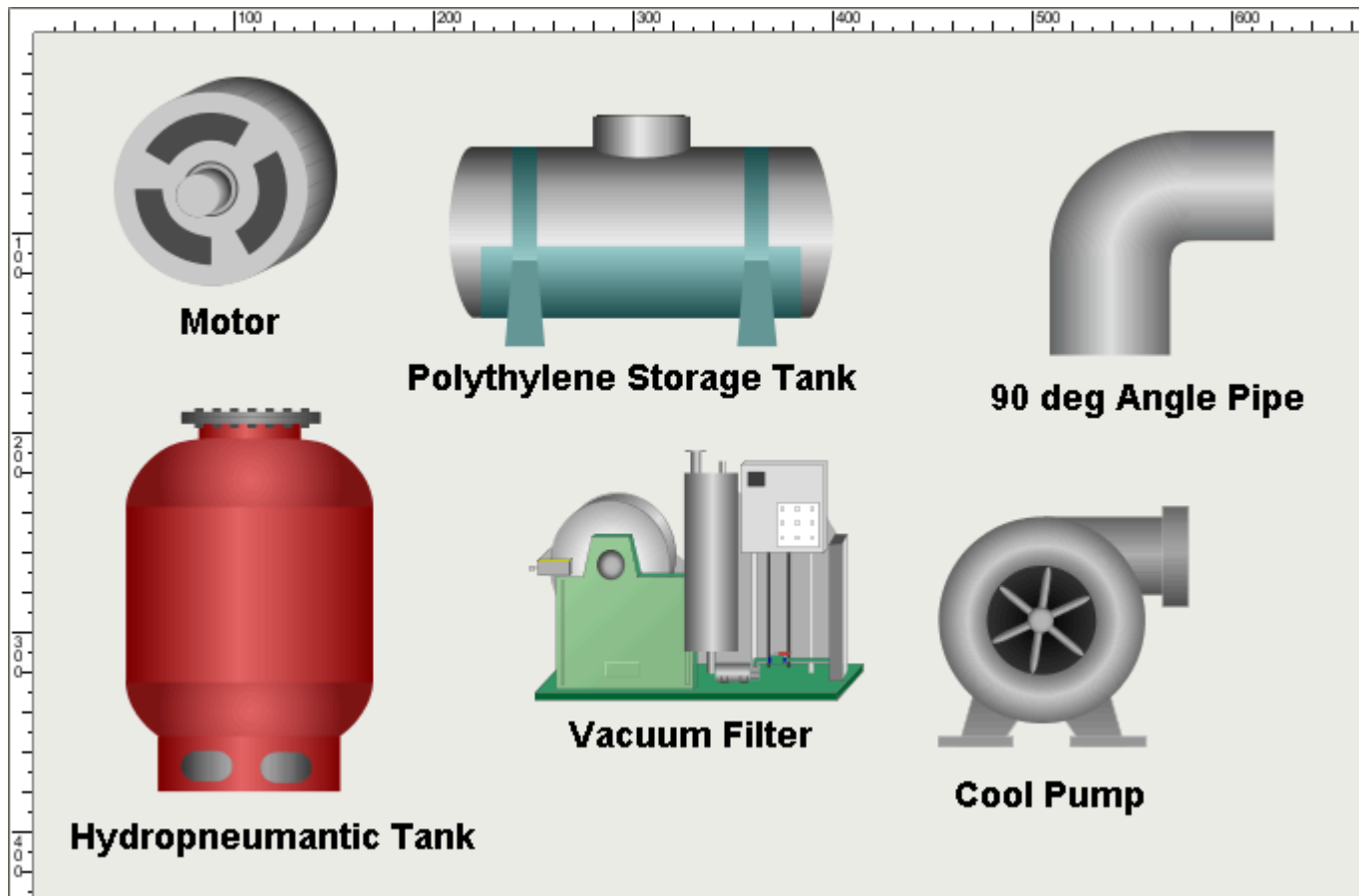
There are also **drawing tools** provided so you can create your own components from shapes such as lines, rectangles, circles, and arrows, to name a few. **Shapes** are components too. Each shape may be individually selected, named, and has its own properties. Shapes have some additional capabilities that other components don't have, such as the ability to be rotated. Shapes are created using the shape drawing tools, not dragged from the component palette.



Symbol Factory

You can even import **Scalable Vector Graphics (SVGs)** into a window. **Symbol Factory**, which is a separate module, contains a variety of 4,000 industrial symbols to choose from for your projects. The Symbol Factory includes symbols for Motors, Tanks, Chemical, HVAC, Food, Material Handling, and many more. Just like components, they can be dragged and dropped on your screen, resized, and

you can even add color to the individual component's pieces. Your own SVG's can even be imported directly into a window to use in your project.



Images

Ignition also comes with a host of [images](#) to use on your components such as PNGs, JPGs, and GIFs. They can spice up a plain button, header, label and more. In addition to the images that come with Ignition, you can import your own too.

Creating Components

Even with all the different component types, creating components in your windows is super easy! All the components are on your component palette; components, shapes, and if installed, Symbol Factory. There are three ways to create components:

- Select your component on the component palette, and use the mouse to draw a rectangle in your workspace where you want your component to be located. When you release the mouse, the component will drop in. It can then be resized, moved or rotated.
- Drag a component's icon from the palette onto a container. The component will be placed where you drop it at it's default size. Then, it can be resized, moved, or rotated if desired.
- The easiest way to make some components that are bound to **Tags** is to simply drag and drop some Tags onto your window. Drag a Tag onto a window. You'll get a popup menu asking you

what kind of component to make. You can display the Tag with some components, and control the Tag with other components.

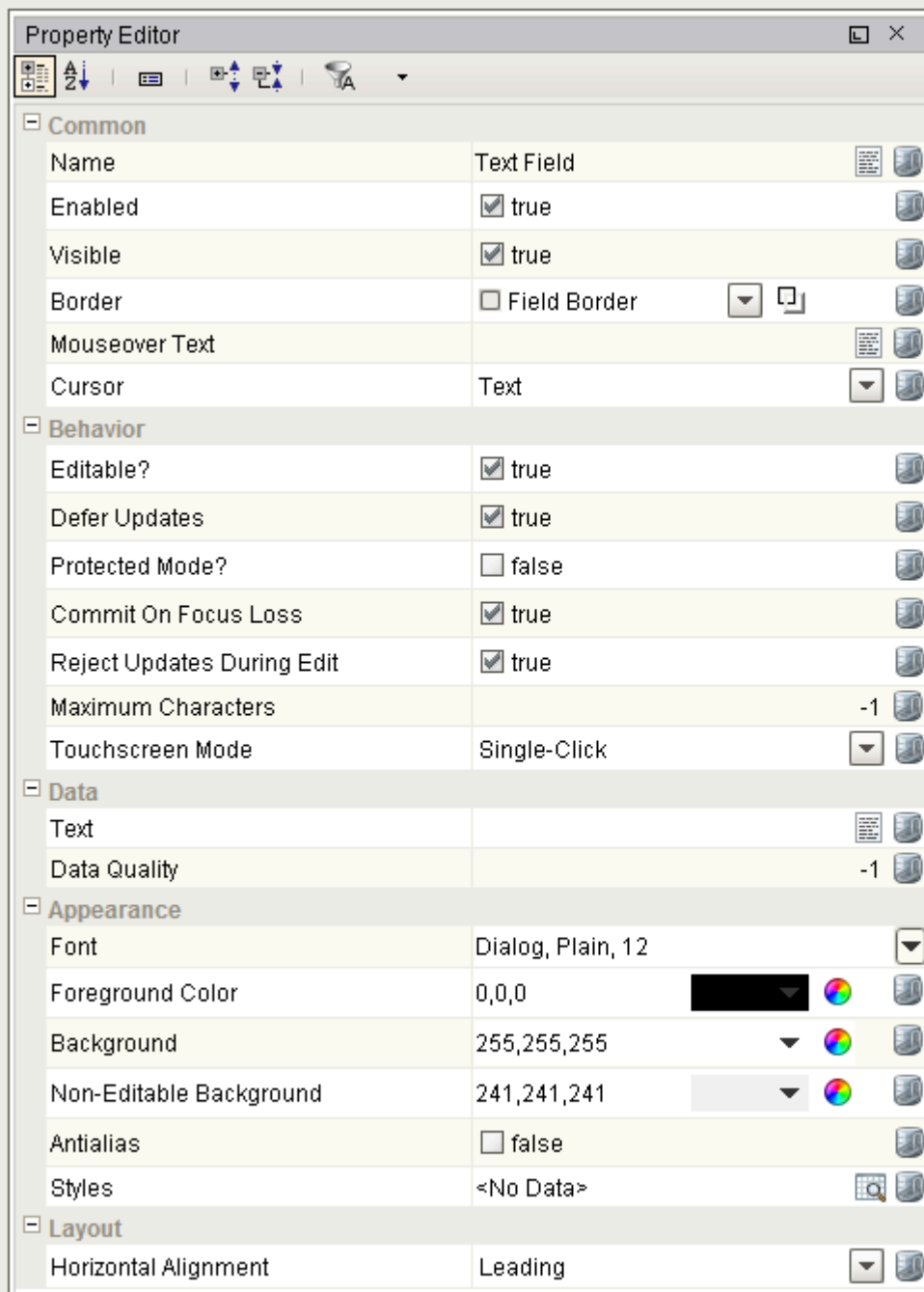
To learn more, go to [Creating Components](#). Once your components are in your workspace, you can move them around, resize, rotate and even group them. [Component layout](#) is a very important concept because of the web-launched deployment of Vision clients launched on many different monitors with many different resolutions. To learn about what you can do with components, refer to [Manipulating Components](#), [Component Grouping](#), and [Component Layout](#) sections.

Component Properties

Each component has its own set of unique [properties](#) which dictate how a component looks and behaves inside of the Ignition window. The Property Editor conveys a lot of information about a component. When you click on a component, you see all the properties that make up that component. By default, Ignition shows you all the properties broken up by categories in the Property Editor. You can view properties in alphabetical order, and filter on specific properties if you prefer. The choice is yours! Each property has a name and a [data type](#). By clicking on the "Show/Hide Description Area" toggle button or mousing over a property, you can see the description of that property along with the data type.

To make a property dynamic you are going to use the component's properties to bind them to various sources of data like a Tag, database query or to a calculation. There is a binding button on the right side of each property where you can edit the property to set or modify a value, font, border, color, size, etc. When you click the bind button for the component, it will open a [Binding](#) window where you can select the source that you want to link it too. If you don't bind that property, the value will be static. If you bind the property to a source like a Tag, the value will change dynamically on the window. That's how Ignition windows come to life, through bindings!

Once you learn how to navigate the Property Editor and understand the various properties that exist for a component, then you know exactly how to configure that component in Ignition. When using a new component, it's always a good idea to scroll through the entire list of properties to see what the component has to offer.



Component Customizers

Some special purpose components have their own special **Customizers**. There are two basic types of customizers that most components have, and some components have a third customizer.

- **Customizers > Custom Properties** - used to create additional properties to enhance existing functionality or add new functionality with data binding, scripting, etc.
- **Customizers > Styles Customizer** - used to define visual styles based on a driving property, like a custom property, that you want to change based on the value of a driving property that draws attention to the user.
- **Customizers > XXX Customizer** - these vary by component and have completely different functions specific to their component (i.e., Multi-State Button Customizer, Easy Chart Customizer,

etc,.).

Custom Properties

The great thing about Ignition is if you can't find a property that you're looking for from the list properties for your component, then you can create it! You can add your own custom component properties for your particular use. You can bind custom properties just like any other property. You can even use custom properties to pass values between windows. Custom properties are extremely important and are used for a variety of different functions in Ignition.

Commonly, these properties are used to create dynamic (bound) values that are very easily accessible in component scripting. For example, an ID value based on the current selection in a table.

To learn how to configure custom properties, go to the section on [Component Customizers](#).

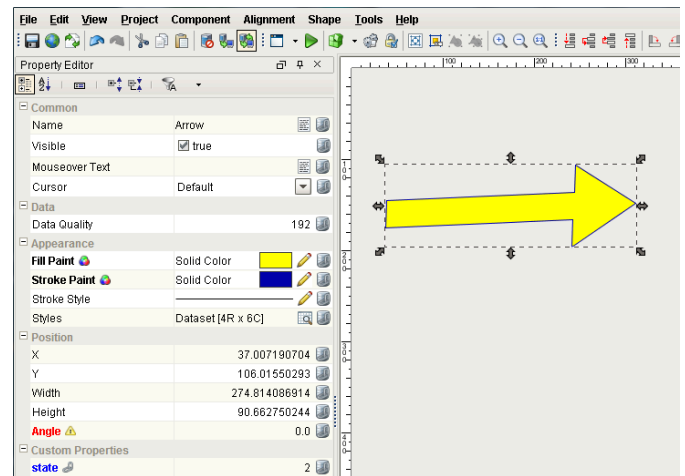
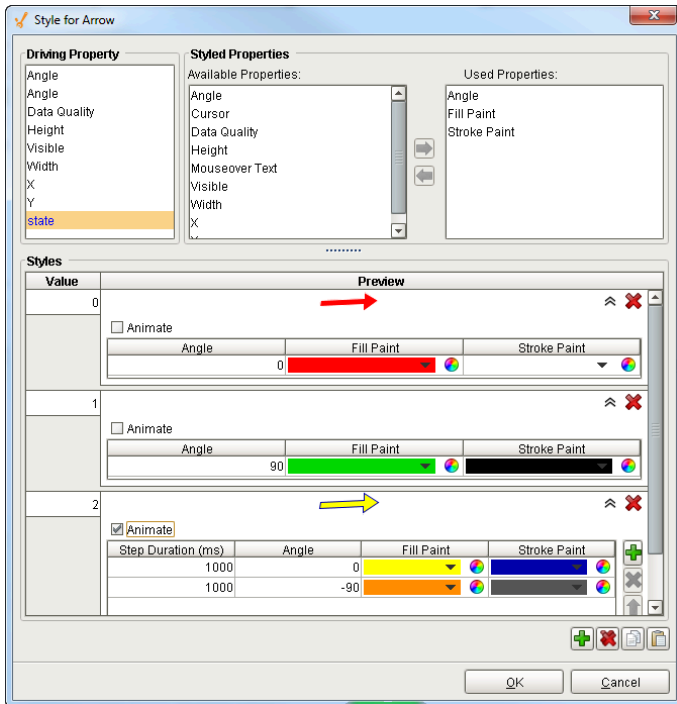
Style Customizer

The [Style Customizer](#) is also used with many Ignition components, and lets you define a set of visual styles that are all based on a single driving property, often a custom property. For example, using the Arrow drawing tool you can change the fill paint color, stroke paint color, and angle of the component (shown below) based on a driving property called '**state**'. Without styles, you would have to go to every property and bind them all individually. Some components 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.

The Styles Customizer saves time when you need to bind multiple properties at once. Once you setup a style, all of the used properties appear in bold in the Property Editor. That is because technically they are bound to a style. The paint canvas icon lets you know the property is used in Styles.

Style Customizer Settings for the Arrow Component

Configured Arrow Component using Style Customizer



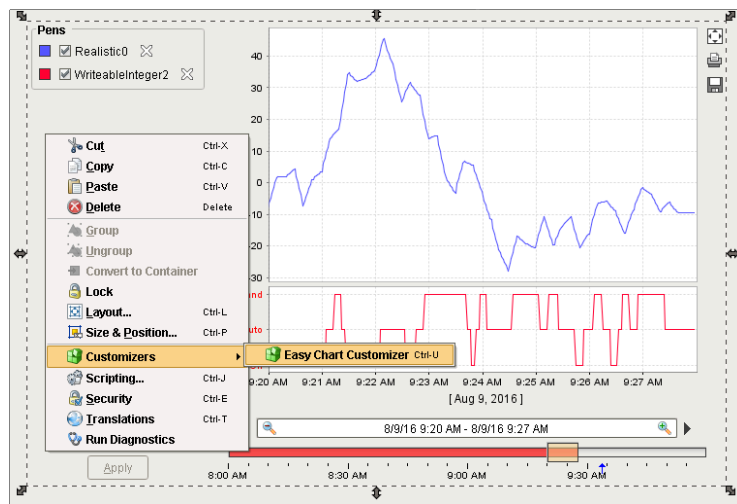
Special Component Customizers

Several components have their own Customizer, which is unique to specific components. You'll notice the first time you use them because they have a unique name. These special component customizers have some default settings to help you get started, but those default settings can always be modified to your requirements. The following components have their own Customizer.

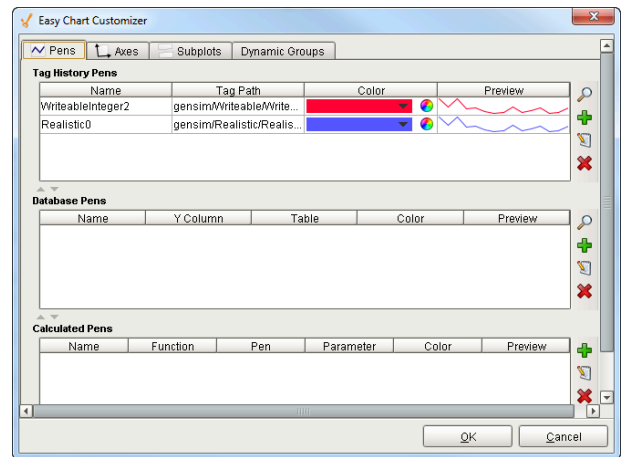
- Tables
- Charts
- Tab Strip
- Multi-State Button

For example, when you drag and drop Tags onto an Easy Chart, it automatically trends the data. Since there are many settings for each pen on an Easy Chart, it has its own Customizer so you can easily modify the chart to your own style. You can edit pen styles, create subplots, axes, and create pen groups in the Easy Chart. You can even preview it to make sure you like it.

Easy Chart Customizer



Easy Chart Customizer Window



Advanced Usage

Ignition components can be made to do some pretty sophisticated things using advanced features such as using **HTML**, **animation**, **multi-line text** and adding **images** to your components. For example, you can use HTML if you want to mix fonts or colors within the text or format multiple lines of text in a component. You can use animation with any of the SVG images and drawing tools in various ways to rotate, change the size, and change position of the component. You can even use the Image Path property to add an icon to a component to make the component visually standout.

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

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

Creating Components



INDUCTIVE
UNIVERSITY

Creating Components

[Watch the video](#)

Adding Components to a Window

There are four primary methods for creating components:

1. Select the component in the palette, and then use the mouse to draw a rectangle in a container. While a component is selected in a palette, the mouse cursor will be a crosshair (+) when hovering over a container that the component can be dropped in. Draw a rectangle in the container to specify where the component should be placed and what size it should be.
2. Drag a component's icon from a palette onto a container. The component will be placed where you dropped it at its default size. It can then be resized.
3. Drag one or more Tags onto a window. You'll get a popup with the kind of component you want to make based on the Tag type.
4. Any shapes that are drawn with the drawing tools, any imported images or SVGs, and any of the Symbol Factory images are also considered components.

Choosing 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 is meant to dock to the north or south edge of the [workspace](#), and the collapsible palette is meant to dock 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 the **Tabbed Palette** or the **Collapsible Palette**.

Creating a 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. Your custom palette has one special button in it, the capture button (). To add components to your palette, select them and press the capture button. This effectively makes 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.

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.

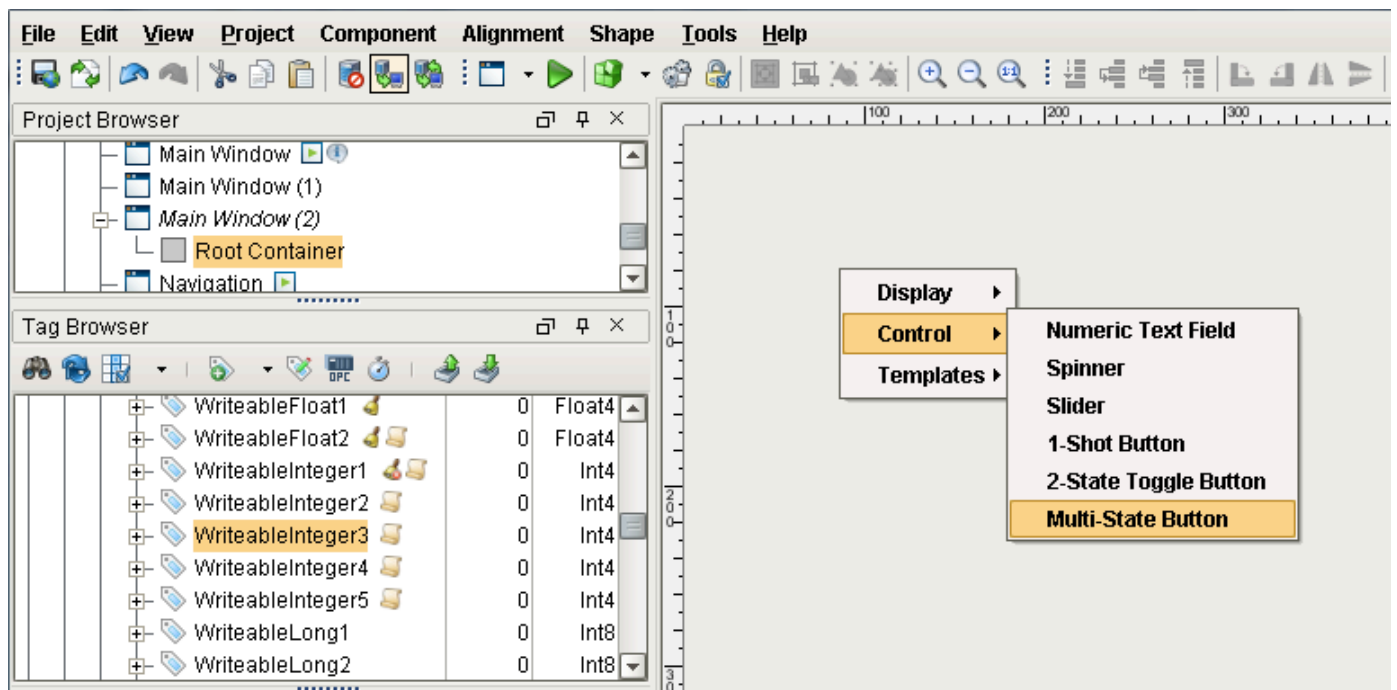
Creating Components using Tags

Components can also be created by simply dragging a tag onto a container. Depending on the datatype 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:

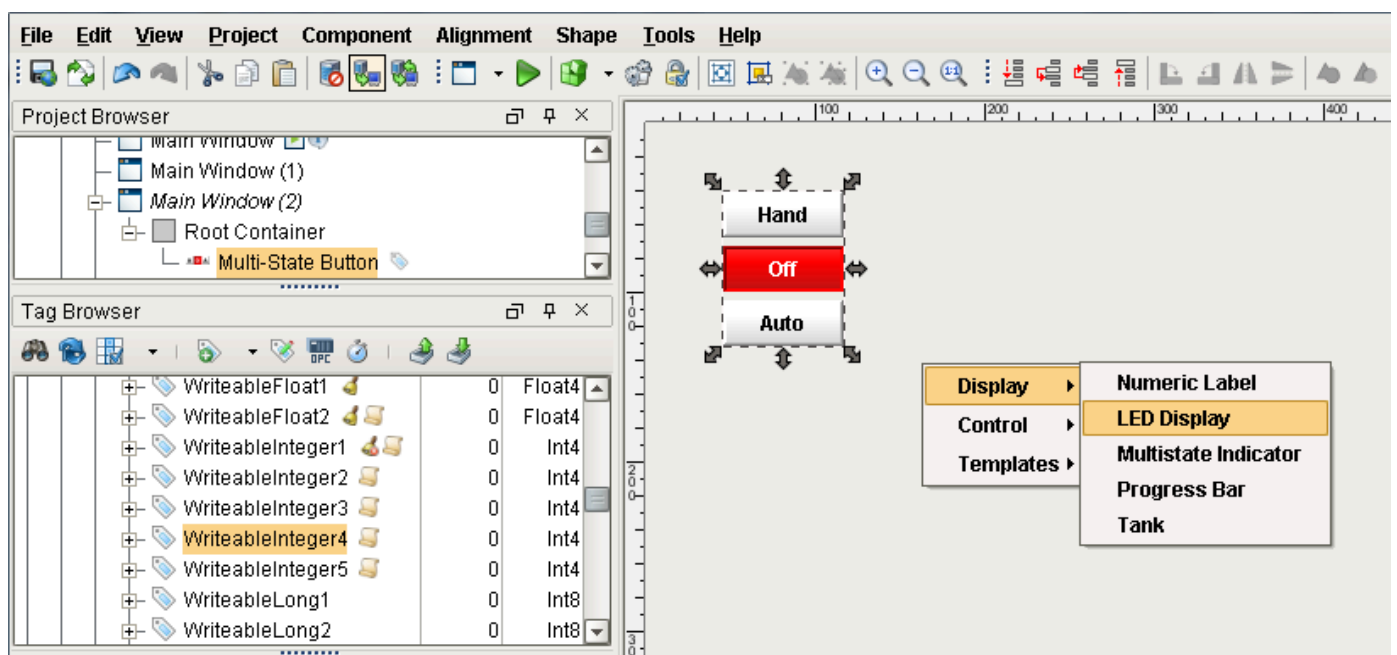
- The component is created at the position you dropped it.
- A variety of property bindings are created automatically.

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 simply drag and drop some Tags onto your window. If you don't have Tags yet, see the [Browsing and Creating OPC Tags](#) page for more info.

1. **Drag a Tag** onto a window. You'll get a popup menu asking you what kind of component to make. You can Display the Tag with some components, and control the Tag with other components.



2. Drag a few more Tags onto the screen to experiment with the different options.



3. As you're editing your project, you can click the **Save** icon on the toolbar to save your changes. Note that unless you started with a new blank project (not one of the project templates) your new window won't be available in the client after saving it. Any new window must be opened, and that means adding a button somewhere to open/swap to this window. See [Navigation Windows](#) for more information.

For example, suppose you have an Int4 type Tag, if you drag the tag from the Tag Browser panel onto a component, you will be prompted either to display or control the Tag with a variety of labels, level indicators, numeric entry fields, and control buttons.

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 using 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.

See also: [Property Binding](#), [Tag Metadata Properties](#)

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. They also have some things that normal components don't.

Binding Shape Position

One such thing that shapes have that normal components don't is a set of properties that control their location. These properties are called `relX`, `relY`, `relHeight`, and `relWidth`. The "rel" prefix stands for "relative". This comes from the fact that the values of these properties are always treated as 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 `relX` 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 `relX` 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.

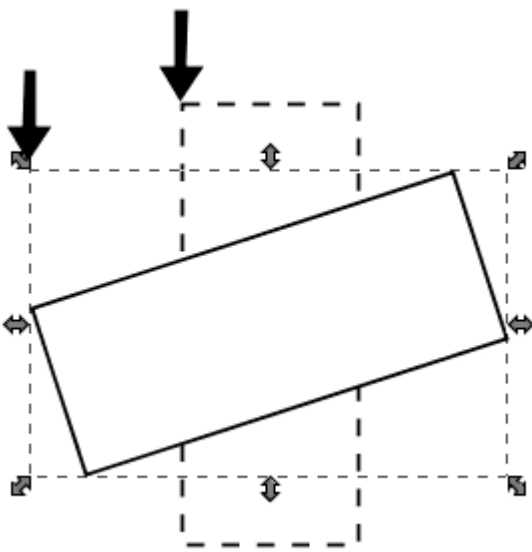
Long story short, using the `rel*` properties let you animate the shape using bindings and not worry about the details of the layout system and how they'll resize the coordinates at runtime.

Binding Rotation

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 a rotation property that can be edited directly or bound to something dynamic like a tag.

BINDING ROTATION WARNING

Binding the rotation comes with one big warning: observe that 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. 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.

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

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

Working with Components

Selecting Components



INDUCTIVE
UNIVERSITY

Selecting Components

[Watch the video](#)

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. Make sure that the selection tool () is the active tool. Simply 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 that are about to become selected will be given a yellow highlight border.

Tree Selection

By selecting nodes in the **Project Browser** tree 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.

Manipulating Components



INDUCTIVE
UNIVERSITY

Manipulating Components

[Watch the video](#)

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 8 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.

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. To nudge the top or left edge of the selection, use **Ctrl-Shift** combined with arrow keys. These nudge actions will resize one pixel at a time. To resize faster, use the **Ctrl-Shift-Alt** key, to move the component 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 arrow keys will move the selection one pixel at a time. Just like resizing with the arrow keys, to move faster, use the **Ctrl-Shift-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.

See also: [Component Animation](#)

Component Grouping



INDUCTIVE
UNIVERSITY

Component Grouping

[Watch the video](#)

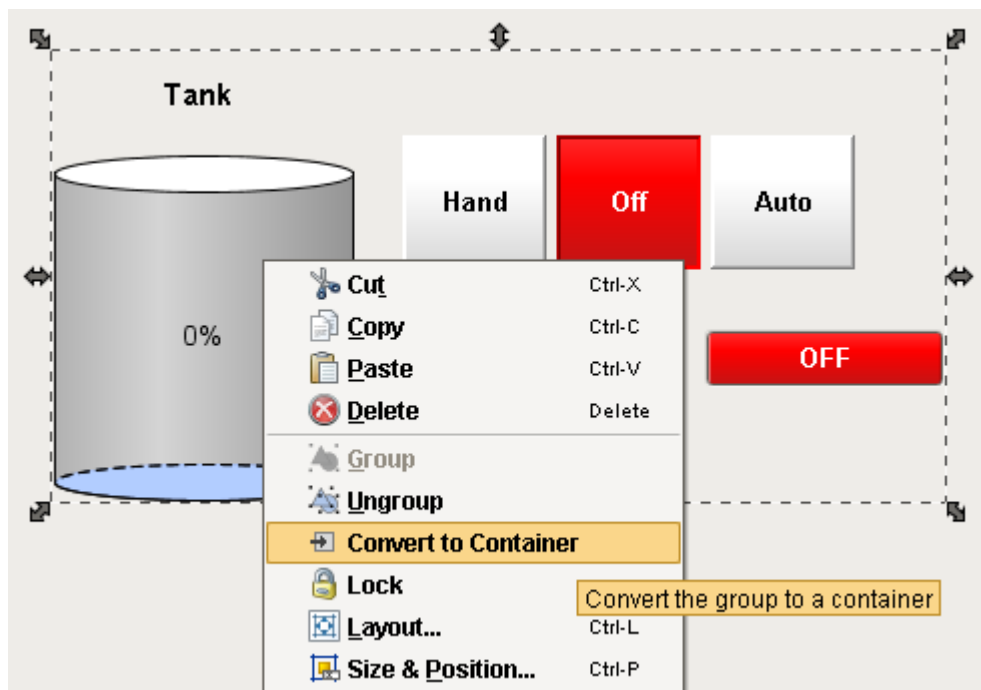
Shapes and components can be grouped together so that they act like a single component in the Designer. Grouping components is very similar to putting them in a Container. In fact, it is the same thing as cutting and pasting them into a perfectly-sized container and then putting that container into group 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 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 Layout



INDUCTIVE
UNIVERSITY

Component Layout

[Watch the video](#)

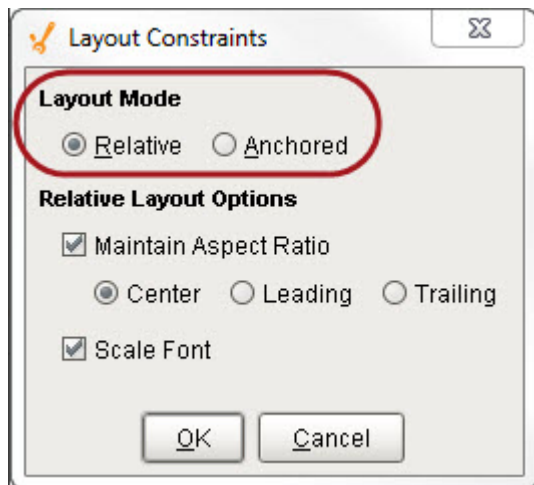
Layout is the concept that a component's size and position, relative to its parent container's size and position, can be dynamic. This allows the creation of windows that resize gracefully using either **Anchored** or **Relative** layouts and can optionally keep the original aspect ratio.

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.

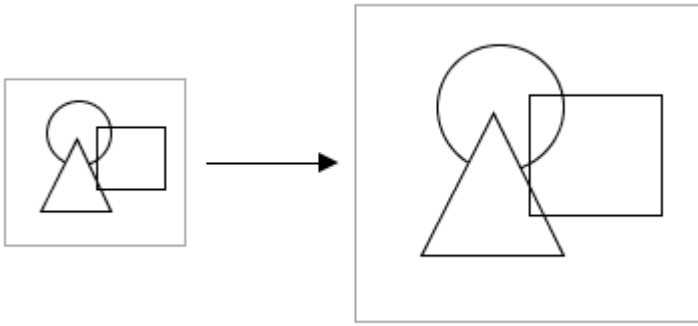


Layout Modes

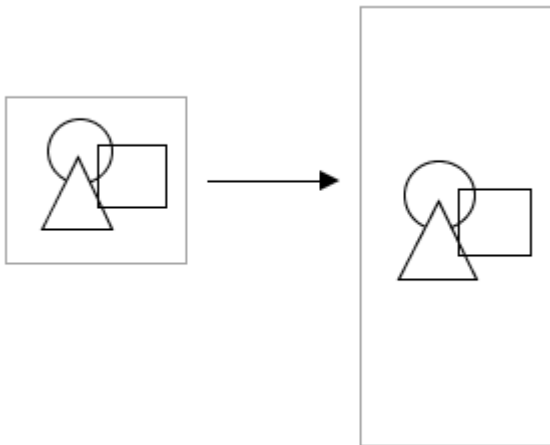
- **Relative** - This mode makes a component's size and location relative to its parent's size and location. When the parent changes size, the component changes accordingly. This creates components that auto-scale.
- **Anchored** - This mode makes the edge of a component's two axes (horizontal and vertical) 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.



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

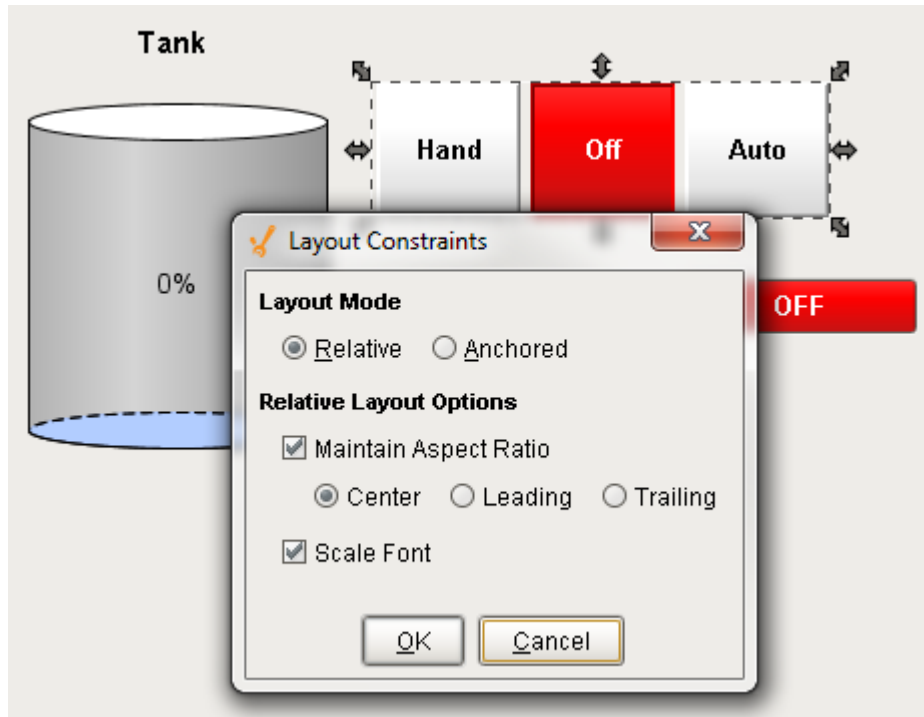
- **Maintain Aspect Ratio** - If selected, the component's original aspect ratio is preserved. Otherwise, it can stretch tall or wide.
- **Center** - When maintaining the aspect ratio, centers the component with respect to its parent.
- **Leading** - When maintaining the aspect ratio, aligns the component 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 **Designer > Window Editing**.

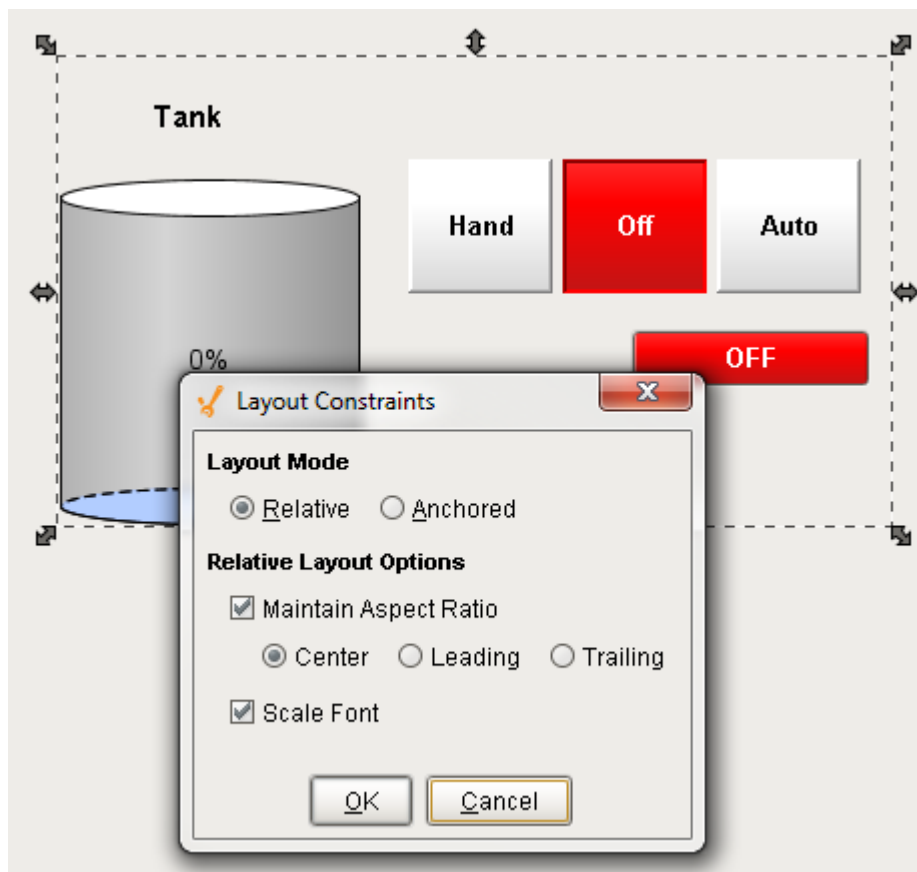
Font Scaling on Individual Components

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

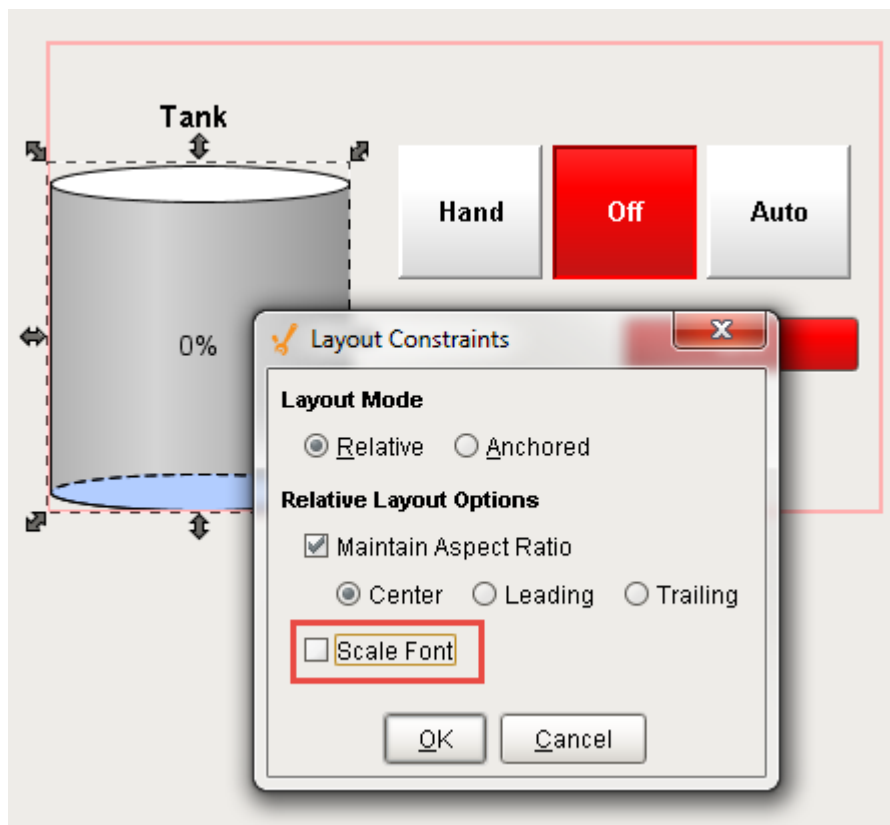
When individual components are grouped together, the default Layout settings are applied to the new group component, and to each component in the group. All the components in the group default to font scaling.



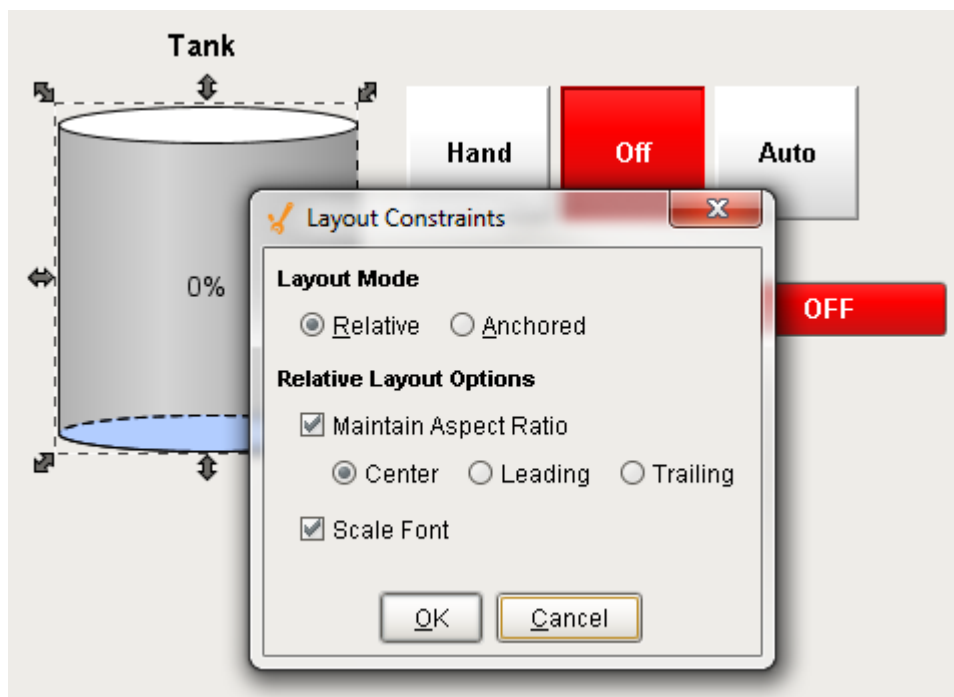
Font scaling on individual components in a group can be disabled by selecting the group, and double clicking on a single component. You will get a red outline around the group, then you can select the individual component(s), and disable font scaling.

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

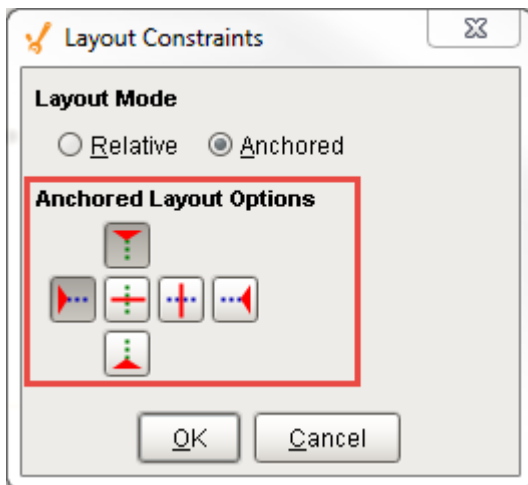
You can convert a group to a container to change the way scaling works. Ignition remembers the last Layout Settings you made to each individual component in that group prior to the conversion to a container (i.e., if any of the components had font scaling disabled, once the group is converted to a container, those same components will still have font scaling disabled).

Anchored Layout

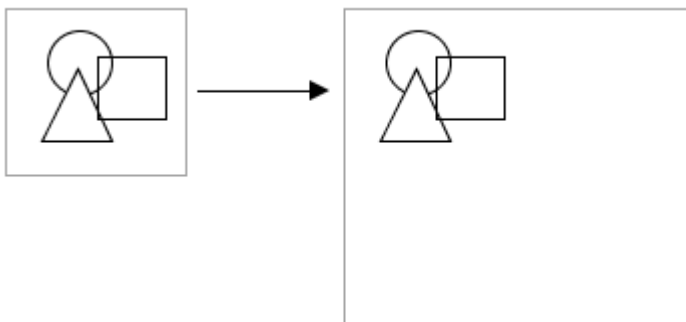
Anchored Layout lets you specify various "anchors" for the component. The anchors dictate how far each of the 4 edges of the component stay from their corresponding edges in the parent container.

Anchored Layout Options

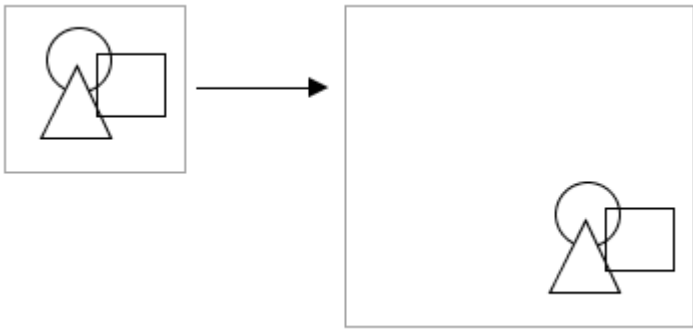
- **North/South** - If one of these is selected, the distance between that edge of the component and that edge of the container is preserved. If both are selected, the component will stretch its **height** to maintain both distances.
- **West/East** - If one of these is selected, the distance between 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.

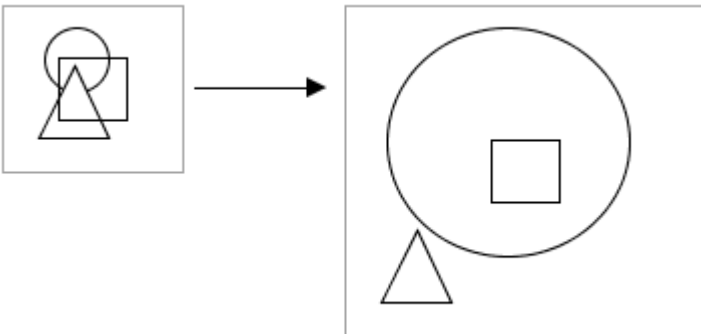


If you anchor bottom and right instead, the components will again stay the same size (since you didn't specify an anchor for their other edges, but they will stay a constant distance from their parent's right and bottom edges).



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.



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

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

Drawing Tools



INDUCTIVE
UNIVERSITY

Drawing Tools Overview

[Watch the video](#)

Drawing Tools Overview

Ignition 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 Ignition. All SVG (Scalable Vector Graphics) images are importable in Ignition and are made up of these basic shapes.

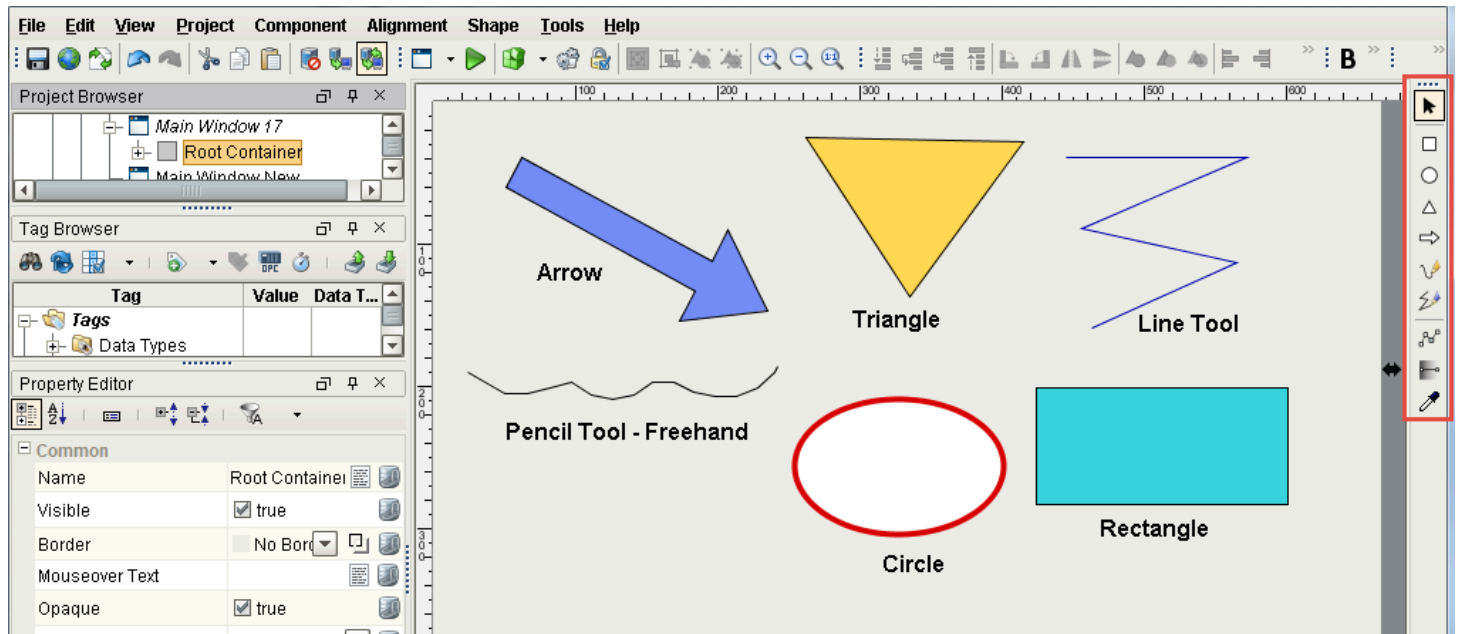
Using Drawing Tools

On the right-hand side of the Designer window is a small toolbar that contains all the 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. Once you draw a graphic, and want to drag a different graphic tool on to your window, click on the Selection tool. In order to switch between the various tools on the screen, you have to click the Selection tool before activating each tool. When a drawing tool is active, a toolbar will appear in the top menubar that has specific settings and actions for that tool. There are a number of tools here for your use, and each one makes the window editing workspace act differently when it's active.

Drawing Tools

By default, the Drawing Tools Toolbar will be on the right side of the Designer. It contains the selection tool, as well as various other tools used for drawing shapes and manipulating their edges and colors.

Normally the **Selection** tool () is active. 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](#).



Shape Tools

Shapes such as lines, rectangles and circles are created using the shape tools. Shape tools allow you to create various shapes as well as edit them after they are created. Click on the tool's icon to make it the active tool. You can also double-click on a shape to change to that shape's native tool. When a shape tool is active, a toolbar will appear in the top menubar that has specific actions and settings for that tool.

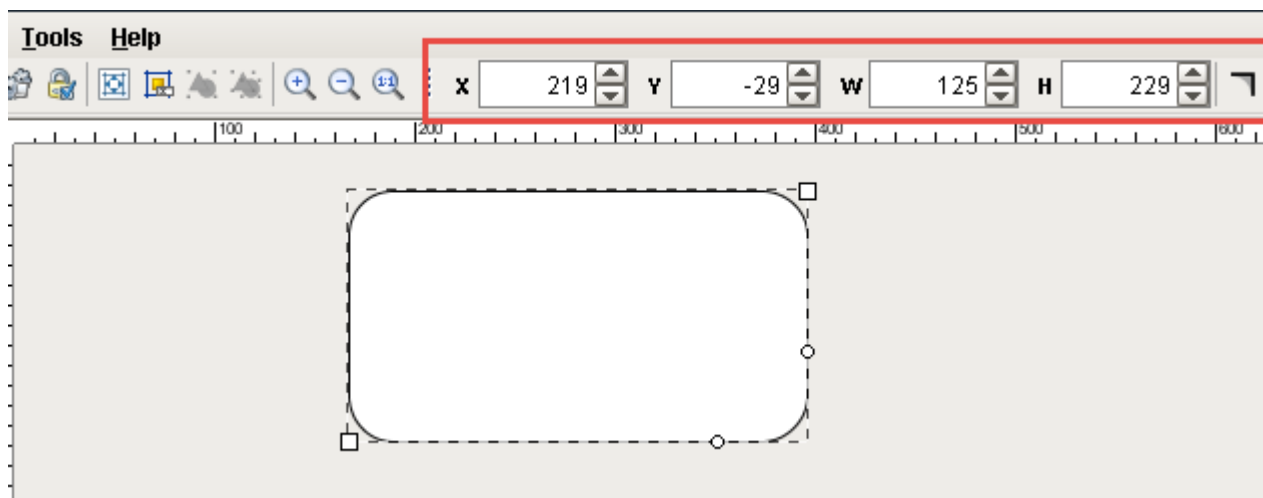
When you create a shape, by default, it has a Fill Paint color of white. After a shape is created, you can change its Fill Paint color, Stroke color, and Stroke Style properties. See [Fill and Stroke](#) for more information. All shapes can be treated as paths and be used with composite geometry functions to alter or create other shapes. For more information, see [Bezier Curves](#), [Editing Shape Paths](#), and [Shape Geometry](#).

Rectangle Tool

The **Rectangle** tool () creates and edits rectangle shapes. To create a rectangle, select the tool and drag it 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.



Ellipse Tool

The **Ellipse** 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 drag it 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/Star 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 3 corners (a triangle) or more. 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°. The Polygon Tool also has a **Star** option that when selected, will give you a second handle that is between each corner, 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



INDUCTIVE
UNIVERSITY

Drawing a Line

[Watch the video](#)

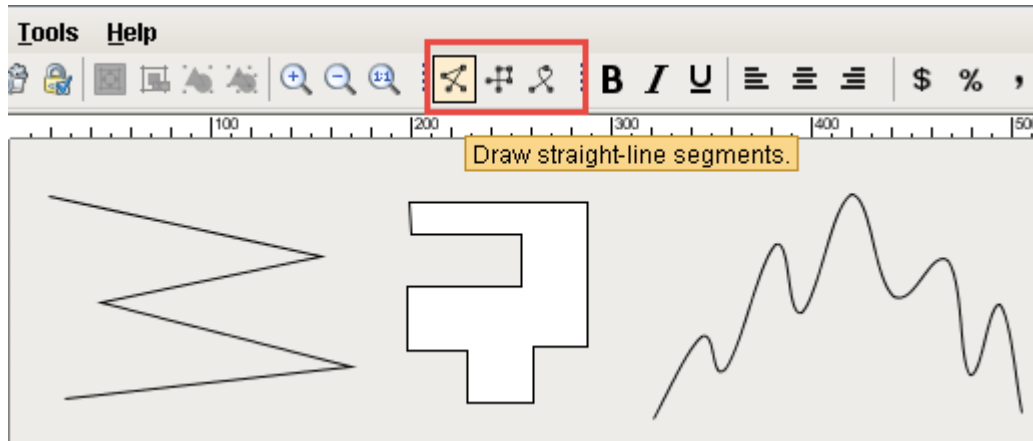
The **Line** tool () can be used to draw lines, arbitrary polygons, or curved paths. Unlike all of the other tools, you don't drag to create new paths with the line tool. Instead, you click for each vertex you'd like to add to your path.

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:

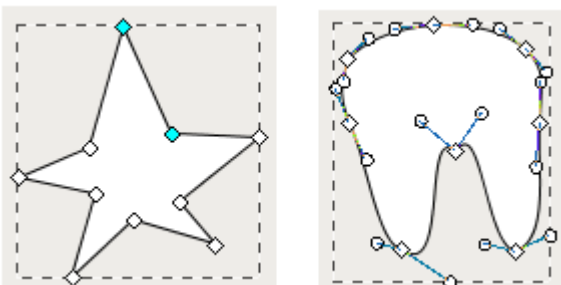
- straight-line segments
- perpendicular-line segments
- curve-line segments



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

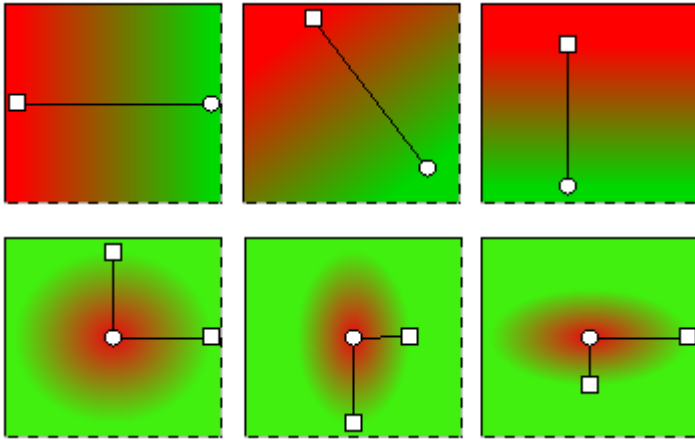
All shapes and paths can be edited directly by using the **Path** tool (). This tool lets you directly modify the nodes in the path, adding new nodes, removing nodes, and toggling segments between straight or curved. Learn more about paths in the [Editing Shape Paths](#) section.



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. To learn more about gradients, go to the [Fill and Stroke](#) section.



Eyedropper Tool



INDUCTIVE
UNIVERSITY

Eyedropper Tool

[Watch the video](#)

The **Eyedropper** tool () is used to set the 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. Don't forget 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. Note, that 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.

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

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

Fill and Stroke



INDUCTIVE
UNIVERSITY

Fill and Stroke

[Watch the video](#)

All shapes have three properties that affect how they look: **Fill Paint**, **Stroke Paint**, and **Stroke Style**.

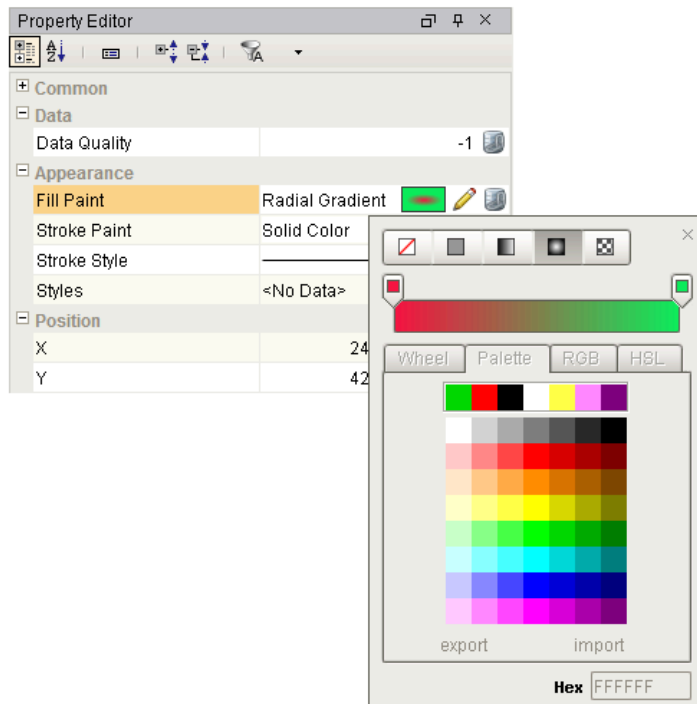
- **Fill Paint** is a property of the type **Paint** that represents the interior or fill color of the shape.
- **Stroke Paint** property (also a **Paint**) represents the color of the shape's outline.
- **Stroke Style** property is a property of type **Stroke** that affects the thickness, corners, and dash properties of the shape's outline.

Fill Paint

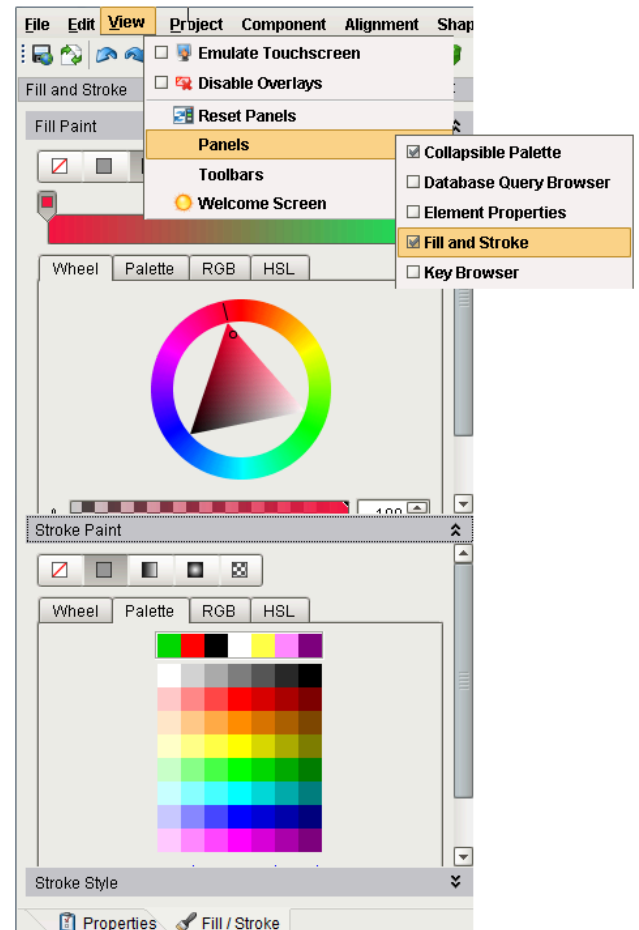
Editing Paints

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 **pencil** icon () or open up the dedicated **Fill and Stroke** panel from the **View** menu.

Property Editor Table using the Pencil



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** (□). If 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 at.
2. **Solid Color Paint** (■). This paint type 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** (▒). Linear gradients smoothly blend any number of colors along a straight line across the shape. Each color is called a Stop. Each stop is represented as a 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** (■). Radial paints are very much like 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** (■). This 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



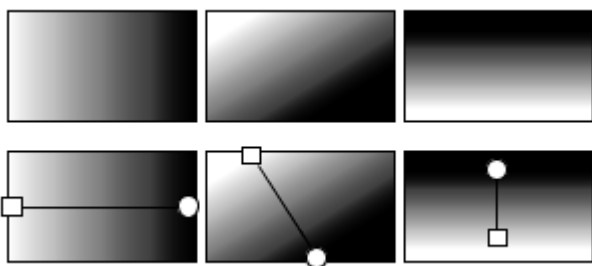
INDUCTIVE
UNIVERSITY

Gradients

[Watch the video](#)

Gradient Paint Bounds

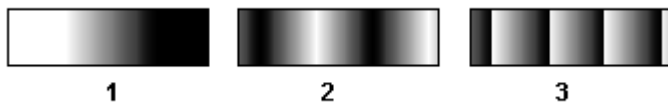
The two gradient paints, **Linear** and **Radial**, are more than a list of colored stops, they also need to be placed relative to the shape. The same gradient may look wildly different depending on how it is placed against the shape. By default, a Linear gradient will run horizontally across the width 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.



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.

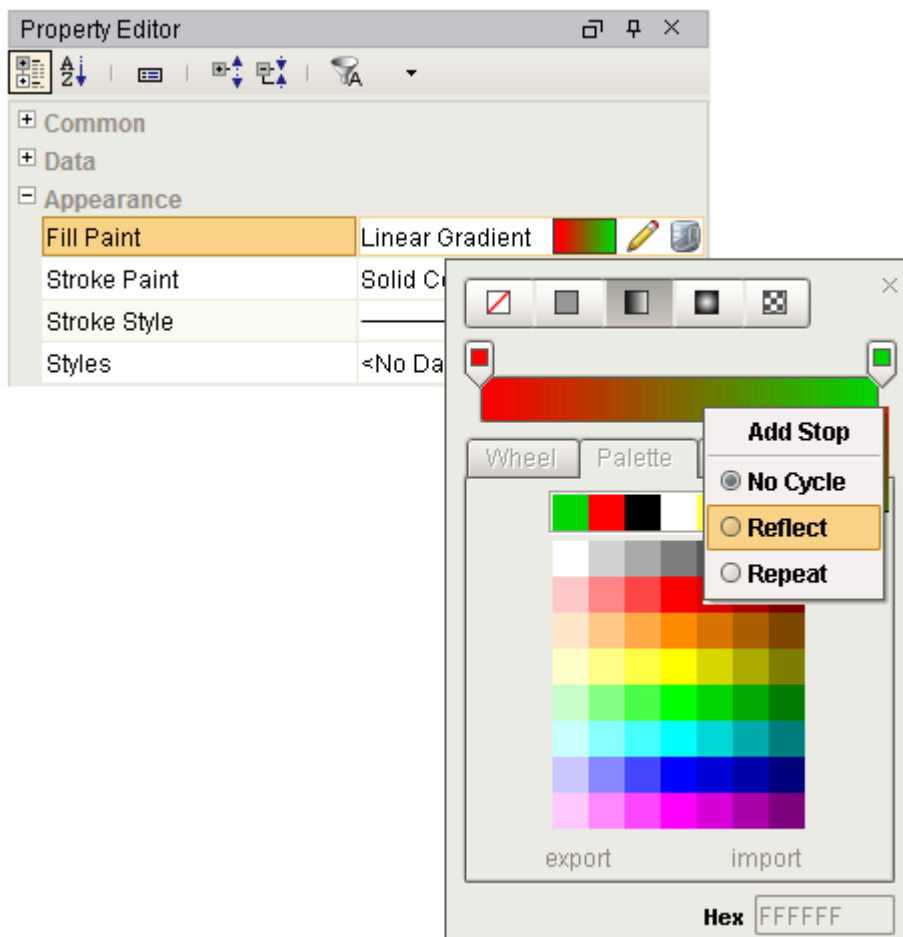
The cycle modes are illustrated below: **No Cycle**, **Reflect**, and **Repeat**.



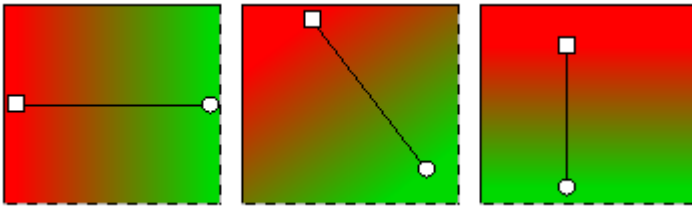
1. **No Cycle** - The first and last stops are repeated forever after the edge of the gradient bounds.
2. **Reflect** - Beyond the bounds of the gradient, it will be reflected and drawn in reverse, and then reflected again, creating a smooth repetition.
3. **Repeat** - Beyond the bounds of the gradient, it will be repeated forever.

To Set a Gradient

1. In the **Designer**, select your component.
2. In the **Property Editor** under **Appearance**, click the **Pencil** icon, and then right click within the **Preview Strip**.
3. Select either the **Linear** or **Radial** gradient.
4. You will see two stops: white and black. You can click on each **Stop** and choose a different color.
5. You can also add/remove **Stops**, and select your desired cycle: **No Cycle**, **Reflect** or **Repeat**.

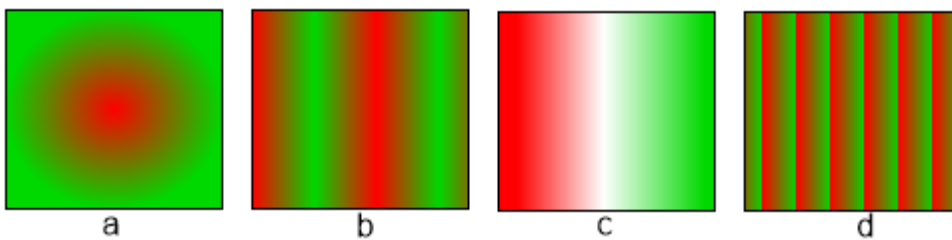


6. With your component still selected on your window, and click the **Gradient tool** in the toolbar. You'll notice a line on your component. Now, you can drag the line's handles to change the orientation and lengthen or shorten the gradient.



7. Here are a few examples of what you can do with gradients:

- i. **Radial** - No cycle
- ii. **Linear** - Reflective
- iii. **Linear** - with a 3rd Stop
- iv. **Linear** - Repeat



Stroke Style

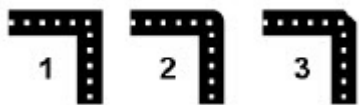
A shape's stroke paint is only half the story. The **Stroke Style** is also an important component of how an outline is drawn. Primarily the style controls the thickness of the line drawn, but it also can be used to create a dashed line. The setting for thickness is specified in pixels, and creating 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 a sharp corner. 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 on the left shows the same miter join with two different miter length settings. The first drawing illustrates the length of the miter join.



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

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

Shape Geometry



INDUCTIVE
UNIVERSITY

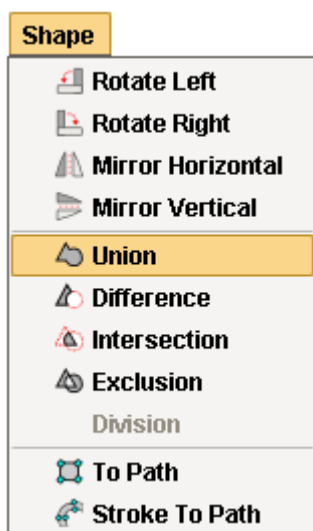
Shape Geometry

[Watch the video](#)

Shape geometry allows you to apply functions such as a union (combine), difference (exclusion), and intersection (division) on multiple shapes to get the resulting shape that you want.

Creating and Editing Shapes Using Constructive Area Geometry

Editing paths directly can be a bit awkward. Using Constructive 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.



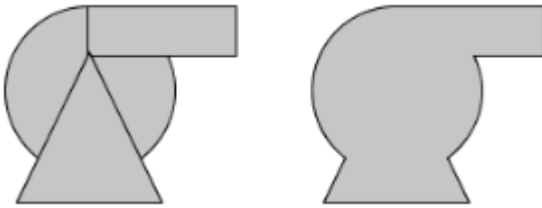
SELECTION ORDER MATTERS

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.

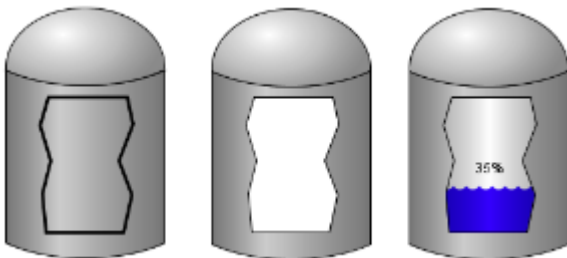
Union

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.



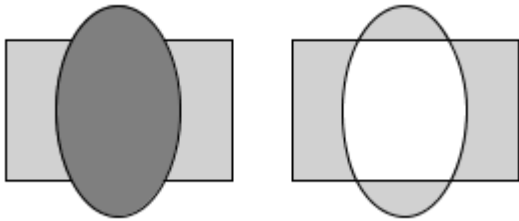
Intersection

The result of an **Intersection** function () will be the area only where two shapes overlap. The example shows how the "top" of the tank in the difference example was easily made using two ellipses.



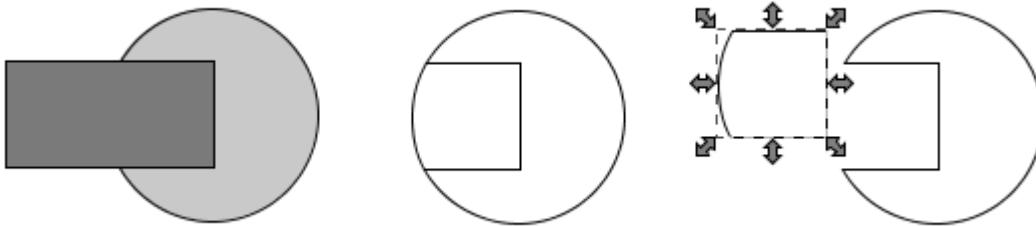
Exclusion

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.



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

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

Bezier Curves

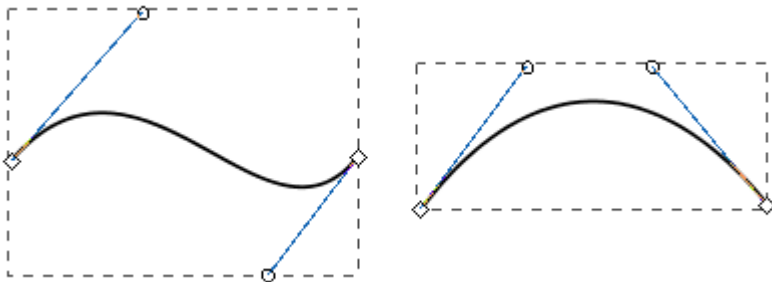


INDUCTIVE
UNIVERSITY

Bezier Curves

[Watch the video](#)

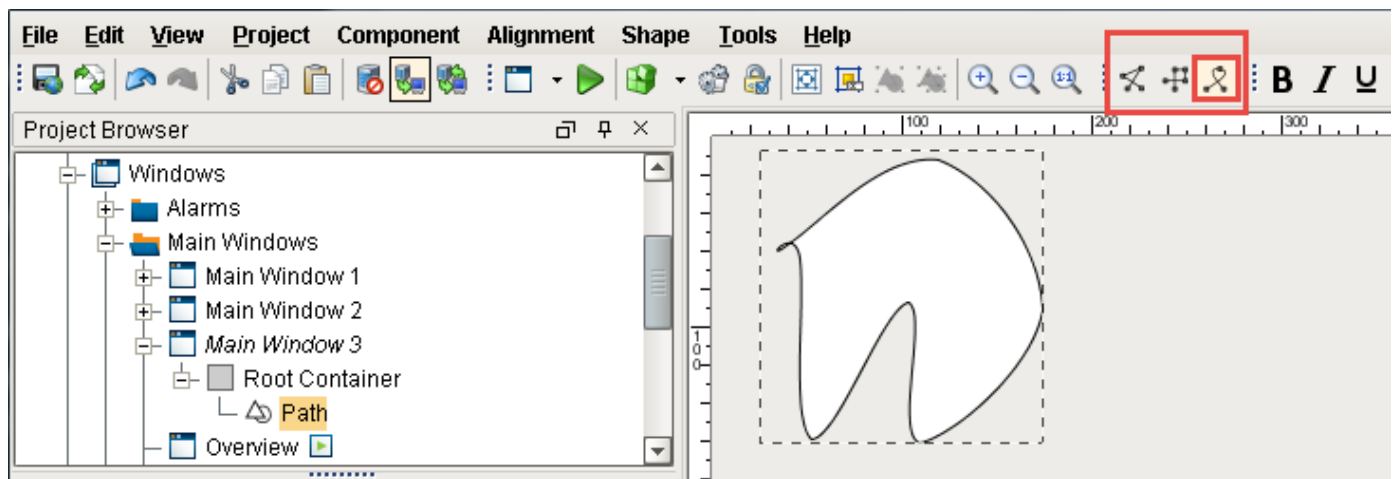
A **Bézier curve**, also sometimes called a quadratic curve, is a type of curved line used in vector graphics that connects two points, allowing you to create smooth vector graphic shapes. A Bézier curve is configured using four points: the two end-points and two control points. The curve starts along 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 Bezier Curves

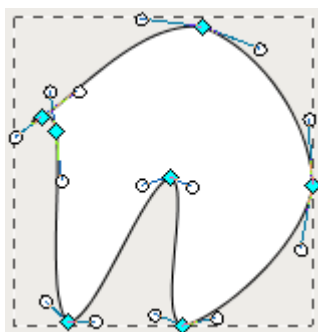
Curves are made using the [Line](#) tool.

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 icon (). 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.



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

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

Editing Shape Paths



INDUCTIVE
UNIVERSITY

Editing Shape Paths

[Watch the video](#)

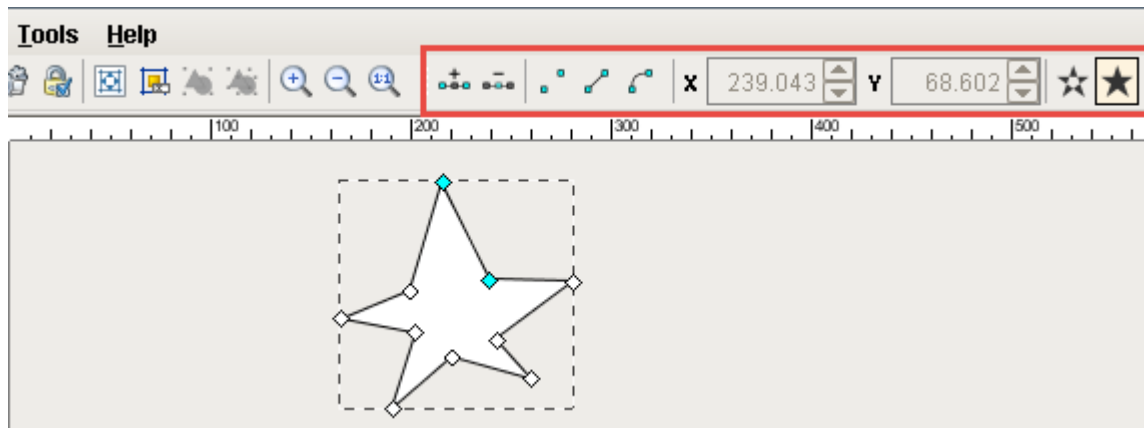
Once you draw shapes using the drawing tools in Ignition, it's possible to alter and edit shapes after they've been created. By default, all shapes have a white fill color. Editing the paths of your vector graphic shapes is done by using the **Path tool** () . Simply select any shape or line while the 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.

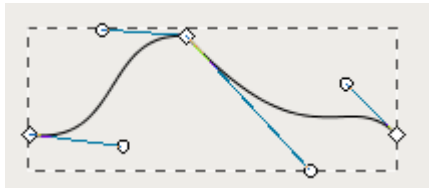
Editing a Shape Path

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, simply use the toolbar functions that become visible when the path editor tool is active. Points can also be added and removed using the functions on the Path Editor toolbar. 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**.



Bézier segments also have handles on their control points, represented by small circles with a line drawn back to an endpoint that can be dragged to change the curvature of the segment.



TIP

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.

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

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

Scalable Vector Graphics (SVGs)



INDUCTIVE
UNIVERSITY

**Scalable Vector Graphics
(SVGs)**

[Watch the video](#)

Using SVGs

Ignition can import SVGs (Scalable Vector Graphic) into a 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.

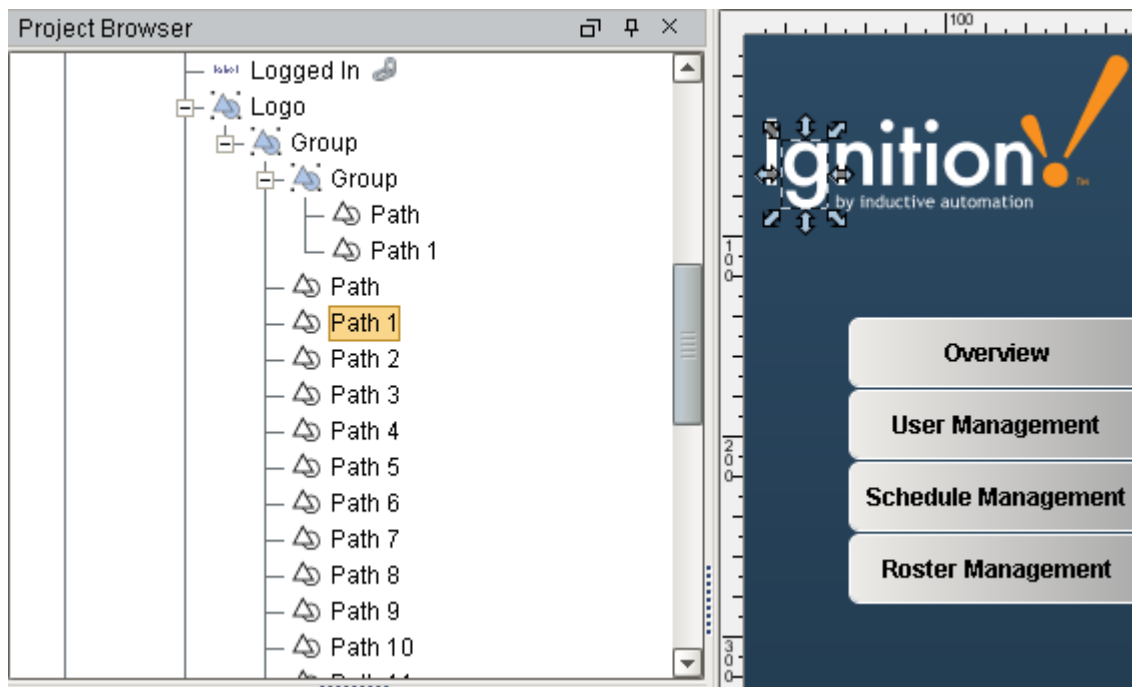
NOTE

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](#).

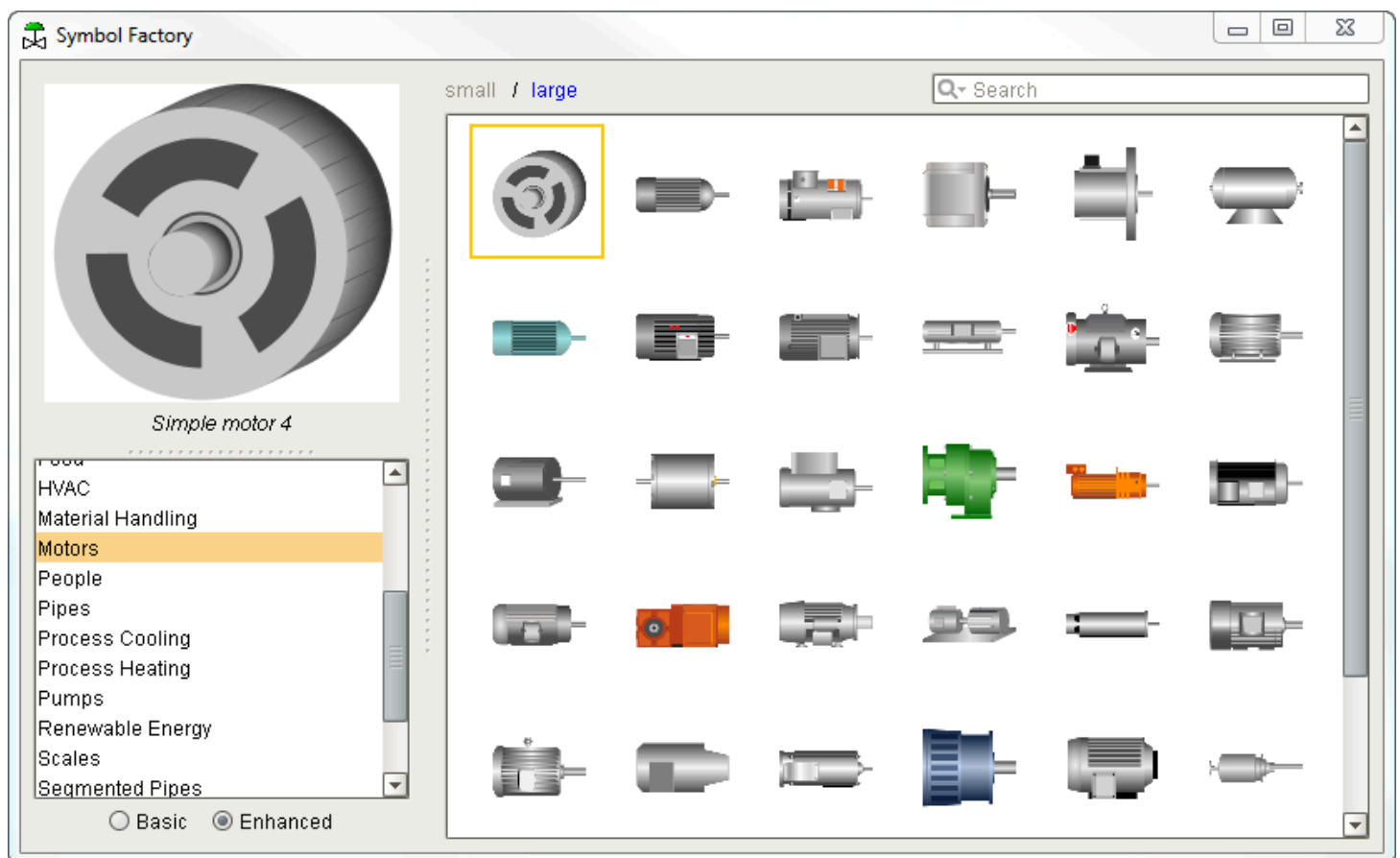
SVGs as Grouped Components

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.



Using the Symbol Factory

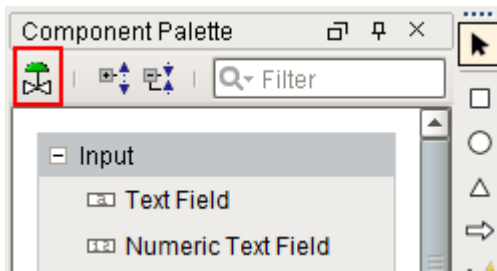
Ignition has a built-in library of images for you to get started on your project with. If you have the Symbol Factory module installed, you have nearly 4,000 industrial symbols at your fingertips to use in your projects. You can add these images to your project as they are, or edit them to be exactly what you need.



To Open the Symbol Factory Browser

1. Launch the Designer and open your project.
2. Choose **Symbol Factory** under the **Tools** menu or the project navigation tree. If you can't find these, the Symbol Factory module probably isn't installed. The Symbol Factory browser opens as a pop-up window that stays on top of the Designer.

There is also an icon for quick access to the Symbol Factory images in your component Palette.



3. Browse the different categories to explore what symbols are available, or search to find a specific symbol.
4. Find a symbol that you'd like to use and drag it onto an open Vision window. The symbol will become a group of shapes. Just like any group, you can double-click it to get to the shapes inside, or simply un-group it. This way you can edit the symbol if you need to.

i NOTE

If the Symbol Factory module is in trial mode, you'll only be able to use the first symbol from each category.

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

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

Color SVG Dynamically



INDUCTIVE
UNIVERSITY

Color SVG Dynamically

[Watch the video](#)

Introduction

In Ignition, one useful HMI technique is to color an individual component of the SVG (Scalable Vector Graphic). Whether you are bringing in a vector graphic from the Symbol Factory or importing an SVG from your computer, you can easily color in individual components of your SVG.

As with almost everything in Ignition, there are multiple ways of accomplishing the same result. For each component of the SVG, there are different binding types you can use to dynamically change the color based on a value.

Reference Information

There are a few pieces of assumed knowledge in this article that are common to all the coloring options. See below to learn more about each.

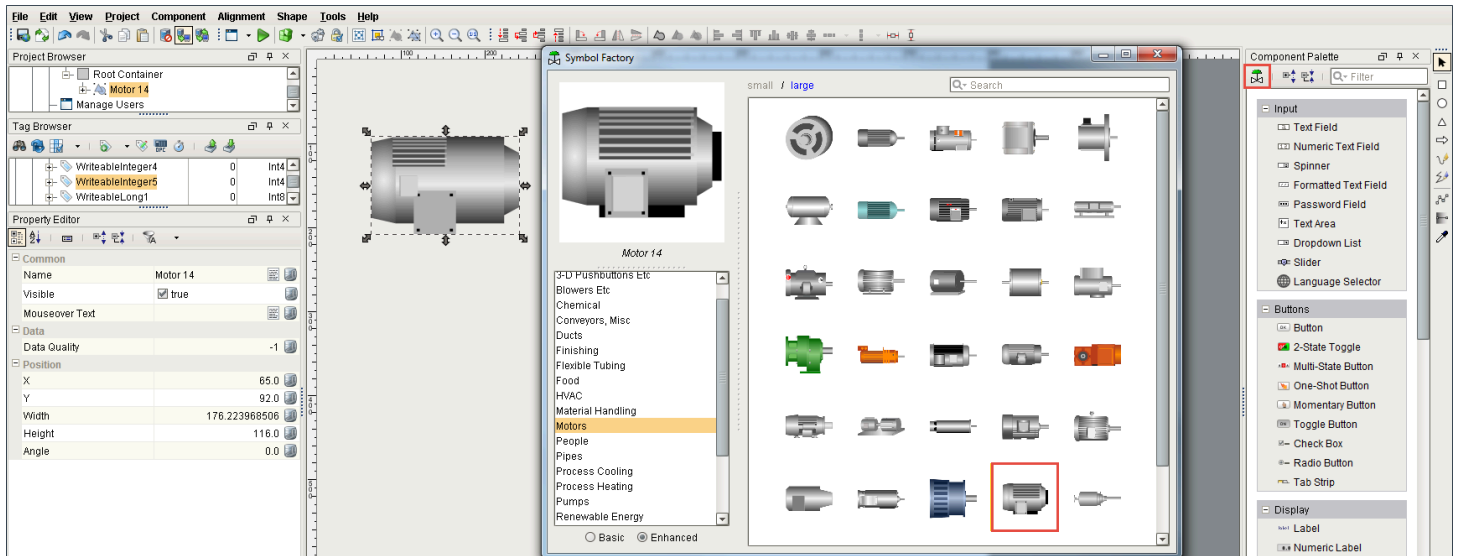
What is a "Component of the SVG?"

There can be many components inside an SVG. These components are often referred to by a number of different names: sub-components, members, parts, pieces, etc. All these names mean the same thing. For clarity, we will refer to an individual item inside an SVG as a '**component of the SVG**'.

Pre-made SVGs from Symbol Factory and selecting components of the SVG

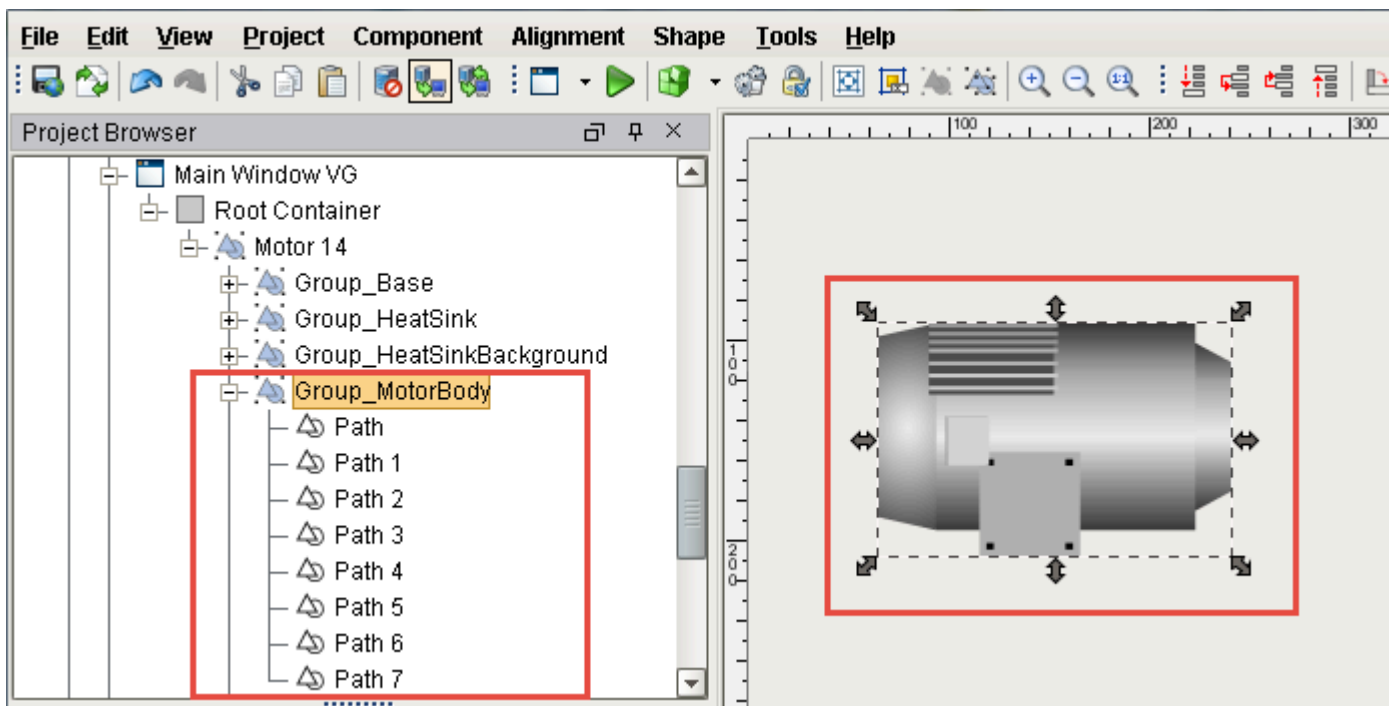
The Symbol Factory has a library of symbols to choose from. Choose your symbol and drag it to your workspace.

In Designer, open the Symbol Factory () in the Component Palette and drag a motor on to your workspace.

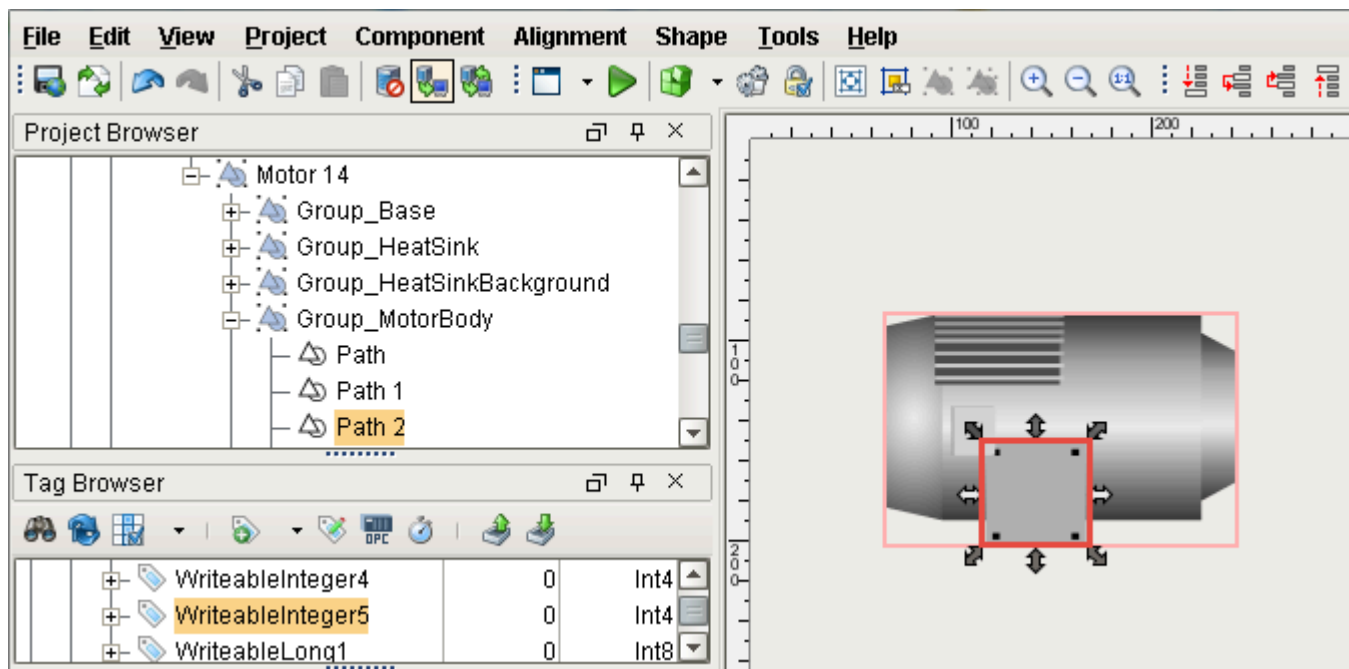


When you drag a vector graphic into a window, it's not just one flat image, it's actually comprised of many individual components.

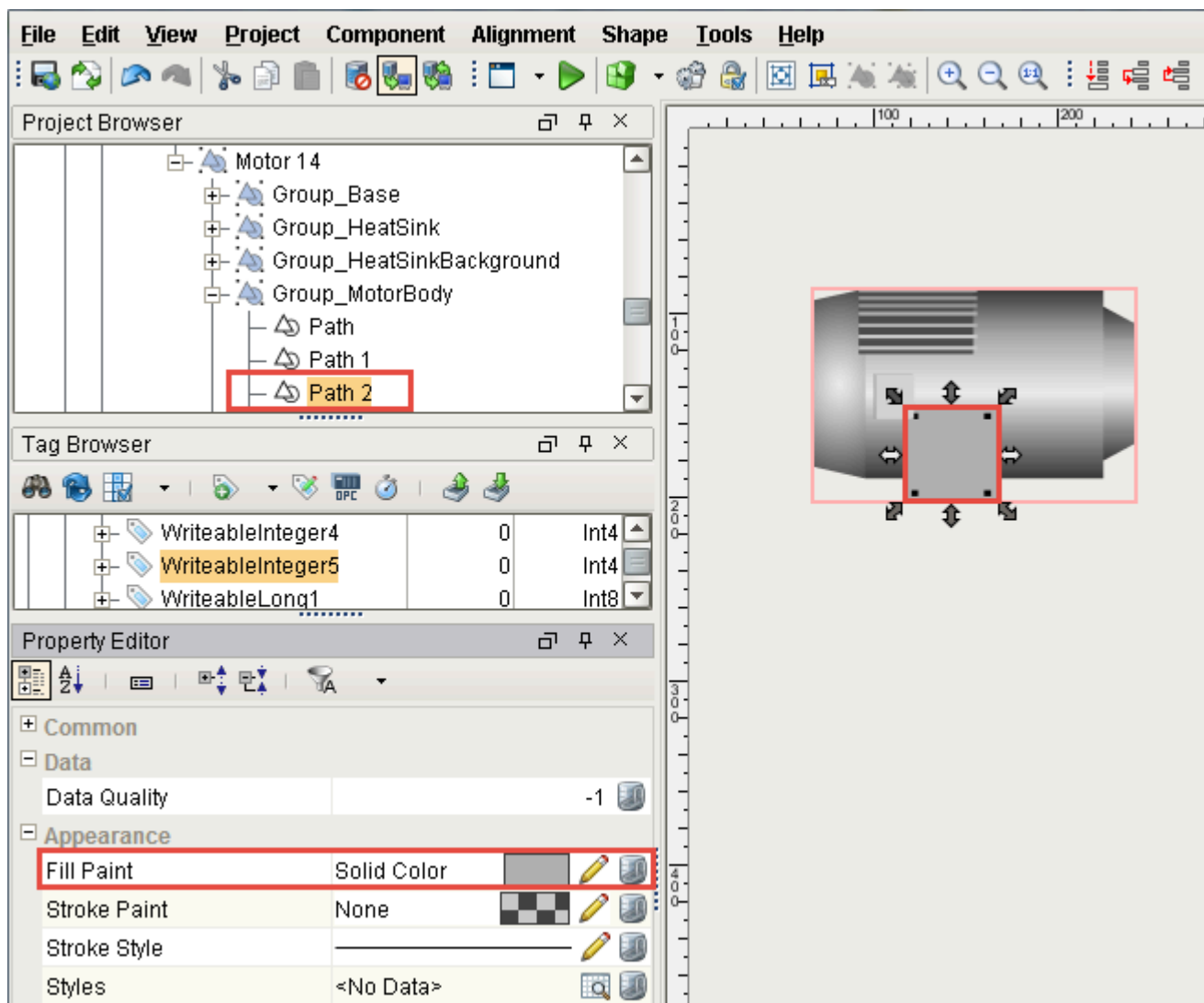
Expand the vector graphic in the **Project Browser**. You can see that the Motor is made up of many different components of the SVG. Each of these components can be colored independently.



In this example, let's select the square as the individual component of the SVG. Double click on the **Motor**, double click the **square**, and then single click on the square again.



Every individual component of the SVG has a Fill Paint property that can dynamically change based on a value. Click on the **Fill Paint** property binding icon.



Once you have the Fill Paint selected, you can bind it to a Tag or anything else you want. Since there are a number of ways to bind the Tag, let's explore our options in the following sections.

Tinting an SVG

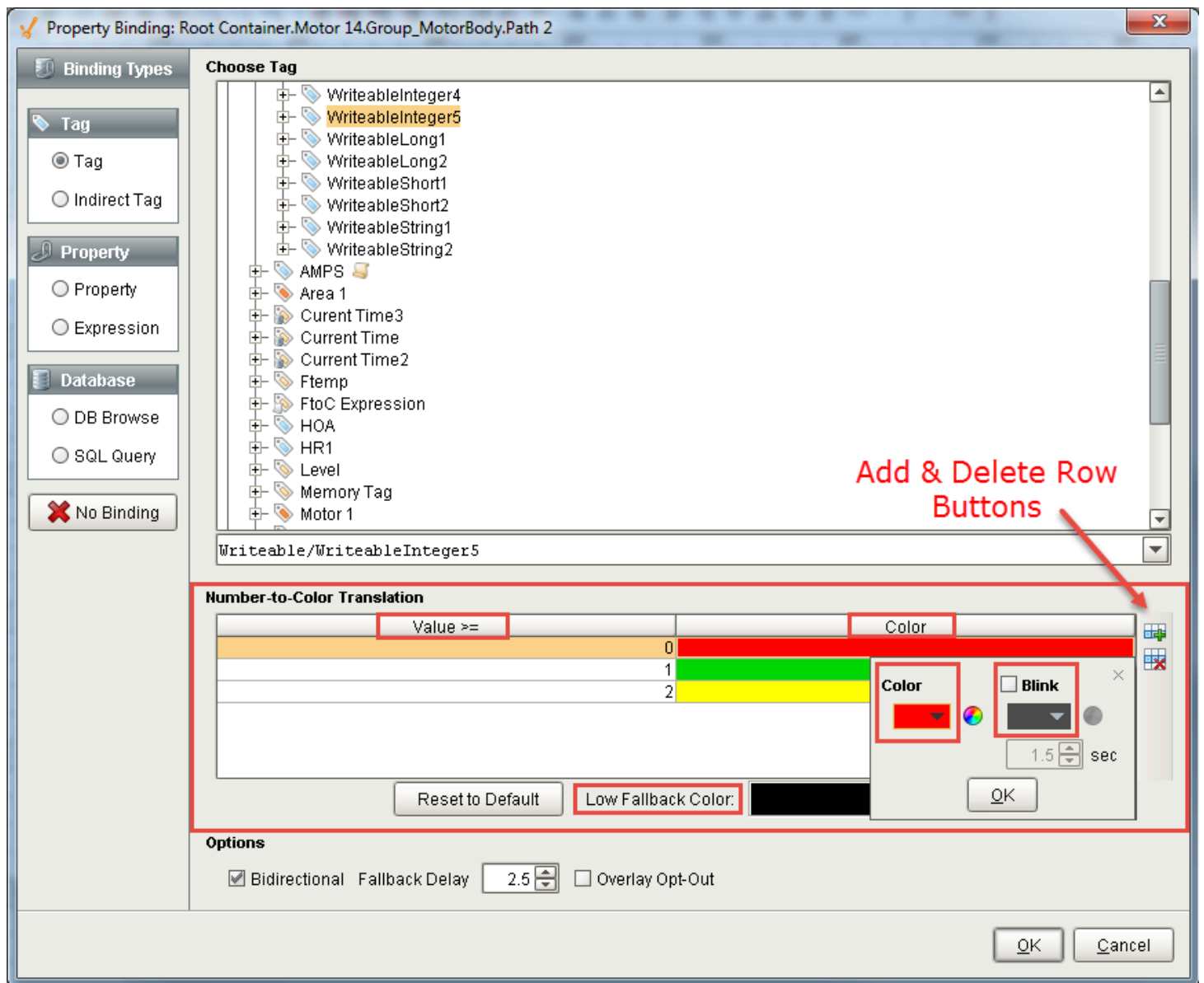
This page talks about coloring a specific part or parts of an existing SVG. There is also a way to tint the entire SVG one color. See applying a [Tint to the entire SVG](#).

Can't Write to your Tag during Testing?

While testing, if you are unable to write to a Tag in the Tag Browser or through buttons in the designer, you might not have it set to allow writes. The first time you try to write, a window will popup in the Designer stating you are in Read Mode. Choose '**Enable Read/Write Mode**' so you can write to your Tags for testing.

Number-to-Color- Translation Table

The Fill Paint property uses the Number-to-Color Translation table in Ignition, commonly known as Color Mapping. This is where you map a value to a color. For every number range there is a set color. 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 a value, simply use the add row () and remove row icons () on the right side of the Number-to-Color Translation table. In this SVG example, value '0' is set to red, value '1' is set to green, and value '2' is set to blink between **yellow** and **orange**. If you are using this as a tint overlay, make sure each color is semi-transparent so you can see the actual symbol behind it. There is a **Low Fallback Color** option so when a value falls below your lowest value, a default color can be set.



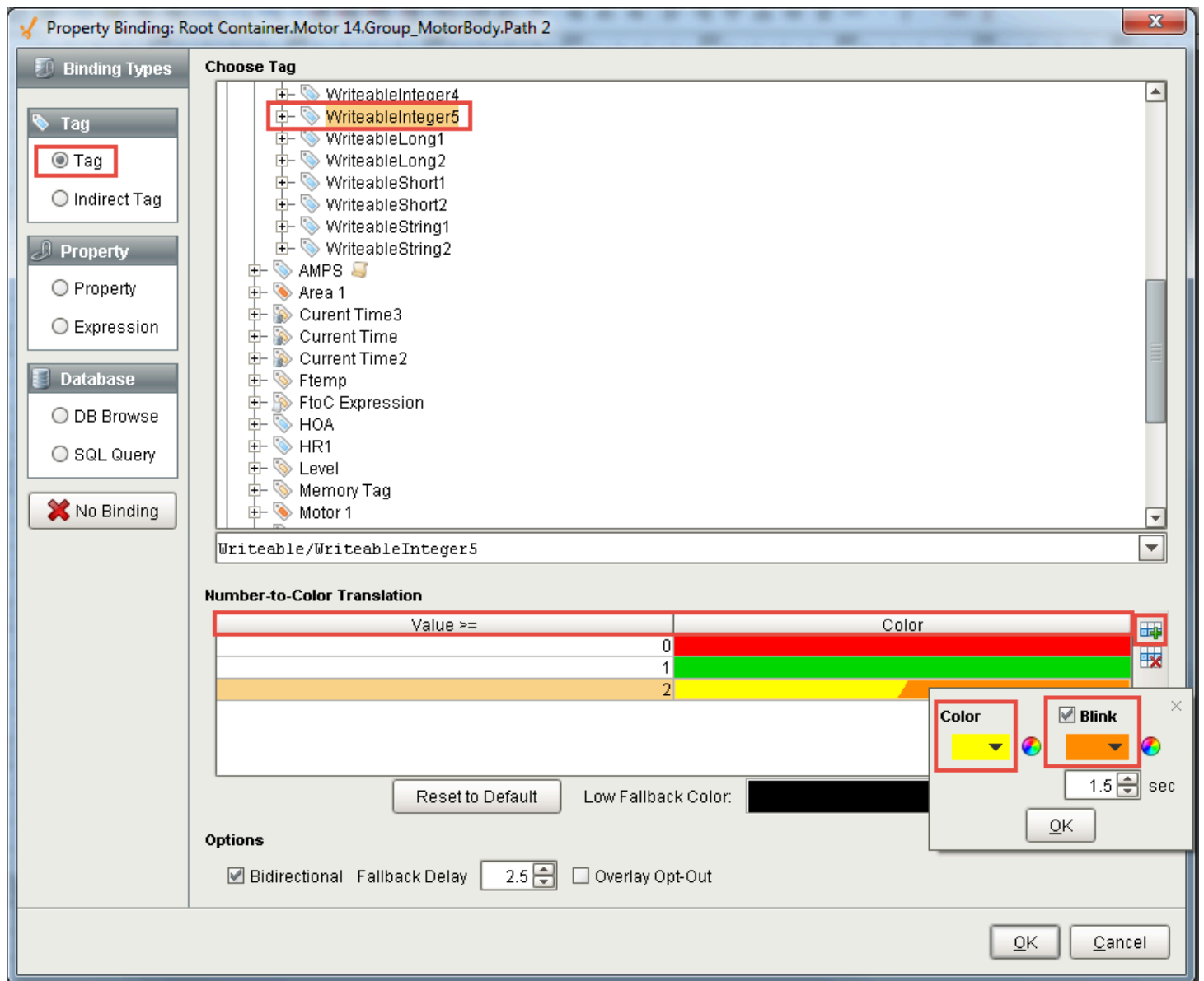
Tag Binding

A Tag Binding is a very straight-forward binding type. It simply binds a single property directly to a Tag. For this example, every time the Tag binding is evaluated, the Fill Paint property changes, pushing the new color to the component of the SVG. To learn more about tags, refer to the section on [Understanding Tags](#).

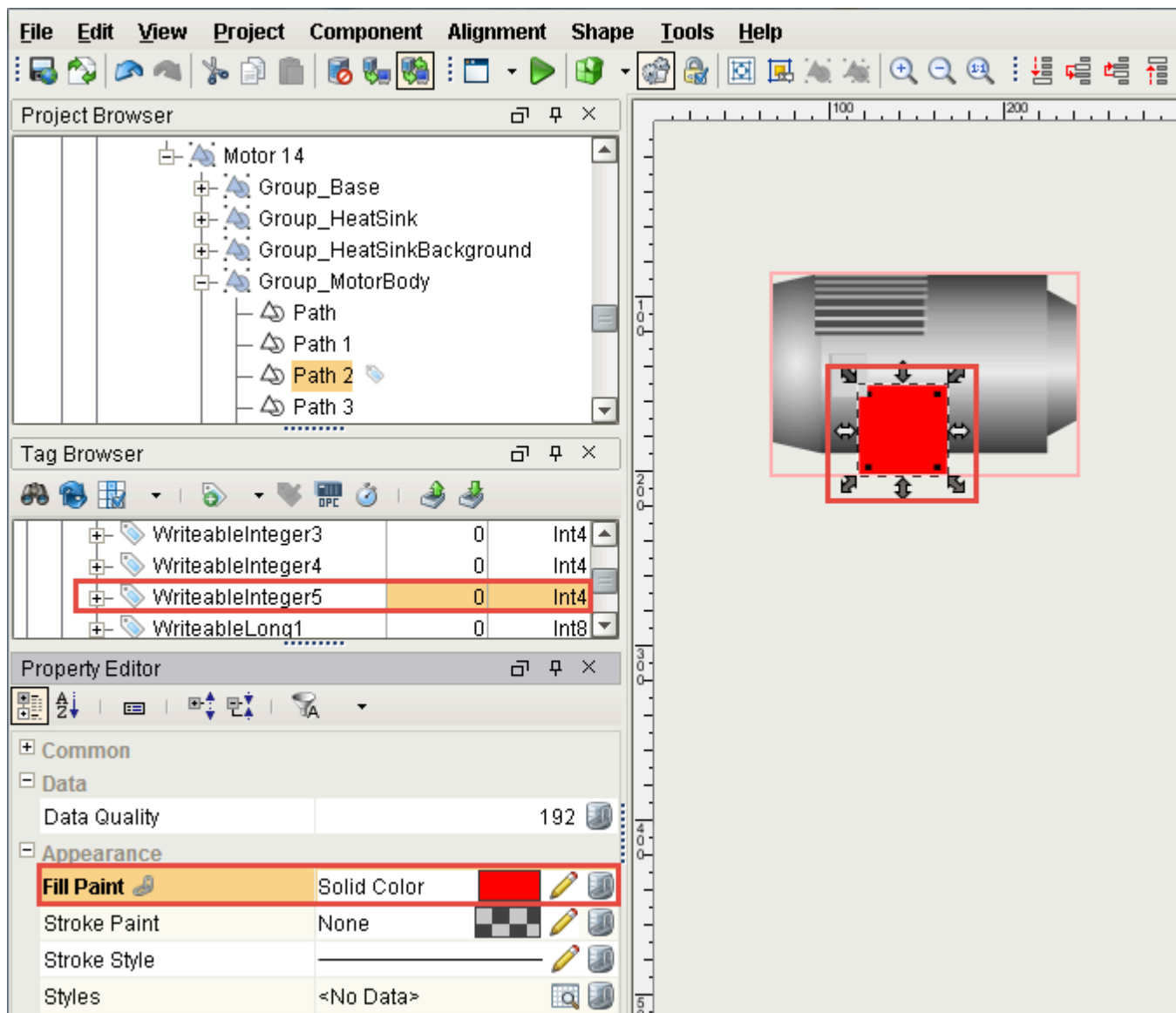
Bind the **Fill Paint** property to any numeric Tag. If you are not familiar with numeric, it's any number value. Integers are the best, but you can also use a boolean, float, etc. In this example, **'WriteableInteger5'** was selected. Set the values in the **Number-to-Color Translation** table to drive the color of a component of an SVG.

- On Value **0**, make the component **red** by double clicking on the first row. Under Color, select **red** and press **OK**.
- On Value **1**, make the component **green**. Double click the second row and under Color, select **green** and press **OK**.
- Add a row by clicking the **Add Row** icon ().

- On Value **2**, make the square component blink between **yellow** and **orange**. Double click the third row and check the **Blink** box, and select the colors **yellow** and **orange**.
- Press **OK**.



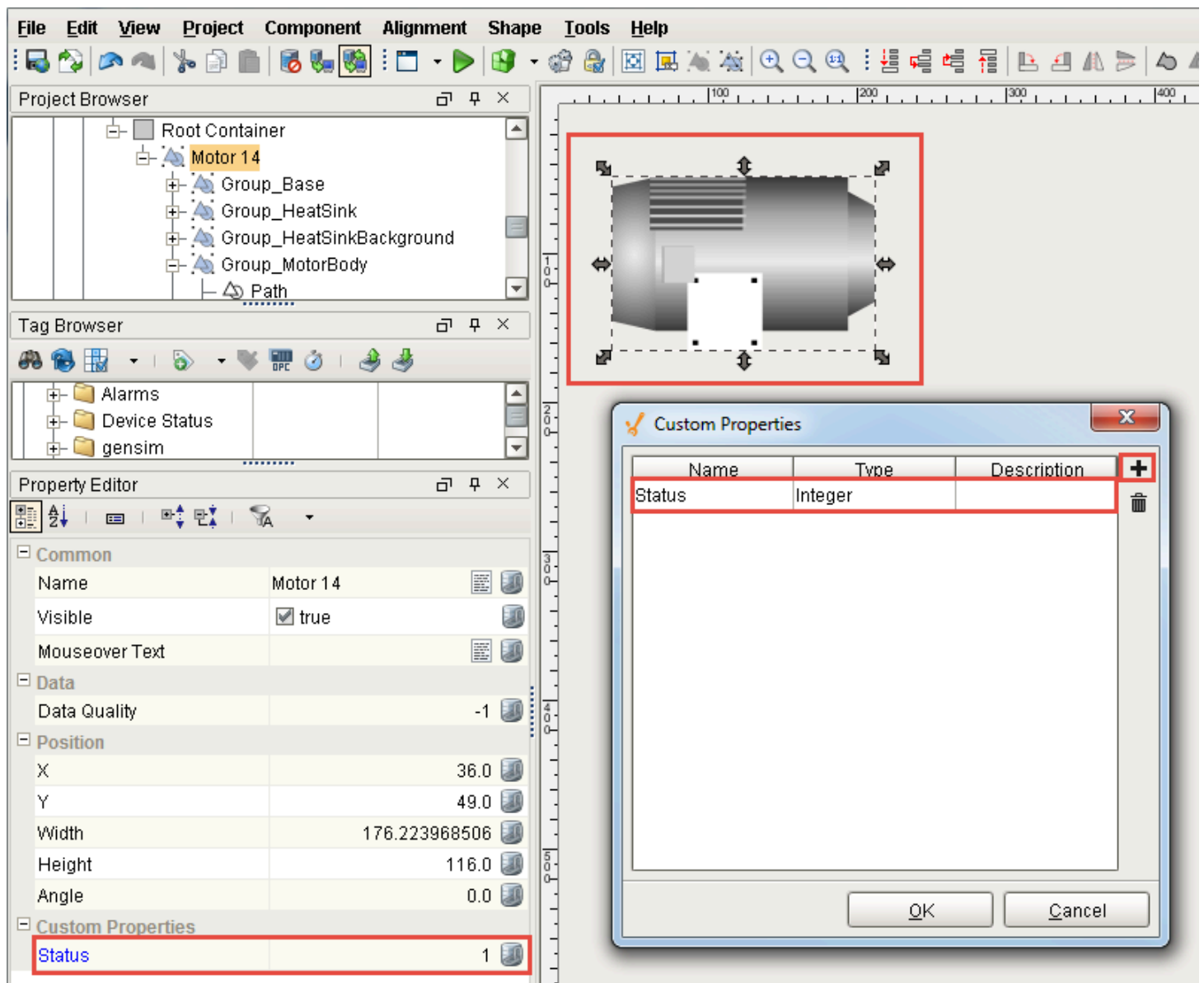
Once you press OK button, you'll notice that the component of the SVG changed color based on the current value of the Tag. The Value is set to '0' and the color is **red**. To change the Tag value, go to the **Tag Browser**, double click on your tag value, and change the value from '0' to '1'. Press **Enter**. The component of the SVG will change from red to green. Now, change the Value from '1' to '2', and the color will blink **yellow** and **orange**.



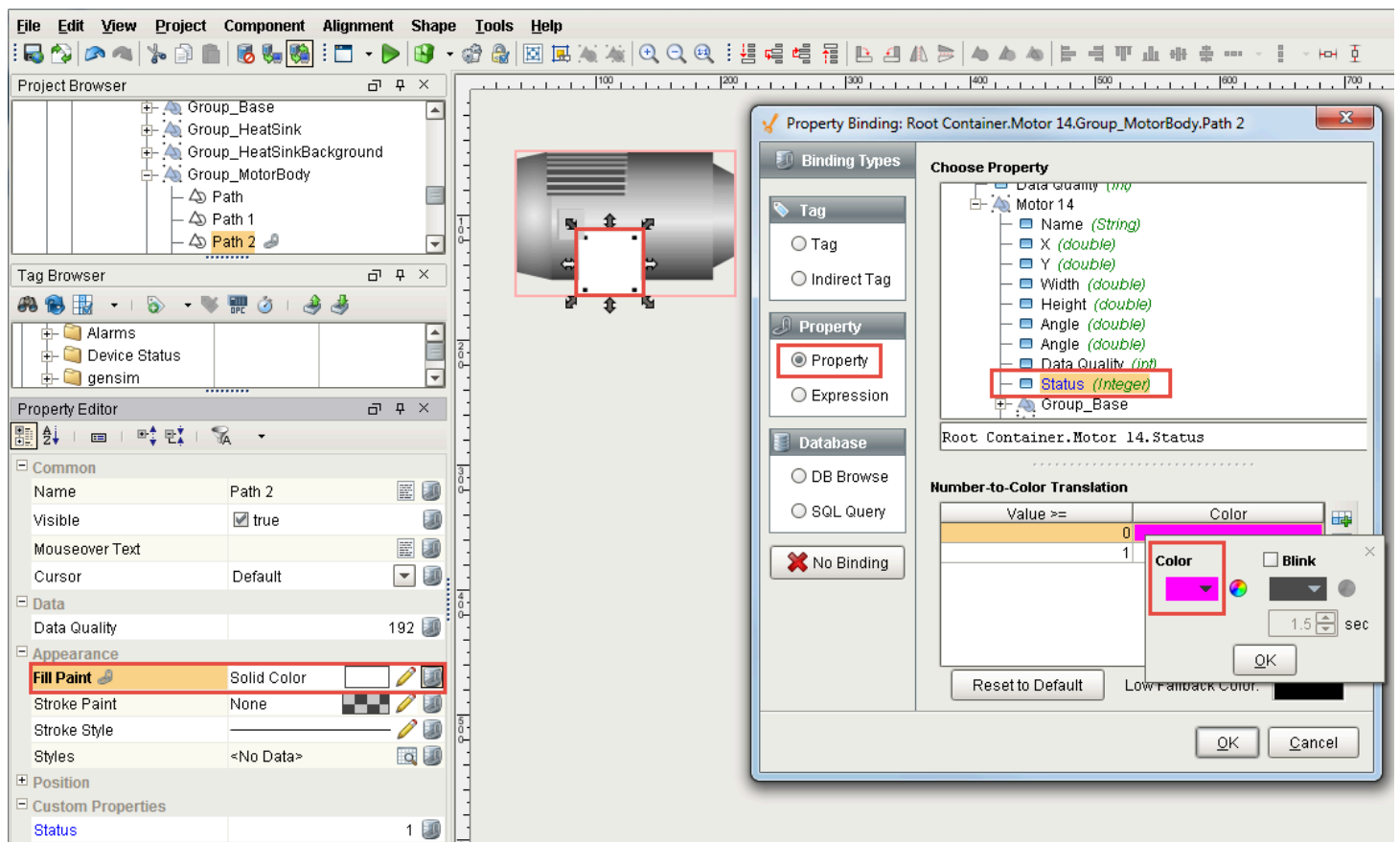
Property Binding

You can use the same number to color translation table with a property binding. Custom properties are extremely useful because they can turn your plain component into one customized for your particular use. Adding Custom properties like Status or Level can transform your SVGs into more than just an image. Refer to sections on [Property Binding](#) and [Component Customizers](#) to learn more.

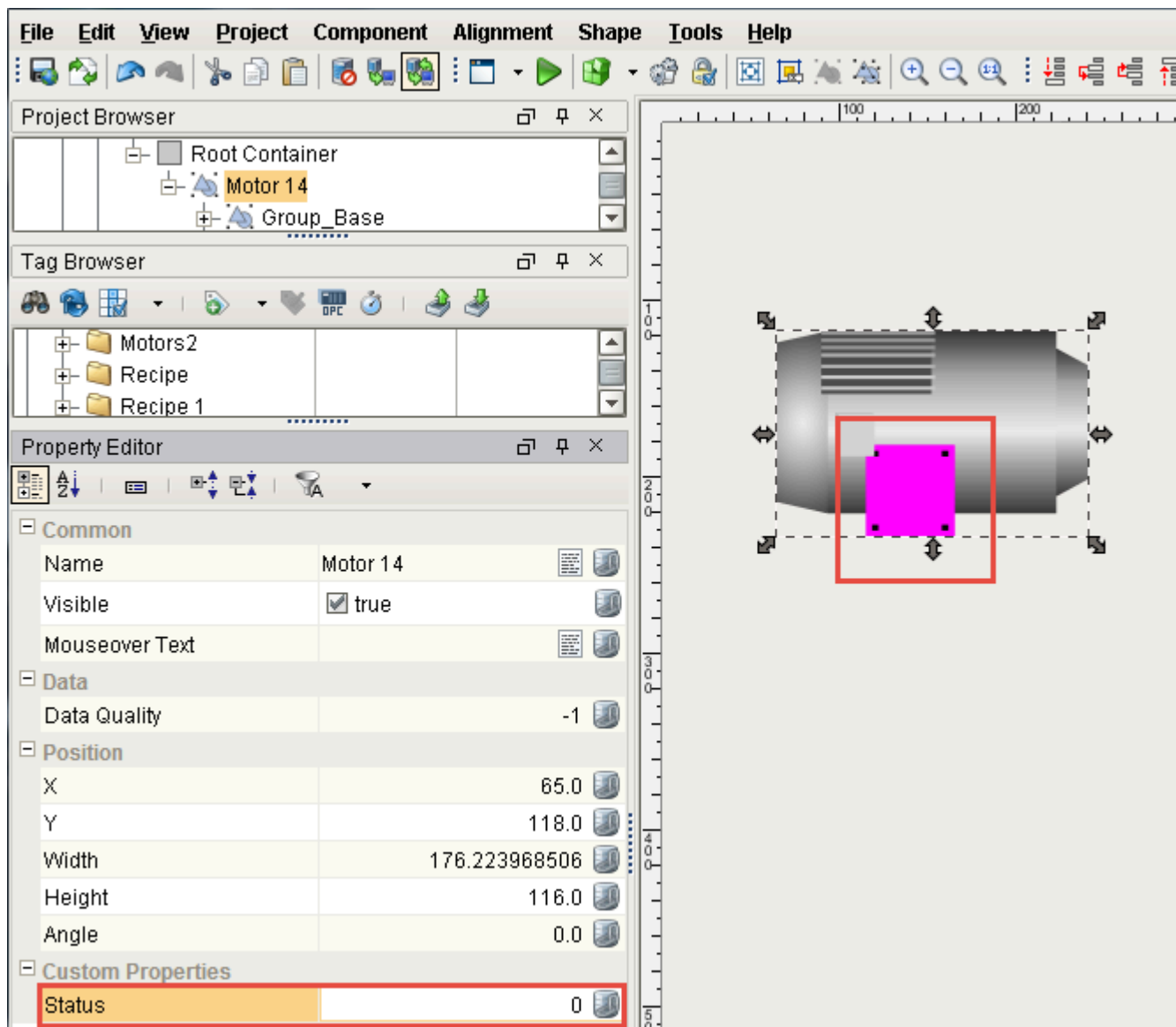
Drag in a Motor SVG and create a custom property on the Motor called '**Status**'. In this example, right click on the **Motor** and select **Customizers > Custom Properties**. Enter the name '**Status**' and use the default '**Integer**' type. Press **OK**.



Click the **Fill Property** icon, and select the **Property** binding type. Under the **Motor** object, select your new '**Status (Integer)**' property. Choose your colors and values. Press **OK**.

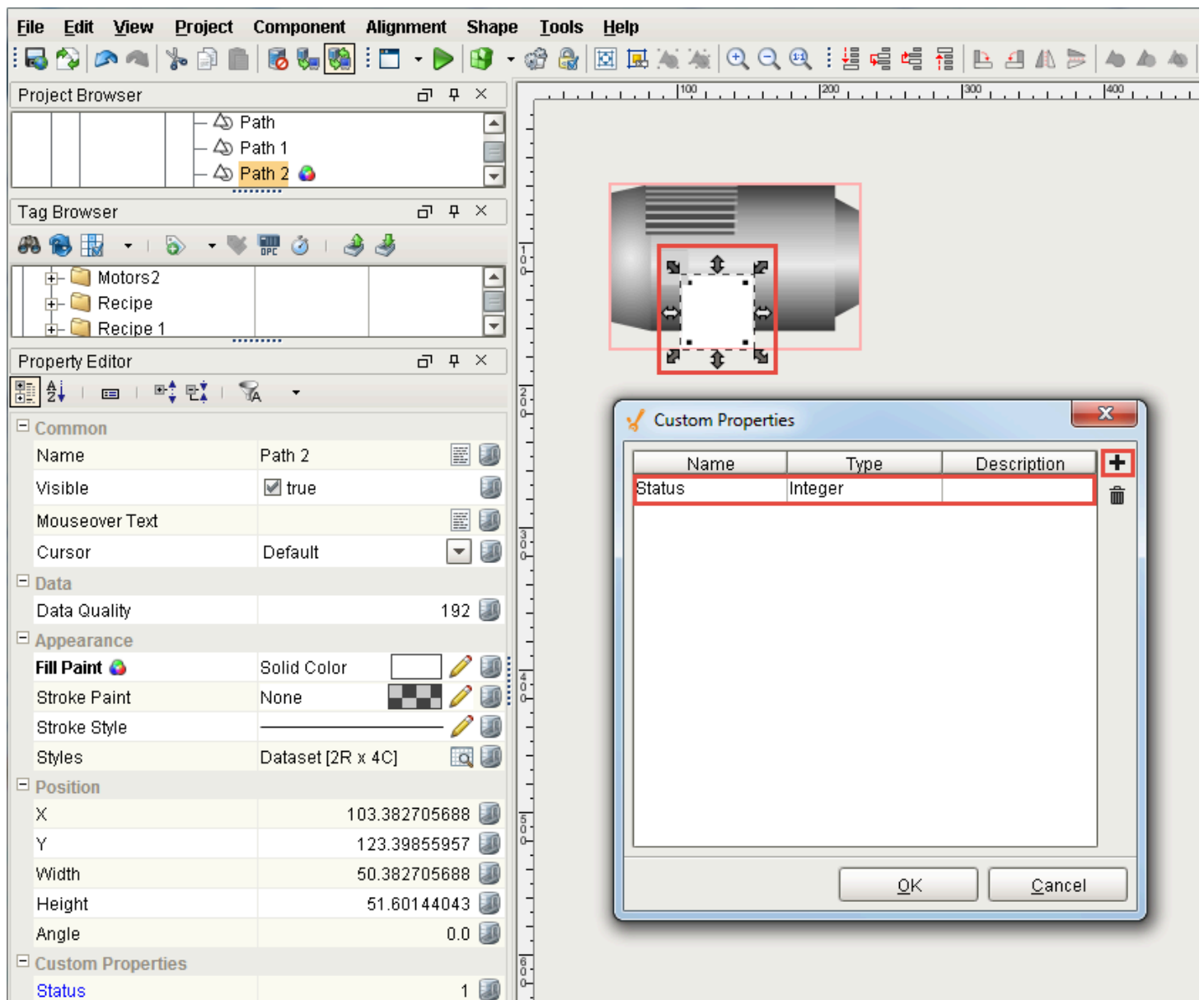


Once you press OK, you'll notice that the square component of the SVG changed color based on the value of the tag. An Integer value defaults to '0' and in this case the color is **purple**. To change the tag value, go to the **Property Editor**, double click on the **Status** property value, and change the value from '0' to '1'. Press **Enter**. The component of the SVG will change from purple to whatever color you set '1' too.

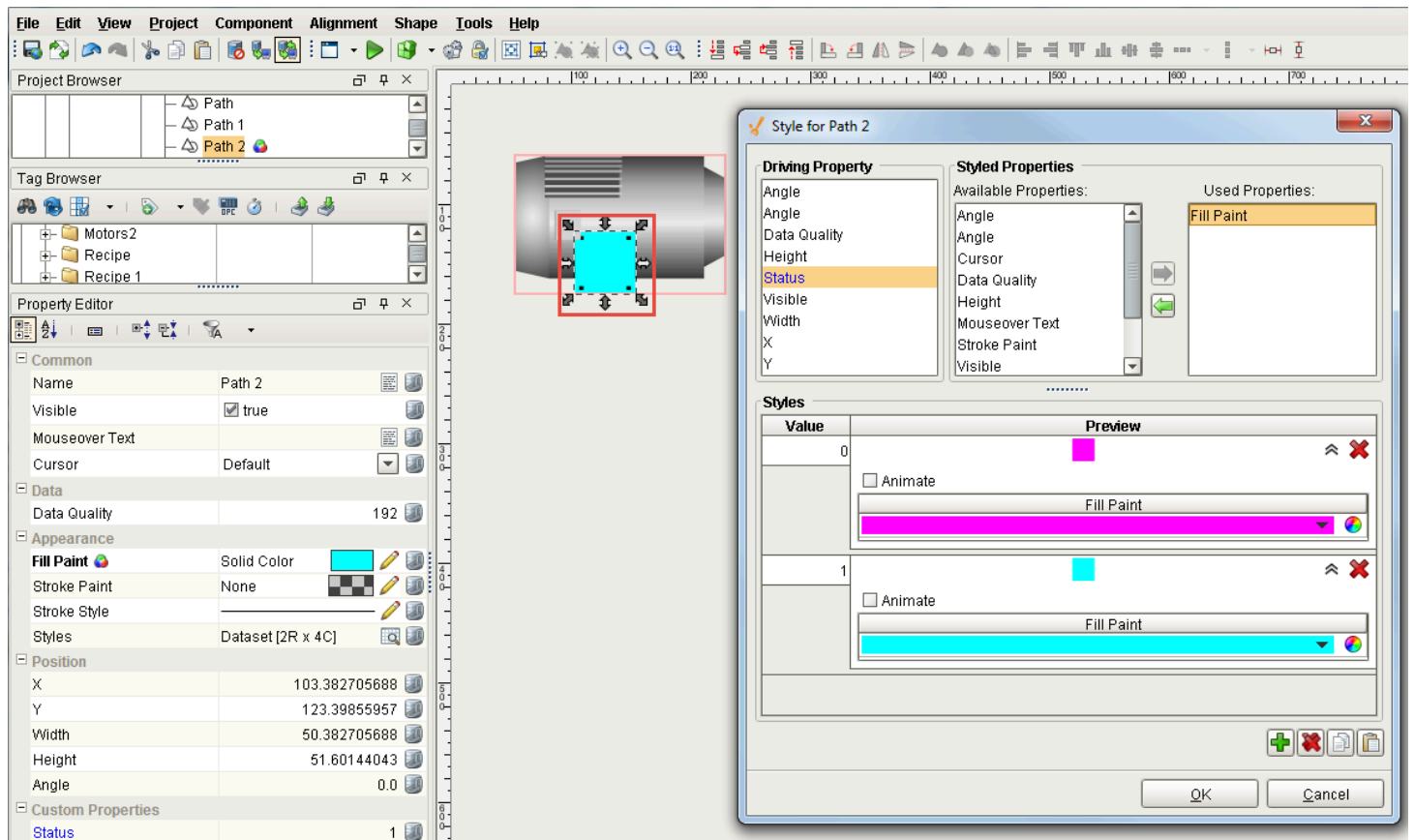


Style Customizer

You can also use the Style Customizer to achieve the same result as with the Property Binding. Like the previous example, we will need a Status property to drive the style. Let's create that same Status property on the inner component of the SVG. Select the component of the SVG, right click and scroll down to **Customizers > Custom Properties**. Enter the name '**Status**' and use the default '**Integer**'. Press **OK**.

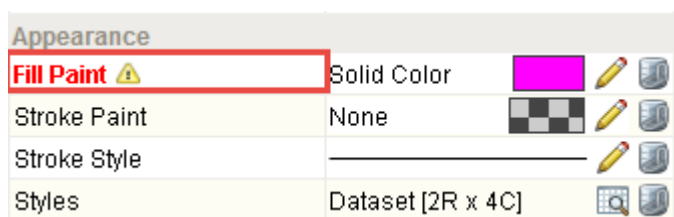


With the inner component still selected, right-click and select **Customizers > Style Customizers**. Select the **Status** property as the driving property, select **Fill Paint** from the list of **Styled Properties**, and then choose your colors for each value, adding values if needed. Press **OK**.



To change the Tag value, go to the **Property Editor**, double click on the **Status** property value, and change the value from '0' to '1'. Press **Enter**. The component of the SVG will change from purple to whatever color you set it too.

In the event you see the Fill Paint property highlighted in red, this is a warning message that you have two competing property bindings that should be resolved. In this case, this warning message popped up because the **Fill Paint** property has both a **Binding**, and a **Style** applied to it.



Direct Bindings

There are several types of Bindings that can be used to set colors directly, without using a mapping table.

Expression Binding

You can use the expression language to calculate a value. If you have a color that depends on multiple properties, then using an express is recommended to evaluate correctly. Here are a few

examples. The first example returns a static color using the Fill Color property.

Expression

```
// binding on the Fill Color property  
// static red color  
color(255,0,0)
```

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  
// fades from white to blue when Tag value goes from 0 to 100 %  
color(255,255,255-({tag value}/100*255))
```

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)
```

SQL Binding

SQL Query bindings only have to return a string that can be translated to a color. Any set of RGB values can be listed using an integer representation ie: "255,0,255" or HEX "FF00FF".

SQL Binding

```
-- A basic static query that returns the hex code for teal  
SELECT "00FFFF"
```

SQL Binding

```
-- A static query that concatenates integers to return a string RGB Code for  
yellow: "255,255,0"  
SELECT CONCAT(255, ',', 255, ',', 0)
```

In most cases you will be using a table in the database. Assuming the following table (where background and foreground are strings):

state	background	foreground
0	255,0,0	255,255,255
1	0,255,0	0,0,0
2	255,255,0	0,0,0
3	255,127,0	0,0,0

SQL Query Binding

```
-- This query returns the background string RGB code for state #3 from the  
colors_table  
SELECT background  
FROM colors_table  
WHERE state = 3
```

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

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

Tank Cutaway



INDUCTIVE
UNIVERSITY

Tank Cutaway

[Watch the video](#)

Using Cutouts on Tanks

The symbols in the Tank Cutaways category work well when combined with the other symbols from the Symbol Factory, especially tanks from the Tank Category. Use the following technique to make a dynamic cutaway tank display:

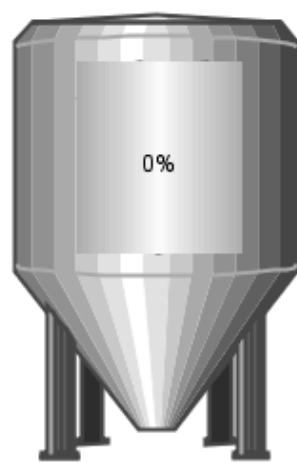
1. From your SVG symbols, Drag a tank and a cutaway symbol onto the window.
2. Align the cutaway symbol on the tank where you'd like the cutaway to be placed.
3. Select the tank symbol, and **then** select the cutaway while holding **CTRL** to select both symbols.
4. Press the **Difference** button () to use the cutaway symbol to (you guessed it!) cut away that area out of the tank.
5. Place a **Level Indicator** component on the area removed by the cutaway.
6. Push the Level Indicator below the tank with the Move to Back icon ()
7. Bind the Level indicator's **Value** property to create a dynamic display.



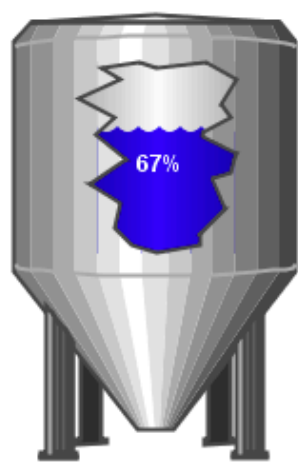
1



2



3



4

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

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

Tint SVG



INDUCTIVE
UNIVERSITY

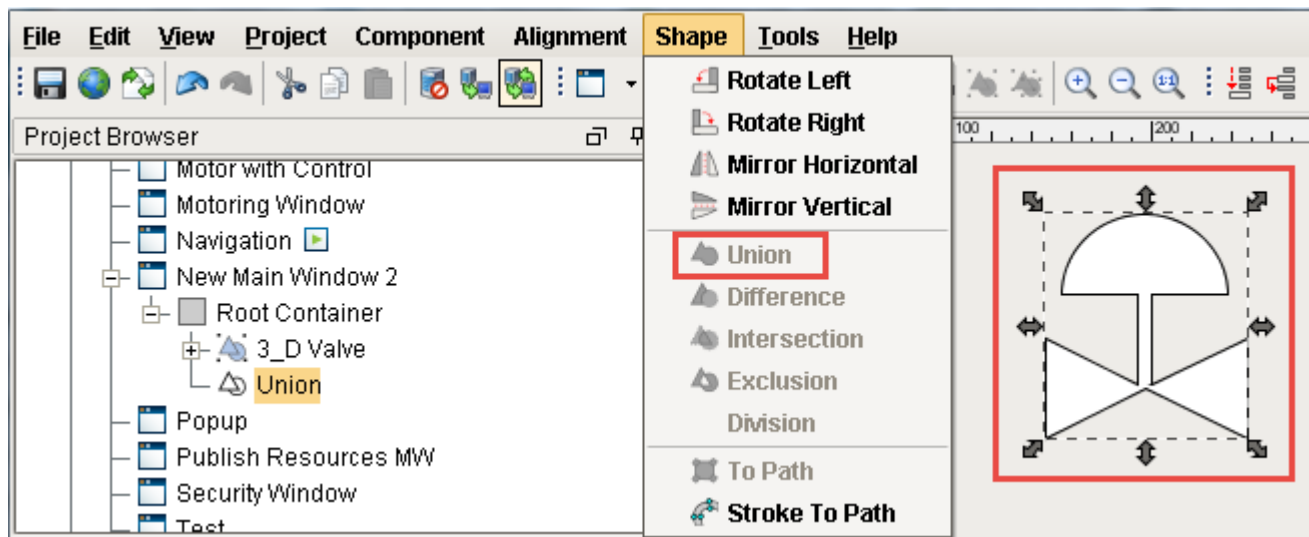
Tint SVG

[Watch the video](#)

Animating the Tint on a Grayscale Symbol

Lets say you have chosen 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. Press the Union button () in the toolbar or find the **Union** item under the **Shape** menu. This will flatten the duplicated shape into a single shape.

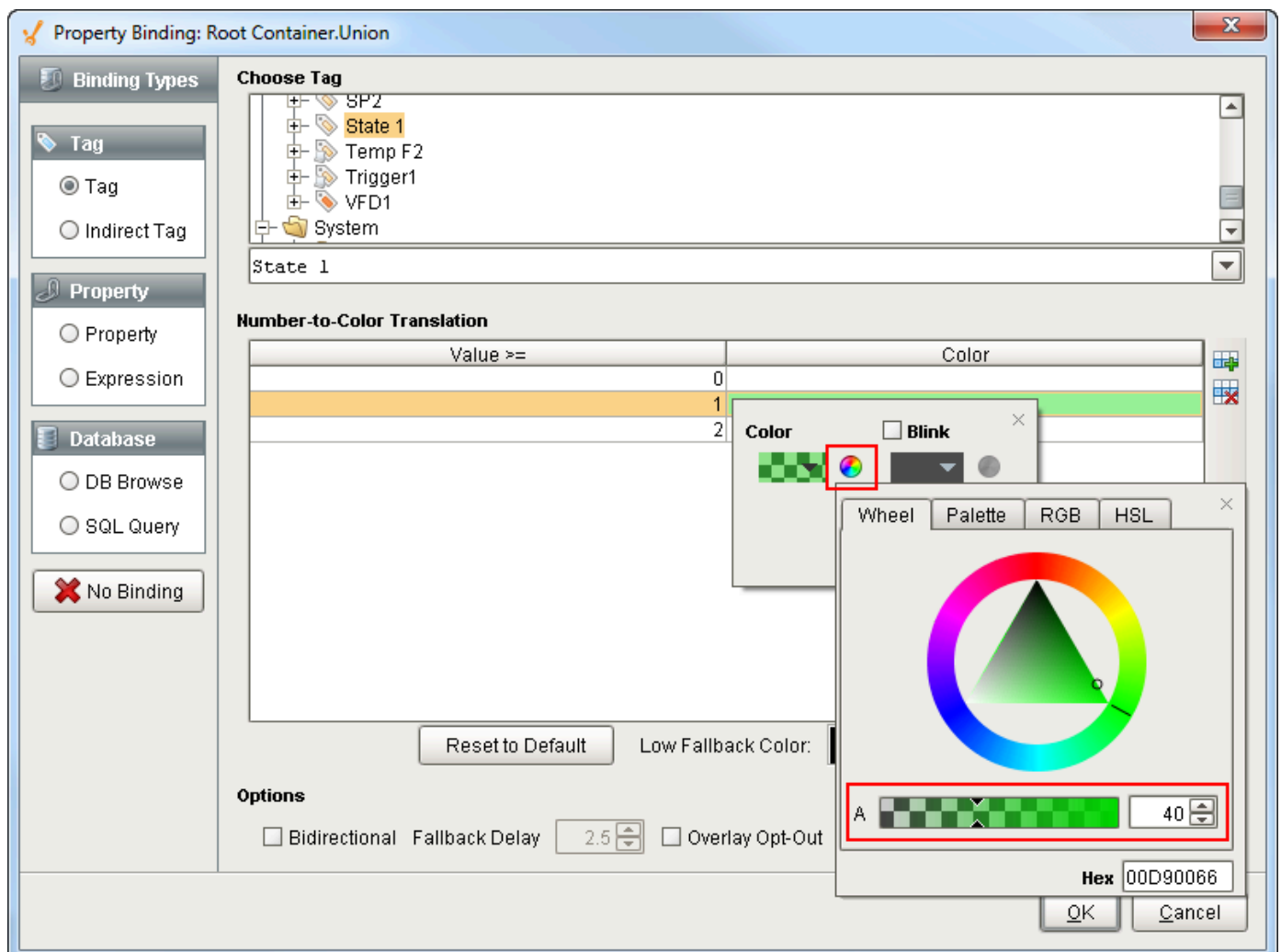


5. Remove the outline by setting the **Stroke Paint** property to **No Paint**.

6. Bind the **Fill Paint** property to your **ValveStatus** Tag.

7. Add three entries into the number-to-color translation table:

- 0% opaque white for 0
- 40% opaque green for 1
- 40% opaque red for 2



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.



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

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

Images in Vision



INDUCTIVE
UNIVERSITY

Images (png, jpg, gif)

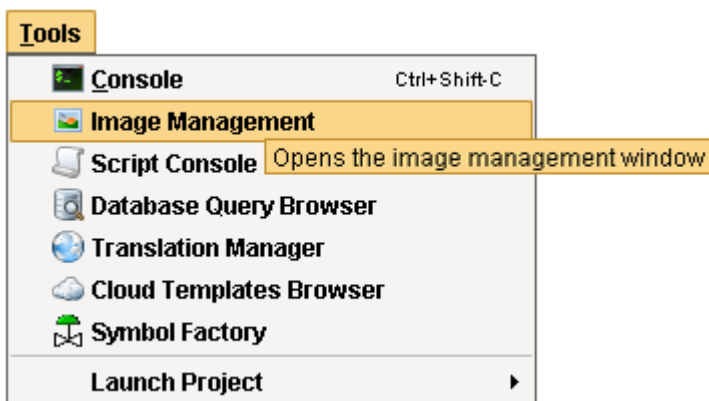
[Watch the video](#)

Image Manager

Images such as PNGs, JPGs, or GIFs can be uploaded to the Image Management tool and used inside of windows in Ignition. Once uploaded, these images may be used on windows and in templates. The Image Manager tool, available from the **Tools > Image Management**, provides an interface to upload, download, or select images.

BITMAP FILE TYPES ARE NOT SUPPORTED

The Image Manager process does not support bitmap files.



Exporting and Importing Projects

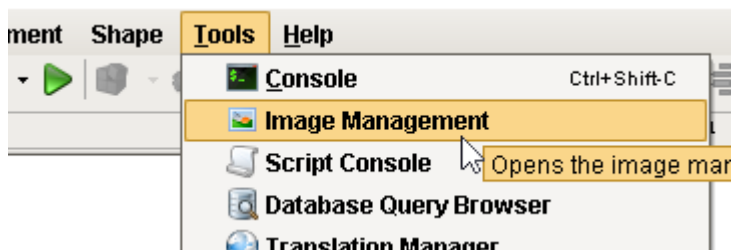
Uploading an image involves storing the file in the Gateway's internal database. This means that project exports do not contain any referenced images.

When exporting a project for use in another Gateway, it is recommended to also export any images that the project uses, and upload them into the new Gateway at the same directory. Examples of uploading and downloading images can be found on this page.

Using Images in the Designer

Uploading an Image to the Image Management Tool

There are two ways to upload an image into the Image Management tool. Both ways involve having the Image Management tool open. At the top of the Designer in the Menu Bar, select **Tools > Image Management**.

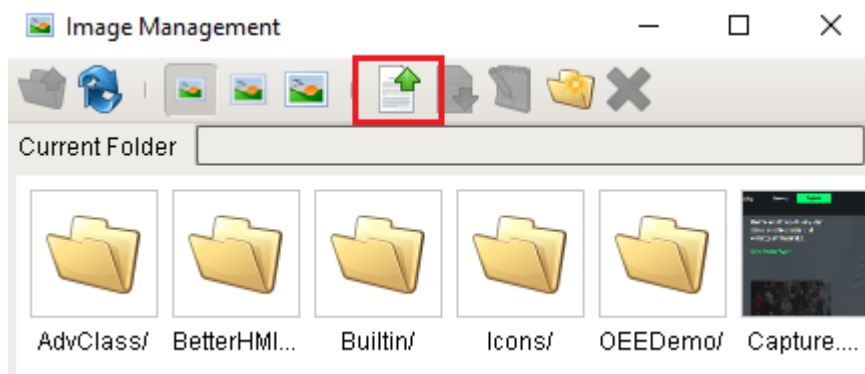


Upload on Drag and Drop

Images can simply be dragged and dropped from the local file system into the **Image Management** window.

Manual Upload

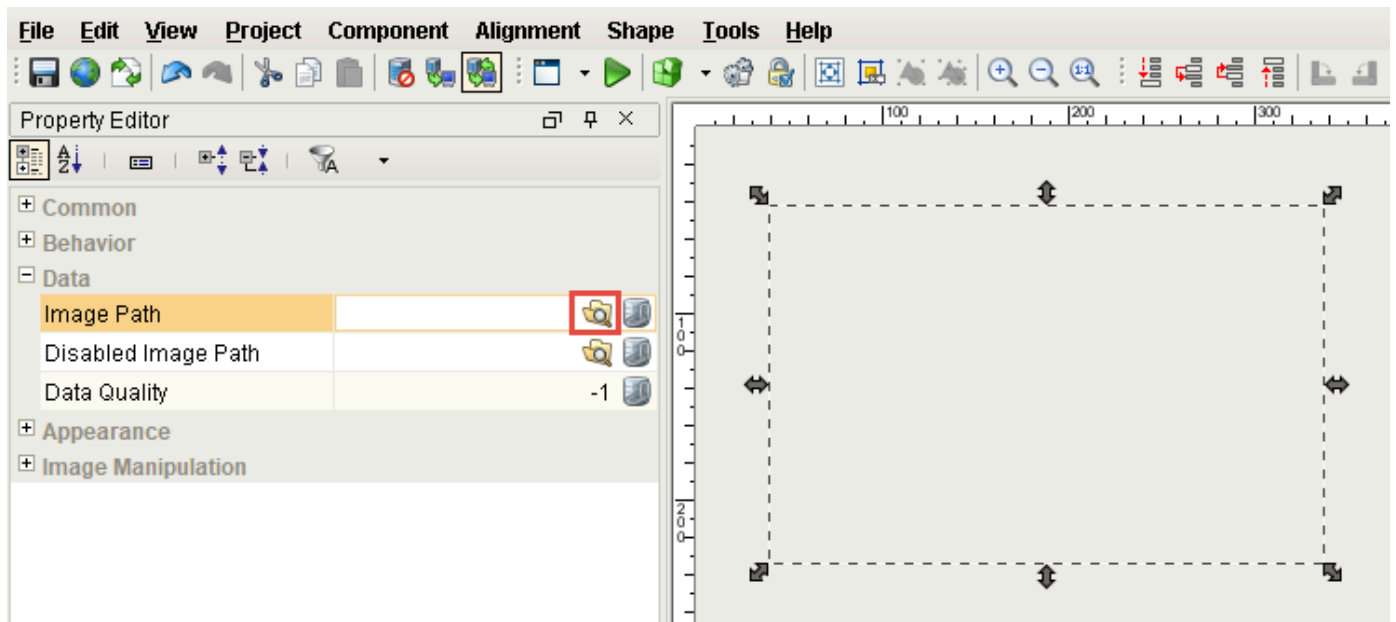
Alternatively, the Image Management window has an Upload button to pass images in. Locate the directory you wish to upload the image. You may use the root folder, or create a new folder to keep your images organized. When ready, click on the **Upload Image** button.



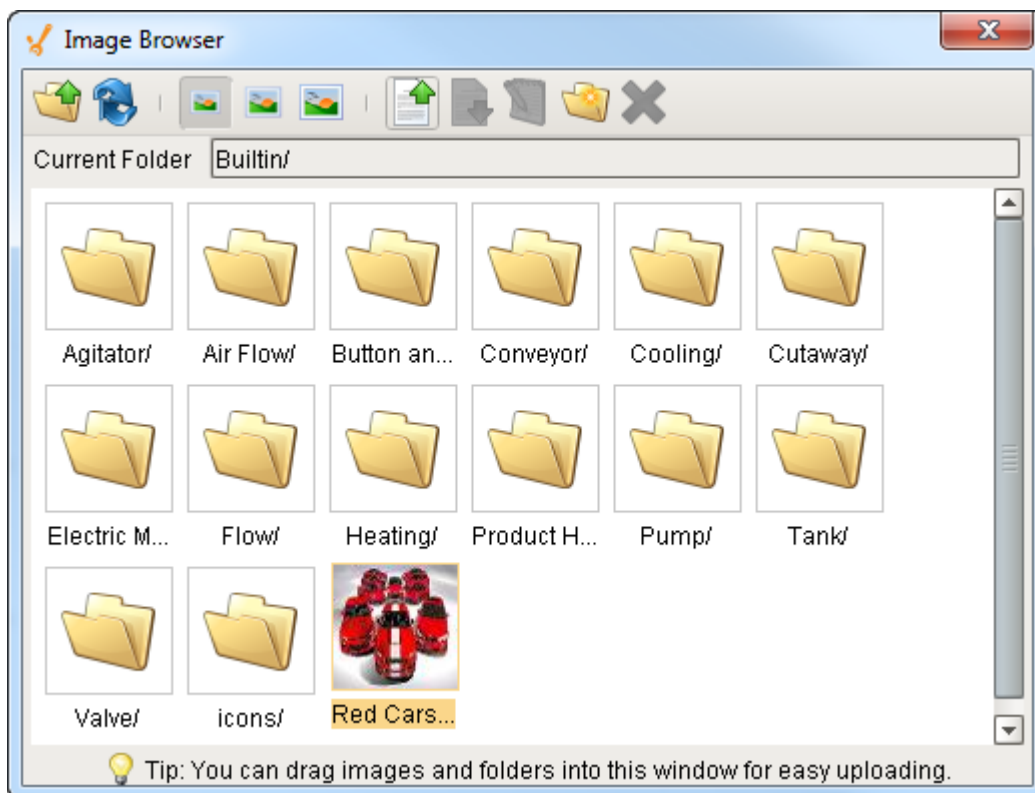
An **Open** dialog window will appear. Simply find your image on the local system and click **Open** to upload the selected image.

Using the Image Component on a Window

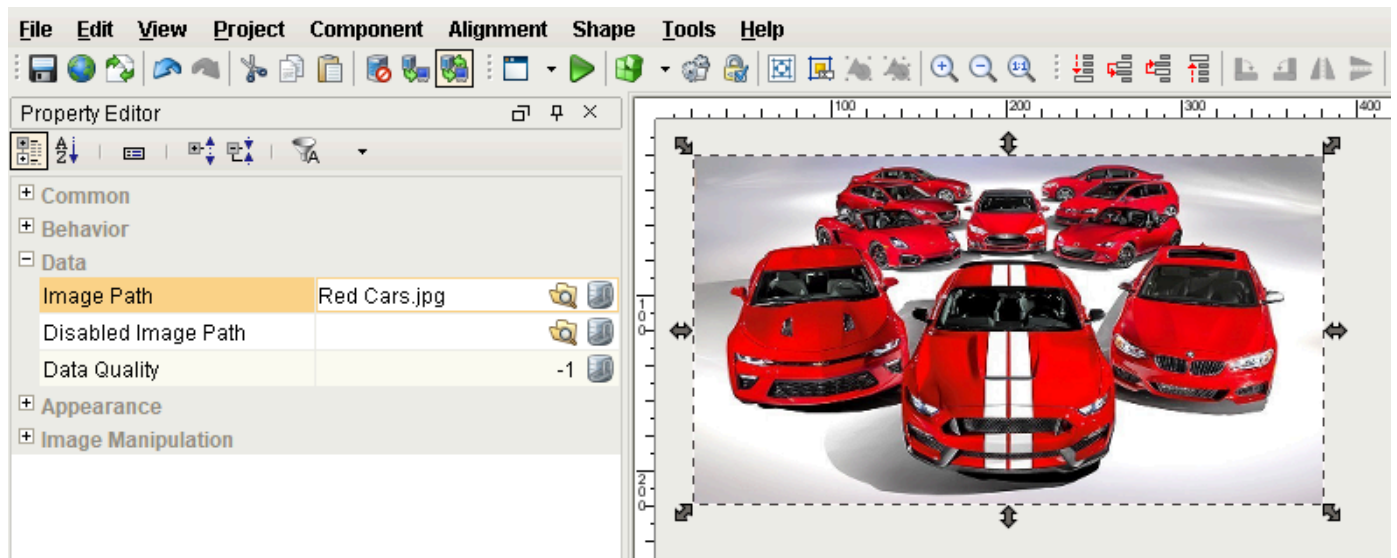
1. Drag an **Image** component from the component palette onto the window.
2. From the Property Editor, select the image component's **Image Path** property, click on the **Browse** icon.



3. From the **Image Browser**, look through the different folders for an image file you want and double-click on it to display it in your window.



4. The path to the image is now displayed in the components **Image Path** property.



Using Local Images

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:

```
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.

Downloading Images from the Image Management Tool

Single images, as well as entire directories, may be downloaded from the Image Management tool. This is useful when migrating a project to another Gateway.

Image downloads can be taken from either the **Image Management** or **Image Browser** windows. If at least one folder or image is selected, the **Save** button () will become enabled. Click the **Save** button, pick a local directory to save the images to, and click the **Save** button again. Note that all images and subfolders in the selected folder will be copied to the selected directory.

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

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

Component Properties



INDUCTIVE
UNIVERSITY

Component Properties

[Watch the video](#)

Every Component Has Properties

Each component has a unique set of properties. A property is simply a named variable with a distinct type that affects something about the component's behavior or appearance. Hover your mouse over the property in the **Property Editor** panel to see its **data type** and scripting name.

Every component in Ignition comes with its own unique set of properties by default, which convey important information. Each component you select inside a window has various properties that you can set and modify in the Property Editor. These properties dictate how a component looks and behaves inside of a window. You can also create your own custom properties for your particular use. It is possible to extend this list of properties by adding our own custom properties to the component. Basically it's like adding variables to the component that we work with in the screen.

The Property Editor

The **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.

Filters

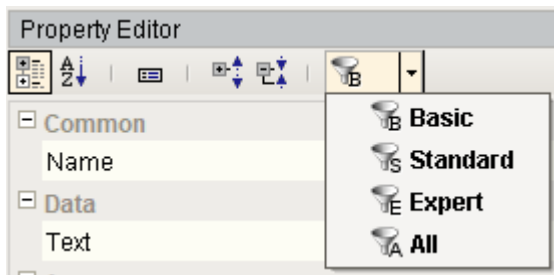
It is common for components to have many properties, so by default, the Property Editor only shows the **basic** properties. These are the properties that you'll most commonly want to set or bind for a given component. There are also the standard properties. This is a larger set of properties that

includes the basic properties and many other useful properties. Some properties are expert properties. These are properties that are either uncommon to set or whose purpose might require an in-depth understanding of the inner-workings of the component. You can change the filter using the **filter** button () in the **Property Editor's** toolbar.

- **Basic:** The Name property and any very commonly used properties. Most components only show 2-4 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

TIP

When you first open the designer on a computer, the property editor is set to only show basic properties. To see all of them, change the property filter to **All**. The designer will remember your selection for future sessions.



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 () 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 is almost assuredly a mistake.

The Binding Button

To the right of most properties is the binding button (). Use this button 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.

Data Types

There are a wide variety of datatypes across all of the Vision Module's components. Each property has a distinct type. Here are the most common types that you'll find on the **Property Editor**.

Numeric Types	Description
Boolean	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 -2 ¹⁵ and 2 ¹⁵ -1. That's -32,768 to 32,767, inclusive.
Integer/int	A 32-bit signed integer. Can hold values between -2 ³¹ and 2 ³¹ -1. That's -2,147,483,648 to 2,147,483,647 inclusive.
Long	A 64-bit signed integer. Can hold values between -2 ⁶³ and 2 ⁶³ -1. That's -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	Description
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 Styles , the <code>toBorder()</code> expression function, or Java scripting.

WHAT IS THE DIFFERENCE BETWEEN INTEGER VS. INT?

The difference is that an Integer property will accept the special null (or None in Python-speak) value, while an int property will not. This distinction holds true for all of the numeric types: the type name that starts with a capital letter accepts null, while the all-lowercase version does not.

EXPERT TIP!

Most of these datatypes are actually defined by Java. For example, the **Date** datatype is really an instance of a **java.util.Date**. This means that you can use the **java.util.Calendar** class to manipulate them, and the **java.text.SimpleDateFormat** class to format and parse them. Learn more about these classes in the Java 2 Platform online documentation at

<http://java.sun.com/j2se/1.5.0/docs/api/index.html>

See also: [Variables](#), [Datatypes](#), and [Objects](#)

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 groups based on what is available. IE: Common, Data, Appearance, etc. Each component has a different list of properties to effect how it behaves, but every component has the **Common** group of properties at the top.

These Common properties will behave the same for all components. Here's a list for each property and when it might be used.

- **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. IE: 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. IE:

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

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

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

Component Customizers

Using the Component Customizers

Ignition provides a number of **Customizers** to configure components in ways that are too complex or cumbersome for basic properties. In this page, the **Customizers > Custom Properties** and the **Customizers > Styles Customizer** are described. 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, such as the [Easy Chart Customizer](#).

To use a customizer, right-click on the component and choose the **Customizer** menu, or use the Customizer split-button () icon in the Vision main toolbar.

EXPERT TIP

Often, a Customizer is really just a user-friendly user interface to one or more expert properties. For example, all the **Easy Chart** Customizer really does, is to modify the contents of the pens, tagPens, calcPens, axes, and subplots Dataset properties. Knowing this is very powerful, because 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



INDUCTIVE
UNIVERSITY

Custom Properties

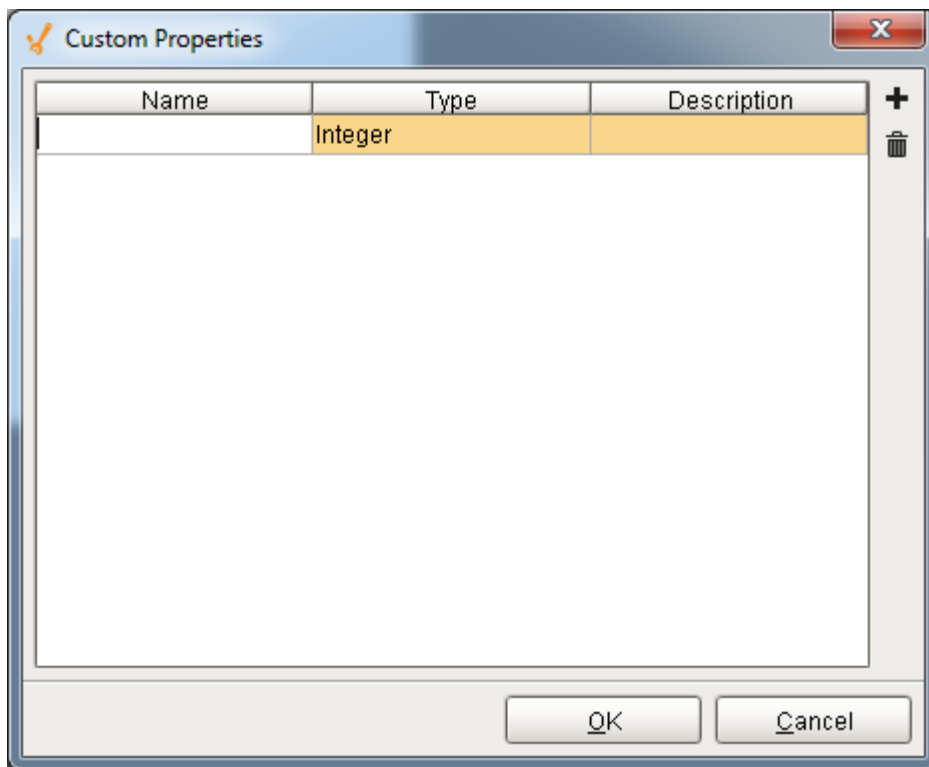
[Watch the video](#)

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 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 to open a popup window, it will pass a certain set of values into the window, and then into the custom property (that you created on the Root Container) for use on that window. For more information, see [Parameterized Popup Windows](#). You can see how custom properties can turn any "plain" component into one customized for your particular use.

To Configure the Custom Property

1. Right-click on a component and go to **Customizer > Custom Properties**. The Custom Properties window is displayed.
2. Click the **black +** icon in the upper right to add a row to the table.



3. Add a **Name**, **Type**, and **Description** for each custom property you want to add.
4. Click **OK**. Notice the custom properties you added are displayed in the **Property Editor** in blue.

Property Editor

Common

Name	Text Field
Enabled	☒ true
Visible	☒ true
Border	☐ Field Border
Mouseover Text	
Cursor	Text

Behavior

Editable?	☒ true
Defer Updates	☒ true
Protected Mode?	☐ false
Commit On Focus Loss	☒ true
Reject Updates During Edit	☒ true
Maximum Characters	-1
Touchscreen Mode	Single-Click

Data

Text	
Data Quality	-1

Appearance

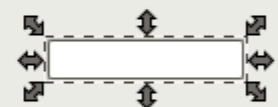
Font	Dialog, Plain, 12
Foreground Color	0,0,0
Background	255,255,255
Non-Editable Background	241,241,241
Antialias	☐ false
Styles	<No Data>

Layout

Horizontal Alignment	Leading
----------------------	---------

Custom Properties

TestProperty1	0
TestProperty2	



Style Customizer



INDUCTIVE
UNIVERSITY

Component Styles

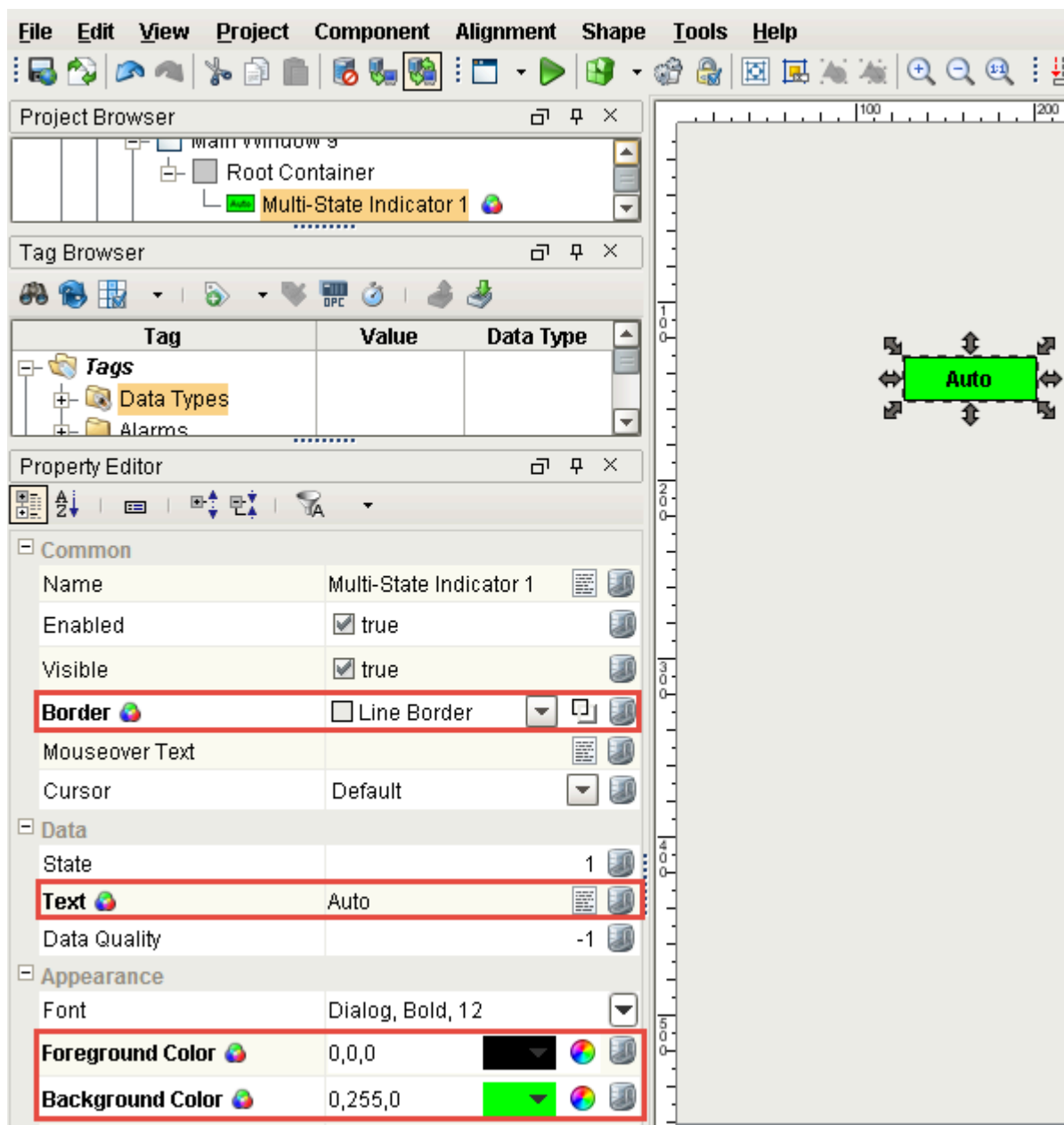
[Watch the video](#)

Many 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 (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. 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.

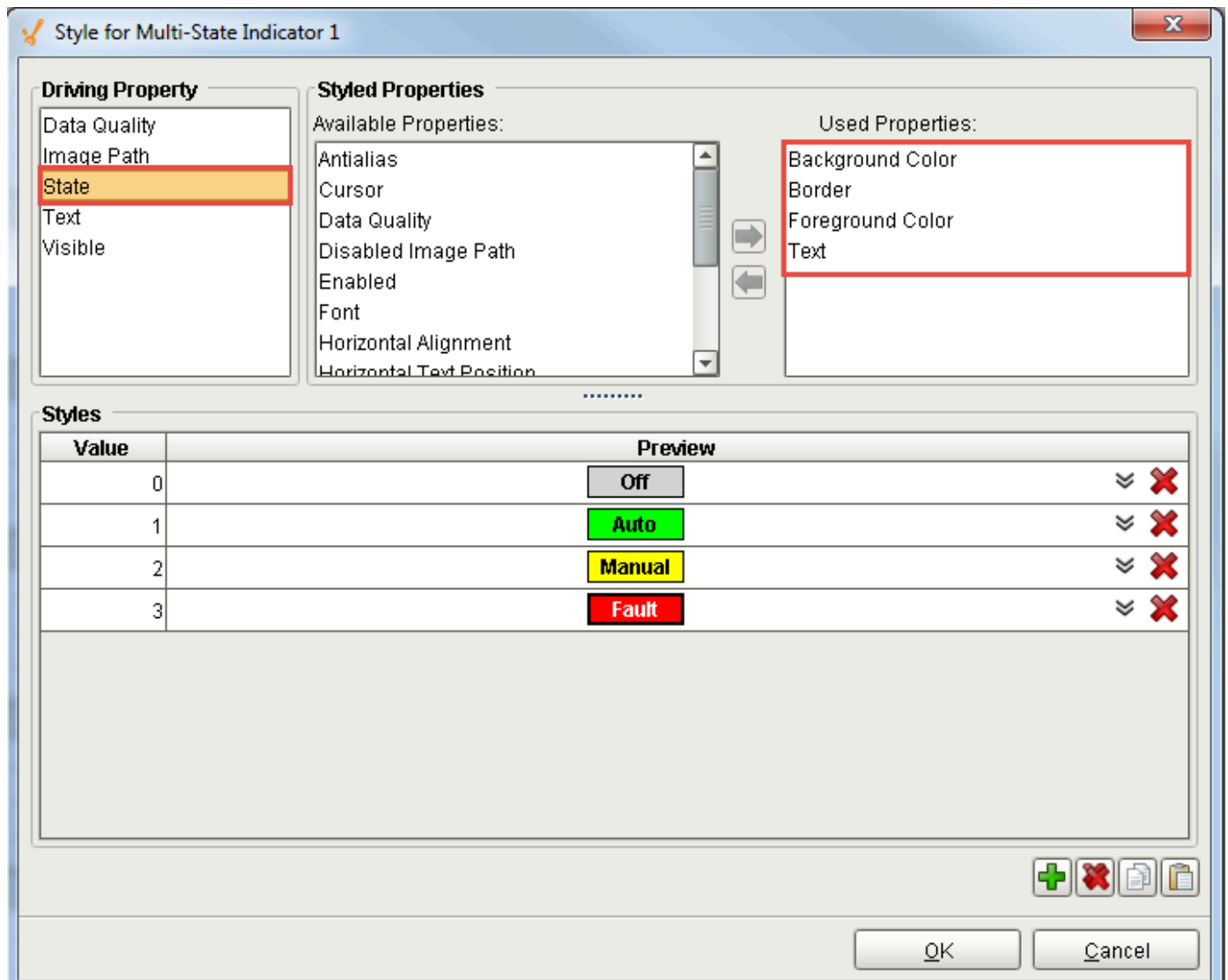
Configuring the Style Customizer

Some components have styles already setup and others do not. The following example involves a component that already has a styles defined.

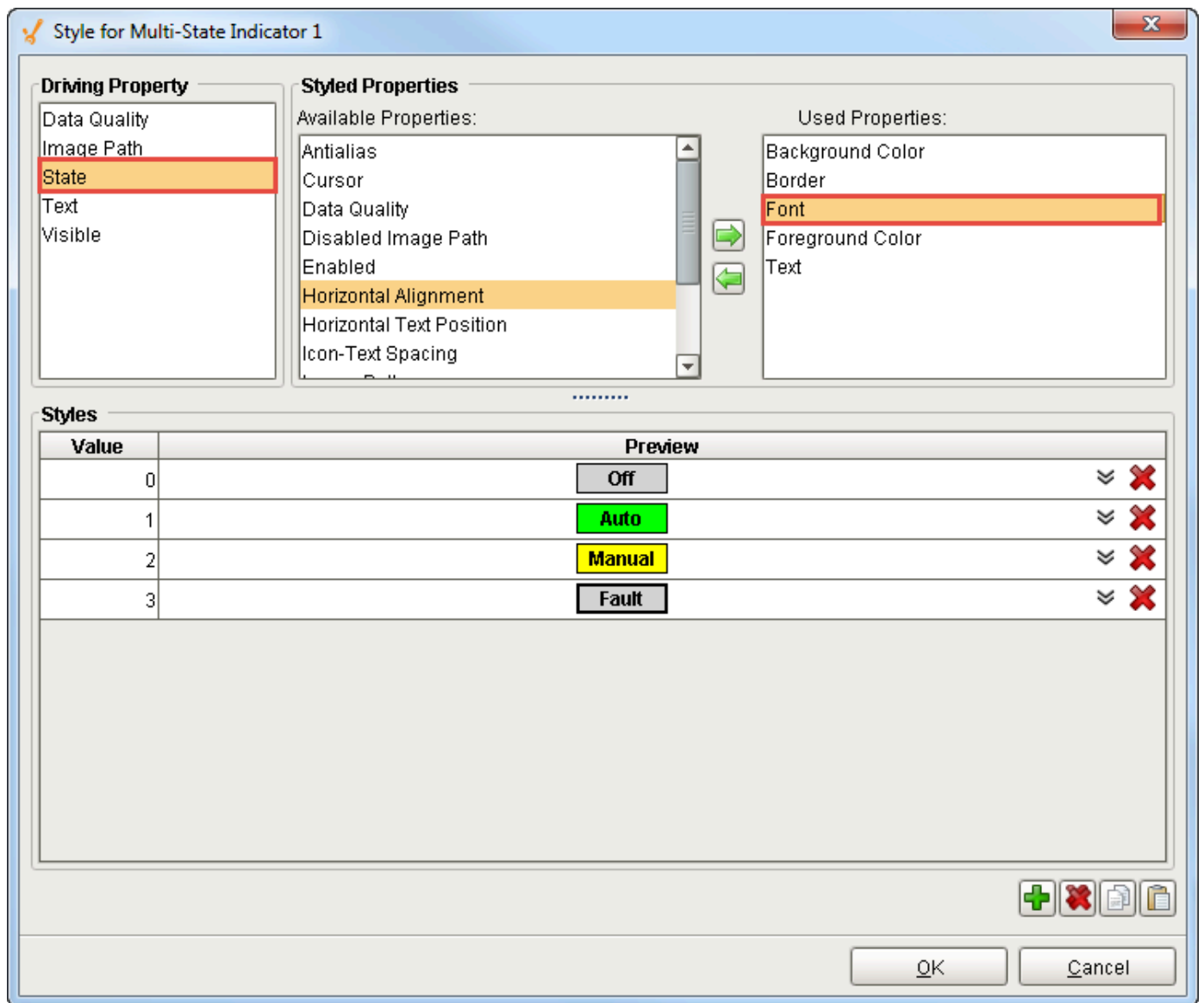
1. Drag in a **Multi-State Indicator** from your **component palette** on to your window. Right click on the component and notice in the Property Editor that several properties appear in bold with a paint canvas icon next to them letting you know that they are used in a style.



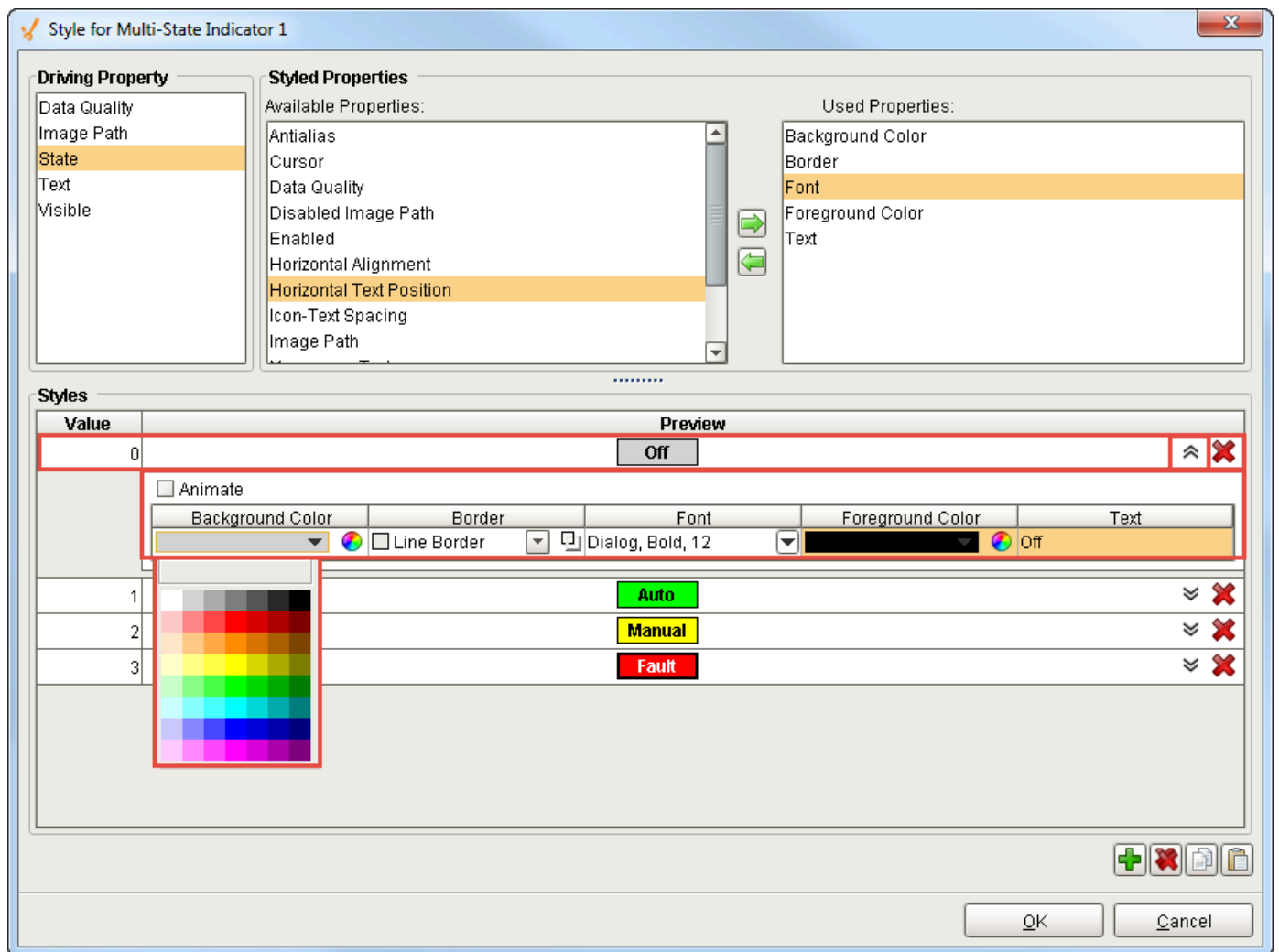
2. Right click on the **Multi-State Indicator component** and scroll down to **Customizers > Style Customizer**. Since styles are already setup on this component, you'll notice that it already has a driving property selected. The driving property is **State** and it uses multiple visual properties like **Background Color**, **Border**, **Foreground Color**, and **Text** which are already configured.



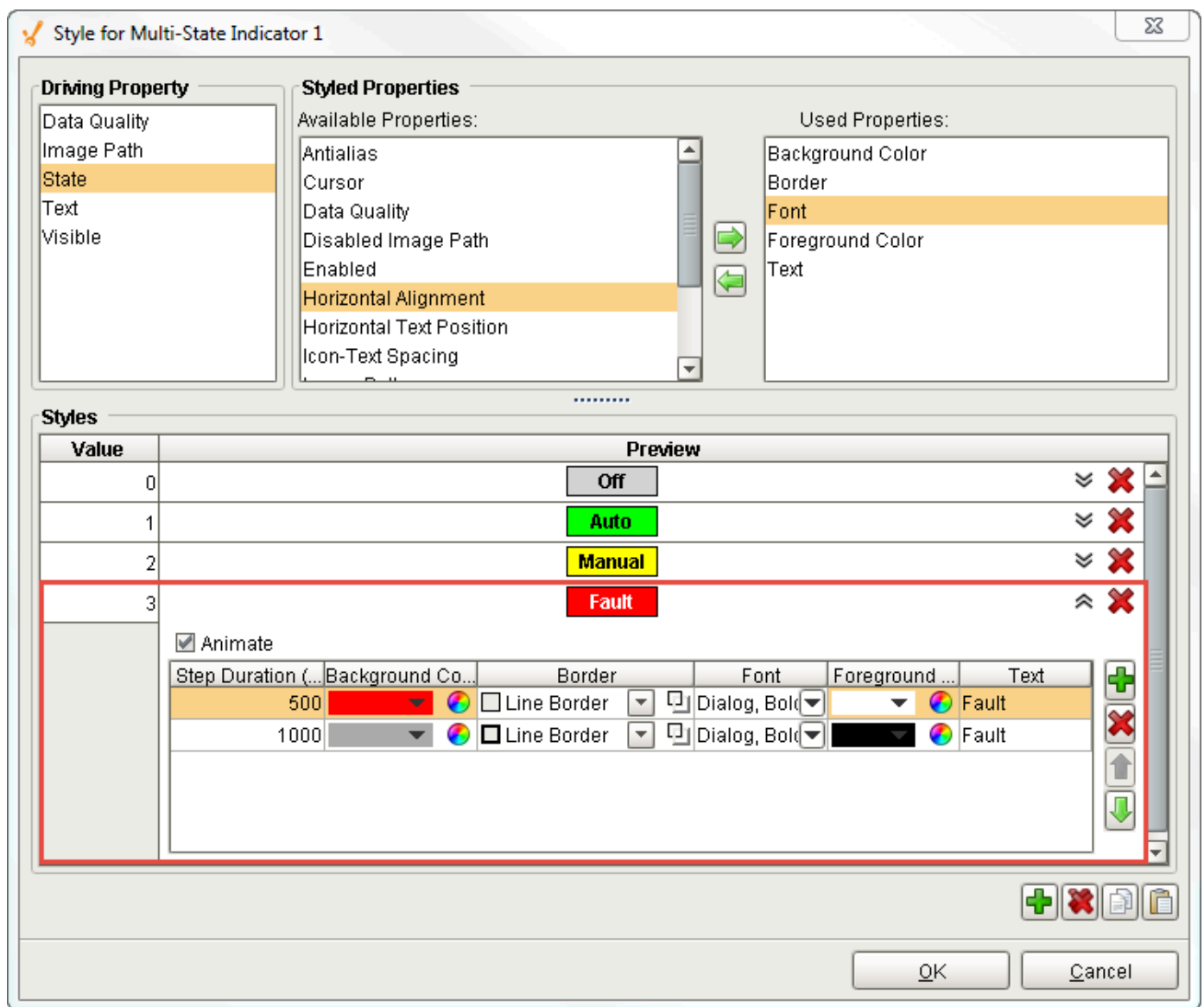
3. Styles are totally customizable. If you don't like the pre-defined style for the preset values for the Multi-State Indicator, you can change it. You can add or remove styled properties. Adding and removing a styled property affects all the **Values** under the component. In this example, the **Font** property was added.



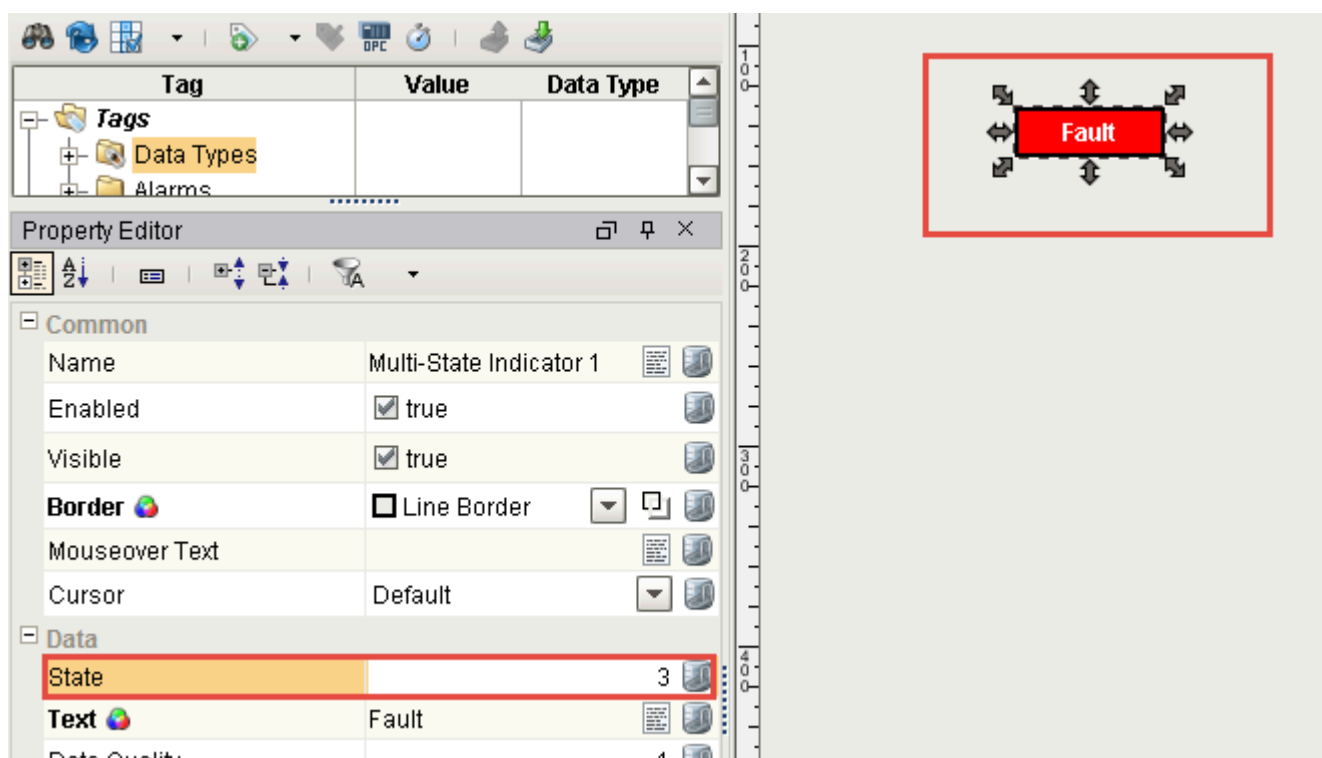
4. For each **Value** property, you can change its styled property to fit your requirements. Click the **Expand icon** (≡) and see all the styles for the **Value '0'**. Starting with Background Color, click on the **dropdown list** to change the color. You can configure all the styles you want to change: Border, Font, Foreground Color and Text. You can change the styles for any Value in this Multi-State Indicator component. You can **Preview** each change after you make it.



5. With the styles feature you can '**Animate**' between different styles on the same **State**. To **Animate**, you can add one or more rows and set the style for what you want those new rows to look like.
 - i. In this example, expand **Value '3'** and you will see that the **Animate** box is checked. Two rows were added and each animation step has a different background and foreground color set along with a **Step Duration** set in milliseconds to control the blinking between the 2 steps. You can add many steps and each step can have a different duration. See how it works in **Preview**.
 - ii. Press **OK**.



6. Change the **State** to '3' to see the component blink '**Faulted**' and switch between the background and foreground colors you setup.



Once your styles are setup, you can see styles coming through on the Multi-State Indicator component as it moves through the different States. In the Property Editor, you will notice that the Border, Foreground Color and Background Color properties are bold letting you know that they are used in styles.

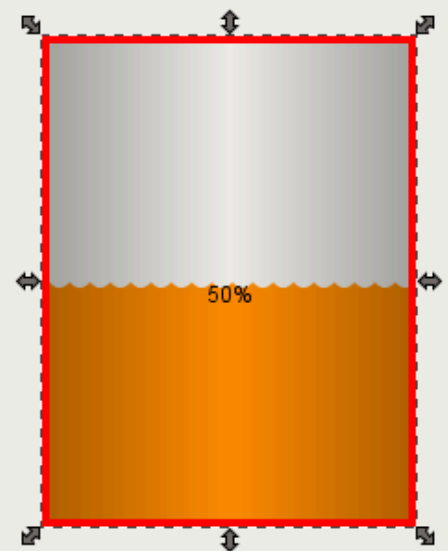
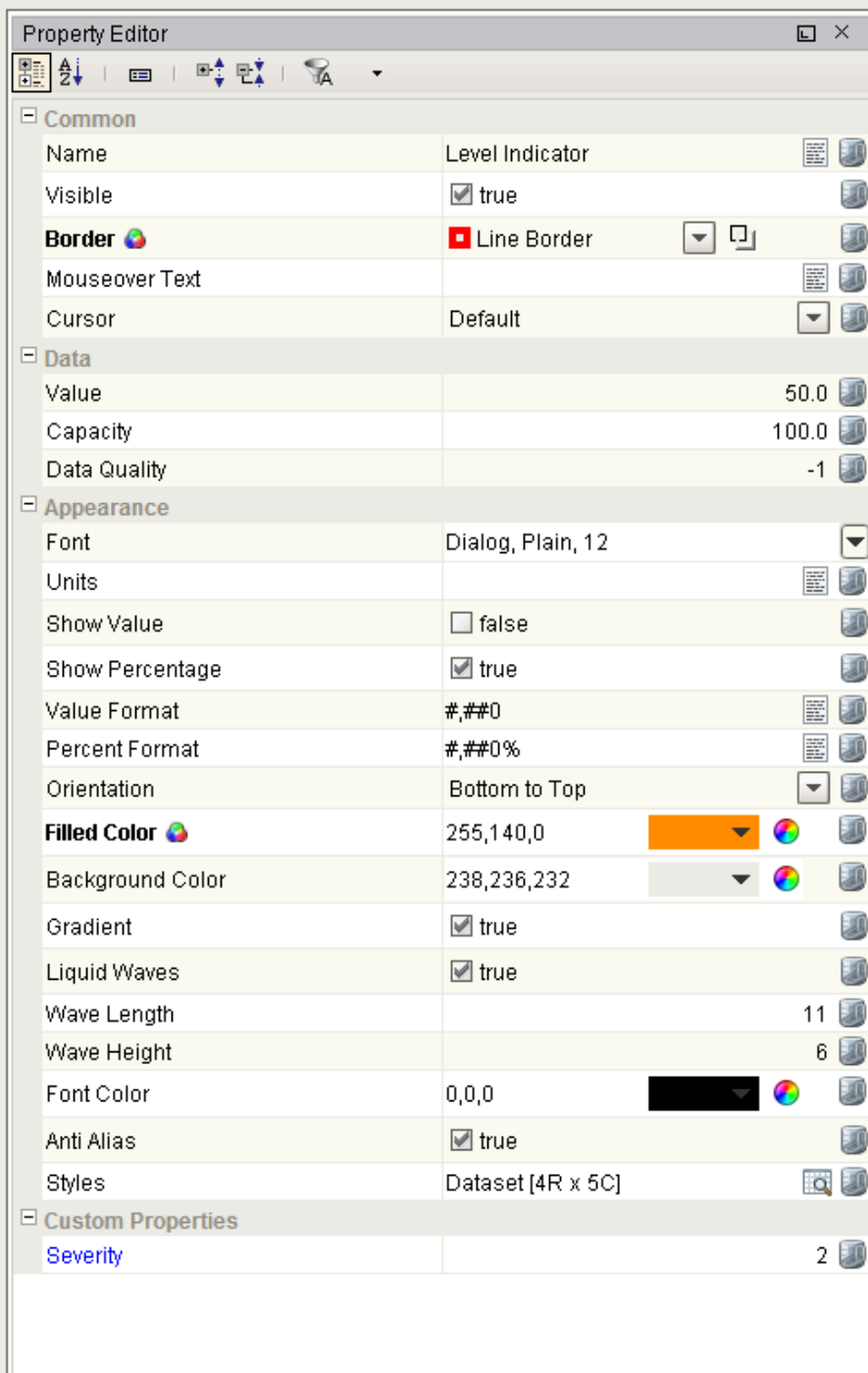
So as you can see, it's easy to change all the properties from one location using the Styles Customizer.

Example 1

Lets say that you have a **Level Indicator** component that is displaying the level in a tank. Lets say that you want to have its appearance change based on the alarm state of the tank's temperature. You can add an integer custom property, call it **Severity**, to the level indicator that you'll bind to the tank temperature tag's **AlertCurrentSeverity** property.

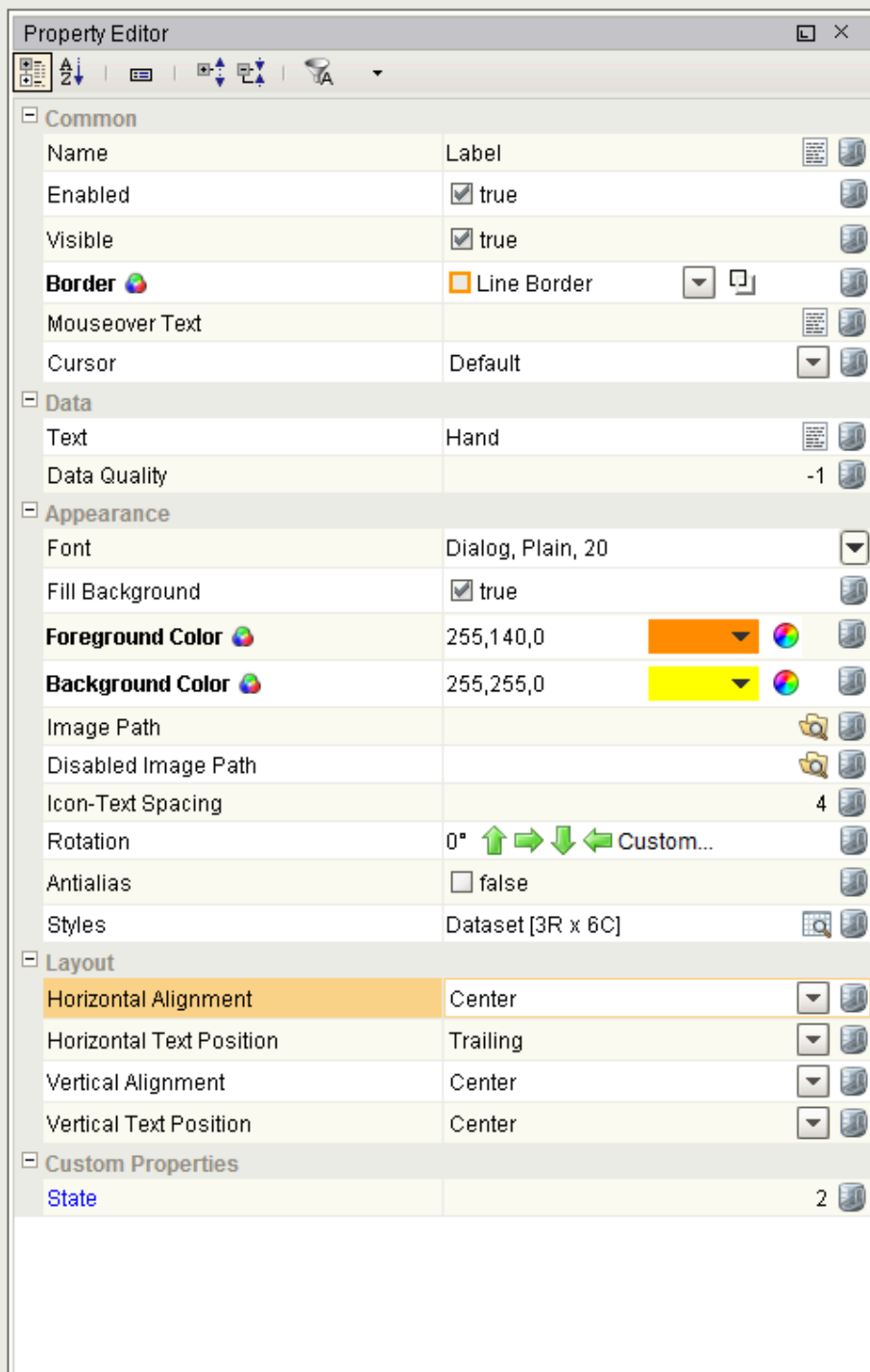
Now go to the **Style customizer**. Choose your **Severity** property as the driving property, and the **Border** and **Filled Color** properties as the styled properties. Add states for -1 (not in alarm), 2 (Medium alarm) and 4 (High alarm). Leave the -1 state alone. Use a red border for state 2 and an orange fill color. For state 4, you can animate it to get a flashing effect. Add two animation frames and set their delay to 500ms each. Configure the frames differently from each other so that you can get a flashing effect.

Click **OK** and notice that the styled properties you chose are now bold and have the styles indicator () 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.



Example 2

Let's look at another example that uses the Custom Properties and the Styles feature together. Take the **Label** component. It seems pretty plain at first - it just displays a String. Of course, you can use its foreground color, background color, and border to make it look interesting. Add an integer custom property on it and name it **state**, and bind that property to a discrete state tag coming out of a PLC. Now use the **State** property to drive its Styles configuration to make the component look different and display different things 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, Tables, Tab Strip, and Multi-State Button/Indicator already 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 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.

Property Editor

2

Common

Name

Multi-State Indicator 1

Enabled

☒ true

Visible

☒ true

Border

☒ Line Border

Mouseover Text

Cursor

Default

Data

State

3

Text

Fault

Data Quality

192

Appearance

Font

Dialog, Bold, 12

Foreground Color

0,0,0

Background Color

213,213,213

Image Path

Disabled Image Path

Icon-Text Spacing

4

Antialias

☐ false

Styles

Dataset [5R x 7C]

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

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

Adding Icons to Labels and Buttons



INDUCTIVE
UNIVERSITY

**Adding Icons to Labels and
Buttons**

[Watch the video](#)

Creating Visible Understandable Components

Most components have an **Image Path** property, which augments the text property of a component by placing an image next to it. The **image** can be an icon that is included with Ignition or an image such as a .jpeg or .png file.

The location of the image in relationship to the component's text is generally handled by the **Horizontal Text Position** property, located in the Layout section on the component.

Adding an Icon to a Label

1. Create a **Label** on your window.
2. Go to the component's Property Editor and select the folder icon () in the Image Path property.
3. From the resulting Image Browser popup, navigate to the icon of your choice and double-click it using your mouse. The component's Horizontal Text Position and Vertical Text Position properties will change the location of the icon in relationship to the text.

Components can be customized so that it is immediately clear what it is trying to convey.

In Alarm ⚠

+ Add New

✖ Delete

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

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

Comparison Charts

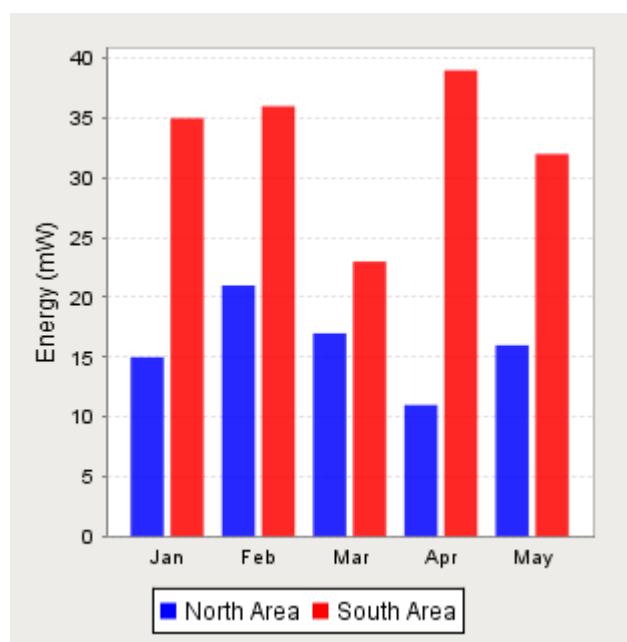
Overview

This page provides an overview of the various comparison charts in Ignition, or charts that allow you to compare difference 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. This page describes several types of comparison charts and how they are used.

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:

Dataset Viewer

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

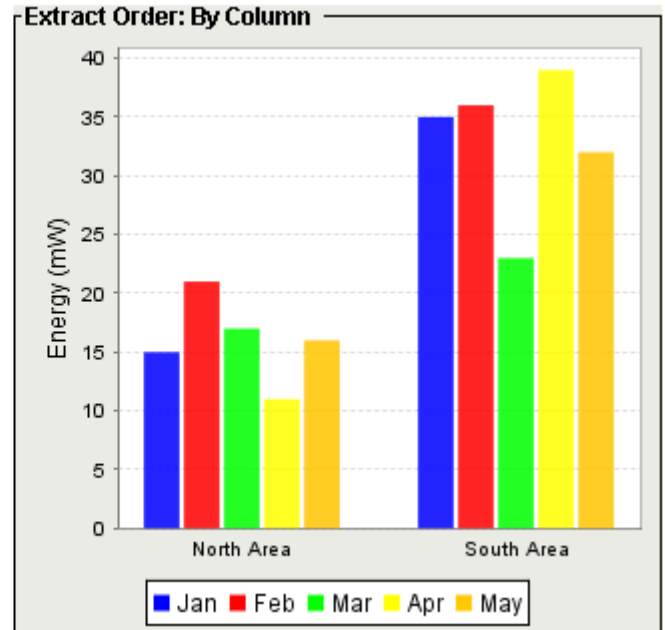
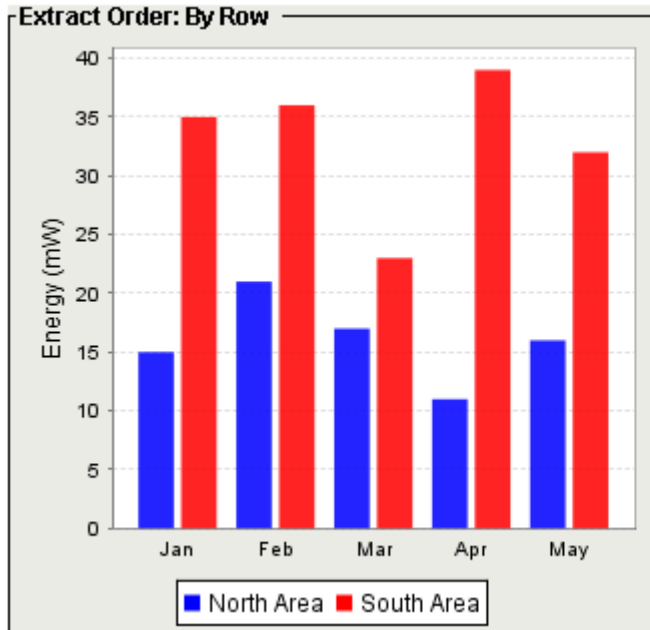
Cancel

Bar Chart Sample Data

```
"#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

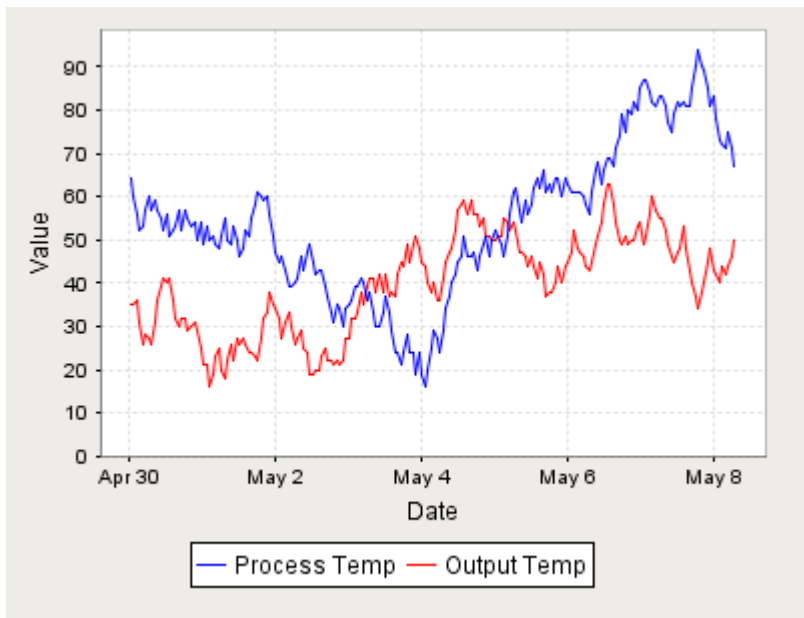
The **Extract Order** property on the chart determines how data series are defined. By default, the property is set to **By Row**, which means each row is a **series**, and each column (except the first) is a **category**. Based on the initial dataset, datapoints are grouped by area (column) and then 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.

▼ Bar Chart Sample Data


```
"#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"

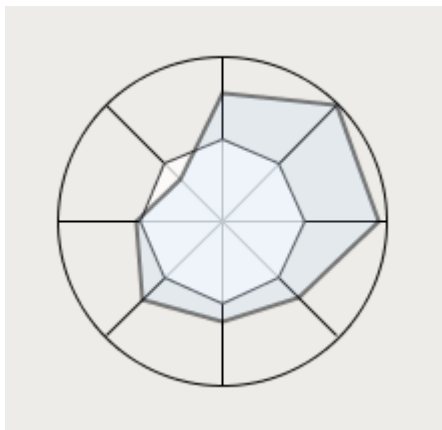
"2018-05-03 19:07:15", "28", "49"
"2018-05-03 20:07:15", "24", "45"
"2018-05-03 21:07:15", "24", "49"
"2018-05-03 22:07:15", "19", "51"
"2018-05-03 23:07:15", "24", "48"
"2018-05-04 00:07:15", "19", "45"
"2018-05-04 01:07:15", "16", "44"
"2018-05-04 02:07:15", "20", "40"
"2018-05-04 03:07:15", "25", "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 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 **Cell 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
72.141	4	92
94.934	2	94.934
86.726	12	90
59.169	2	88
67.164	23	95
68.668	24	90
50.648	9	90
45.044	15	100

Column Name: ---- Column Type: ----

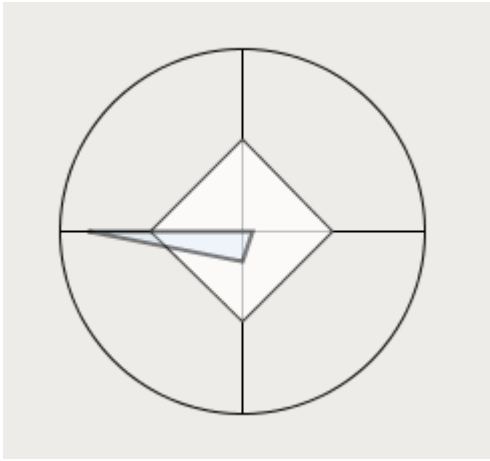
OK

Cancel

Min and Max

The **Min** and **Max** column, aside from determining the limits on the 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**.

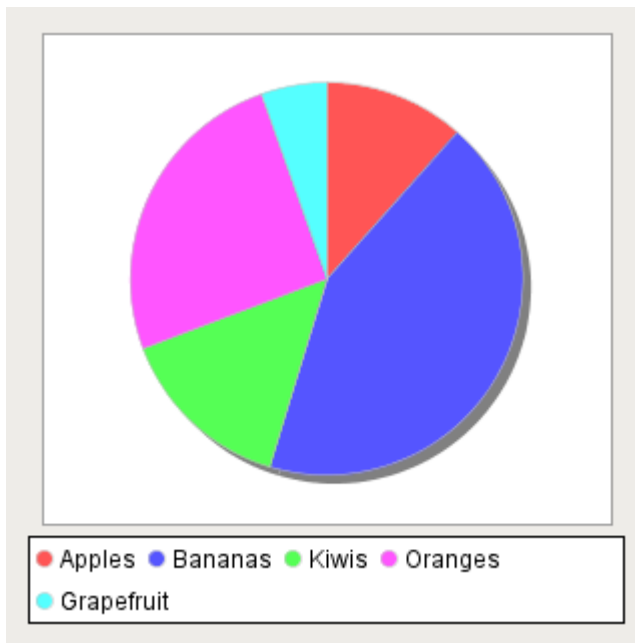


▼ Radar Chart Sample Data

```
"#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.

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.

▼ Pie Chart Sample Data Extracted by Row

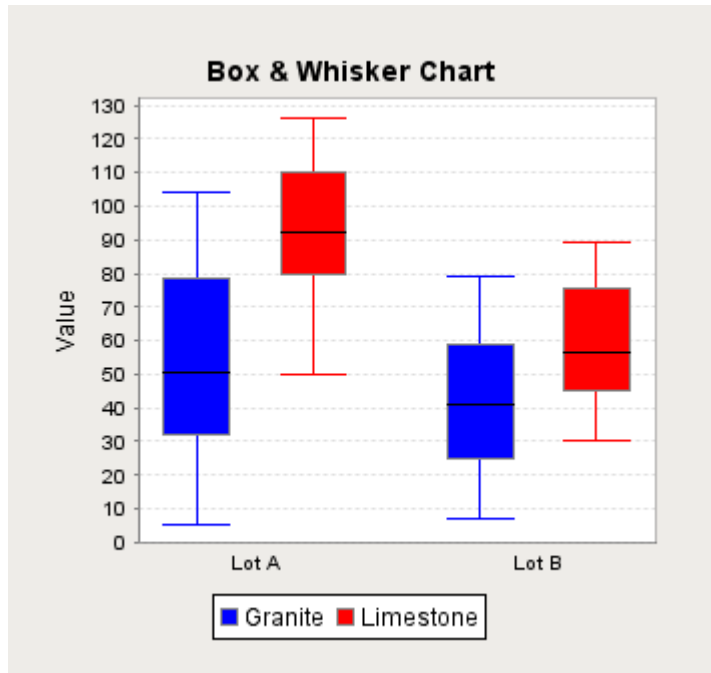
```
"#NAMES"  
"Grapefruit","Apples","Bananas","Kiwis","Oranges"  
"#TYPES"  
"I","I","I","I","I"  
"#ROWS","1"  
"7","15","56","19","33"
```

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.

 Dataset Viewer 

Key	Granite	Limestone
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

Column Name: ---- Column Type: ----

▼ Box and Whisker Sample Data

```
"#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"

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

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

HTML in Ignition



INDUCTIVE
UNIVERSITY

HTML in Ignition

[Watch the video](#)

Using HTML in Components

Many Ignition 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.

HTML stands for HyperText Markup Language. It is commonly used to style text within webpages. The features that HTML brings to style web pages can be applied to many components within Ignition to style the text within components.

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 Ignition 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.

NOTE

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>...</code>)
<code>...</code>	Font	Colors the contents red. Works with standard color names, hex numbers, or RGB numbers.

Applying HTML to Components



INDUCTIVE
UNIVERSITY

Multi-Line Labels and Buttons

[Watch the video](#)

In Ignition, 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.

Adding HTML into a Label

Instead of having a long label, you can have it auto wrap text by simply adding `<\html>` to the beginning of its text property.

1. Select the label that needs the automatic line wrapping.
2. Select the component's Text property.
3. Insert `<html>` before the start of the text and press enter.

Or, supposed you want to make a label appear like this: This is a **bold** word

1. Create a label on your window.
2. Select the label's text property and enter the following text:

```
<html>This is a <b>bold</b> word
```

And if you wanted to have a line break in a specific spot in the text.

1. Create the label and select the Text property.
2. Insert `<html>` before the start of the text. Select the part of the text that you want to see the break occur and type `<br>` in this location and press enter.

```
<html>I would like to have a new line <br> so that the rest of my text will be below it.
```

Adding HTML into a Button

Suppose you want to make a button appear with Stop the Process, with the word `process` appearing in red. The word "Process" starts on a new line. Also, the words must be centered on the button.

1. Create a button on your window.

2. Select the buttons text property and enter the following text:

```
<html><center>Stop the <br><font color="red">Process</font>
```

Adding HTML into a Component's Mouseover Text Property

The mouse over text property creates a popup when the user places their mouse over the component for an extended period of time. The mouse over text is an excellent way of giving the user additional information. The mouse over text property also handles HTML formatting.

Suppose you want a list of instructions to appear in the mouse over text of a component as the following:

These are the instructions:

1. Stop this process.
2. Check on this.
3. Remove that.

To create this type of mouseover text, do the following:

1. Create or choose a component on your window.
2. Select the component's mouse over text property, and select the property's text editor icon () to edit the text.
3. Enter the following text:

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>
```



Note that this technique allows you to format the text across multiple lines. Line breaks do not occur unless they are explicitly declared with the `<br>` element or the component wraps the contents because of the lack of space for its contents.

Adding HTML into a Table

The contents of a table will render HTML formatting. The HTML formatting has to be part of the specific cell. If you want to have HTML formatting affect all rows of the data then it will be necessary to concatenate the HTML tags into the raw data that is populating the table.

Suppose that you wanted to edit a cell of a table in order to render its contents with HTML formatting. Simply access the table's underlying data and enter the HTML tags into the cell that you would like to format. The same techniques used for HTML formatting throughout Ignition apply to the cells of a table as well.

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

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

Component Animation



INDUCTIVE
UNIVERSITY

Component Animation

[Watch the video](#)

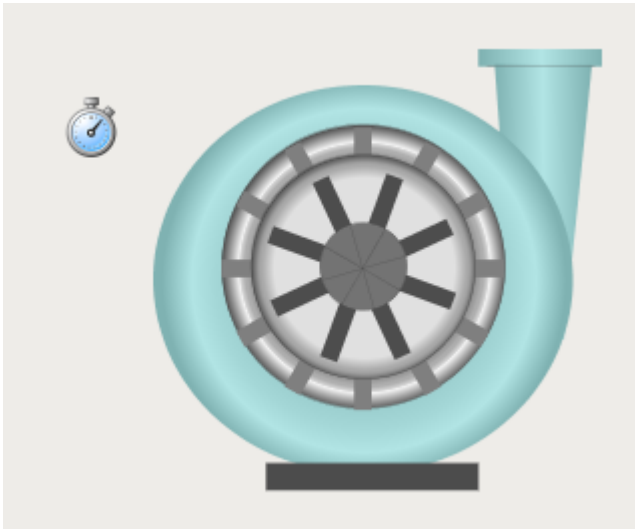
Making Components Move

Animation can be a useful tool to help visualize what is happening at any given time. Animations make it easy to tell if a machine is running, or a conveyor is moving with a quick glance at the screen, and it can help highlight which components aren't currently in use. There are two main ways of creating animation: actually moving all or part of a component, or cycling through a few different static images of a component that give the illusion of moving. Each method has its advantages. For example, making a dump truck move forward and backwards may be better accomplished with actually moving the component. It would be fairly simple to make the whole component move forward on its x coordinate, and then back the same amount. But, if the truck was instead stationary, and just its bed were moving up and down to simulate the truck dumping its cargo, it may be difficult to seamlessly move the component, since it would not just be moving on one axis, but would require careful rotation and movement combined. In this instance, it would be far easier to create a few static images of the truck with its bed in various states, and then cycle through those images fairly quickly so it looks like it is moving. Let's go over both methods below.

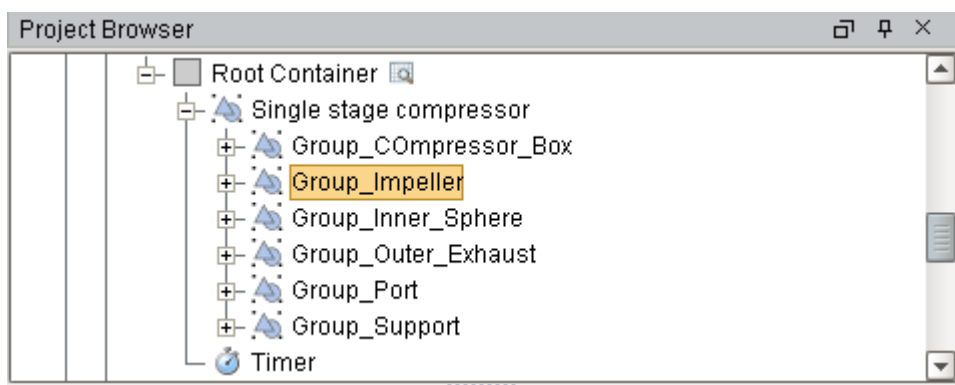
Components that Actually Move

To actually move components around during runtime usually involves binding all or part of an image or svg to the value property of a timer or signal generator. The symbol factory components all contain an Angle property. Binding the Angle property of a spinning part of a motor can help us visualize when the motor is running or not.

1. Pick out a component that you would like to animate. I used the Single Stage Compressor from the Symbol Factory. You will also want to grab a Timer Component.



2. Expand out the Single Stage Compressor component. You will notice that the whole component is made up of many smaller pieces. We want to select the piece called Group_Impeller.



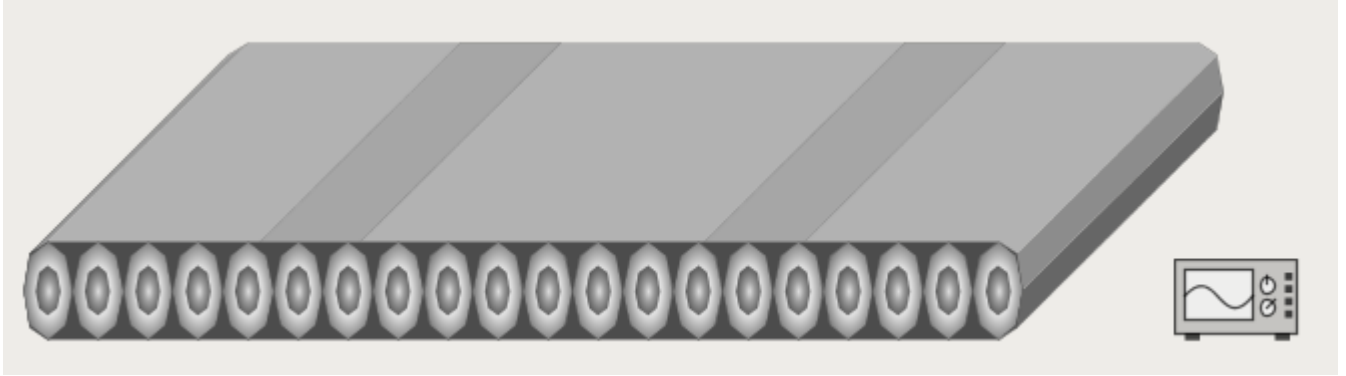
3. Bind the Group_Impeller's Angle property to the Value property of the Timer.
4. We now need to modify the properties of the timer to ensure a good rotation. Set the Delay (ms) to 200, the Step By to 10, and the Bound to 360.
5. You should now be able to set the Timer's Running? property to True, and your component will now animate. You can adjust the Delay (ms) to be lower or higher to adjust the speed at which the component rotates.

Giving the Illusion of Moving

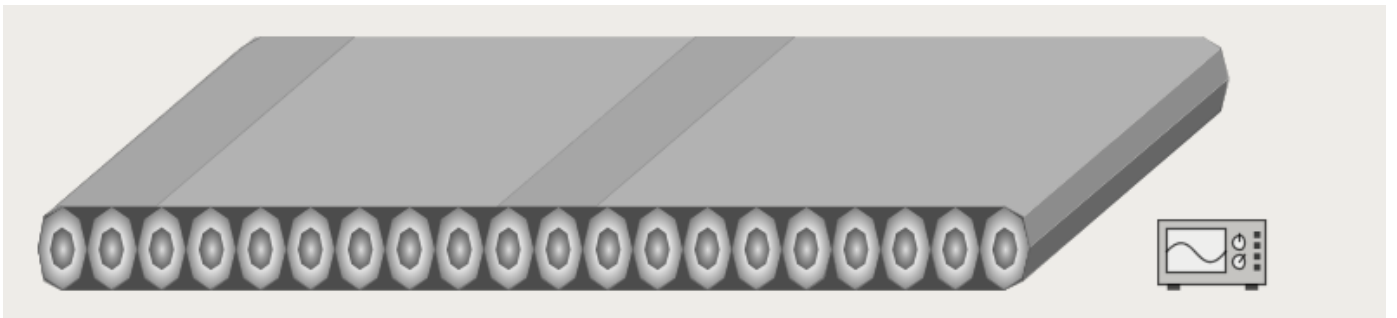
With this method, you can make components look like they are moving without actually moving anything. What we do here is duplicate the component multiple times and modify each component to be a little different than the others, and then show and hide them in the correct order to make it seem like they move.

The following example uses a Signal Generator component to drive the animation, but any incrementing value can be used, such as the Value property on a Timer component, an accumulating value in a PLC, or the current time in seconds.

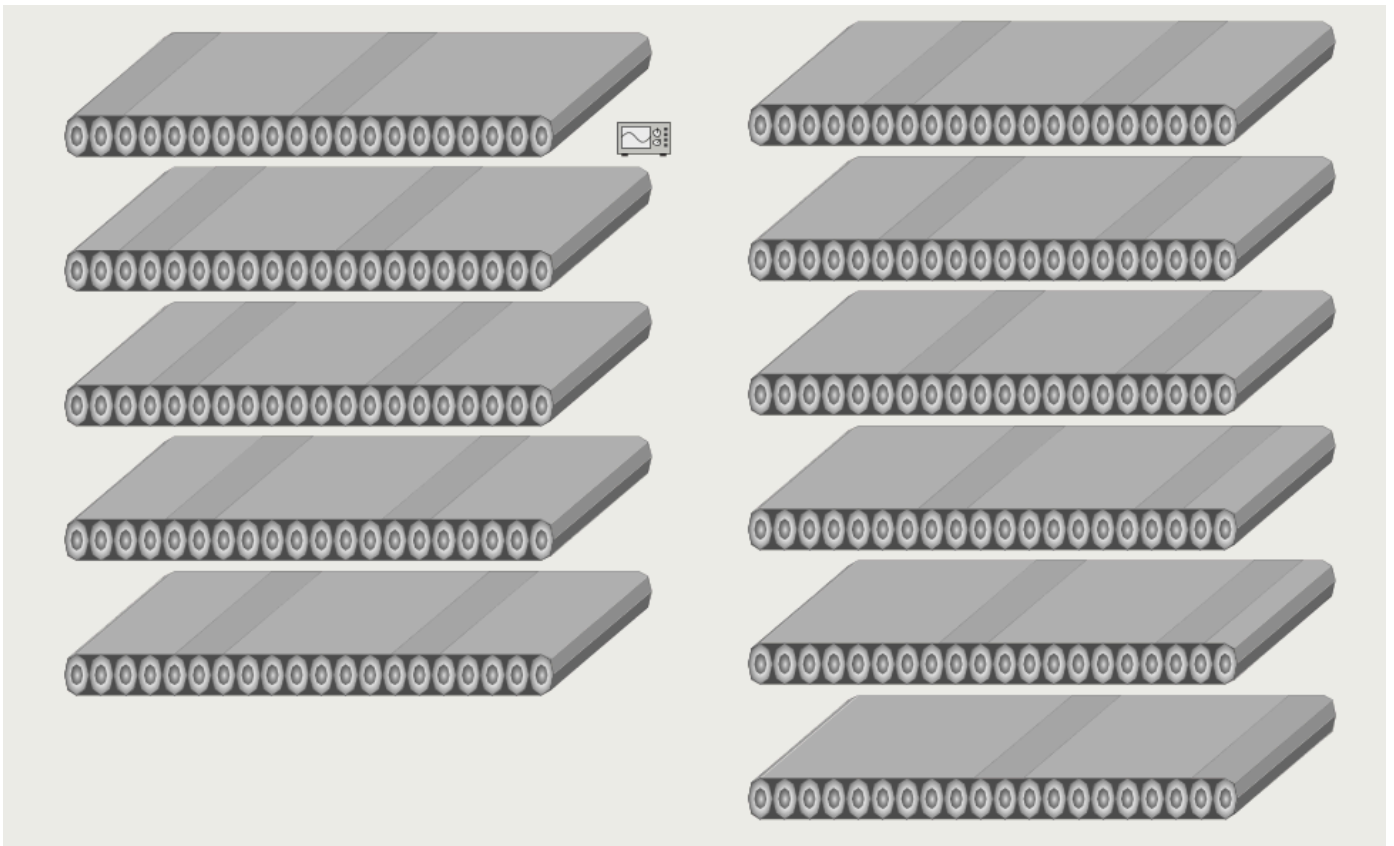
1. Pick out a component to animate. For this example, I selected the Horizontal Conveyor with Perspective from the Symbol Factory. I also dragged a signal generator on the screen.



2. Duplicate this component 10 times for a total of 11 conveyors.
3. Select the first instance, expanded the Group_Conveyor_Belt, and moved the Path 4 and Path 5 Components. On this instance, I moved them both an equal distance left so that the leftmost component is on the left edge of the conveyor.



4. Repeat this with the rest of the components, except move them slightly to the right of the previous instance. Notice how the gray bars slowly move from left to right over all the components when viewed in sequence.



- Next, stack them all on top of each other exactly. This is easily done by selecting all of the conveyors, then going into the Alignment menu and selecting both Align Centers Horizontal and Align Centers Vertical.
- Once stacked, select the first Conveyor from the Project Browser (the one that has the bars on the far left), and place an expression binding on its Visible property that looks like this.

```
if({Root Container.Signal Generator.value} = 0, 1, 0)
```

Duplicate this across all of the conveyors, but increment the first number by 1 each time. The last conveyor (the one that has the bars on the far right) should have the following expression:

```
if({Root Container.Signal Generator.value} = 10, 1, 0)
```

- Select the Signal Generator component. Set the Signal Type to Ramp, the Period to 1000, the Values/Period to 11, the Upper Bound to 11, and the Lower Bound to 0.
- Setting the `Running?` property of the Signal Generator to True will now cycle through showing all of the conveyors, which will make it look like it is moving.

Animation in Scripting

Additionally, the `system.gui.transform` function can move and resize components from a Python script. More details can be found in the Scripting Appendix.

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

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

Control Components



INDUCTIVE
UNIVERSITY

Control Components

[Watch the video](#)

When creating real-time status and control screens, operators typically use control components to write back to the PLC. You can find all the control components that come with Ignition under the Input and Buttons tabs in the Component Palette of the Designer. You can use various **Input** components like the Numeric Text Field, Spinner, or Slider; and **Buttons** such as 2-State Toggle, Multi-State, One-Shot, or Momentary to write values back to a database or PLC. These control components are the core of data entry in Ignition, and the most commonly used that allow data entry.

When choosing control components, be sure to look at the component's properties. Each component has different properties that affect its behavior and can dictate which component to use for your project.

THE EXAMPLES ON THIS PAGE ASSUME YOU HAVE SOME TAGS IN YOUR TAG BROWSER

This section describes several control components and how to use them in a controls environment. These examples assume you have some Tags created. If not, you can drag some simulator Tags from the [OPC-UA server](#) by going to your Tag Browser, clicking on the OPC icon () , and dragging some Tags into your Tags folder, or creating some [Memory Tags](#). Any Tag will work.

Binding Tags

Another thing to mention as you read through the examples, there are the three ways of binding Tags. There is no right or wrong way, simply choose if you want to bind multiple Tags, individual tags, or which ever way is best for you.

- Drag a Tag from the Tag Browser to the component. A yellow outline will appear around the component, release the mouse, and the Tag is instantly bound to multiple properties of the component.
- In the Property Editor, click the Binding icon () next to one particular property you want to bind to a Tag. The Property Binding window will open, select a Tag Binding Type, navigate to the Tag you want to use, and click OK. This one particular property of the component is now bound to the selected Tag.
- Drag a Tag from the Tag Browser on to one individual property in the Property Editor of the component. For example, select an Integer Tag from the Tag Browser and drop it on the Value (Integer) property of a Numeric Text Field. The Tag is now bound to that individual property.

Here are a few examples of control components and how they can be used in a controls environment.

Input Components

Input components allow users to enter or select data. One characteristic that Input components have is that they allow users to modify a numeric or string value and write back to the PLC. Different components have different properties so make sure to look at all of them. When you drag a Tag on to a numeric component, you'll see that it often binds multiple properties. Often there are **Minimum** and **Maximum** values that control what can be entered.

Defer Updates

Before we get started looking at some Input components, we need to mention the Defer Updates property. The Defer Updates property exists on most Input components, and by default is set to **true**. Users typically have specific cases where they want to use it, and cases where they don't want to use it, so it's important to understand how it works with each component.

- If True, then any edits to the component's value will not be submitted until the enter key is pressed. For example: if you type a word into a Text Field, the text property will not be updated until enter is pressed.
- If False, then the component's value will be submitted when each key is pressed. For example: if you type a word into a Text Field, the text property will be updated once for each keystroke in the word.

You can leverage this when you use an Input component as part of a filter for a SQL query on another component. In this case, turning **off** Defer Updates would re-run the query every time a key is pressed. This allows you to make your tables filter what rows are shown as the operator starts typing.

You should not turn off `Defer Updates` when using a bi-directional Tag binding for two reasons.

- Writes are sent immediately but these bindings update at a set rate. This means the operators may not see the update and type in odd values that may trigger unnecessary alarms.
- When using Input components that have a numeric or text `Format Pattern`, this formatting will be applied at every keystroke. That means decimals can be truncated if the number is entered incorrectly.

It's a good idea to test the `Defer Updates` property on the component you intend to use so you know exactly what to expect.

Numeric Text Field



The most important property of the **Numeric Text Field** is the **Number Type**. The `Number Type` property dictates what type of numbers this component will display. You can select from the following value types: Integer, Double, Long, or Float. You can change the `Number Type` (i.e. from a Double to an Integer), but should not bind this property. Instead, bind the corresponding `Value` property. Be careful to select your `Number Type` properly, otherwise the wrong value might get written back to your bound Tag.

Here are descriptions of some of the `Numeric Text Field` properties:

- **Number Type** - this property dictates the numeric formatting that is displayed on the component's value. You'll notice that every time you enter a new value, you will see all the `Value` properties change: `Value (Integer)`, `Value (Double)`, `Value (Long)`, and `Value (Float)`. So it's important to be careful to select the appropriate `Number Type` to match the value property binding.
- **Defer Updates** - it's a good idea to have the `Defer Updates` property set to **'true'** (don't update until enter is pressed), otherwise you're entering each number into the Tag and triggering different values that may have other consequences.
- **Protected Mode** - when set to **'true'**, the `Numeric Text Field` is disabled. The user must double click on the field or press enter in order to edit the field. When editing is complete, press enter again or leave the field, and the field will become disabled again.
- **Reject Updates** - when set to **'true'**, updates coming from property bindings are ignored while the the `Numeric Text Field` component has focus (is being edited).
- **Use Bounds** - when set to **'true'**, values that are outside the **Minimum** and **Maximum** properties will be blocked. Entering a bad value will instead set the value to the closest bound (i.e., entering a value above the maximum will instead set the maximum value).

- To see an **Out-of-Bounds Message** when an operator enters a value out of range, you need to set both the **Use Bounds** and **Error on Out-of-Bounds** properties to **'true'**. When an operator enters a value that is out of bounds, the text in the **Out of Bounds Message** property will appear and the new value will not be entered. If the **Error on Out-of-Bounds** property is false, this message will be ignored.
- **Touch Screen Mode** - When Touch Screen Mode is enabled, a numeric keyboard will be displayed when a user touches the Numeric Text Field component.

Numeric Text Field Example

Drag a Numeric Text Field component onto your window, and drop a Tag on to the component. In the **Property Editor**, you'll notice that it binds multiple properties (in bold). (In the image below, you can see that we selected the WriteableDouble1 Tag from our simulator and it is bound to the **Value (Double)** property bidirectionally).

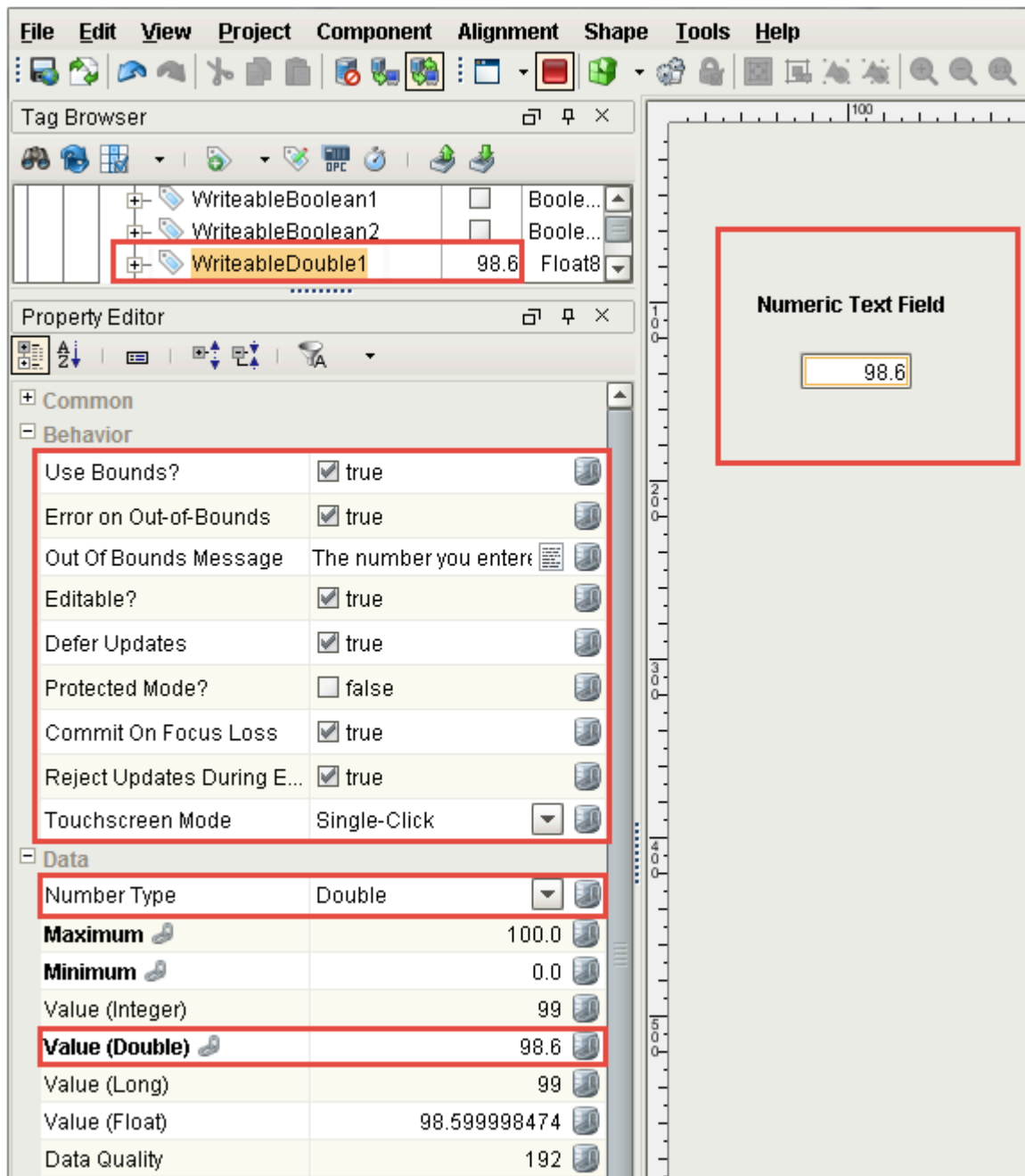
In the **Property Editor**, check the component's properties to see if you want to enable, disable, or change any default values or styles for any of the component's properties.

1. Set the **Number Type** property to **Double** (but don't bind it).
2. Keep the default values set for **Maximum** and **Minimum** at **100** and **0**, respectively.
3. To send an Out of Bounds Message, set the **User Bounds** and **Error on Out-of-Bounds** properties to **'true'**. You can use the default Out-Of-Bounds message, or create one of your own.

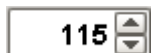
Test it out!

Make sure your **Gateway Communication Mode** is set to read/write. You can enter a value for the Numeric Text Field in three ways:

- In **Preview Mode** of the Designer, enter a value (i.e., 98.6) in the component, and you'll see the value change in the **Tag Browser** next to your bound Tag.
- In the **Property Editor**, enter a value directly in the **Value (Double)** field making sure it corresponds to your Number Type setting. (i.e., If your Number Type is **Double**, then enter a value in the **Value (Double)** field). If you enter a valid value, the Tag will update. If you enter an out-of-bounds value, an Out-of-Bounds Message will popup, and the value will return to the last known good value.
- Once you save and publish your project, login to the **Client**, and enter a new value in the Numeric Text Field component. If the value is within bounds, the Tag will update along with the input component, otherwise, you will receive an Out-of-Bounds Message.



Spinner



The **Spinner** component is another Input component that allows users to modify a numeric value in the PLC. It not only allows users to edit a numeric value directly, but to "spin" the value up or down by some defined step size, using the up and down buttons that are part of the component.

The most important property of the Spinner is the Spinner Mode. The **Spinner Mode** dictates what type of numbers this component will display: Integer, Double, or Date. When setting up property bindings, make sure to use the value property that corresponds to the Spinner Mode. For example, if you choose an Integer as your **Spinner Mode**, you need to bind a Tag to the **Value (Integer)** property. Another convenient property of the Spinner component is the **Numeric Step Size** property. This

property dictates how much you step up or step down when using the buttons and the Integer or Double Spinner Mode.

The Spinner component has **Numeric Minimum** and **Numeric Maximum** properties that you can set forcing an operator to enter a value between the bounds. If an operator enters a value out of the minimum or maximum range, the value property in the Spinner component will not accept the value. The Spinner component does not have an error message property that alerts you when a value is out of bounds, so it's not always obvious when entering a value directly in the Spinner component that it is out of bounds. When typing in a value directly, a user would have to click away from the spinner to change focus, then they would see the value go back to the last known good value. If the up and down buttons were used to change the value and the new value is out of bounds, the spinner value will not change, thus alerting an operator that the value is out of bounds.

The **Touch Screen Mode** is an available property on the Spinner component. When enabled, this overrides the Client Touchscreen Mode to force a keyboard to be displayed when a user touches the Spinner. Setting this property to False does not block the keyboard when the Client is in Touchscreen Mode. Lastly, the **Defer Updates** property is not an available option on the Spinner component, it will always behave as if Defer Updates was True.

Spinner Example

Drag a Spinner component onto your window, and drag a Tag on to the component. In the **Property Editor**, you'll notice that it binds multiple properties (in bold), and binds the **Value (Integer)** property bidirectionally if you use an Integer Tag. (In this example, a WriteableInteger5 Tag was selected from our simulator and it bound the **Value (Integer)** property bidirectionally).

In the **Property Editor**, check the component's properties to see if you want to enable, disable, or change any default values or styles for any of the component's properties.

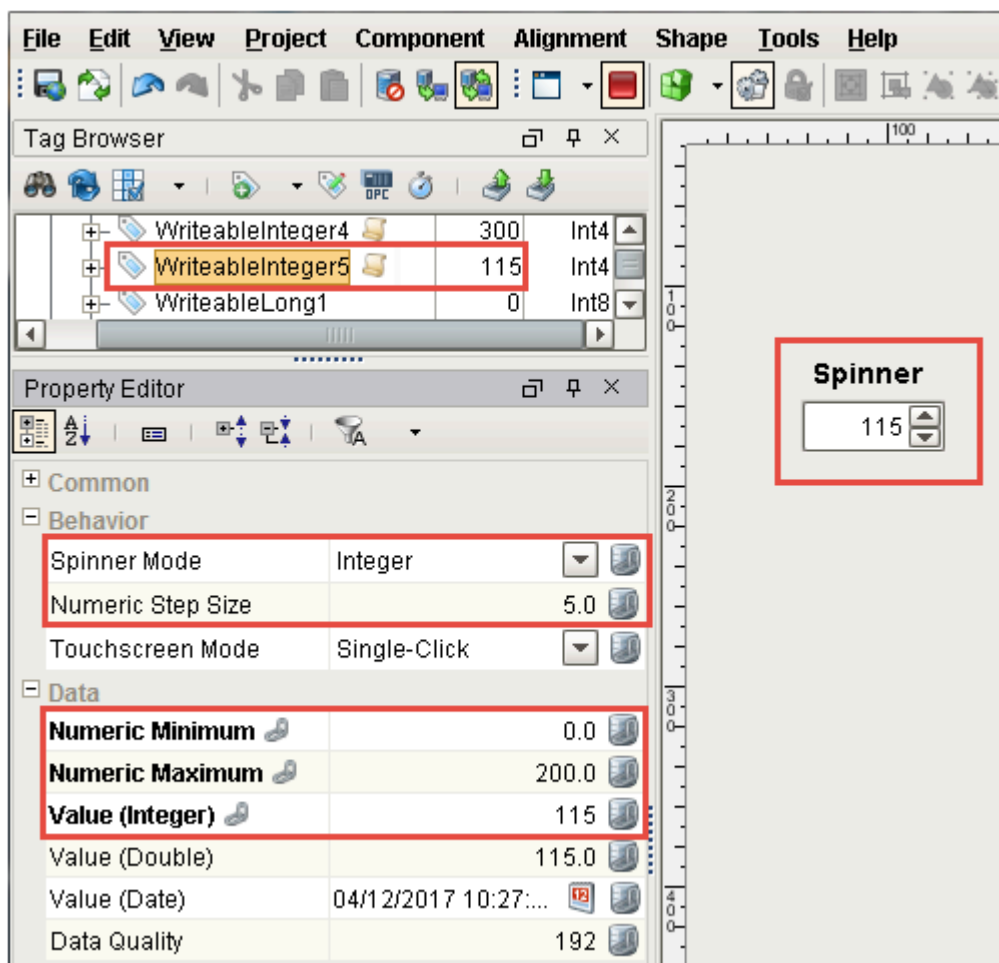
Let's make a few changes.

1. In the Property Editor, set the **Spinner Mode** to **Integer**.
2. Set the **Numeric Step Size** to **5** or whatever you want. (The default is 1).
3. Next, change the **Numeric Maximum** property to ***200**, so any value over 200 is out of bounds and doesn't write to the property.
4. Keep the **Numeric Minimum** value at the default of **0**.

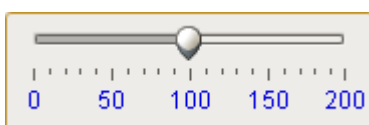
Test it out!

Make sure your **Gateway Communication Mode** is set to read/write. You can enter a value for the Spinner in three ways:

- In **Preview Mode**, enter a value (i.e., 115) directly in the component, or use the up or down buttons to change the value. Once you enter a new value, you'll see the value change in the **Tag Browser** next to your bound Tag.
- In the **Property Editor**, enter a new value directly in the **Value (Integer)** field making sure it corresponds to your Number Type setting. (i.e., If your Number Type is an **Integer**, then enter a value in the **Value (Integer)** field). If you enter a valid value, the Tag will update. If you enter an out-of-bounds value and press enter, the new value will not update. When you change focus from the component, the value will return to the last known good value.
- Once you save and publish your project, login to the **Client**, and enter a new value in the Spinner component. If the value is within bounds, the Tag will update along with the component. If not, change focus and the value will return to the last known good value.



Slider



The **Slider** component allows a user to drag an indicator along a scale to choose a specific value within a given range and write back to the PLC. Unlike the **Numeric Text Field** and **Spinner** components that use various Tag types, the **Slider** component uses only an **Integer** type for its **Value**.

property. It is very easy to use and has quite a few appearance properties you can change. For example, you can choose to display your Slider horizontally or vertically, change the foreground and background colors, tick spacing, labels, and lots more. You can set the Minimum and Maximum Values for your Slider. If the Value property holds a value outside of the minimum and maximum values, an overlay appears on the component. The overlay will not be removed until a valid value is entered.

The Slider component has the **Defer Updates** property set to **'true'** by default because as you drag the indicator around on the slider, you probably don't want to see the value constantly updating.

Slider Example

Drag a Slider component on to your window, and drop a Tag on to the Slider component. In the **Property Editor**, you'll notice that the Tag binds multiple properties (in bold), and binds the **Value** property bidirectionally. (In this example, a WriteableInteger4 Tag was selected from our simulator and it bound the **Value** property bidirectionally).

In the **Property Editor**, check the properties to see if you want to enable, disable, or change any default values or styles for any of the component's properties.

Let's make a few changes.

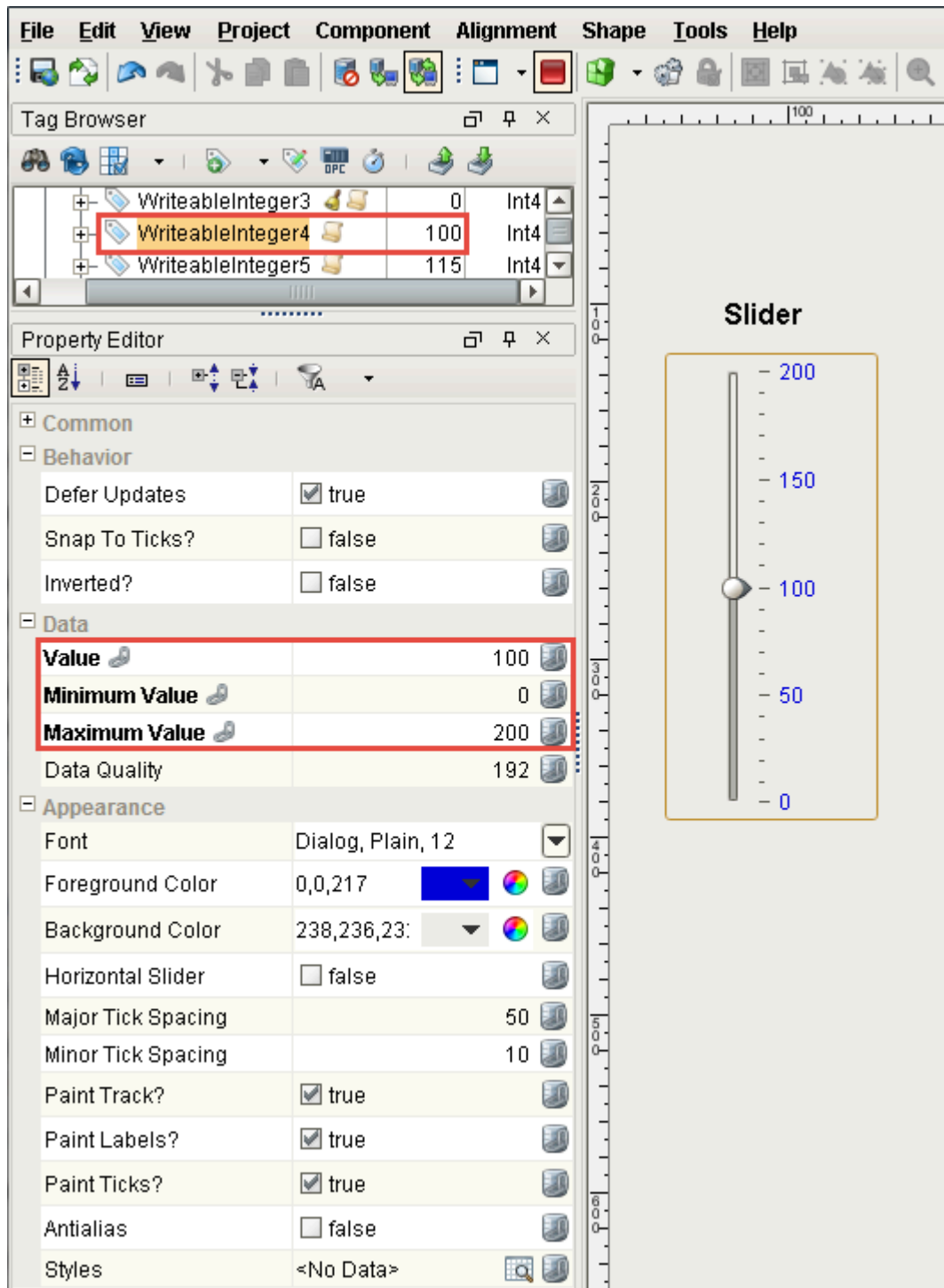
1. The Slider is set to display horizontally by default. Set the **Horizontal Slider** property to **false** so it displays vertically.
2. Set a range on the Slider. Click on the binding icon for the **Maximum Value** and set it to **200**. Keep the **Minimum Value** set to the default value of **0**.
3. Change the **Foreground Color** to **blue**.
4. Set the **Major Tick Spacing** to **50**.
5. Set the **Minor Tick Spacing** to **10**.

Test it out!

Make sure your **Gateway Communication Mode** is set to read/write. You can change a value for the Slider in three ways:

- In **Preview Mode**, use the indicator and drag it along the scale to set a value (i.e., 100). Once you enter a new value, you'll see the value update in the **Tag Browser** next to your bound Tag.
- In the **Property Editor**, enter a new value directly in the **Value** field. The slider indicator will move to the specified value on the scale and the bound Tag will update. If you enter value that is out of range of the Minimum and Maximum values, a component overlay will appear on the component. When a valid value is entered, the overlay disappears.

- Once you save and publish your project, login to the **Client**, slide the indicator to any value on the slider, and the Tag will update.



Formatted Text Field

(212) 455-9191

The **Formatted Text Field** is an excellent Input controls component that allows you to format data added to an input field. The Formatted Text Field is actually a specialized text field used for alphanumeric text input that matches a specific pattern or format in a specific way. It operates in two modes:

- **Formatted Mask** - user input is automatically formatted and restricted to a pattern. There are several commonly used formatted mask patterns to choose from such as phone number, percent, date, etc.
- **Regular Expression** - user input is validated against a special string that defines a set of allowed strings. There are several commonly used regular expression patterns like email address, IP address, HTTP URL, and more. Input that matches the regular expression is allowed, and input that doesn't match, is restricted. RegEx is very complicated, so don't worry if you don't understand each of the sample patterns.

The Formatted Text Field has an extensive list of properties. Here are descriptions of some of the commonly used Formatted Text Field properties:

- **Validation Mode** - required. Select Regular Expression or Formatted Mask Mode.
- **Formatted Mask Pattern** - if Validation Mode is set to Formatted Mask, fill in this property. You can choose from the list of commonly used formatted masks.
- **Reg Ex Pattern** - if Validation Mode is set to Regular Expression, fill in this property. You can choose from the list of commonly used regular expressions.
- **Commit While Typing** - commits valid text while user is typing. The default is set to '**false**'. It's recommended not to commit text while typing because it may have unintended consequences.
- **Touch Screen Mode** - when Touch Screen Mode is enabled, a keyboard will be displayed when a user touches this component expecting a value to be entered.
- **Allows Invalid Text** - when '**true**', allows the user to type in a value but will not commit the value unless it matches the selected pattern.
- **Committed Value** - holds the current committed value.

Formatted Text Field Example

Drag a Formatted Text Field component on to your window, and drop a string Tag on to the Formatted Text Field component. (In this example, a WriteableString2 Tag was selected from our simulator).

In the **Property Editor**, check the properties to see if you want to enable, disable, or change any default values or styles for any of the component's properties.

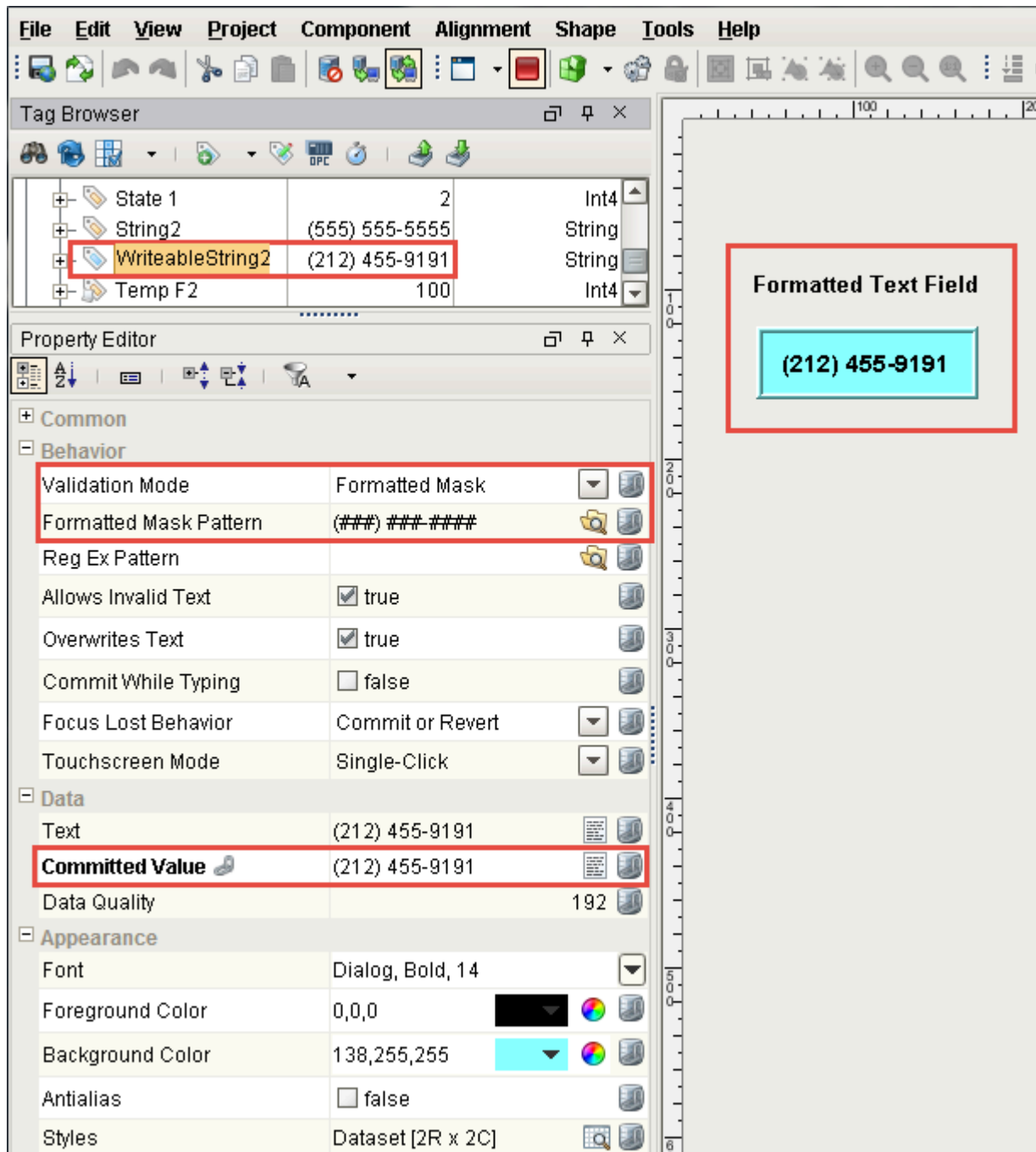
The following examples uses the Formatted Mask Mode.

1. Set the **Validation Mode** property to **Formatted Mask Mode**.
2. Click on the folder next to the Formatted Mask Pattern (📁) and select any pattern from the list (i.e., **Phone Number**). You'll notice how the Formatted Mask Pattern property and component display the format selected.
3. Next to the **Committed Value** property, click the binding icon and select a **Tag** binding type, select a **string** Tag, mark it **Bidirectional**, and click **OK**.

Test it out!

Make sure your **Gateway Communication Mode** is set to read/write. You can test out the component in two ways:

- In **Preview Mode**, enter a phone number (or whatever format you setup), hit return, and you'll see the phone number in the **Tag Browser** next to your **string** Tag. The Committed Value property is also updated with the phone number.
- Once you save and publish your project, login to the **Client**, enter a phone number in the component and press **Enter**.



Button Components

Button components are very versatile components that also work great in a controls environment. They can do things like change the state of a machine, and be configured to accept user input. Here are a few examples of buttons used as control components and how they can be used in a controls environment.

2-State Toggle Button



The **2-State Toggle** button is an excellent button to use to toggle a value between two states on a machine like On/Off, Stop/Run, etc. It is easy to setup and uses four integer values: Control Value, Indicator Value, and the values that define the two different states, State 1 and State 2. Each time the 2-State Toggle button is pressed, one of the State Values is written to the Control Value. The Indicator Value property determines which state the machine is currently in.

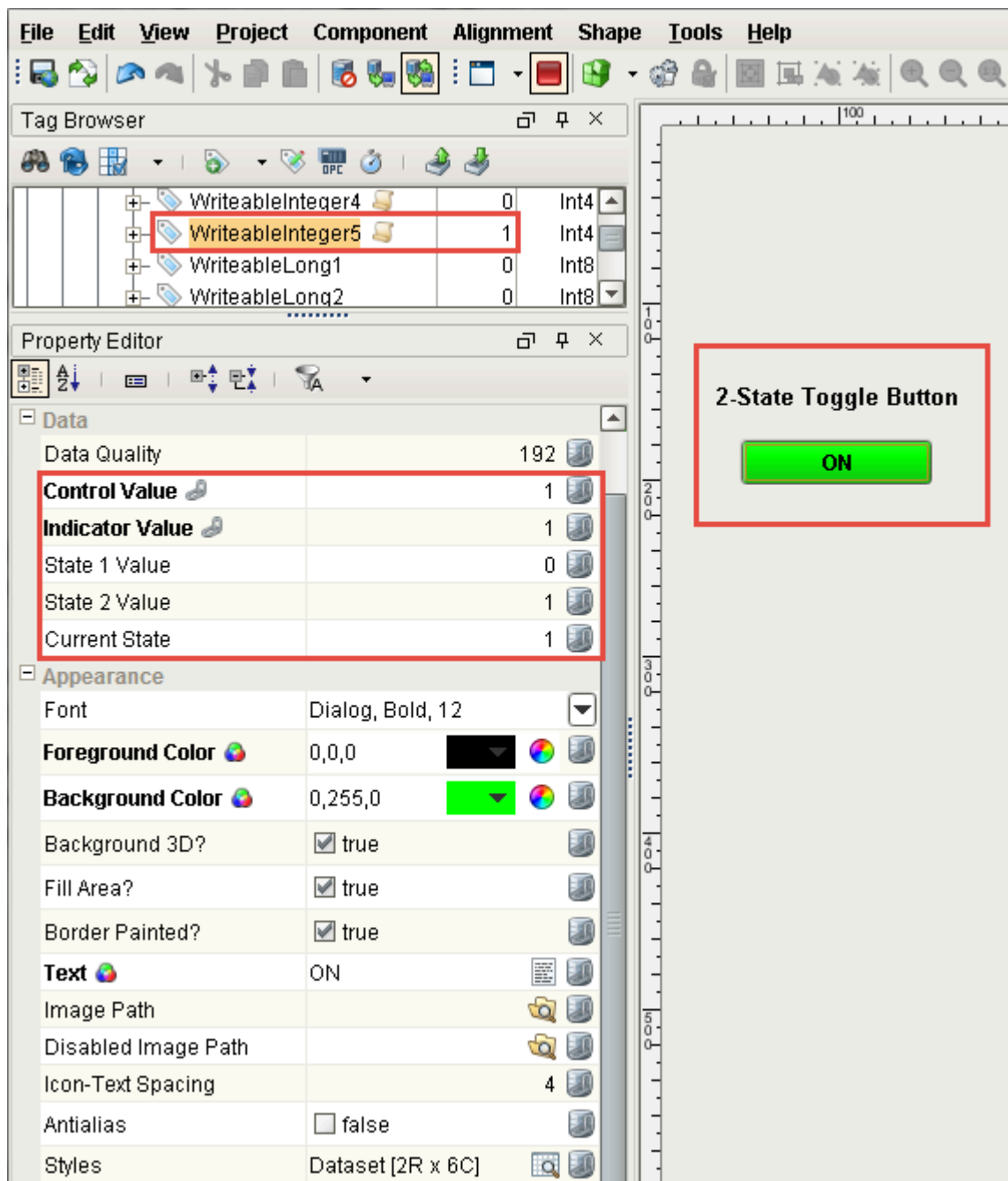
The 2-State Toggle button also has a Style Customizer that allows you to define the visual styles for the two states such as background and foreground colors, image path, font, border, and lots more. This helps an operator quickly identify at a glance which state a machine is currently in.

2-State Toggle Button Example

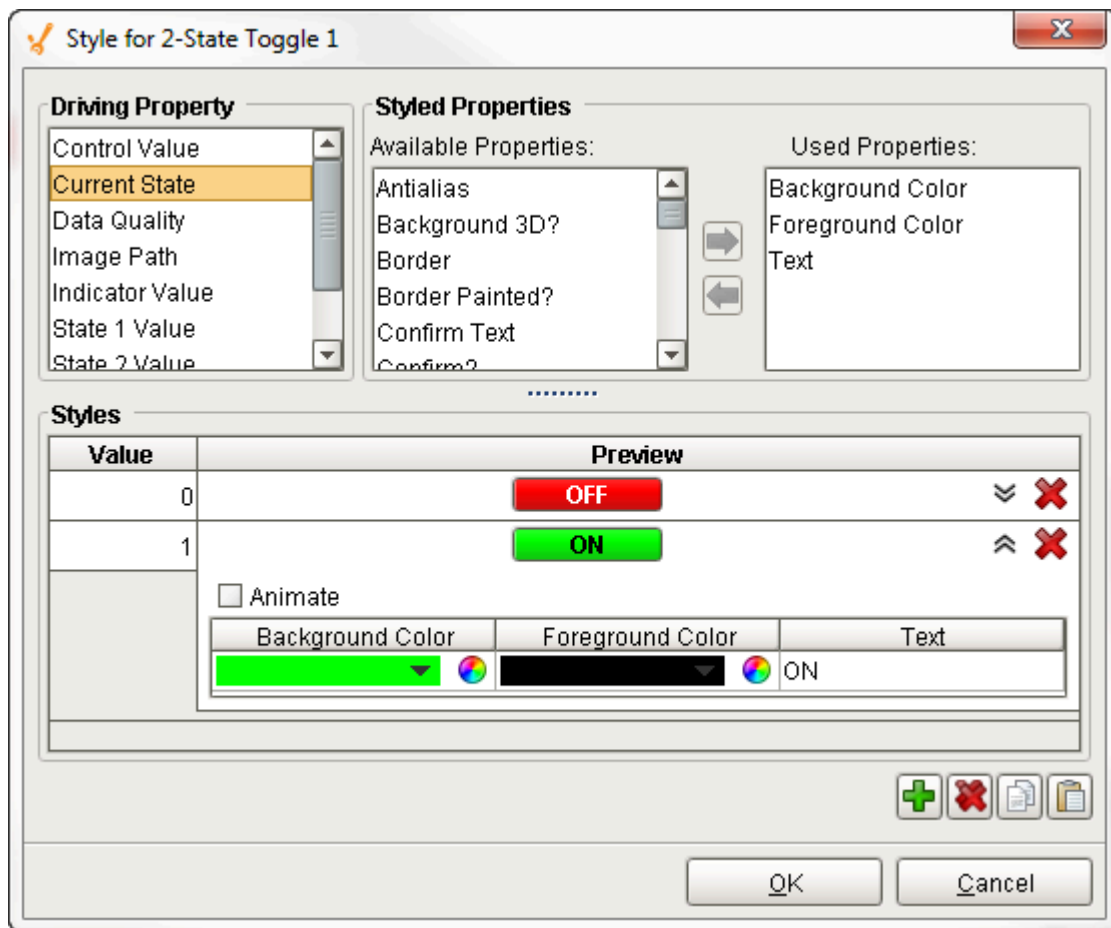
Drag a 2-State Toggle button on to your window, and drop a Integer Tag on to the component. In the Property Editor, you'll notice that the Tag binds to multiple properties (in bold), including the **Control Value** and **Indicator Value** properties. (In this example, a WriteableInteger5 Tag was selected from our simulator and bound it to the **Control Value** property bidirectionally).

In the **Property Editor**, check the component's properties to see if you want to enable, disable, or change any default values or styles for any of the properties.

1. Set the **State 1** and **State 2** properties, but you can use the default values of **0** and **1**, respectively. (This example uses the default values for these properties).



2. Double click on the component to open the Style Customizer. Here you change the visual styles of the button by changing the color, font, border, make them blink, etc. The Style Customizer already has some default styles set, so if you don't like them, change them, otherwise, you are good to go!



Test it out!

Make sure your **Gateway Communication Mode** is set to read/write. You can enter a value in three ways:

- In **Preview Mode**, press the the 2-State Toggle button, and you'll see the value change in the **Tag Browser** next to your Integer Tag. Each time you toggle the button, it will change your Integer Tag to either **0** or **1**, and the button will change from **OFF** to **ON** and vice versa.
- In the **Property Editor**, enter a value of **1** directly in the Control Value property. The 2-State Toggle button will change to **ON**. The Indicator Value will also change to **1** showing the current state of the machine. (If you used the default colors in the Style Customer, the button will change from red to green).
- Once you save and publish your project, login to the **Client**, and toggle the button to see the button change from **ON** to **OFF** and vice versa.

Multi-State Button



The **Multi-State Button** is a commonly used component in a controls environment. It's easy to configure, and has it's own Style Customizer that lets you define a set of visual styles that change based on the current state. It's similar to the 2-State Toggle Button, but you can have more than two states. The Multi-State Button is a series of two or more buttons, arranged in a column, row, or grid. Each button represents an integer-valued state. Each state defines two styles for a button: the selected style, and the unselected style. Each button is automatically displayed with the correct visual style based on the current state (Indicator Value). When a button is pressed, its state's value is written to the Control Value.

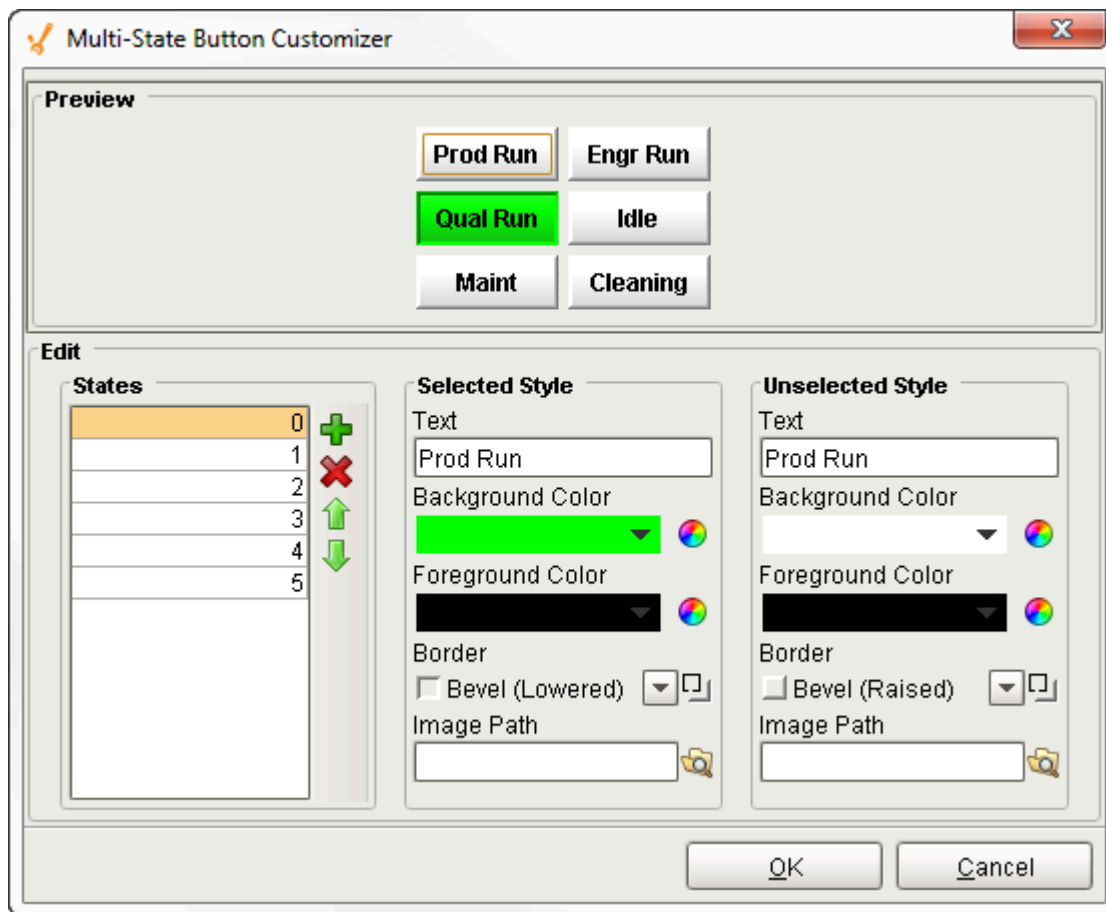
Multi-State Button Example

Drag a Multi-State Button on to your window, and drop a Integer Tag on to the component. In the Property Editor, you'll notice that the Tag binds to the **Control Value** and **Indicator Value** properties (in bold). (In this example, a WriteableInteger5 Tag was selected from our simulator and bound it to the **Control Value** property bidirectionally).

In the **Property Editor**, check the component's properties to see if you want to enable, disable, or change any default values or styles for any of the properties.

Let's make a few changes.

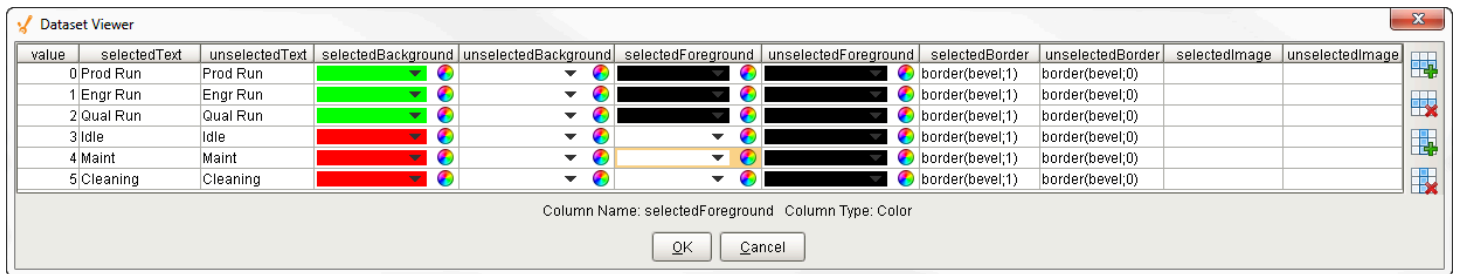
- Open the **Multi-State Button Customizer** by right-clicking on the component and selecting **Customizers -> Multi-State Button Customizer**.
- Add three additional states by clicking on the green plus icon.
- Enter a name for the selected and unselected text that will be displayed on the button.
- You can have a different visual style when the button is selected and unselected. Edit the visual styles for each button when it is selected and unselected. This is optional, but does help an operator quickly identify the current state of the equipment.
- Click the OK button to close the customizer when you are finished with your changes.
- In the **Property Editor**, you can change the **Display Style** property from a column to rows, or a grid. Our example uses a Grid.



NOTE

You have a couple of choices on how to add and remove states, and change the visual styles of each state. You can use the **Multi-State Customizer** or the **States** property by opening the **Dateset Viewer**.

- **Multi-State Button Customer** - As above, Double click on the component, add additional states by clicking the green plus icon () . The state will automatically assign the next number. We recommend you organize your states in numerical order to make it easier to configure and update in the future. You can enter the name for the selected and unselected text that will be displayed on the button. You can even edit the foreground, background, border, and add an image to selected and unselected styles for each button. The Multi-State Button Customizer has a Preview area that shows the text and the styles as you configure them. An example of the Multi-State Button Customizer is shown below.
- **States** property - In the **Property Editor**, click on the the Dataset Viewer icon () next to the States property. Here you can view all states, add, remove, and edit states. When adding a state, by default, the value assigned is **0**. We recommend you organize your states in numerical order to make it easier to configure and update in the future. Enter the name for the selected and unselected text that will be displayed on the button. You can edit the foreground, background, border, and even add an image to the selected and unselected styles for each button. Once you close the Dataset Viewer, the styles for your buttons will update. The Dataset Viewer for the States property is shown below.

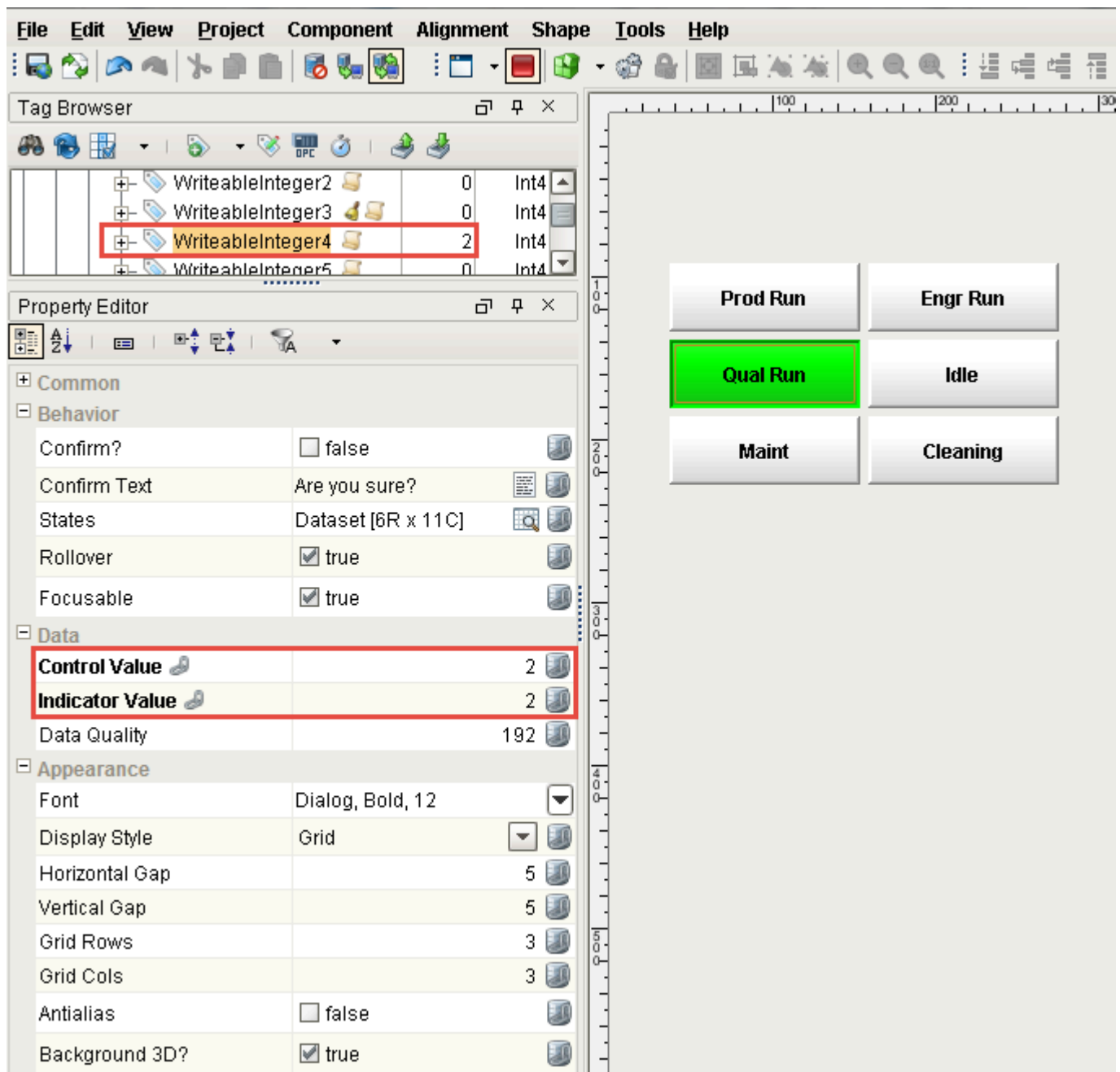


ABOUT STATES...

You'll notice when you first add your Multi-State button and open either the Customizer or the Dataset Viewer, the states are in random order. It's a good idea to first put your states in numerical order, then match each state value with the corresponding button, and finally enter the text and style for the selected and unselected styles for your buttons.

Make sure your **Gateway Communication Mode** is set to read/write. You can enter values in three ways:

- In **Preview Mode**, press one of the buttons and you'll see the value change in the **Tag Browser** next to your bound Tag. You'll notice the Control Value and Indicator Values will also change corresponding to the state value. In the **Property Editor**, enter one of your state values in the **Control Value** field (i.e., 2), and the Tag will update. You'll notice that the **Indicator Value** will also update with the current state (i.e., 2). The button will change text and color depending on how it was configured. One you save and publish your project, login to the **Client**, and press any button. You will see the button changes state from selected to unselected as you press different buttons.



One-Shot Button

The **One-Shot Button** allows the operator to only press the button once to write a value to the PLC, and then it waits for the PLC to reset the value before becoming available again. This component is considered safer than the Momentary Button because it receives positive feedback from the PLC that the signal was received. The PLC must be programmed to reset the Value property after reading it, otherwise, the button will remain in a "Writing..." state until it is manually reset.

One-Shot Example

Drag a One-Shot Button on to your window, and drop a Boolean Tag on to the component. In the Property Editor, you'll notice that the Tag binds to multiple properties (in bold), and the Value property is bound bidirectionally. When the On-Shot Button is pressed, it writes a value in its Set Value property to the Value property. By default, the Set Value is **1** and the Value is **0**, which is the ready state. The

button disables itself when it is writing, and displays the text "**Writing...**" When the button is "Writing" you'll see that both the Set Value and Value properties are both equal to 1.

In the **Property Editor**, check the component's properties to see if you want to enable, disable, or change any default values or styles for any of the properties.

How the One-Shot Button Works

- By default, the Set Value property is **1** and the Value property is **0**. When the One-Shot Button is pressed, it writes the value in the Set Value property (i.e., **1**) to the Value Property. Now both values are equal.
- When both values are equal, the One-Shot Button will disable itself, and the following text will be displayed on the button, "**Writing...**"
- Uncheck the Boolean Tag to reset to it, so you can execute the task again using the One-Shot button.

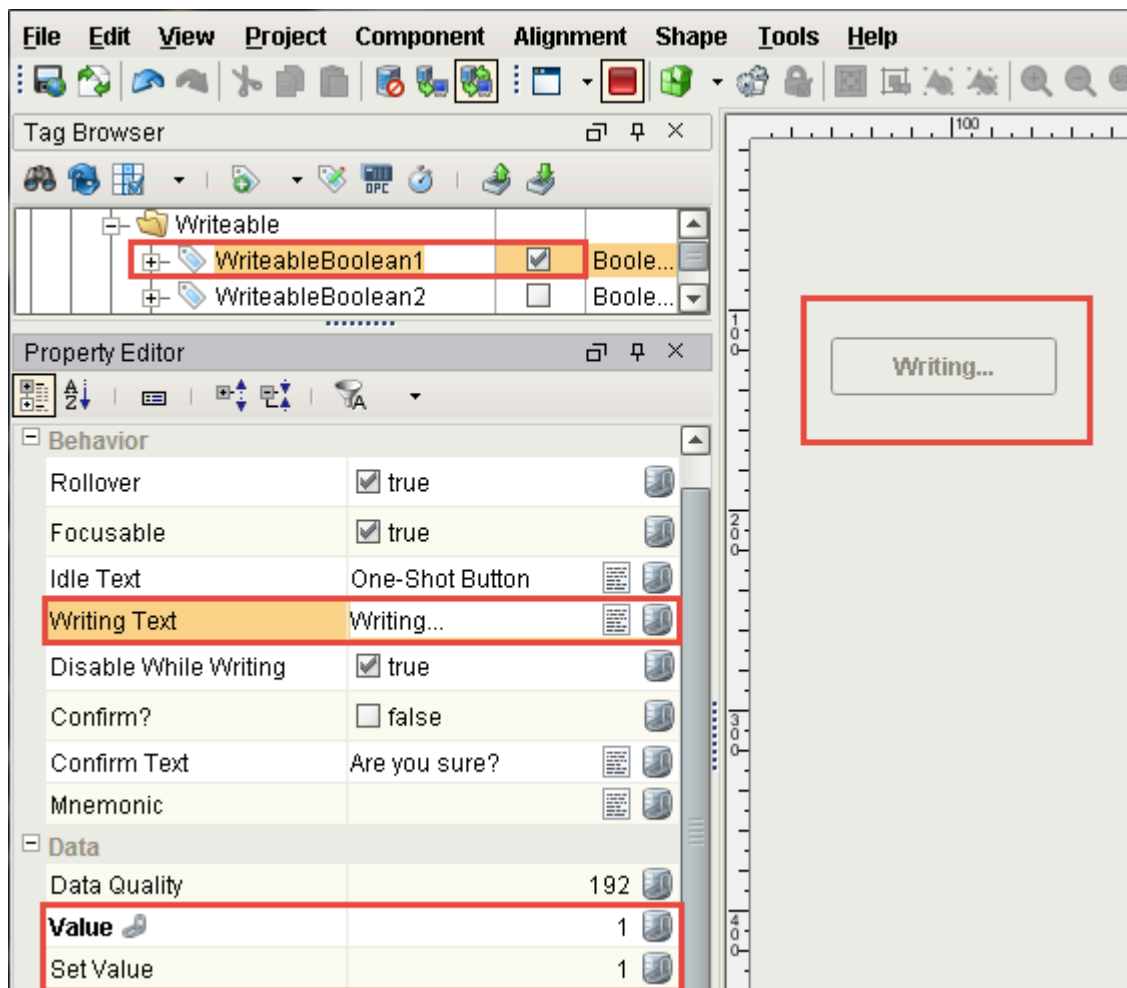
Test it out!

NOTE

This example uses a WriteableBoolean1 Tag so you will need to uncheck the Tag to can clear the "Writing..." state so you can execute the task again. As mentioned earlier, the PLC can be programmed to reset itself after the value is read.

Make sure your **Gateway Communication Mode** is set to read/write. You can change the value in three ways:

- In **Preview Mode**, press the One-Shot button, and you'll see your Tag change to 'true' in the Tag Browser. You'll also see the button change to "Writing..."
- In the **Property Editor**, enter a value of **1** directly into the **Value** property. The button will display "**Writing....**". Set your Tag to '**false**' which resets the Value property back to the ready state.
- Once you save and publish your project, login to the **Client**, and press the One-Shot Button. The button will display "**Writing...**". It will not change back until the PLC resets the Value property to the ready state, or you manually set the your Tag to 'false.'



Momentary Button

Momentary Button

The **Momentary Button** is a controls component that sets the value in the PLC for specified number of seconds, or however long the button remains held down, whichever is longer. It uses several properties such as the Control Value and Indicator Value properties, in addition to the On Value and Off Value properties that define two different states. The On Value and Off Value properties are set to 0 and 1, respectively.

The Momentary Button uses a Min Hold Time property that resets the value after a minimum number of seconds which you can specify. The default Min Hold Time is 1 second and the Max Hold Time is 0, but you can set both properties to whatever you want. When the Momentary Button is pressed, it writes the On Value to the Control Value and Indicator Value properties. (The Indicator Value is used to determine which state the machine is currently in). When released, it will either write the Off Value to the Control Value immediately, or wait until the Min Hold Time has elapsed.

Note that the Momentary Button does **not** force the value or continually write a value during the on time. It only writes the on value once, then writes the off value some time later.

LATENCY

If you are using the Momentary Button to start a motor or move a piece of equipment, make sure to account for network latency.

Momentary Button Example

Drag a **Momentary Button** on to your window, and drop a Tag on to the component. In the Property Editor, you'll notice that the Tag binds to the Control Value and the Indicator Value properties (in bold). The Control Value is bound bidirectionally to the Tag. (In this example, a WriteableBoolean1 Tag was selected from our simulator and it bound the Control Value property bidirectionally).

In the **Property Editor**, check the component's properties to see if you want to enable, disable, or change any default values or styles for any of the properties.

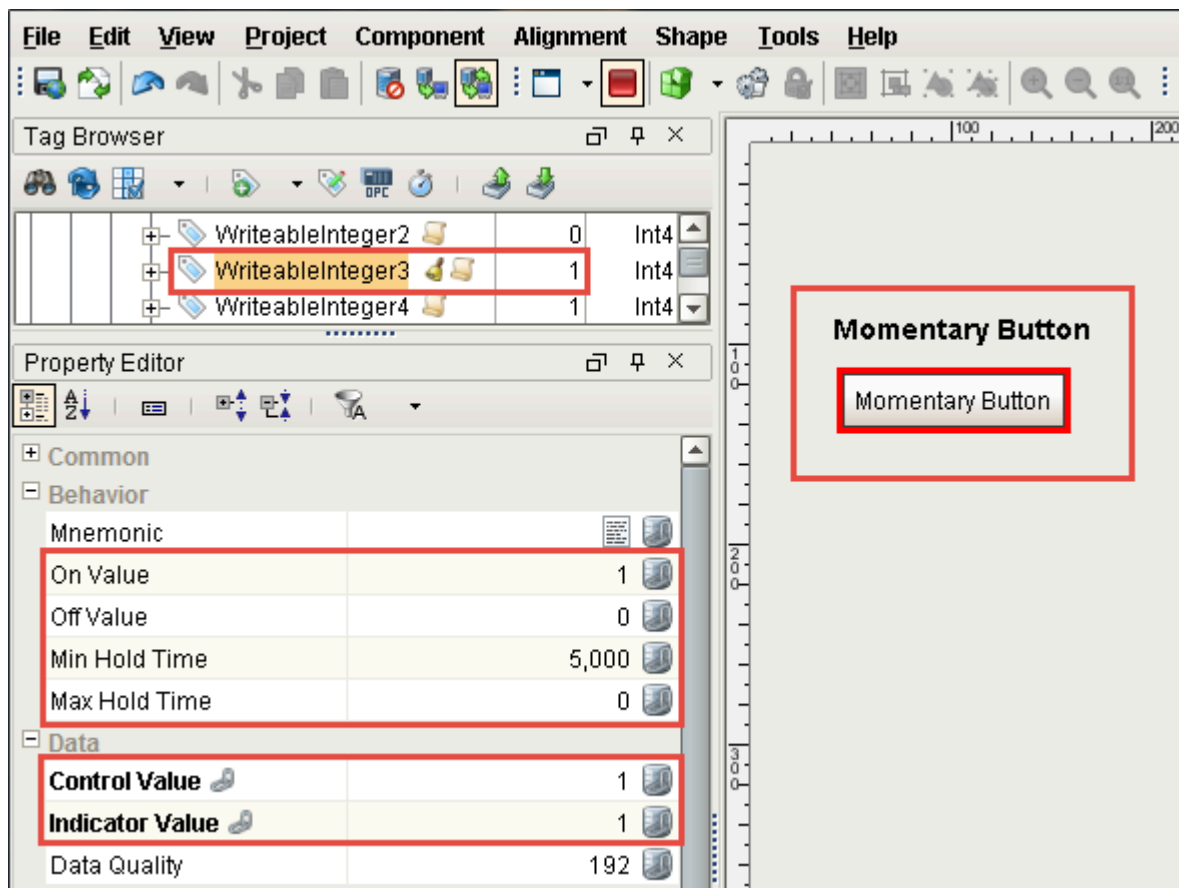
Let's make a few changes.

- Use the default settings for the **On Value** and **Off Value** properties of **1** and **0**, respectively.
- Set the **Min Hold Time** for **5 seconds** and use the default **Max Hold Time** of **0**.

Test it out!

Make sure your **Gateway Communication Mode** is set to read/write. You can change the value in two ways:

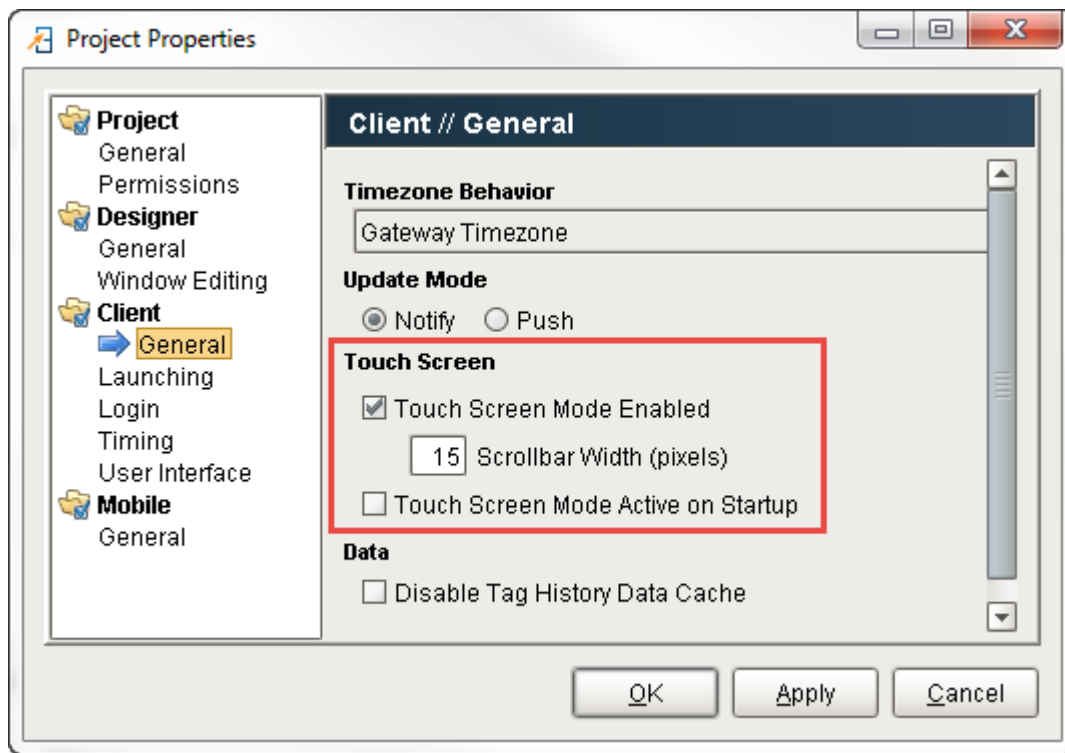
- In **Preview Mode** - press and release the Momentary Button. The button will become highlighted, and you'll see the value change in the Tag Browser next to your Tag. In the Property Editor, you will see the Control Value and Indicator Values properties get set to the On Value (i.e., 1) for the specified Min Hold Time (i.e, 5 seconds). After 5 seconds, it writes the Off Value to the the Control Value (i.e., 0).
- Once you saved and published your project, login to your **Client**, press your Momentary Button. The button will become highlighted and remain highlighted for the Min Hold Time (i.e., 5 seconds). You can also hold down the button instead of releasing the button immediately. However long the button remains held down, it sets the value in the PLC. If it is less than 5 seconds, it will wait the until the Min Hold Time is elapsed.



Touch Screen Mode

Ignition has a built-in **Touch Screen Mode** that allows access to Input components on computers that don't have a keyboard or mouse. This means that whenever Touch Screen Mode is turned on, a keyboard will be displayed when a user touches an Input component that expects a value to be entered such as a Text Field, Numeric Text Field, Spinner, etc... You can find the Touch Screen Mode under the Property Editor for most Input components. It has three settings so you can select from to open the Touch Screen keyboard: single-click, double-click, or disable it all together. Depending on the data type of a component, one of three keyboards will automatically be opened: alpha-numeric, numeric, or password.

To enable Touch Screen Mode, go to the menubar and select **Project Properties > Client > General**, and select **Touch Screen Mode Enabled**.



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

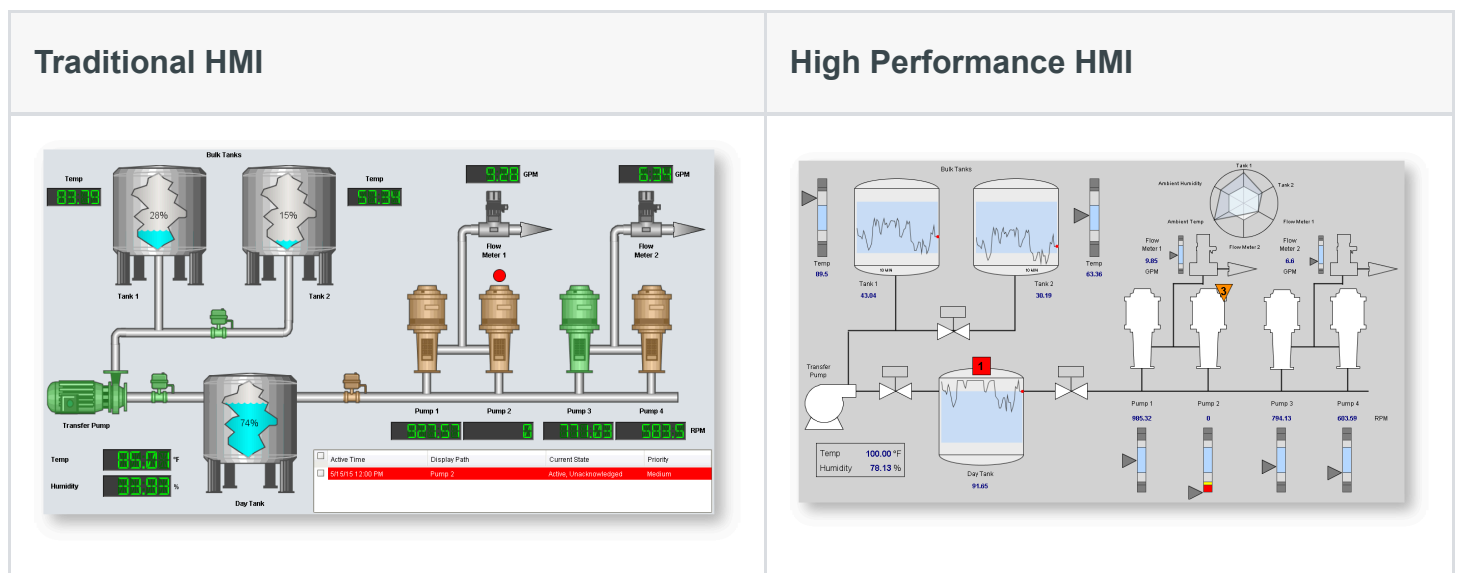
For documentation on the latest stable release, see the [latest version](#) (8.3).

High Performance HMI Techniques

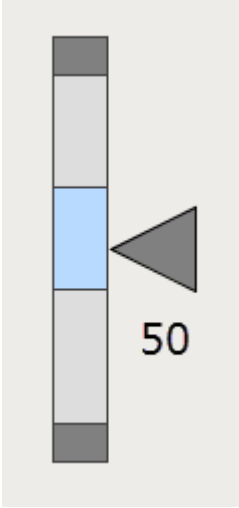
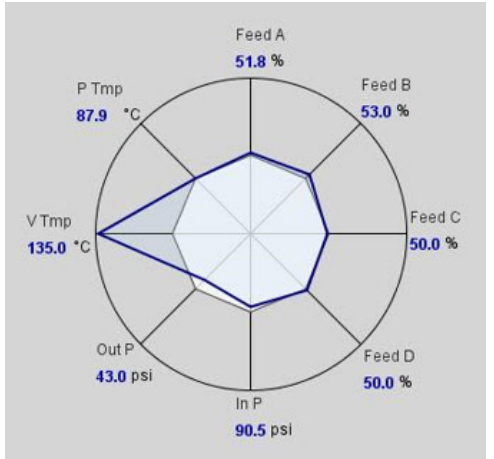
About High Performance HMI Techniques

High performance HMI techniques and practices call for designs and displays which help the viewer make the best decision in the shortest amount of time after interacting with the HMI.

High performance HMIs often look basic and simplistic. They typically use gray-scale colors rather than the traditional graphics and bright colors for their displays. Conceptually, the High Performance HMI operates under the idea of visually contrasting critical and non-critical states. The power of this design philosophy is when something does go wrong, a high performance HMI will quickly guide the user to the source of the problem.



In Ignition, you create the high performance HMIs by using components such as [moving analog indicators](#), [sparkline charts](#), and [radar charts](#). Here is a comparison of a traditional HMI next to a high performance HMI.

Moving Analog Indicator	Sparkline Chart	Radar Chart
		

Use of Color



INDUCTIVE
UNIVERSITY

Use of Color

[Watch the video](#)

Colors are an important consideration when designing a high performance HMI. Gray-scale colors are used instead of the traditional bright red, green, and blue colors.

The person that designs the HMI must understand the diverse audience that may view the HMI. For example, an operator may look at a motor on a traditional HMI. If it the HMI has colored the motor green, the operator may conclude that the motor is running. However, if a maintenance technician looks at the same motor, he may conclude that the motor is not faulted. These are logical conclusions. They reflect the respective interest of the person viewing the HMI where the operator wants the motor to run and the maintenance technician wants it to be working.

In reality, the motor is simply "scheduled" to run, meaning when it runs depends on its control mechanism. For example, the motor may run when there is product on the line or boxes on the conveyor. In other words, the motor may be periodically starting and stopping automatically.

A high performance HMI can eliminate this confusion by introducing a color that signifies a state of "scheduled." A common high performance HMI practice is to use a dark gray to signify a "scheduled" state for equipment. This color should never compete with more alerting colors that "pop" from the

HMI resulting in the viewer's eye's being drawn to the portion of the HMI where the problem may be occurring.

Colors and Alarm Indicators

Color can play an important role in how an operator responds when problems do occur. High performance HMI design refrains from coloring equipment when the equipment is in a state of fault. For example, some equipment may still run when faulted. Instead, an optimized solution is to place an alarm indicator near the equipment in such a way that when the equipment is undergoing some fault, the alarm indicator renders with the appropriate color and shape. The object and the consequential color should signify the most important alarm state occurring for the equipment at that current time.

Technical Considerations with Colors

Some HMIs in industrial settings may temporarily lose their ability to render color because of various environmental factors. High performance HMI design incorporates this possibility by encouraging the use of descriptive text with color. For example, motors of two different colors may look the same on a color deficient HMI resulting in confusion for the viewer. Even worse, the viewer may misinterpret the motor state and assume everything is fine. However, if each motor has a descriptive text such as "Motor 1 is Faulted" and "Motor 2 is Running", the problem associated with a faulty HMI failing to display color is largely reduced by the HMI's high performance design.

Accommodating Color Blind Viewers

Common color combinations such as red and green and blue and purple cannot adequately be distinguished by those with color blindness. High performance HMI design accommodates color blind users by combining colors with descriptive text as well as incorporating alarm indicators in unique shapes.

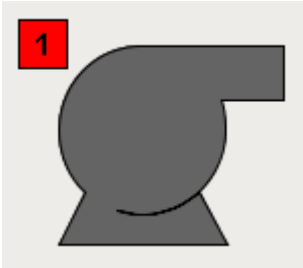
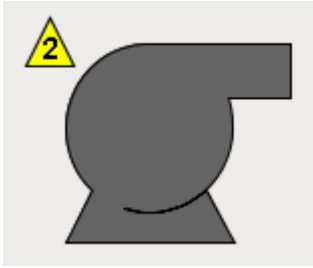
Alarm Indicator

The high performance HMI design techniques make use of an object called the **Alarm Indicator** which displays a colored shape when there is a problem. This works well with the high performance HMIs as color is only used when there is a problem. The Alarm Indicator can contain descriptive text in addition to the shape and color, and is usually placed near the component that is causing the problem. You can import the Alarm Indicator, shown in the examples below.

The Alarm Indicator represents different levels of alarm with different shapes, color, and descriptive text. For example, a motor that exists in an industrial setting is monitored by a high performance HMI. There are two alarms on the motor. The first is a critical alarm associated with the motor becoming

seized, will display as a red rectangle with the letter 1. The second is a high priority alarm associated with the motor when overheating, will display as a yellow triangle with the number 2. An Alarm Indicator is placed near the motor, positioned in such a way to clearly show motor that the indicator is referring to.

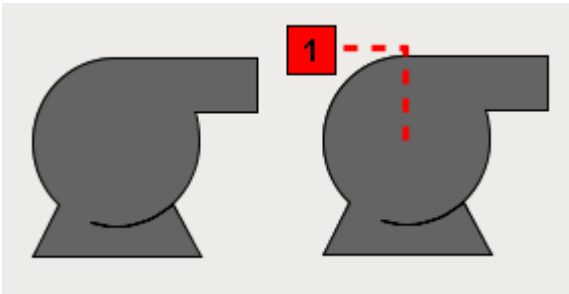
Given this scenario, a high performance HMI will show the Alarm Indicator as in the following examples:

Example 1	Example 2
The motor is critically faulted. This is the highest priority. 	Motor is overheating, resulting in a high alarm. 

Reducing Ambiguity

A high performance HMI design technique to reduce ambiguity incorporates a line from the Alarm Indicator to the object experiencing the alarm.

The following example shows how a simple dotted line can ensure that the viewer associates the correct Alarm Indicator with the correct motor.



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

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

Property Binding Types

Property 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 do useful things. A property binding simply links one component's property to another on the same window.

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 property binding adds.

Property binding, as its name suggests, lets you bind a property to something else, and that something else could be:

- A Tag
- The results of a SQL query executed against a remote database
- Some other component's property
- An expression involving any of these things
- The results of a Python script
- and so on...

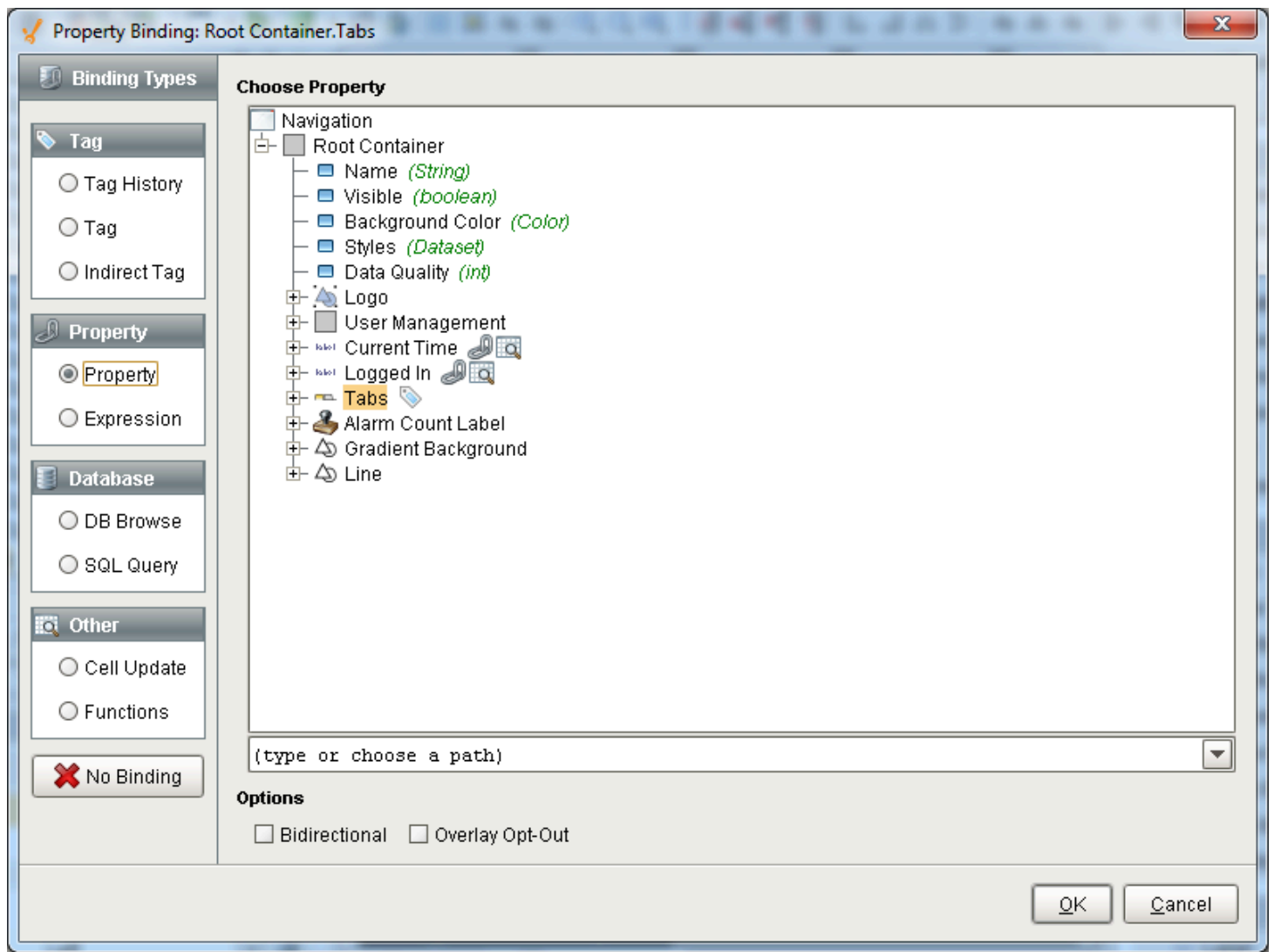
For example, bind the Value property of an LED Display to an OPC Tag, and voilà - the value property will always be the value of that Tag - creating a dynamic display. Bindings can also work the other way, using a bidirectional binding. Bind the value of a numeric text box to a Tag, and that Tag will be written to when someone edits the value in the text box.

The power of property 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 (`glyphForeground`) color to a Tag's `AlertCurrentSeverity` 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, quite literally, endless.

Property Binding Types

A property can have many different types of bindings, for example it can have a Tag or an Expression binding. Instead of setting a label statically, the text might change based on a PLC value or on-screen selection. There 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:

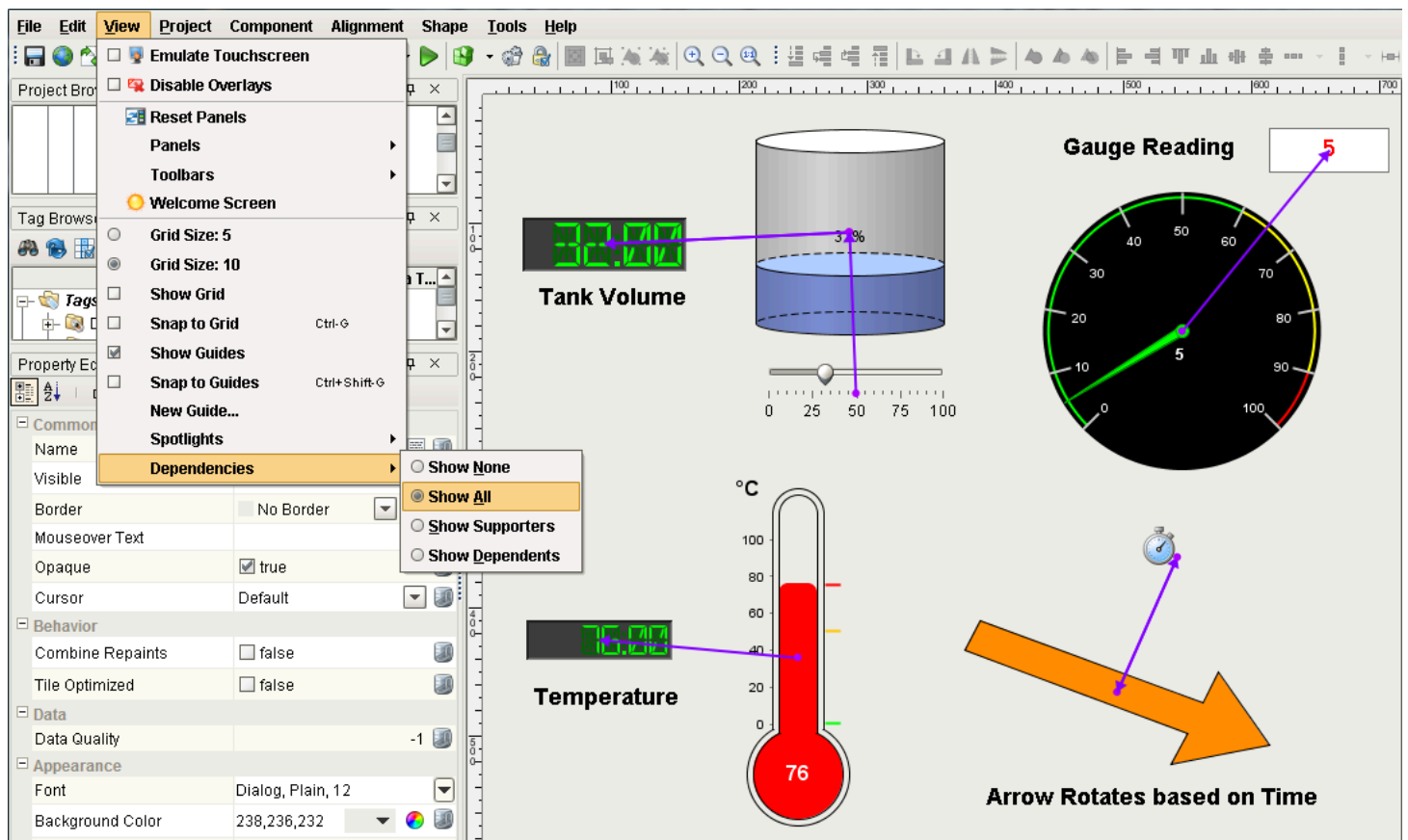
- **Property** - simply binds one property to another. When that property changes, the new value is pushed into the property that the binding is setup on.
- **Tag** - binds a property directly to a Tag which sets up a Tag subscription for that Tag, and every time the chosen property changes, the binding is evaluated, and pushes the new value into the inbound property.
- **Indirect Tag** - 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** - used for Dataset type properties. It runs a query against the Tag Historian.
- **Expression** - is the most powerful type of property binding. It uses simple expression language to calculate a value which can involve lots of dynamic data.
- **Database Browse** - is equivalent to the SQL Query binding except that it helps write the queries for you.
- **SQL Query** - is a polling binding type that runs a SQL Query against any of the database connections configured in the your Gateway.
- **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.
- **New in 7.9.4** **Named Query** - similar to a SQL Query binding, except it links to a previously created Named Query.



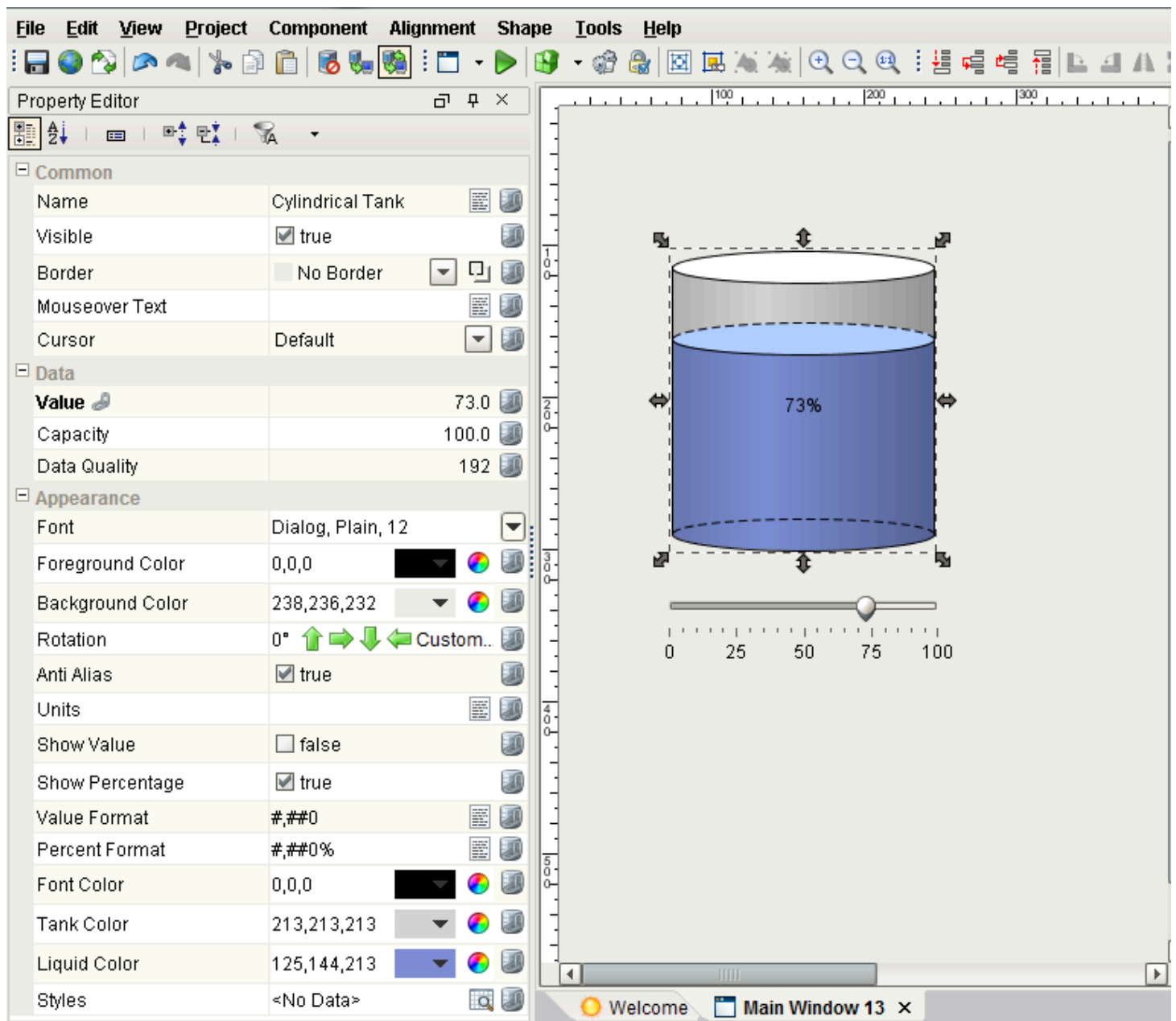
Basic Bindings

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 **property bindings** on the component. It's the Property binding that brings your components to life and have them do useful things. **Components** can be configured to do just about anything using property bindings.

In this screenshot, property bindings were set to make these random components do something. In the Designer, you can quickly view dependencies to determine what is linked to what. As shown below, a line is drawn from the Slider to the Tank letting you know the Tank is bound to the Slider. You can graphically see what components are bound to each other by using the **View > Dependencies > Show All** command.



The simplest of all the binding types is the **Property** binding type. You can link one property on the screen to another property in the same window using the bind button next to the property. When the property you've bound to changes, the new value is pushed into the property that the binding is setup on. For example, say you want a Tank to reflect the value of a Slider component. You can bind the '**Value**' property of the Tank to the '**Value**' property of the Slider. So each time the Slider value changes, the Tank will update to the same value.



You can click on the bind button to link to a Tag directly or indirectly, to another property in the same screen, to an expression or calculation, or to a SQL query. For example, you can place a Label on a window and link it to an Expression to make it blink red and yellow when an alarm is active. You can bind an LED Display to a Tag so the value property will always be the value of that Tag, thus, creating a dynamic display. You can even bind properties to Tags inside UDTs or to a whole UDT.

How Bindings Work: Event-based vs. Polling

While there are quite a few different binding types, they all boil down 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, updates the tank's value. The following bindings are event-based:

- Tag bindings
- Property 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:

- SQL query bindings
- Some expression functions, like `runScript()` or `now()`

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 red code at the bottom is a property binding inside of the query binding. Every time this (event-based) binding fires, the query will run again. Using bindings like this, you can create highly dynamic and interactive screens with no scripting whatsoever.

Polling Options



INDUCTIVE
UNIVERSITY

SQL Query Binding - Polling

[Watch the video](#)

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.

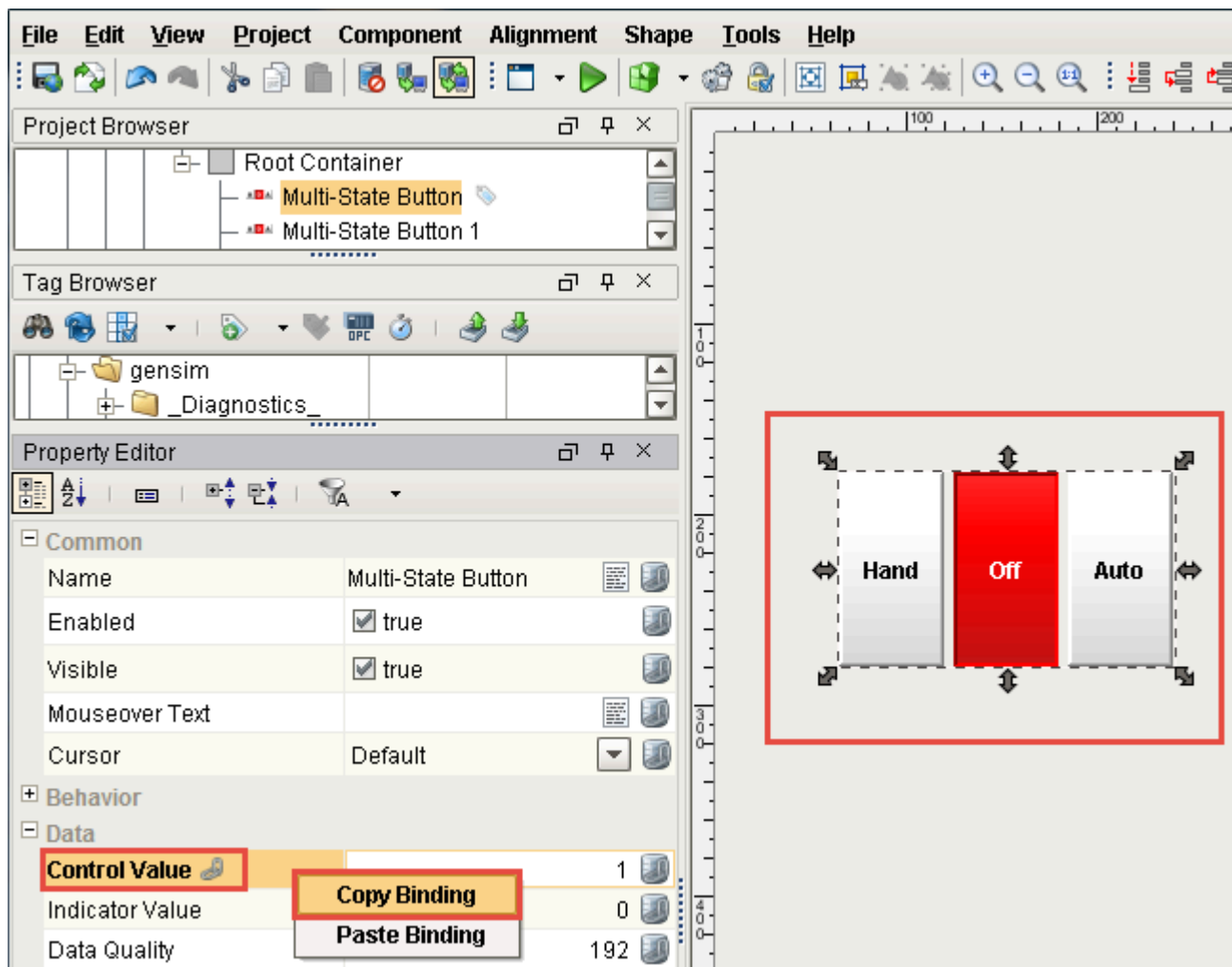
Additionally, all three types will **always** update if the binding contains a reference to something else, such as a Tag or property value (noted with the brace-notation ""), and the value of that reference changes. Typically this is seen in SQL query bindings: polling can be turned off, and the query 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. 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. (i.e., A binding that resolves to a string will not work on an integer property, etc.).

Here's how to copy a binding property:

1. In the Designer, click on a **component**.
2. Select the **bound property** and right-click.
3. Choose **Copy Binding**.
4. Select another property on the same component or any property on another component.
5. Right-click and select **Paste Binding** to apply the binding to the new property.



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

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

Property Binding



INDUCTIVE
UNIVERSITY

Property Binding

[Watch the video](#)

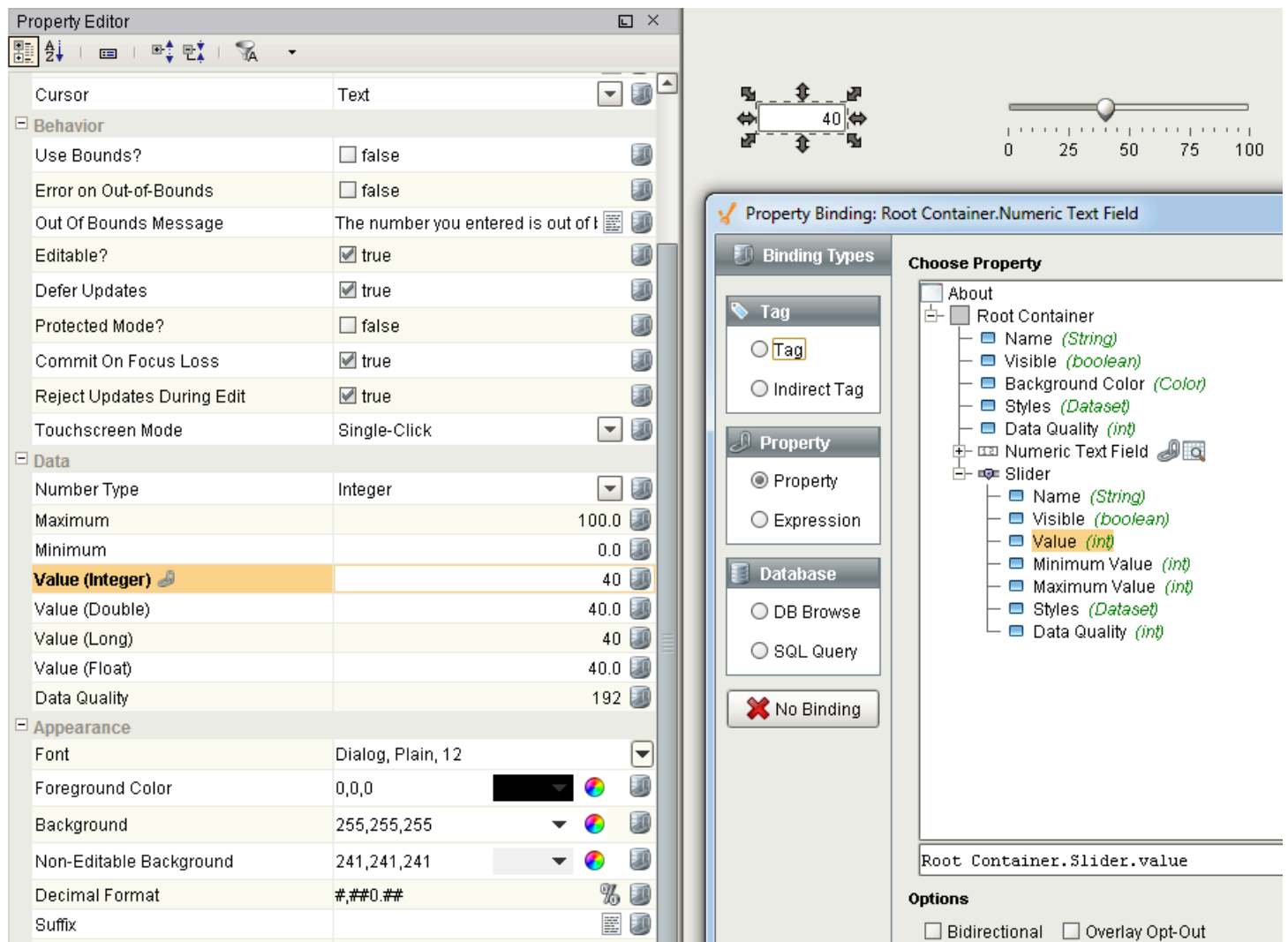
Property to Property Binding

A property binding is a very simple type of binding. It simply 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.

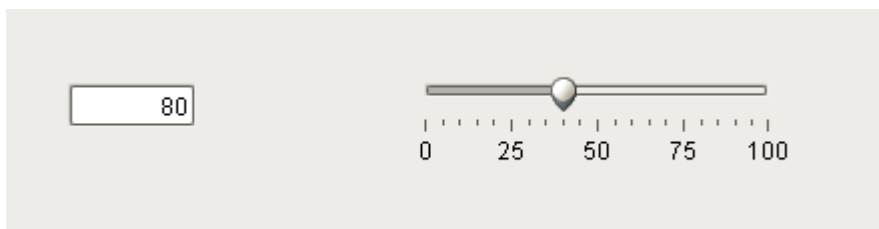
WHY AREN'T ALL PROPERTIES LISTED?

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 for which a **propertyChangeEvent** is fired may be bound to.

For example, I have bound the Value (Integer) property of a Numeric Text Field to the value property of a slider.



Notice how changing the value of the slider, will then adjust the value of the numeric text field as well. The Numeric Text Field is changing to match the value set by the slider. 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

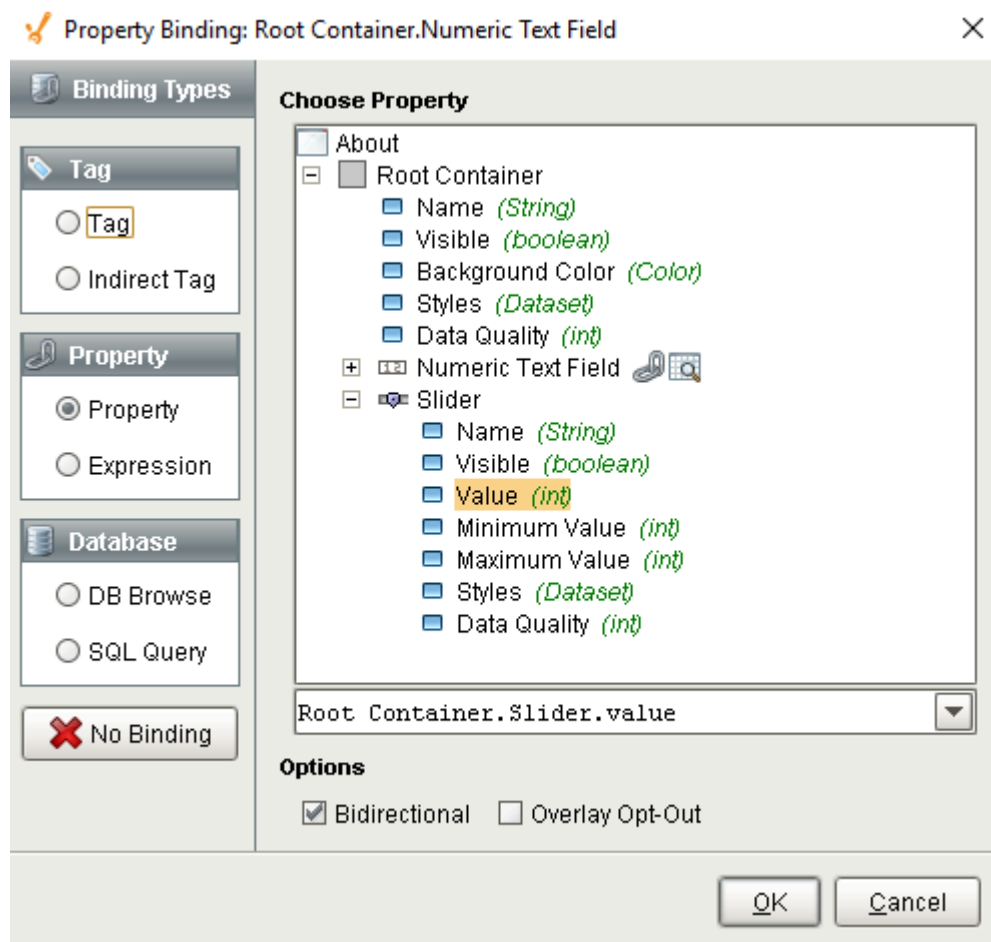


INDUCTIVE
UNIVERSITY

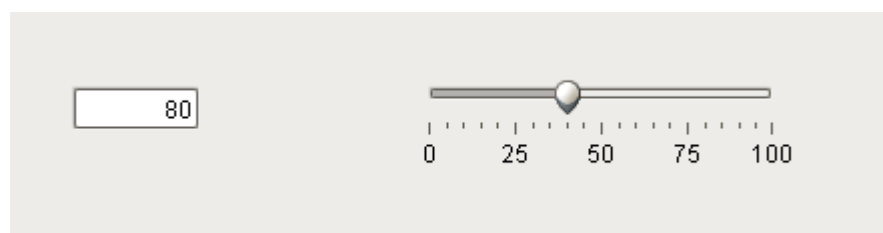
Property Binding - Bidirectional

[Watch the video](#)

Certain 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 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.



If we check, and save the binding, it will now be a bidirectional binding. Notice now how changing the value of the Numeric Text Field will also update the Slider component.



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

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

Tag Binding



INDUCTIVE
UNIVERSITY

Tag Binding

[Watch the video](#)

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



INDUCTIVE
UNIVERSITY

Tag Binding – Drag and Drop

[Watch the video](#)

You can quickly bind a Tag by dragging it from the Tag Browser into a component or into the component property. In the Property Editor, the properties that have the new bindings will display in bold font along with a binding icon.

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 . 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 datatypes 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 on to 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.

Bidirectional



INDUCTIVE
UNIVERSITY

Tag Binding – Bidirectional

[Watch the video](#)

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.

Bindings and Tag Properties

Aside from binding a property to a Tag's value, you can also bind to properties on a Tag, such as **Tooltip** 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 two boolean Memory Tags. Each Tag has an alarm that will go active when the value on the Tag is high.

Tag	Value	Data Type
Tags		
Data Types		
Alarm_Active_1	☐	Boolean
Alarm_Active_2	☐	Boolean
Value	false	
AlarmActiveAckCount	0	Integer
AlarmActiveUnackCount	0	Integer
AlarmClearUnackCount	0	Integer
AlarmHighestAckName	null	String
AlarmHighestAckPriority	-1	Integer
AlarmHighestUnackName	null	String
AlarmHighestUnackPriority	-1	Integer

The simplest approach involves a **Tag Binding**. This can be achieved by simply **dragging-and-dropping** the Tag Property onto a Window, component, or component property: 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 below the **Choose Tag** tree:

Property Binding: Root Container.Numeric Label

Binding Types

- ☒ Tag
- ☐ Indirect Tag

Property

- ☐ Expression

Choose Tag

- Tags
 - Data Types
 - Alarm_Active_1
 - Value
 - AlarmActiveAckCount
 - AlarmActiveUnackCount
 - AlarmClearUnackCount

Alarm_Active_1.AlarmActiveAckCount

Options

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

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

Indirect Tag Binding



INDUCTIVE
UNIVERSITY

Indirect Tag Binding

[Watch the video](#)

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. Indirect Tag bindings can use bidirectional mode just like standard Tag bindings. Making bindings indirect is an important part of the binding system. Indirect Tag, Expression, and SQL Query bindings can all be made indirect. All this means is that what the binding is linking to can be changed based on the value of something else.

For example, instead of binding straight to a Tag's path, like

```
[TagProvider]MyPlant/EastArea/Valves/Valve4/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}Area/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 Indirect Tag Bindings

Indirect Tag bindings are an important part of Ignition and the binding system. This means that what the binding is bound to, can be changed based on the value of something else. For example, you can have a selection from a drop-down list to choose which Tags to look at on the screen.

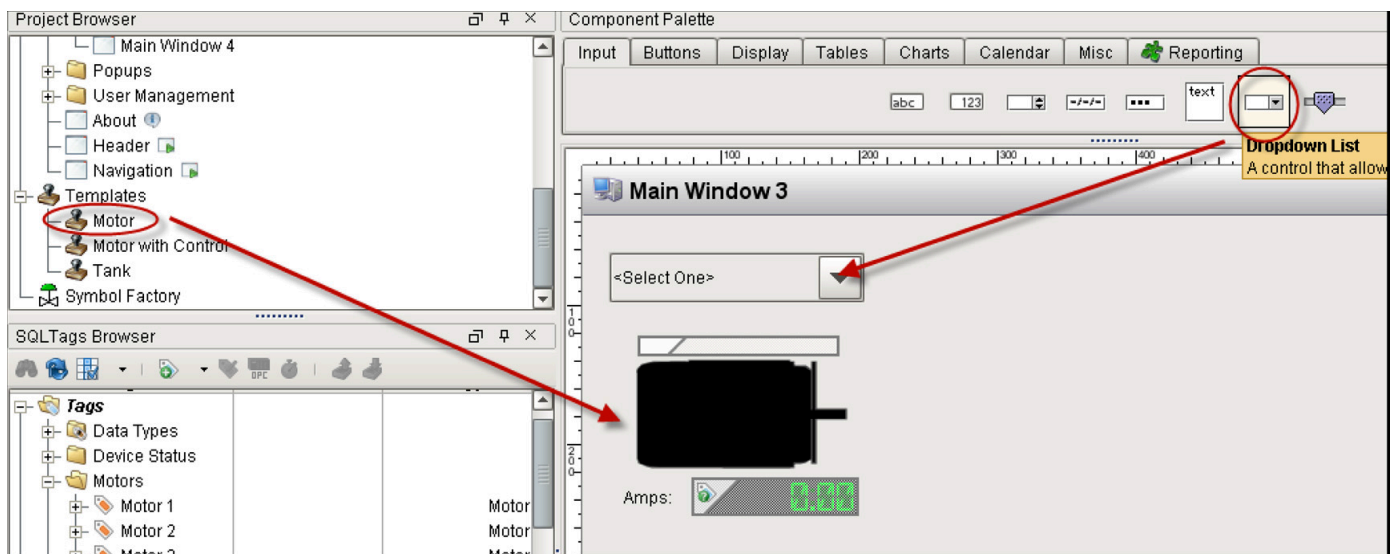
We have four motor UDT Tags and a motor template. Let's create a single window with one drop-down list and one motor template. The drop-down list gives the option of which motor to look at. The motor Tags look like this:

Motors/Motor 1 Motors/Motor 2 Motors/Motor 3 Motors/Motor 4

The only difference between each motor is the number.

To Create a Window with a Drop-down list and a Motor Template

1. From the Project Browser, under **Templates**, drag in a template. In this example, it is a **Motor** template.
2. From the **Input** tab of the **Component Palette**, drag a **Dropdown List** component.



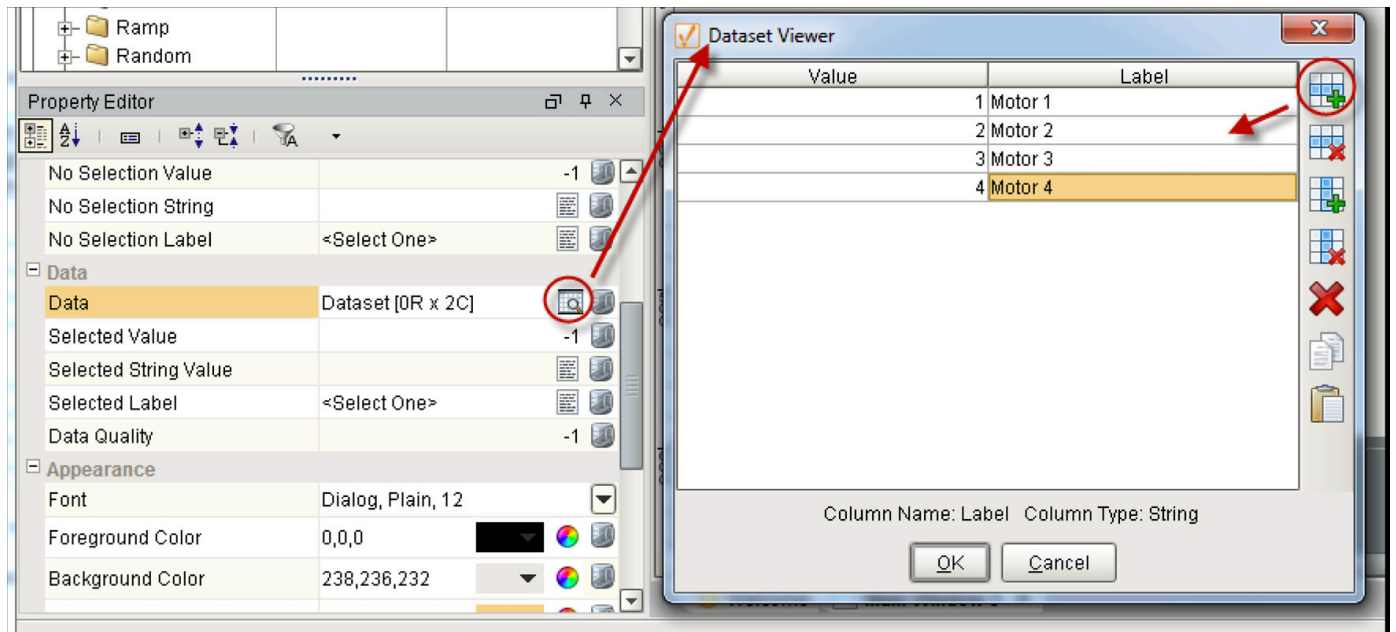
Now we need to setup the options for the drop-down list in the **Property Editor**. The drop-down component has the **Data** property which is a dataset. There are three ways to set up the dataset and the corresponding behavior is as follows:

Column Details	Example	Description
One Column [String]	Apples Oranges Bananas	- Drop-down displays values from the first column. - Selected Value is undefined. - Selected String Value represents value from first column. - Selected Label represents value from the first column.
Two Column [Integer, String] (Default)	201 Apples 202 Oranges 203 Bananas	- Drop-down displays values from second column. - Selected Value represents value from first column. - Selected String Value represents value from second column. - Selected Label represents value from second column.
Two Column [String, String]	APL Apples ORN Oranges BAN Bananas	- Drop-down displays values from second column. - Selected Value is undefined. - Selected String Value represents value from first column. - Selected Label represents value from second column.

Let's use the default mode which is the second option above. **Two Columns [integer, string]**, to set up the options in the drop-down list.

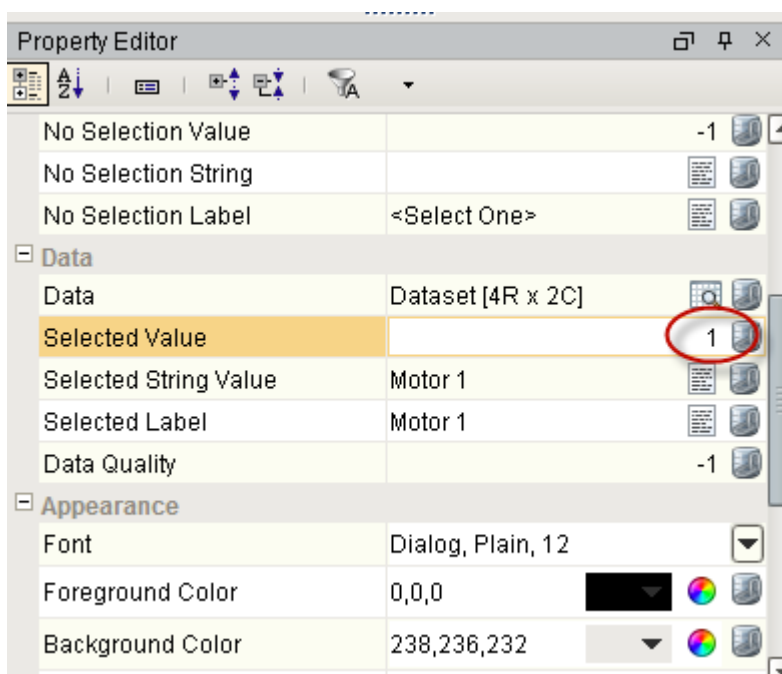
3. Select the drop-down component and click the **Dataset view** icon next to the **Data** property.
4. On the right side of **Dataset Viewer** window, click the + **green row** icon four times to add 4 rows.
5. Set the rows to the following values:

Row Number	Value
1	Motor 1
2	Motor 2
3	Motor 3
4	Motor 4



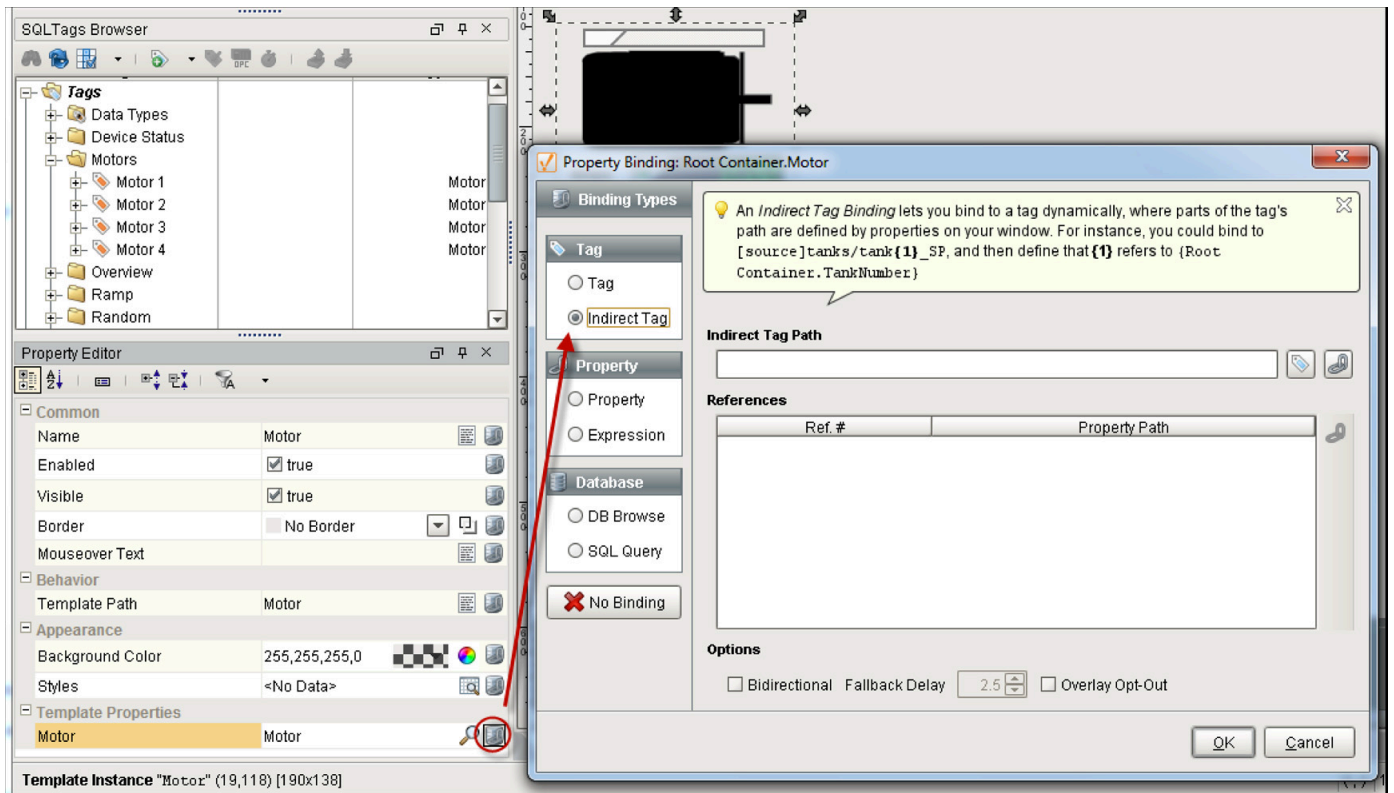
6. Click **OK** to save.

7. Set the **Selected Value** property to **1**. Notice the values for **Selected String Value** and **Selected Label** fill in for you.



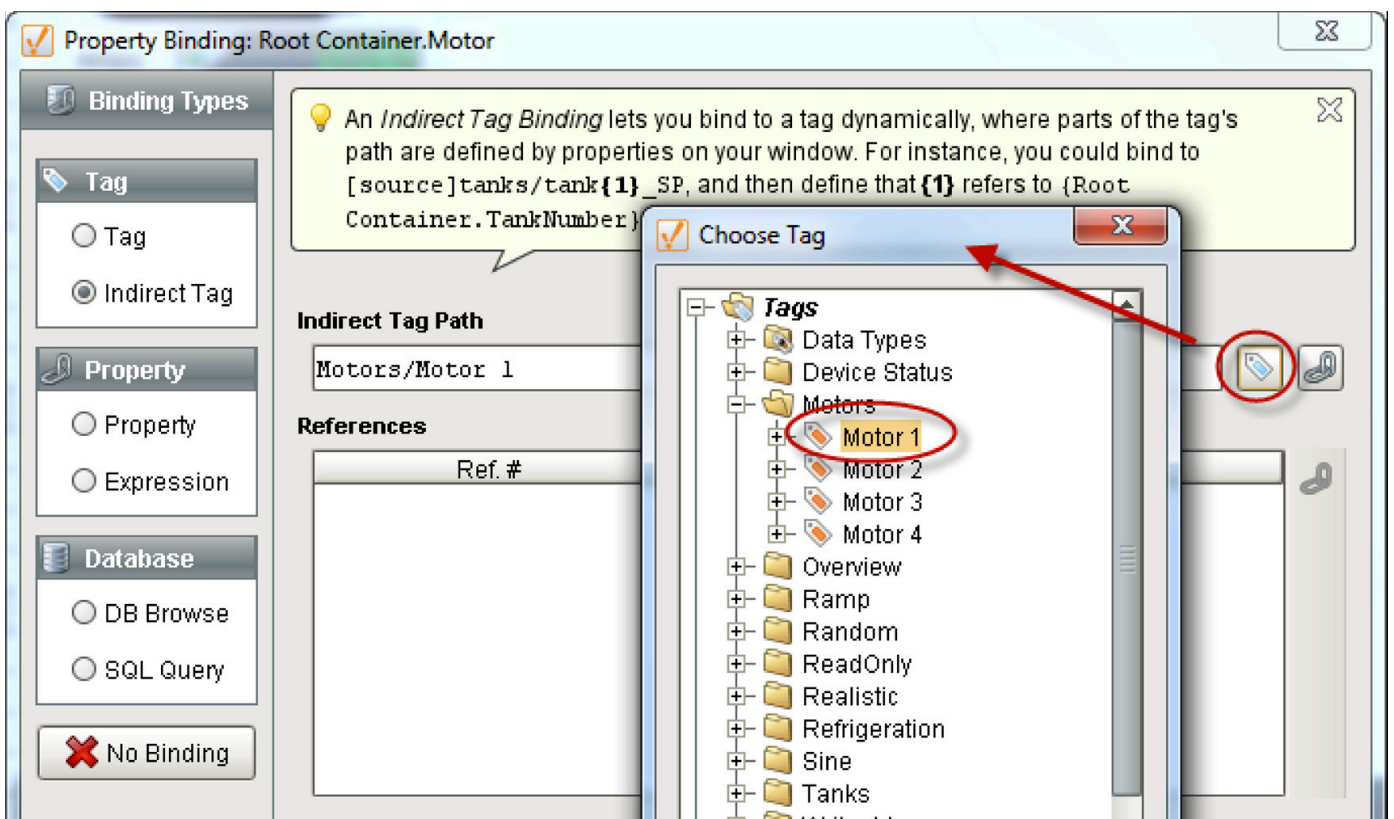
We want to use the Selected Value in an indirect binding on the template to dynamically select which motor to look at.

8. Select the Motor template on the window and bind the Motor UDT property to an Indirect Tag.



The easiest way to use the indirect Tag binding, is to select a Tag as a starting point and then make it dynamic.

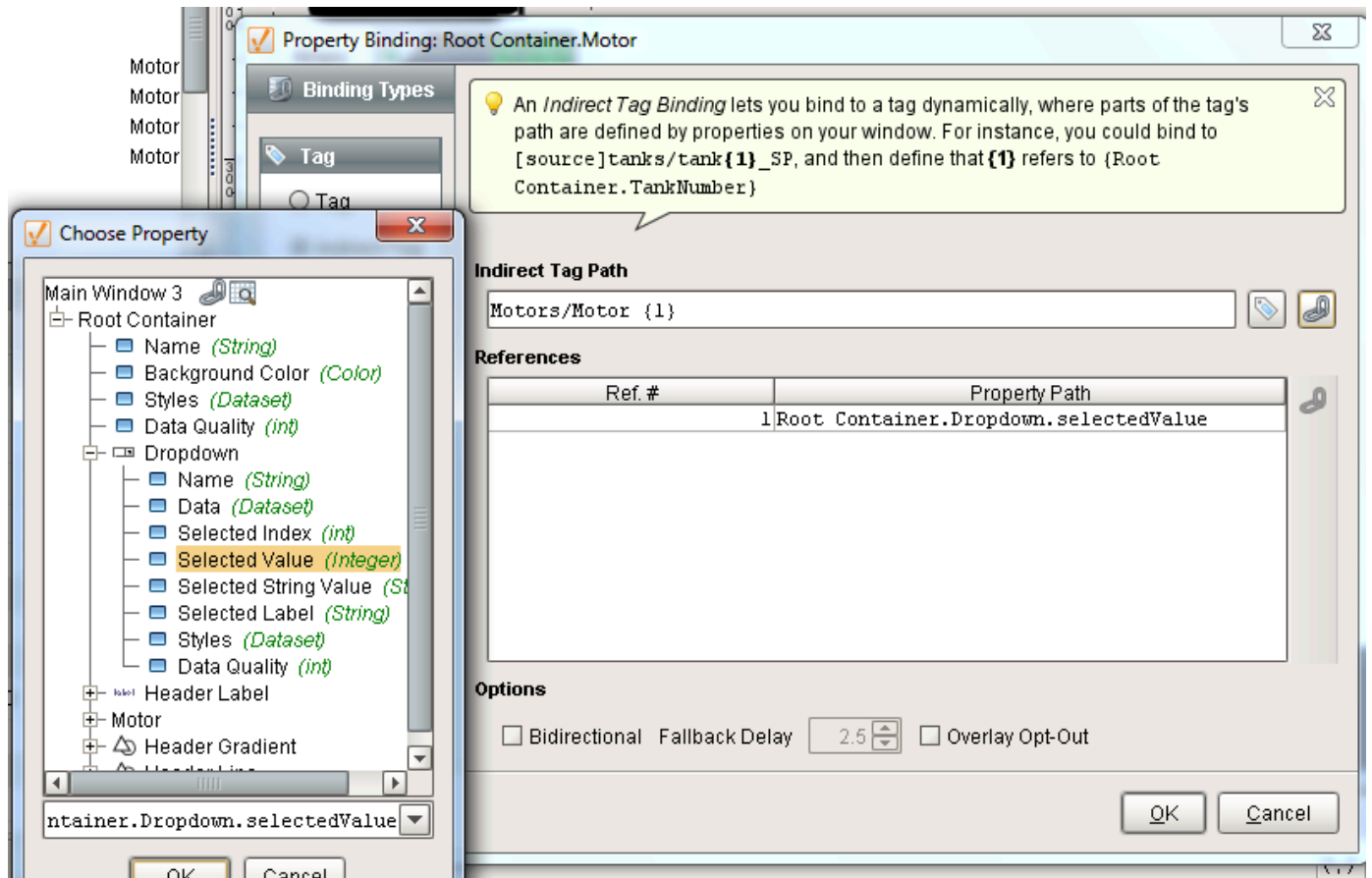
9. Click the Tag button on the right of Indirect Tag Path and select the Motor 1 Tag as the reference.



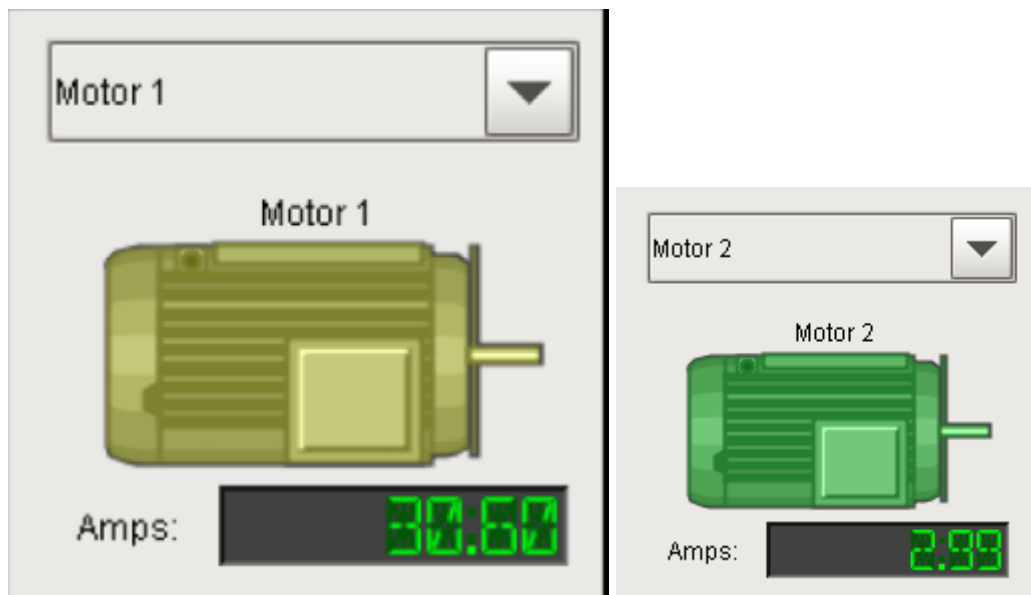
The drop-down list contains which motor number to look at. So we need to link the Motor Tag number to the drop-down's selected value.

10. Highlight the number 1 (in Motors/Motor 1) in the Indirect Tag Path, then click on the link button to the right to bring in the drop-down's Selected Value property.

Ignition replaces the 1 with a 1 and links it to the drop-down. That means Ignition substitutes 1 with the value of the drop-down allowing it to select which motor Tag to look at.



11. Click **OK**. Now you can try it out.
12. Save your project and open the runtime. You can change the drop-down list and it will select a different motor.



It is as easy as that to setup Tag indirection. You can apply this method to popups or anywhere in Ignition.

See also: [Parameterized Popup Windows](#)

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

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

Tag History Binding



INDUCTIVE
UNIVERSITY

Tag Historian Binding

[Watch the video](#)

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 to query. The Dataset returned by the query will have a timestamp column, and then a column for each path that you select here.

These paths may use indirection following the same rules as the Indirect Tag Binding. 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.

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.

Sample Size and Aggregation Mode

NOTE

For complete information on the sample sizes and aggregation modes, see the [Sample Size and Aggregation Mode](#) page.

The sample size determines how the query results will look.

- **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.
- **On Change/Raw** - 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. This will always return a row from every changed column, even if the timestamps are identical.
- **Fixed** - A Fixed query will return the given number of rows. Where data was sparse, interpolated values will be added. Where data is dense, the Aggregation Mode will come into play.
- **Interval** - Similar to fixed, but with the spacing based on time, rather than the number of requested results.

The aggregation mode dictates what happens when multiple raw values must be processed into a single value. Here is a description of the most commonly used options.

- **Min/Max** - Both the minimum and maximum values will be returned for each timestamp.
- **Time-weighted Average** - The values are averaged together, weighted for the amount of time they cover in the interval.
- **Closest Value** - The value closest to the ending time of the interval will be returned.
- **Simple Average** - The values are summed together and divided by the number of values.

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.

NOTE

Tags Historian information is often easiest to work with in the [Easy Chart](#) component, which handles all of these options automatically.

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

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

Expression Binding



INDUCTIVE
UNIVERSITY

Expression Binding

[Watch the video](#)

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. You can use expressions for many different purposes. Anytime information needs to be massaged, manipulated, extracted, combined, split, and so on - think expressions!

For a complete list of the Expression Functions, see the [Expression Functions](#) page in the Appendix.

Example 1

You have a button that starts a batch, but you only want to let it be pressed after the operator has entered a scale weight. Bind the button's enabled property to:

```
{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, use one of these two expressions (they're equivalent)

```
if ({CurrentProcessState} = 0, "Not Running",  
if ({CurrentProcessState} = 1, "Warmup phase - please wait",  
if ({CurrentProcessState} = 2, "Running", "UNKNOWN STATE")))
```

```
switch ({CurrentProcessState},  
0,1,2,  
"Not Running",  
"Warmup phase - please wait",  
"Running",  
"UNKNOWN STATE")
```

See also [Expression Overview and Syntax](#).

Expression Function Examples

Concat Strings



INDUCTIVE
UNIVERSITY

**Expression Binding - Concat
Strings**

[Watch the video](#)

You can use an expression binding to concatenate strings resulting in a new string that reflects the concatenation of different strings.

The function starts with `CONCAT()` and inside the function is a list of strings. They can be manually typed like "42" or they can come from Tags or properties.

Example 1

```
concat("The answer is: ", "42") //returns "The answer is: 42"
```

Example 2

You have a Date, and need to extract the year, and concatenate the word "Vintage" to the end for a label display. Bind a label's text property to:

```
dateExtract({Root Container.VintageDate}, 'year') + ' Vintage'
```

Celsius to Fahrenheit



INDUCTIVE
UNIVERSITY

Expression Binding - Celsius to Fahrenheit

[Watch the video](#)

Use an expression binding to convert a temperature from Celsius to Fahrenheit. This is an example of how expression binding can handle calculations.

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

The reference to the Celsius temperature can come from a property or a tag. As the property or the tag changes, so does the expression binding.

Format Date



INDUCTIVE
UNIVERSITY

Expression Binding – Format Date

[Watch the video](#)

You can format a date in an expression binding by using the `dateFormat` and `now` functions.

To make a label that updates to show the current time:

1. Drag a **Label** component onto the window.
2. Select the label's **Text** property binding icon and select **Expression** binding.
3. Enter the following code into the expression, and click **OK**.

```
dateFormat(now(1000), "MMM d, yyyy hh:mm:s a")
```


The `dateFormat` function takes two arguments. The first argument is any date type variable. This can include another function (like `now`) that returns a date type. The second argument refers to the date format that you want returned. The `now` function returns the current time and in this case it will update every second.

For more information on the date formatting expression, see the [Appendix](#).

Date Manipulations



INDUCTIVE
UNIVERSITY

Expression Binding – Date Manipulations

[Watch the video](#)

You can manipulate dates in expression bindings such as a date addition/subtraction with the `dateArithmetic` function. This is important when you want to use the expression bindings to select a date that is offset by a certain amount.

The following example will return the time 15 minutes ago by using the `dateArithmetic` expression function:

```
dateArithmetic(now(), -15, "minute")
```

Bit Functions



INDUCTIVE
UNIVERSITY

Expression Binding – Bit Functions

[Watch the video](#)

You can use various bit functions in expression bindings like `getBit` to return individual bits of a word.

Example 1

Assuming a tag path 'Folder/BitTag', the following would return the binary representation of the tag's value at the 0 position

```
getBit({Folder/BitTag}, 0)
```

For more details on the getBit function please see the [getBit](#) page in the Appendix.

Example 2

You have 3 bits in a PLC, only one of which will be on at a time. You want to turn these three bits into a single integer (0,1,2) to drive a component's Styles. Bind a custom integer property to:

```
binEnum({MyTags/Bit1}, {MyTags/Bit2}, {MyTags/Bit3})
```

Switch



INDUCTIVE
UNIVERSITY

Expression Binding – Switch

[Watch the video](#)

You can use the switch function in expression bindings to evaluate conditional statements. This function acts like the switch statement in C-like programming languages. It takes the value argument and compares it to each of the case1 through caseN expressions.

The following example returns the string "Running" when it is given the value of 1. Its options are 0, 1, and 2. And when comparing the value to the options the switch statement returns one of the corresponding results. If a result cannot be found, a fail-over option is returned.

```
switch(  
  1,  
  0, 1, 2,  
  "Off", "Running", "Fault",  
  forceQuality("!BAD STATE!", 0))
```

Checking Conditions



INDUCTIVE
UNIVERSITY

Expression Binding – Checking Conditions

[Watch the video](#)

You can use expression bindings to return true or false based on different conditions.

Example 1

Consider the following expression that references the Tag with a path of 'Folder/Machine State':

```
{Folder/Machine State} = 0
```

The above expression simply tests the value of the Machine State tag. If the value of the Machine State Tag is ever equal to 0 then the above expression would return true. In every case where Machine State is not equal to 0, then the expression would return false.

Example 2

It is possible to check for multiple conditions in the same expression. If you have two boolean tags and you only want the expression to return true if both tags are true then the binding would look like:

```
{boolTag1}=True && {boolTag2}=True
```

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

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

Expression Overview and Syntax

Overview

The expression language is used to define dynamic values for component properties and expression tags. Expressions often involve one or more other values that are used to calculate a final value. Expressions don't do anything, other than return a value.

The classic example for an expression is to change a temperature that is stored in Celsius to Fahrenheit in order to display it. Suppose you had a tag, Tank 6/Temp that was in Celsius. If you wanted to display that tag in Fahrenheit on a Label, you would use an Expression Binding on the label's text property using the following expression:

```
1.8 * {Tank 6/Temp} + 32
```

Every time that the temperature tag changes, the expression will re-calculate the value and push it into the Label's text property. Now lets say that you wanted to append a "°F" to the end of the label so that the user knew the units of the temperature. You could simply use some string concatenation in your expression, like this:

```
(1.8 * {Tank 6/Temp} + 32) + " °F"
```

Lets suppose that you wanted to give the user an option to display the value in Celsius or Fahrenheit, based on checking a checkbox. You could add a **Check Box** component to the screen called DisplayFahrenheit. Then you could use this expression to dynamically display either unit, based upon the user's selection:

```
if({Root Container.DisplayFahrenheit.selected}, (1.8 * {Tank 6/Temp} + 32) + " °F", {Tankf/Temp} + " °C")
```

Syntax

As its name suggests, everything in the expression language is an "expression". This means that everything returns a value. 5 is an expression. So is 5+1. So are `{MyTags/TankLevel}` and `{MyTags/TankLevel}+1`. Expressions can be combined in many powerful ways. Lets take a look at how expressions are written.

More formally, an expression is any one of the following:

- Number
- Boolean
- String
- Bound tag
- Bound property
- Function call
- Dataset access
- Equation involving any of these

Literal Values

Literal values are things like numbers, booleans, and strings that are represented directly in the language. In the expression language, numbers can be typed in directly as integers, floating point values, or using hexadecimal notation with a 0x prefix. Examples:

```
42
8.456
0xFFC2
```

Strings are represented by surrounding them with double or single quotes. You can use the backslash character to escape quotes that you want to be included in the string. Examples:

```
"This is a regular string"
'This one uses single quotes'
"This string uses \"escaping\" to include quotes inside the string"
```

Operators

You can use these arithmetic, logical, and bit-shifting operators to combine expressions.

Operator	Name	Description
--	Unary Minus	Negates a number.
!	Not	Logical opposite of a boolean.
^	Power	Raises a number of the power of another number. a^b is a^b .
%	Modulus	Modulus or remainder of two numbers. $a\%b$ is the remainder of $a\div b$
*	Multiply	
/	Divide	
+	Add/Concatenate	If both operands are numbers, this will add them together. Otherwise treats arguments as strings and performs concatenation.
-	Subtraction	
&	Bitwise AND	
&\#124	Bitwise OR	
xor	Bitwise XOR	
<<	Left Shift	A signed bitwise left shift.
>>	Right Shift	A signed bitwise right shift.
>	Greater Than	Logical greater-than test between two numbers. Returns a boolean.
<	Less Than	
>=	Greater Than or Equal To	

Operator	Name	Description
<=	Less Than or Equal To	
=	Equal	Tests for equality between two operands.
!=	Not Equal	Tests for equality, returning true when not equal.
&&	Logical AND	Returns true when both operands are true. Anything non-zero is considered true.
||	Logical OR	Returns true when either operand is true. Anything non-zero is considered true.
like	Fuzzy String Matching	Compares the left-hand value with the pattern on the right side. The pattern may consist of %, *, and ? wildcards.
//	Comments	Allows for comments following this operator.

Bound Values

Bound values are paths to other values enclosed in braces. These will appear red in the expression editor. When you are writing an expression for a **Expression Binding**, you can reference tag values and property values using the brace notation. When you are writing an expression for an **Expression Tag**, you can only reference other tag values. You can use the Insert Property () and Insert Tag Value () buttons to build these references for you.

Dataset Access

If you have an expression that returns a dataset, you can pull a value out of the dataset using the dataset access notation, which takes one of these forms:

```
Dataset_Expression ["Column_Name"] //returns the value from the first row at
the given column name
Dataset_Expression [Column_Index] //returns the value from the given column at the
first row
Dataset_Expression [Row_Index, "Column_Name"] //returns the value from the given
row at the given column name
Dataset_Expression [Row_Index, Column_Index] //returns the value from the given
row at the given column index
```

For example, this expression would pull a value out of a **Table** at row 6 for column "ProductCode":

```
{Root Container.Table.data}[6, "ProductCode"]
```

Note that you'll often have to convince the expression system that what you're doing is safe. The expression language can't tell what the datatype will be for a given column, so you may have to use a type-casting function to convince the expression language to accept your expression, like this:

```
toInt({Root Container.Table.data}[6, "ProductCode"])
```

Expression Functions

The expression language's functions are where much of the real power lies. A function may take various arguments, all of which can themselves be any arbitrary expression. This means that you can use the results of one function as the argument to another function. In general, the syntax for a function call is:

```
functionName(expression1, expression2, ...)
```

Whitespace and Comments

Whitespace, such as spaces, tabs and newlines, are largely ignored in the expression language. It is often helpful to break your expression up onto multiple lines for clarity. Comments are delimited by two forward slashes. This will make the rest of that line be ignored. This example shows an if function spread over 4 lines with comments annotating the arguments.


```
if( {Root Container.UseTagValueOption.selected},  
    {MyTags/SomeValue}, // Use the tag value  
    "Not Selected"      // Use default value if the user doesn't check the box  
)
```

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

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

DB Browse Binding



INDUCTIVE
UNIVERSITY

DB Browse Binding

[Watch the video](#)

Binding Properties to Database Tables

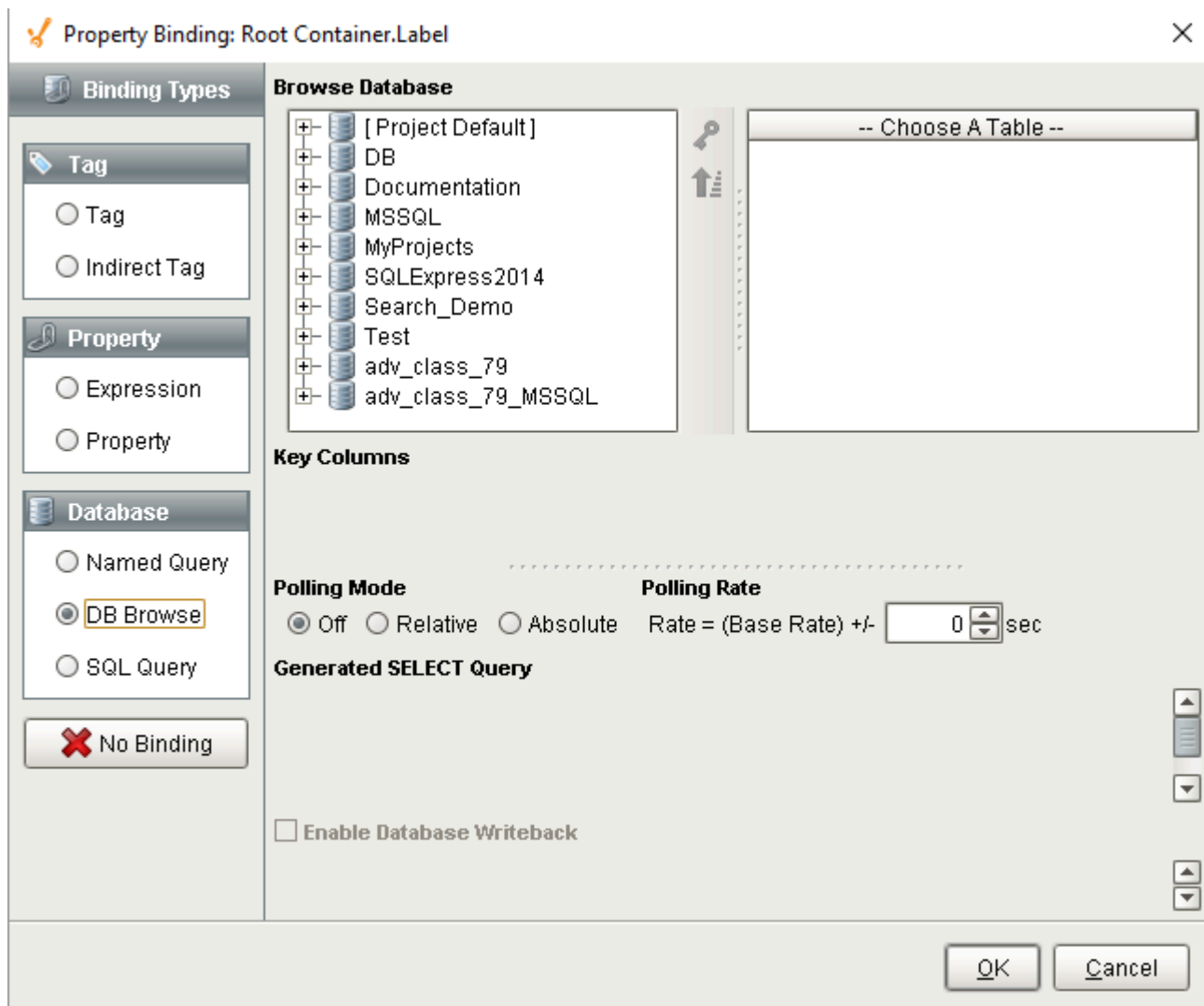
The DB Browse binding is technically equivalent to the [SQL Query binding](#), except that it helps write the queries for you. Using the Database Browse binding type, you can pick the table from a list of tables in each database that you want to pull content from. If you have a fixed range of data you 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).



TIP

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.



NOTE

Note, the examples in this section assume Ignition is connected to a Database, and the connection contains some database tables.

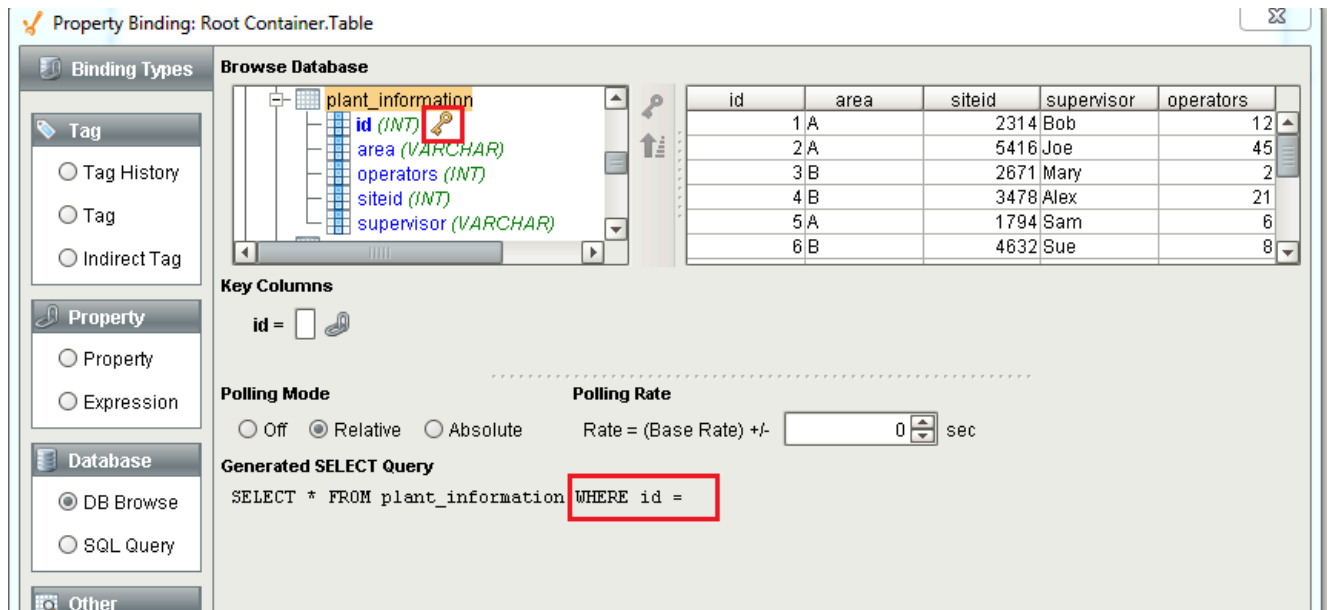
Configuring a DB Browse Binding

Key Column

The DB Browse binding is easy to get started, and helps create queries for you. Let's get started.

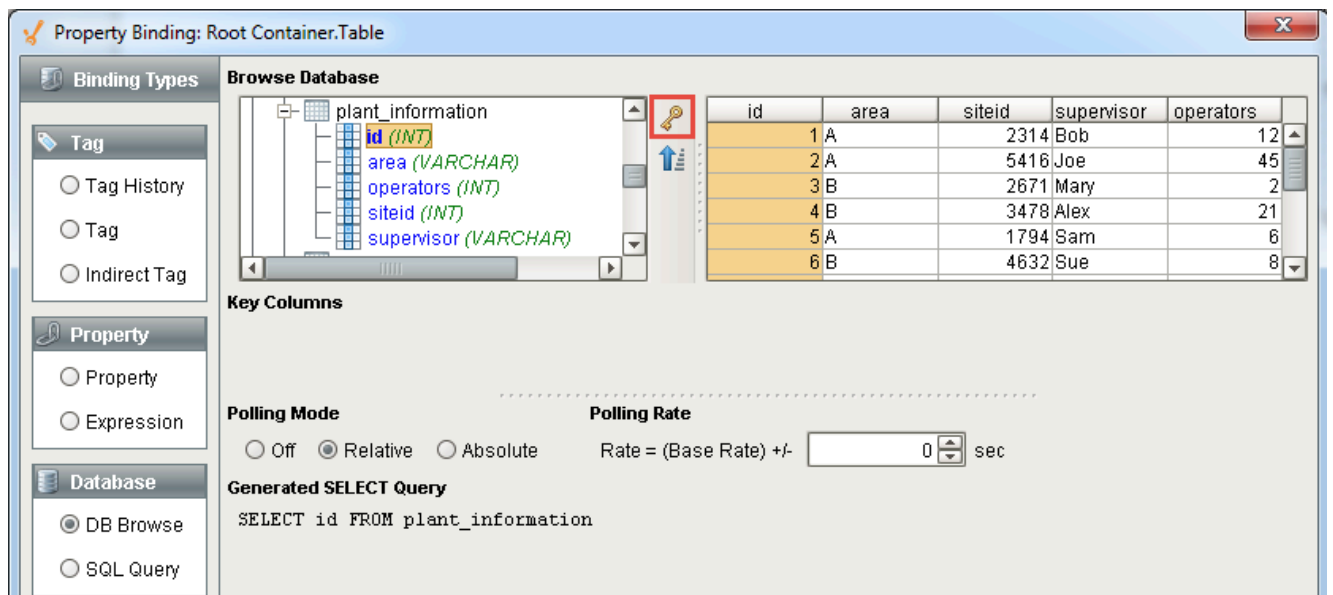
1. Place a Table component on the window.
2. With the Table component selected, click the binding icon  next to the **Data** property in the **Property Editor**.
3. The Property Binding window will appear. Select the **DB Browse** radio button.

4. A tree of available database tables will appear under the **Browse Database** heading. Once a table is selected, you can see a sample of the data in the Preview pane.
5. If you expand a database table (click the + button next to the table name) you may find a Key button . This denotes that the binding will attempt to use this column in a WHERE clause in the resulting query that is automatically generated.
 - i. In this example, you'll notice that the **Generated SELECT Query** is returning all columns from the table. This is because the plant_information table is selected instead of a column, so the binding will attempt to return every column from the selected database table.



id	area	siteid	supervisor	operators
1	A	2314	Bob	12
2	A	5416	Joe	45
3	B	2671	Mary	2
4	B	3478	Alex	21
5	A	1794	Sam	6
6	B	4632	Sue	8

6. To remove the WHERE clause, simply select the **id** column in the Browse Database section, and click the **Remove Key** button  to the right of the Browse Database window. You'll notice a couple of things:
 - i. The Key icon  next to the id column in the Browse Database disappeared since it is no longer being used as a Key Column.
 - ii. The table Preview panel on the right highlighted the entire id column.
 - iii. Since the Browse Database is not set to identify a Key column, there is no longer any filtering of our query under the **Key Columns** section, so it will bring back all of the data in the columns that we select.
 - iv. Notice that our **Generated SELECT Query** changed. It no longer has the **WHERE** clause, and since the id column is the one currently selected, we have gone from selecting all columns to only selecting the id column.



Sort Order

In the DB Browse binding, you can also do ordering of data in the table. There is a Sort button  that allows you to sort in ascending and descending order. Using the same example, let's have the area column of our table be the key column so you can only see the entries for **area B**, and return all the operators, siteid, and supervisors columns in the table. We'll also order the data by the number of operators at each site in ascending order.

1. Select the **area** column in the Browse Database section, and hit the Key button  to have the query filter by area.
2. Select the **operators** column in the Browse Database and then click the **Sort**  button. This will sort based on the column in ascending order. (Click the Sort button again to sort in descending order, and click it a third time to remove the ordering).
3. Select the **operators**, **siteid**, and **supervisor** columns from the Browse Database section since those are the columns we want returned. No need to return the area column since we only want to see the entries for **area B**.
4. Enter the area key to equal 'B' in the **Key Columns** section.

Property Binding: Root Container.Table

Binding Types

Tag

☐ Tag History

☐ Tag

☐ Indirect Tag

Property

☐ Property

☐ Expression

Database

☒ DB Browse

☐ SQL Query

Browse Database

plant_information

- id (INT)
- area (VARCHAR)
- operators (INT)
- siteid (INT)
- supervisor (VARCHAR)

Key Columns

area = B

Polling Mode

☐ Off ☒ Relative ☐ Absolute

Polling Rate

Rate = (Base Rate) +/- sec

Generated SELECT Query

```
SELECT siteid, supervisor, operators FROM plant_information WHERE area = 'B' ORDER BY operators ASC
```

id	area	siteid	supervisor	operators
1	A	2314	Bob	12
2	A	5416	Joe	45
3	B	2671	Mary	2
4	B	3478	Alex	21
5	A	1794	Sam	6
6	B	4632	Sue	8

5. Confirm the binding by clicking **OK** to produce the specified data in the table. Only **area B** rows are shown, and our data is being sorted in ascending order of our operators column.

siteid	supervisor	operators
2,671	Mary	2
2,549	Dave	6
4,632	Sue	8
6,457	Chris	14
3,478	Alex	21

NOTE

This example only had one key and one order column, but you can add as many as you want. Just select a second column and hit the Key or Sort buttons.

Dynamic Filters



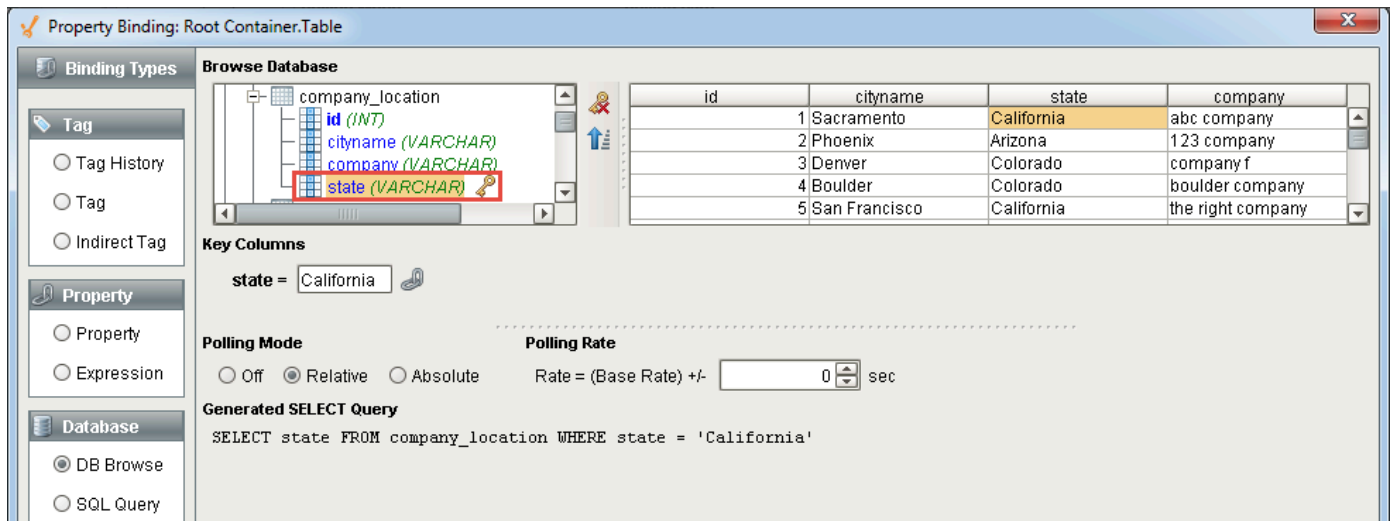
INDUCTIVE
UNIVERSITY

DB Browse Binding - Dynamic Filters

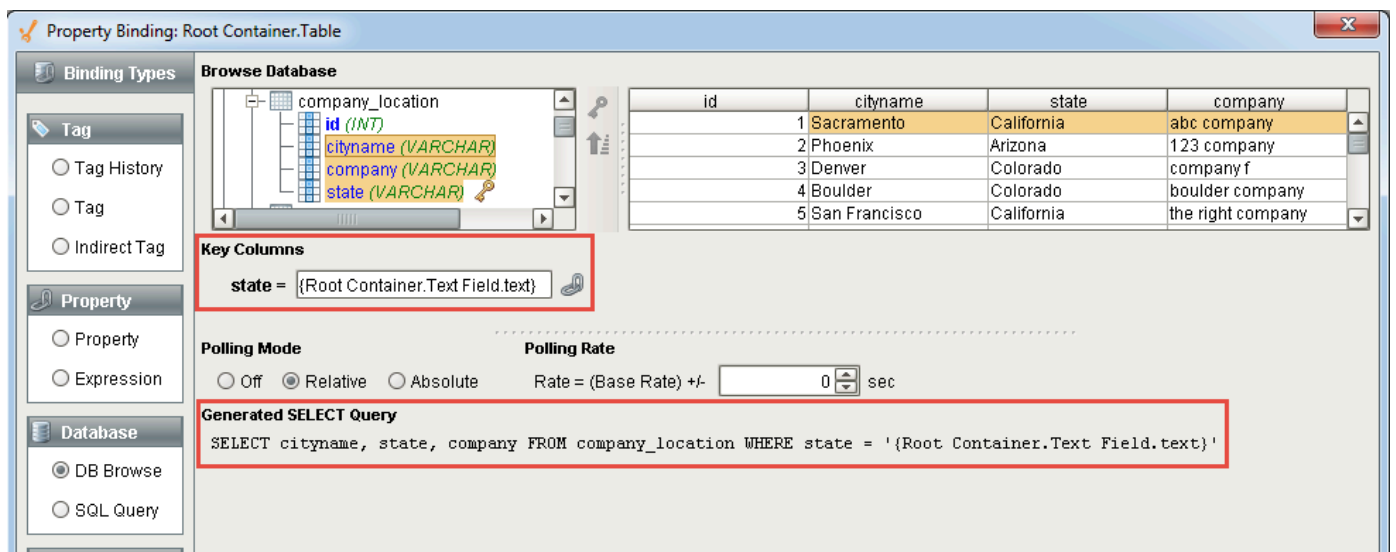
[Watch the video](#)

DB Browse bindings also give the ability to bind a property to the key column to allow for dynamic filtering of the returned data. This allows you to give the operators some control over the data they are seeing. This example is using a list of companies that has their respective city and state.

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. Let's pull all the data from this Table except for the id, and filter on state. Remove the Key  from the **id** column and place it on the **state** column.



4. Select the **cityname**, **company**, and **state** columns. You can do this with Control+Click, or by clicking and dragging in the results table in the upper right.
5. Instead of statically typing in a value like we in did in the above example, let's make it dynamic using the Text Field. Click the Binding button  next to the value in the **Key Columns** section, and select the Text property of the Text Field. Notice there is now a property reference in the **Key Column** as well as the **Generated SELECT Query**.



6. Confirm the binding by clicking **OK**, and you'll see the Table change based on the value typed in the Text Field.

Dynamic Filter: Arizona

Arizona

cityname	state	company
Phoenix	Arizona	123 company
Scottsdale	Arizona	stacking inc
Flagstaff	Arizona	towers industries
Yuma	Arizona	udc
Tuscon	Arizona	jobs+
Chandler	Arizona	tech2u
Gilbert	Arizona	groceries 4 less
Glendale	Arizona	wrapping paper im...

Dynamic Filter: California

California

cityname	state	company
Sacramento	California	abc company
San Francisco	California	the right company
Los Angeles	California	g industries
Oakland	California	international imports
San Jose	California	compuplace
Bakersfield	California	fink and fink
Napa	California	quick to you

Dynamic Filter: Colorado

Colorado

cityname	state	company
Denver	Colorado	company f
Boulder	Colorado	boulder company
Estes Park	Colorado	beta productions
Colorado Springs	Colorado	global exports
Vail	Colorado	vester team
Golden	Colorado	outdoor restoration
Aspen	Colorado	fun times productio...
Fort Collins	Colorado	military surplus

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

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

SQL Query Binding



INDUCTIVE
UNIVERSITY

SQL Query Binding

[Watch the video](#)

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.

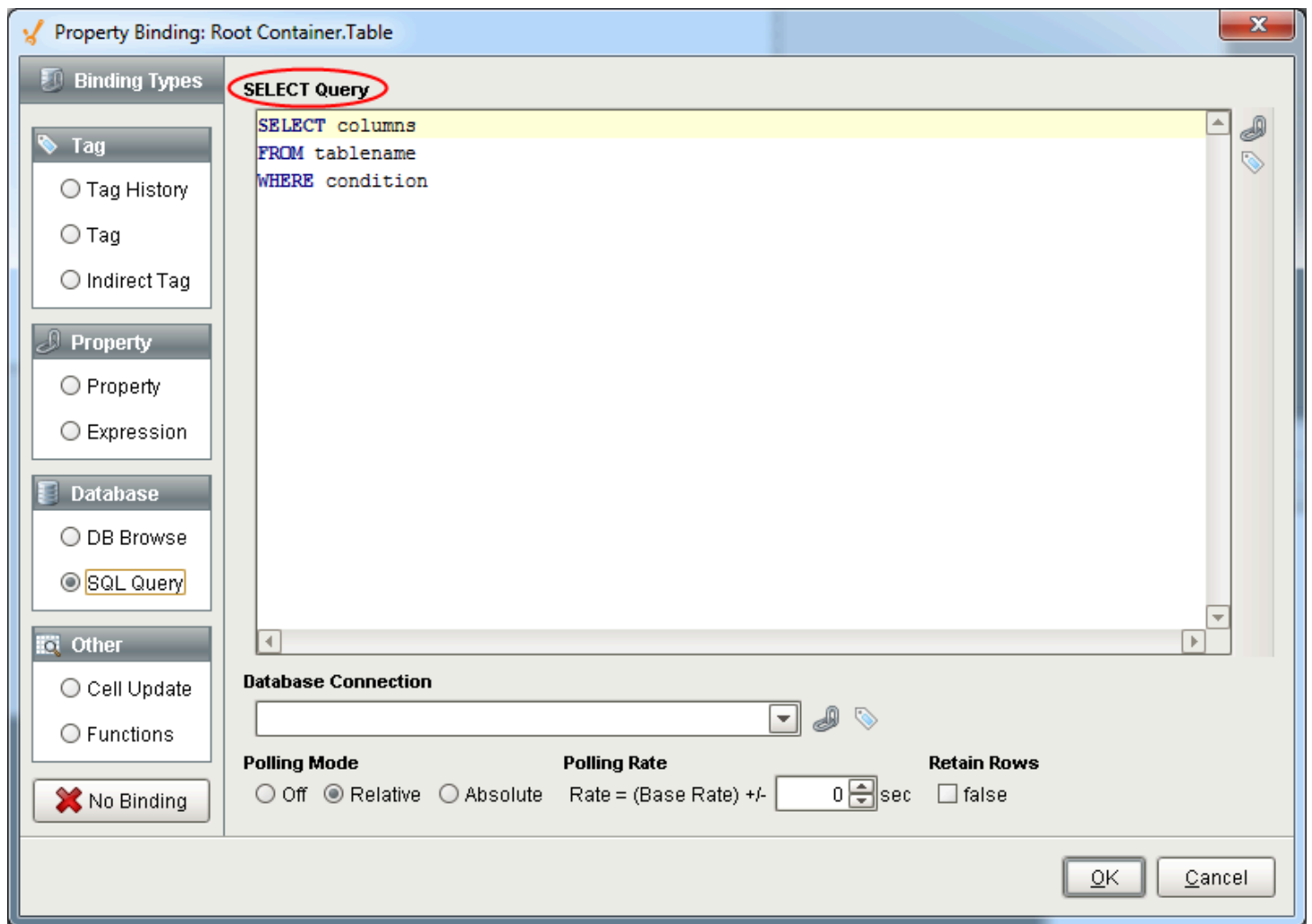


TIP

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 (and are usually Dynamic). These will usually return many rows with multiple columns. For example, showing all customer details from a certain account, or all downtime events in the facility. The only important distinction here is that if you are returning multiple rows and/or columns, you need to put that binding on a property of type **dataset** like the Data property on a **Table component**. Basic data types like integers or strings can only show the first value of a query result, not the whole set. You can easily see the difference in the binding window because only basic data types will show two query entry fields, SELECT and UPDATE. Dataset properties only show SELECT.



Dynamic Filters



INDUCTIVE
UNIVERSITY

SQL Query Binding - Dynamic Filters

[Watch the video](#)

Using the 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.

Note that 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.

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

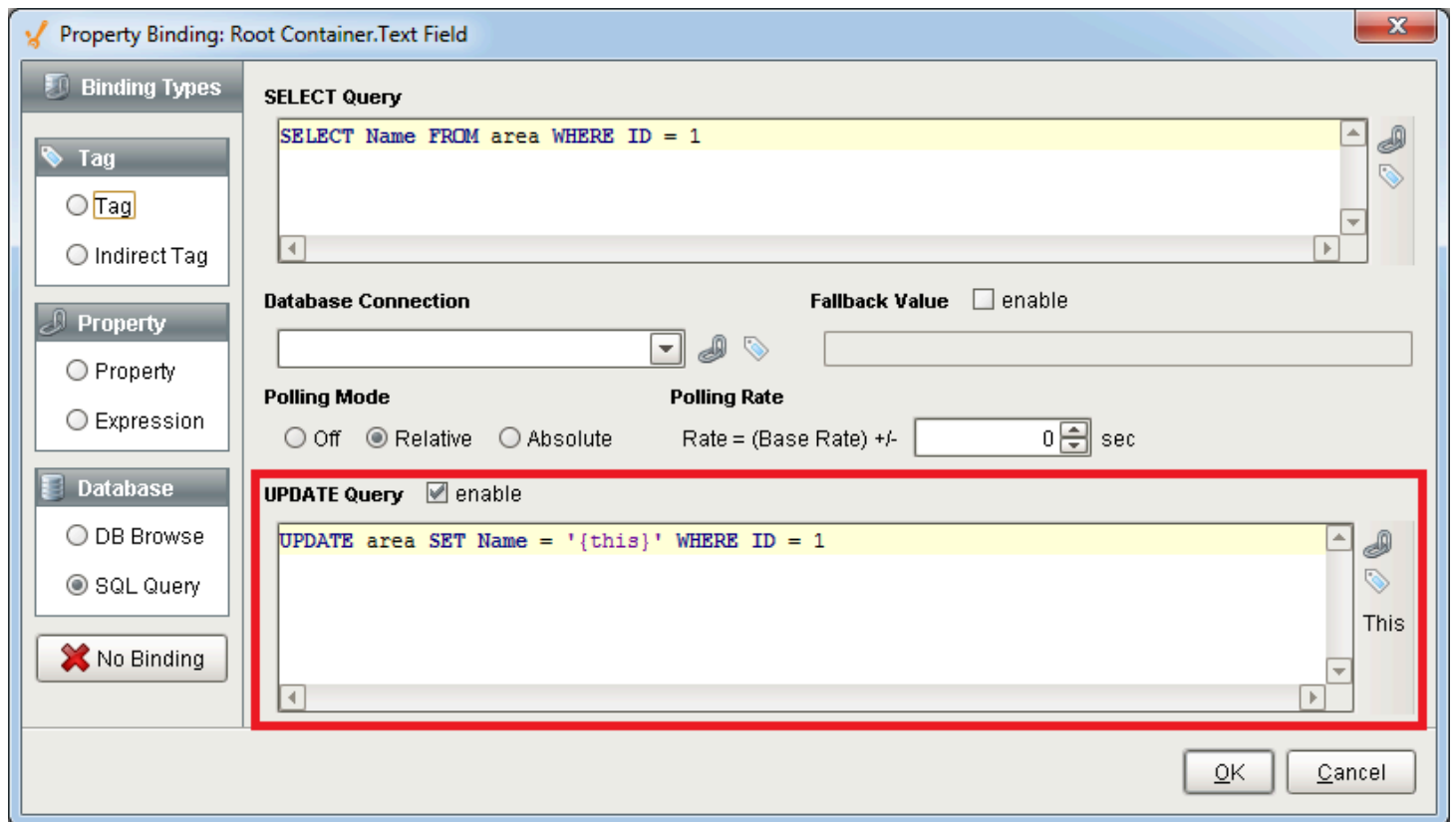


INDUCTIVE
UNIVERSITY

**SQL Query Binding - Scalar
Query and Update**

[Watch the video](#)

You can bind an input component bidirectionally to a SQL query to allow data be retrieved and updated in the database. This can only be done with a Scalar Query on a non Dataset property. 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 to denote the new value from the bound property. If 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
```

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.

IT WORKS FOR DB BROWSE TOO!

The [DB Browse Binding](#) also features a database writeback that provides the same functionality as the SQL Query binding update query, and allows you to make your DB Browse Queries bidirectional as well, as long as your binding is a scalar query on a non dataset property. The update query on the DB Browse binding will also transfer to the SQL Query binding much like the select query, to help you get started with building your query!

Scalar Query Fallback



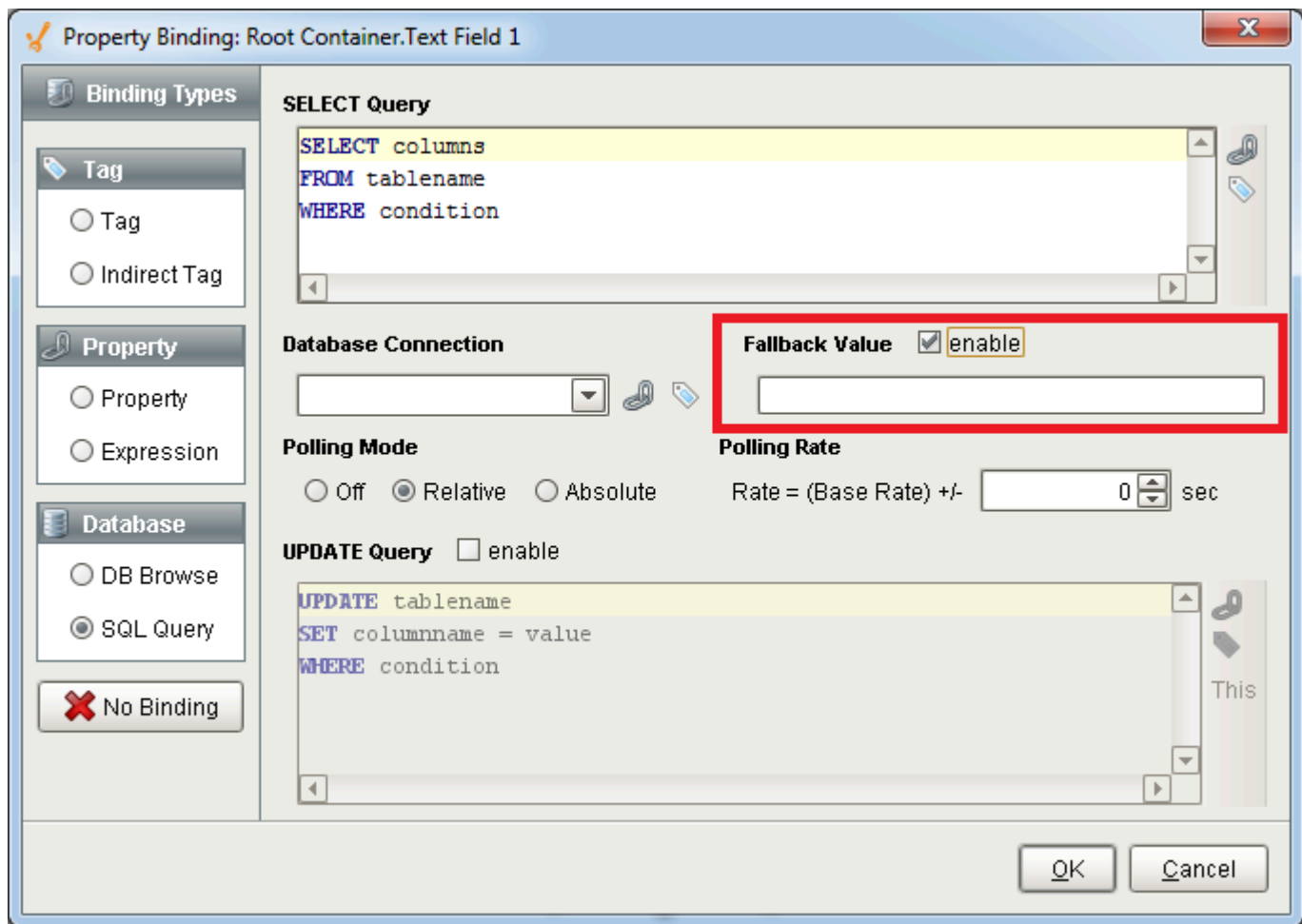
INDUCTIVE
UNIVERSITY

**SQL Query Binding - Scalar
Query and Fallback**

[Watch the video](#)

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 happily contain zero rows.



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, that 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
```

For more information, please see the [SQL Stored Procedures](#) page.

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

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

Cell Update Binding



INDUCTIVE
UNIVERSITY

Cell Update Binding

[Watch the video](#)

Binding a Dataset Property to Realtime Values

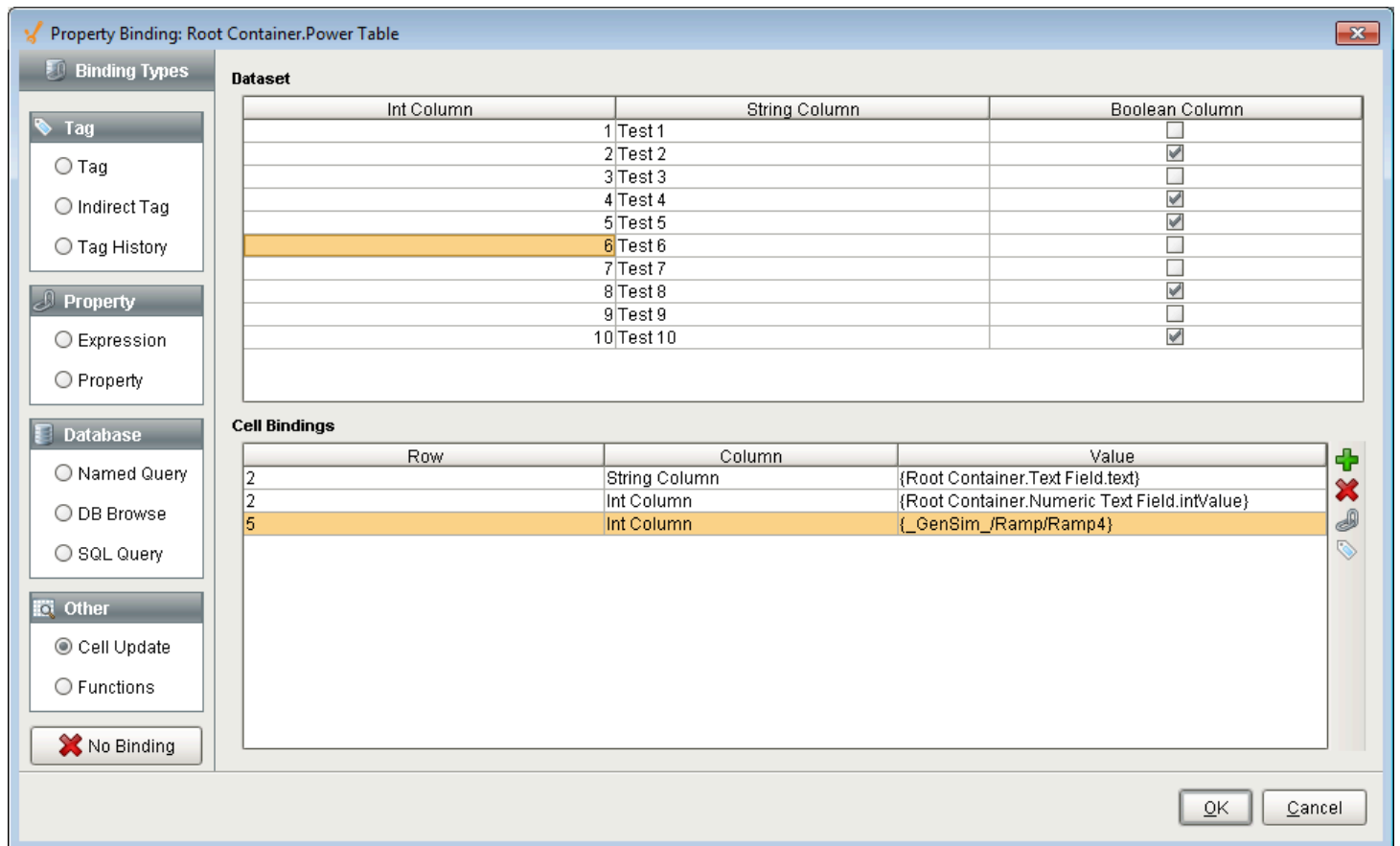
The Cell Update binding enables you to easily make one or more cells inside a dataset dynamic. This is particularly useful for components such as the Table to easily display realtime Tag information, or in the Linear Scale which stores configuration information in a dataset. The Cell Update binding allows 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.

CELL UPDATE BINDING

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 four 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.

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.
- 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.

NOTE

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.

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=Test 1	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 eight rows for eight pumps, and each having a pressure

value. As an alternative, you could use eight numeric text fields and labels, but the Table component looks much nicer compared to all of those labels and numeric text fields.

Pump	Pressure
Pump 1	0
Pump 2	0
Pump 3	0
Pump 4	0
Pump 5	0
Pump 6	0
Pump 7	0
Pump 8	0

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 () on a **Data** property, click the **Cell Update** binding type, and you'll see your dataset along with a **Cell Bindings** area.
2. Select the first pressure reading, and click the add () button 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.

Property Binding: Root Container.Table

Binding Types

Tag

☐ Tag History

☐ Tag

☐ Indirect Tag

Property

☐ Property

☐ Expression

Database

☐ DB Browse

☐ SQL Query

Other

☒ Cell Update

☐ Functions

No Binding

Use the *Cell Update* binding to dynamically bind specific cells inside a dataset to tag or property values. Simply select the cell you want to bind, and hit the "Plus" button. Rows can be specified explicitly by their index (e.g. "15") or by referencing another column (e.g. "Name=Compressor")

Dataset

Pump	Pressure
Pump 1	0
Pump 2	0
Pump 3	0
Pump 4	0
Pump 5	0
Pump 6	0
Pump 7	0
Pump 8	0

Cell Bindings

Row	Column	Value
0	Pressure	

OK

Cancel

3. Select the first **Row** and first **Value** cell in the Cell Binding area. Use the Tag () button to the right to select the Tag reference. You can also manually type in your Tag reference, or place in a property reference instead.

Property Binding: Root Container.Table

Binding Types

- Tag**
 - ☐ Tag History
 - ☐ Tag
 - ☐ Indirect Tag
- Property**
 - ☐ Property
 - ☐ Expression
- Database**
 - ☐ DB Browse
 - ☐ SQL Query
- Other**
 - ☒ Cell Update
 - ☐ Functions

☒ No Binding

Use the *Cell Update* binding to dynamically bind specific cells inside a dataset to tag or property values. Simply select the cell you want to bind, and hit the "Plus" button. Rows can be specified explicitly by their index (e.g. "15") or by referencing another column (e.g. "Name=Compressor")

Dataset

Pump	Pressure
Pump 1	0
Pump 2	0
Pump 3	0
Pump 4	0
Pump 5	0
Pump 6	0
Pump 7	0
Pump 8	0

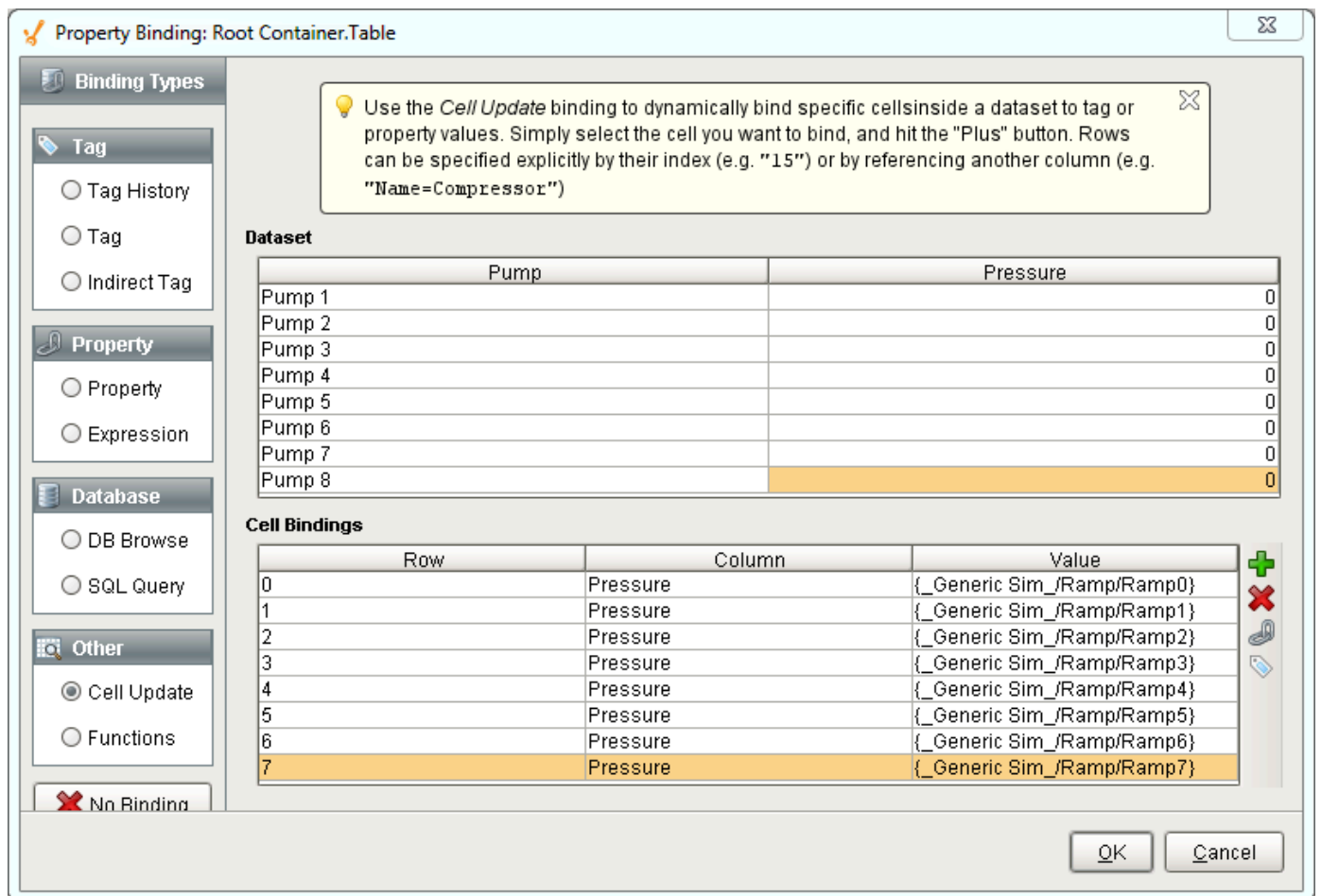
Cell Bindings

Row	Column	Value
0	Pressure	{_Generic Sim_/Ramp/Ramp0}

OK **Cancel**

4. Now, you have the value of that Tag to the Pressure column in row 0. Repeat Step 3 for the other seven Pumps so that the Cell Binding area looks like the image below, then click **OK**.



5. You can see that the Table values are now updating with the value of their respective Tag.

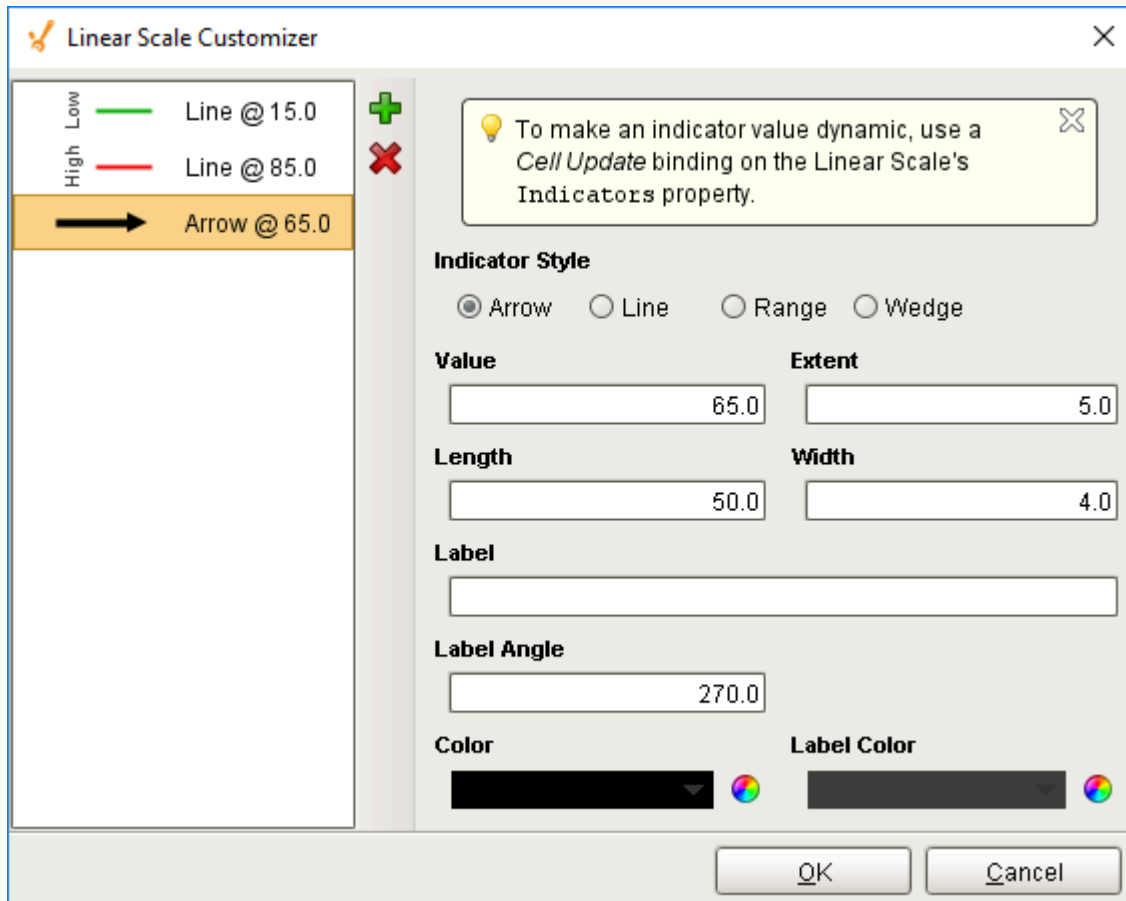
Pump	Pressure
Pump 1	202.73
Pump 2	52.05
Pump 3	-22.27
Pump 4	238.07
Pump 5	152.05
Pump 6	126.71
Pump 7	129.84
Pump 8	503.2

Adding a Realtime Indicator to the Linear Scale Example

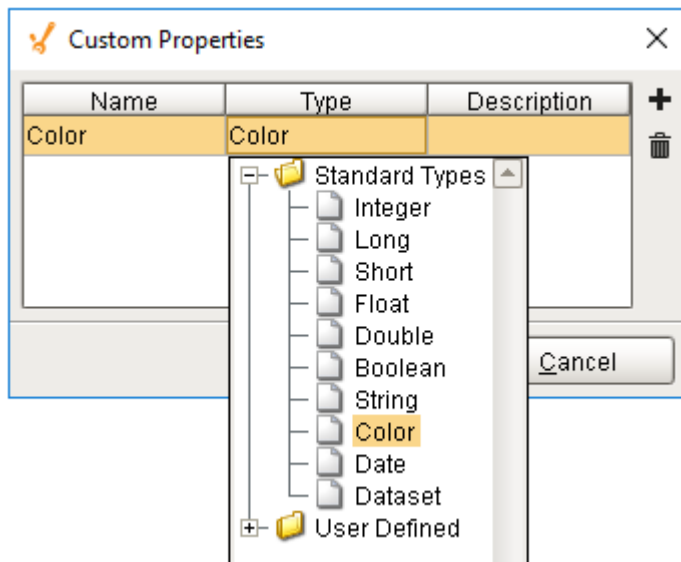
The **Linear Scale** is a component that has a special Scale Indicators customizer that allows you to configure setpoints. You can setup the value, color, style, and more for each of the indicators. During runtime, these values are normally static, as there is no way to setup a binding in the customizer. 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.

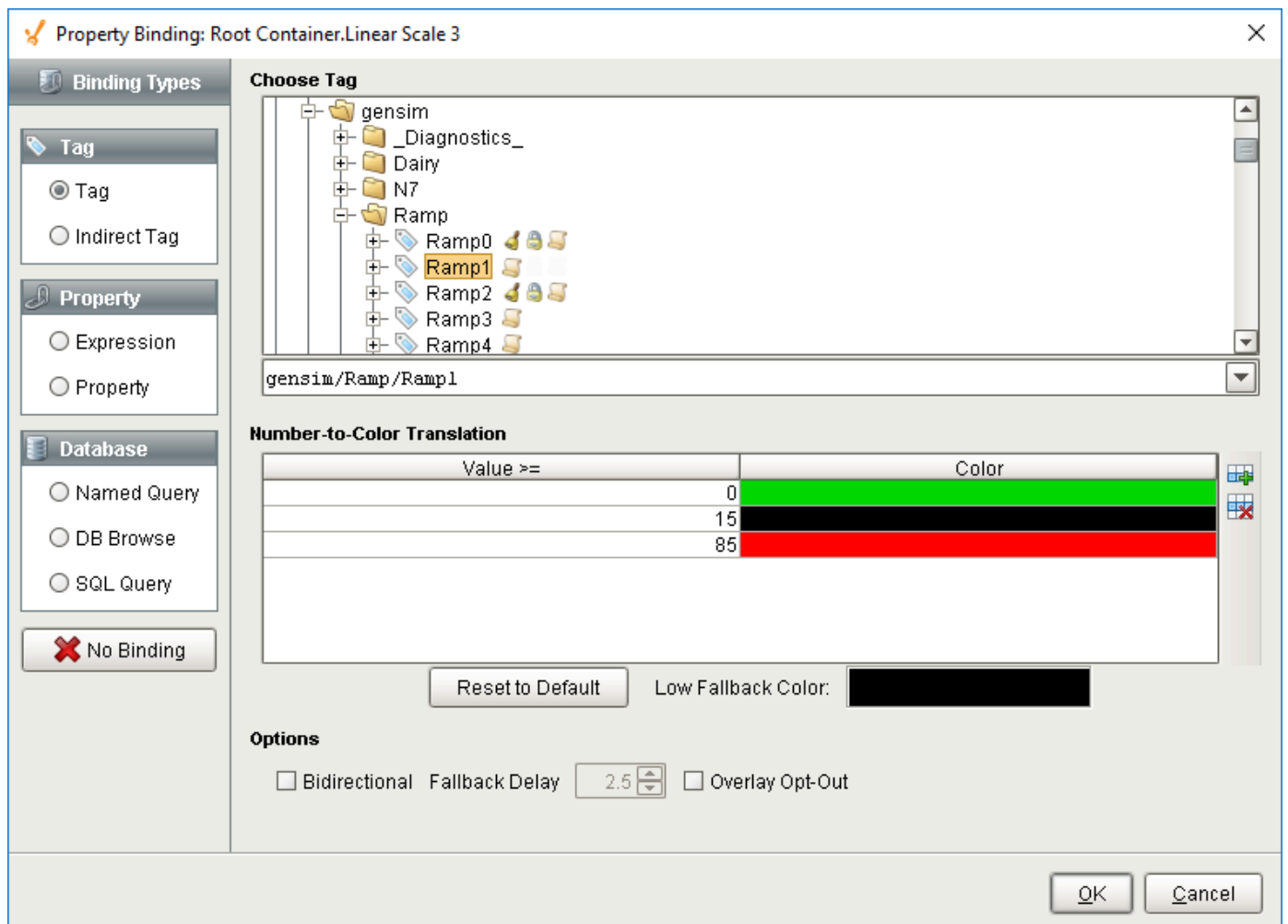
1. Select the Linear Scale and right click on the **Customizers > Scale Indicators**.
2. Create a new indicator using the **Arrow** Style. This new indicator will be whatever the current value is. Set the following properties for the Arrow, and click **OK**.
 - i. **Value**: 65.0
 - ii. **Length**: 50.0
 - iii. **Width**: 4.0



3. 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**:
 - i. With the Linear Scale selected, right click on the **Customizers > Custom Properties**.
 - ii. Choose **Color** from the dropdown list, and click **OK**.

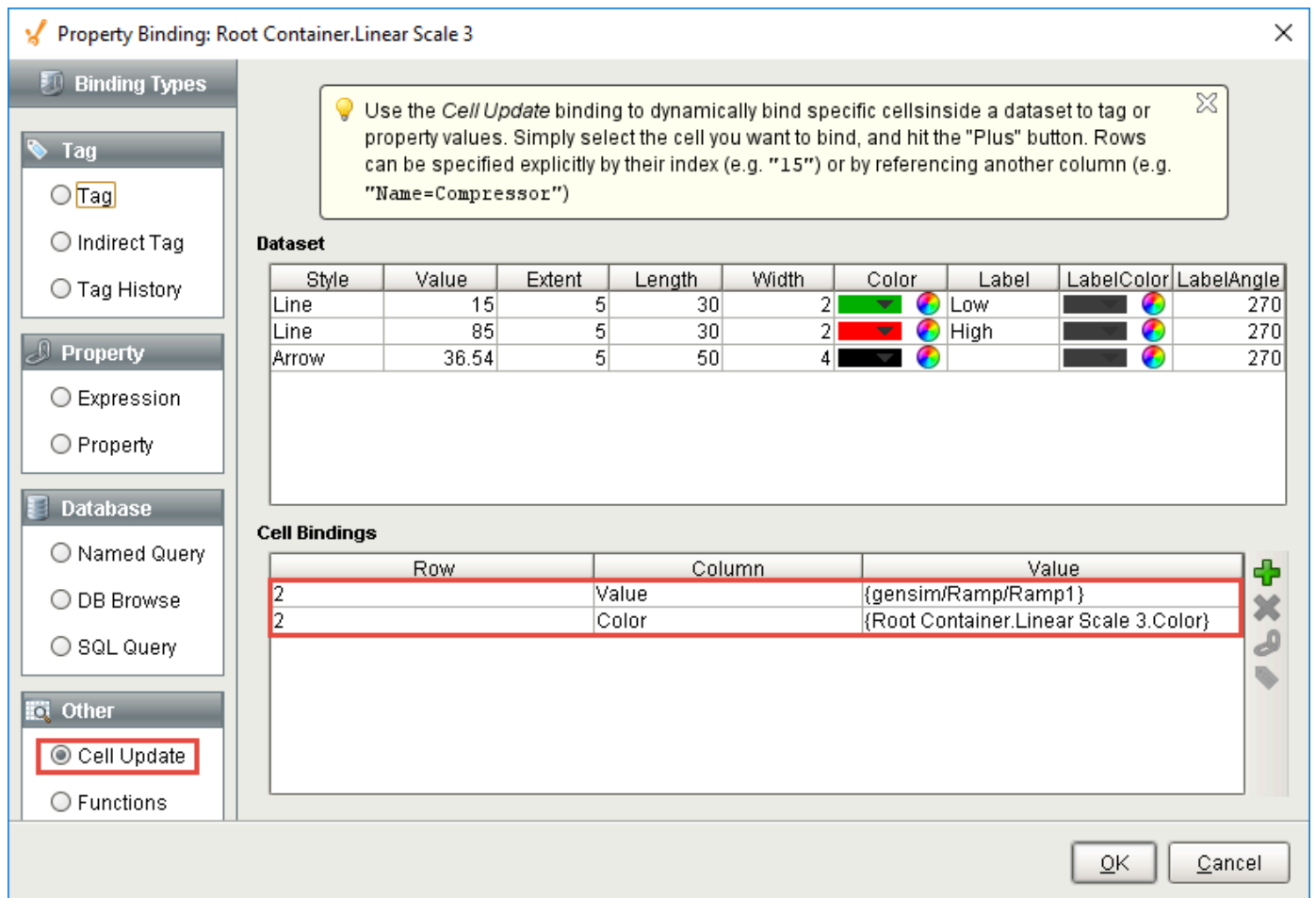


4. Next, setup a Number-to-Color Tag binding on the custom **Color** property. The values shown in the image below are based on the static setpoints that were set in the Linear Scale customizer.

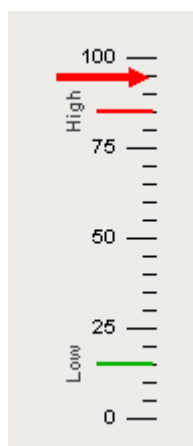


5. Finally, setup a Cell Update binding on the **Indicators** dataset property of the Linear Scale. Notice below that both of the cell bindings are for the same row, just different columns. This is fine, as long as there aren't duplicate bindings for the same cell.
- Bind the Value cell to the same Tag (i.e., Ramp1) used in the custom property Color binding to get the value.

ii. Bind the Color cell to the Color custom property we just created.



6. 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 is out of range and the arrow turned red after exceeding the setpoint.



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

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

Function Binding



INDUCTIVE
UNIVERSITY

Function Binding

[Watch the video](#)

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, and using 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. I just used the default settings for the function binding, but 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.

Property Binding: Root Container.Table

Binding Types

Tag

☒ Tag History

☐ Tag

☐ Indirect Tag

Property

☐ Property

☐ Expression

Database

☐ DB Browse

☐ SQL Query

Other

☐ Cell Update

☒ Functions

☒ No Binding

Binding Function

Alarm Status

Path Filter

Min Priority

Active and Unacked?

true

Clear and Unacked?

true

Display Path Filter

Max Priority

Active and Acked?

true

Clear and Acked?

false

Polling Mode

☐ Off ☒ Relative ☐ Absolute

Polling Rate

Rate = (Base Rate) +/- 0 sec

OK Cancel

Now our table 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 shelve alarms using our own scripting functions. This may be more work in setting it up then 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
cfb014fa-79b...	prov.default/tag...	Test	Sep 8, 2016 8:47 AM	2	4
202e1955-09...	prov.default/tag...		Sep 8, 2016 8:46 AM	2	4
8a2eea2c-0a...	prov.default/tag...		Sep 8, 2016 8:46 AM	0	4
7de7411a-fc0...	prov.default/tag...		Sep 6, 2016 10:16 AM	2	1

☐ Active Time	Display Path	Current State	Priority
☐ 9/8/16 8:47 AM	Test	Active, Unacknowl...	Critical
☐ 9/8/16 8:46 AM	momentary test/Testing	Active, Unacknowl...	Critical
☐ 9/8/16 8:44 AM	momentary test/Testing	Cleared, Unackno...	Critical
☐ 9/6/16 10:16 AM	_SLC Sim_JB3/B3:0/Alarm	Active, Unacknowle...	Low

Acknowledge Shelve

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

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

Named Query Binding

New in 7.9.4



INDUCTIVE
UNIVERSITY

Named Query Binding

[Watch the video](#)

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 has 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.

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 magnifying glass (🔍) button to get a list of all available Named Queries that you can choose from.
- **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 property (📌) button or the Tag (🏷️) button.
- **Query:** In the query section, there is nothing to configure, but you can see what the Named Query you have selected looks like.
- **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.

Property Binding: Root Container.Table

Binding Types

- ☒ Tag
 - Tag
 - Indirect Tag
 - Tag History
- ☐ Property
 - Expression
 - Property
- ☒ Database
 - Named Query**
 - DB Browse
 - SQL Query
- ☐ Other
 - Cell Update
 - Functions

SELECT Query

Path

BayItems 🔍

Parameters

Type	Name	Data Type	Value
Value	BayNum	Int4	{Root Container.Nume...}

Query

```
SELECT * FROM containers WHERE StorageBay = :BayNum
```

Polling Mode

☒ Off ☐ Relative ☐ Absolute

Polling Rate

Rate = (Base Rate) +/- 0 sec

Retain Rows

☐ false

No Binding (with red X icon)

OK **Cancel**

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 also works much the same as the SELECT query in that you need to select the path to the Named Query and setup 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.Table

Binding Types

- Tag
- Indirect Tag

Property

- Expression
- Property

Database

- ☒ Named Query
- ☐ DB Browse
- ☐ SQL Query

SELECT Query

Path
Contacts/Find Contact

Parameters

Type	Name	Data Type	Value
Value	BadgeNumber	Int4	{Root Container.Dropdown.se...}

Query
SELECT name FROM contacts WHERE badgeid = :BadgeNumber

Polling Mode
☒ Off ☐ Relative ☐ Absolute

Polling Rate
Rate = (Base Rate) +/- 0 sec

UPDATE Query ☒ Enabled

Path
Contacts/Update Contact

Parameters

Type	Name	Data Type	Value
Value	BadgeNumber	Int4	{Root Container.Dropdown.se...}
Value	NewName	String	{Root Container.Text Field.text}

Query
UPDATE contacts SET name = :NewName WHERE badgeid = :BadgeNumber

OK **Cancel**

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

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

Templates



INDUCTIVE
UNIVERSITY

About Templates

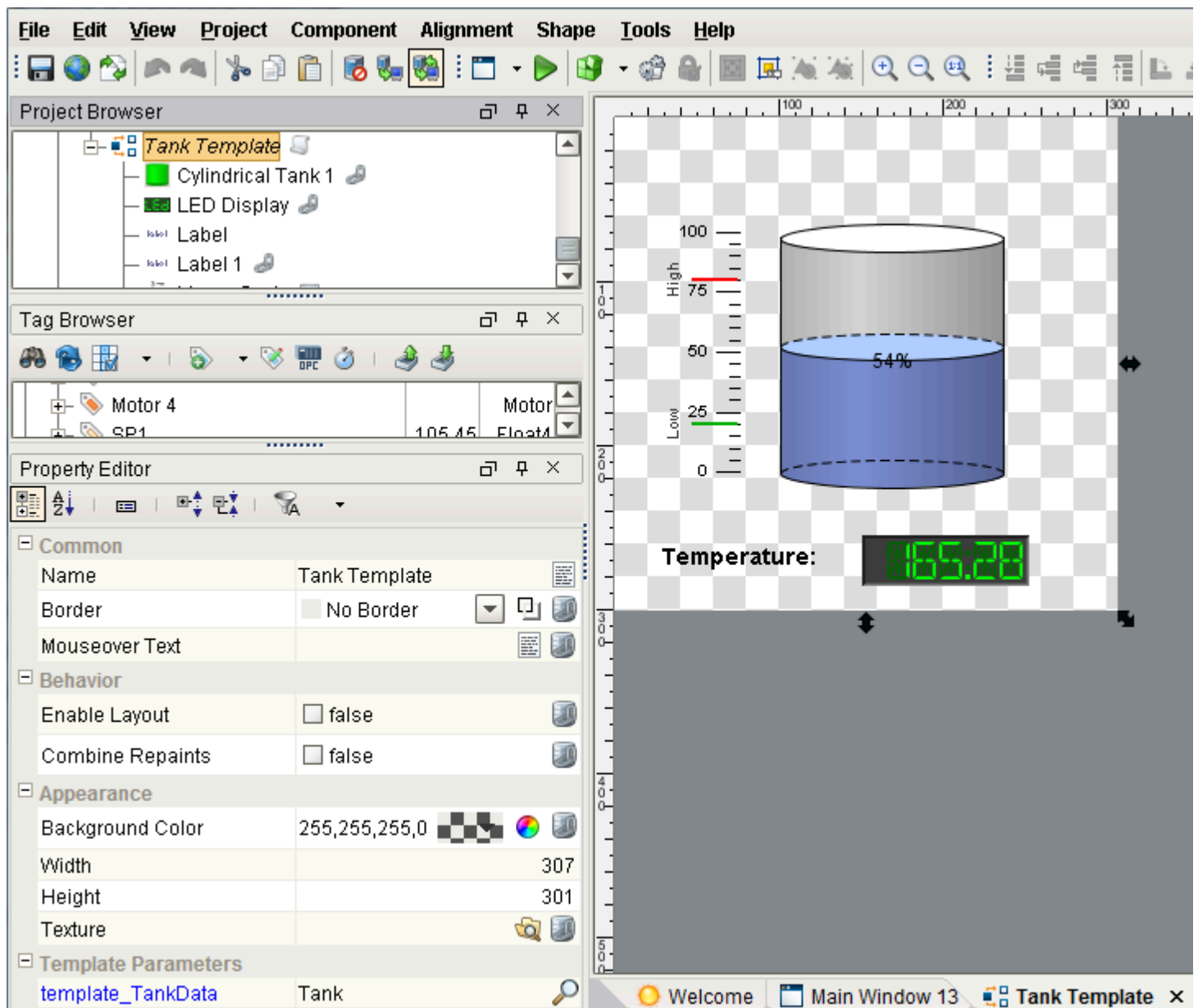
[Watch the video](#)

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 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.

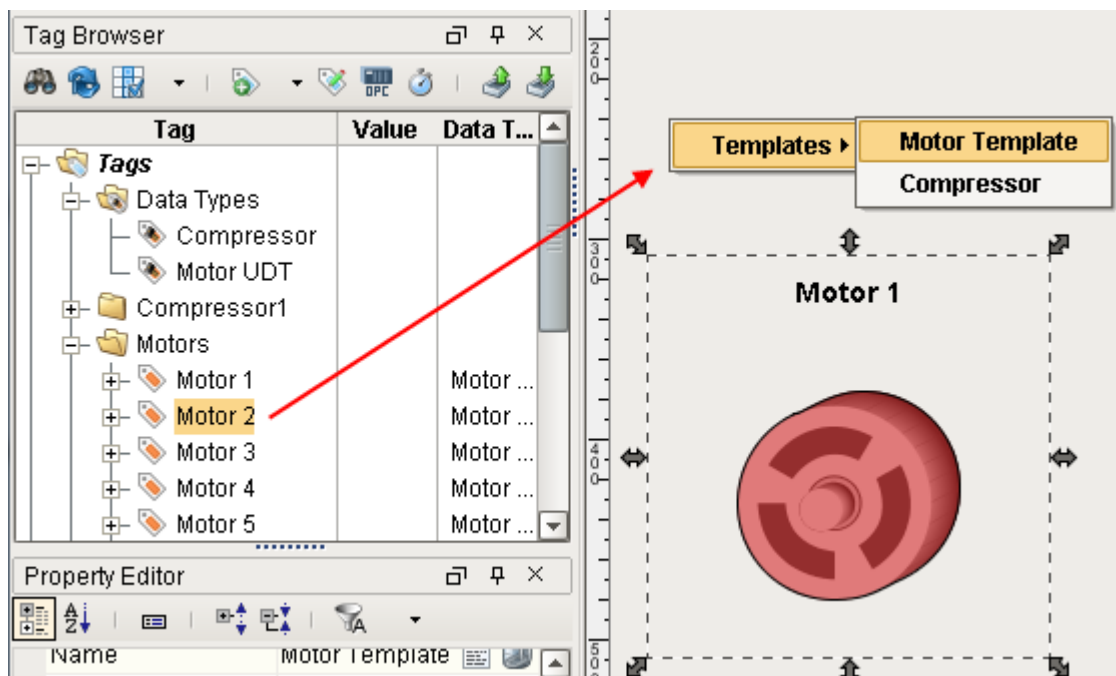
UDT Parameter

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.

The Drop Target Parameter

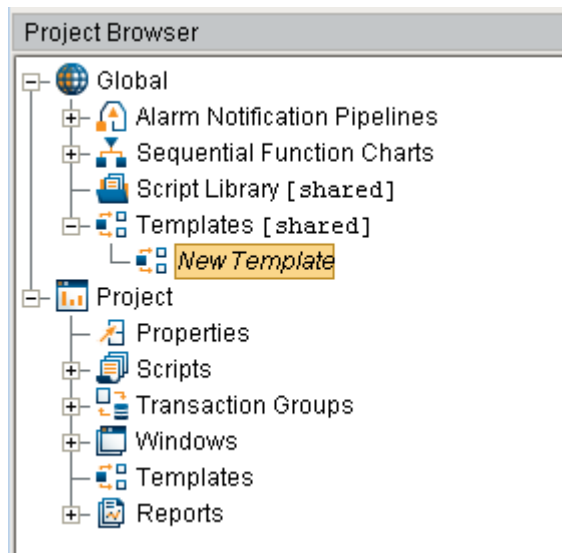
When you specify parameters in the **Custom Properties** window (right-click the checkered-box of the template and select **Customizers > Custom Properties**), you can set one of the parameters as the Drop Target. This allows you to drop a Tag of that type onto your template instances or onto a window 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.



Global Templates

Global Templates work the same as any other Template with the added bonus that they are globally scoped and available to be used and edited across all Projects. This is useful for very generic Templates which will be used across multiple Projects. For Templates specific to a certain Project, it is still recommended that they be created in the Project Templates space, so that they are not accidentally modified in a Project that is not using them.



Online Shared Templates

Ignition allows you to save and retrieve templates from Inductive Automation's online repository, the Ignition Exchange. In the [Ignition Exchange](#), you can access resources, templates, and tools that you can use in your own Ignition projects. Ignition assets designed by others in various industries can be shared through the Exchange. This collection encompasses anything that can be built inside of Ignition including screens, graphics, templates, views, reports, alarm pipelines, scripting functions, database backups, projects, full systems, Ignition demos, and more.

Changed in 7.9.13 The Cloud Templates browser has been removed as of 7.9.13. All Cloud Templates have been automatically moved to the Ignition Exchange and are still available.

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

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

Create a Template



INDUCTIVE
UNIVERSITY

Basic Template

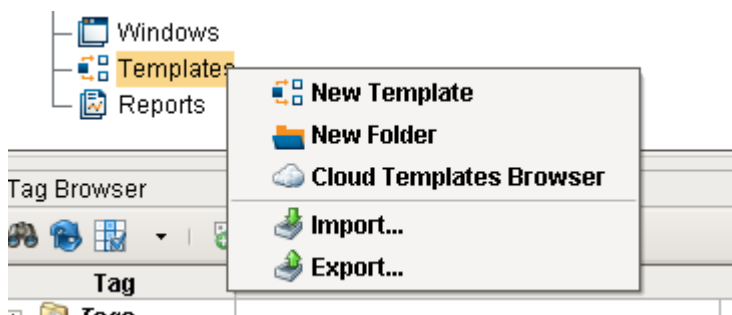
[Watch the video](#)

Templates are a simple but very powerful feature that you can use with your Vision windows. Using templates, you would define the graphics and parameters in one place, called the master template. You then create instances throughout your project that update whenever the master/parent template is changed.

Creating a 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**.

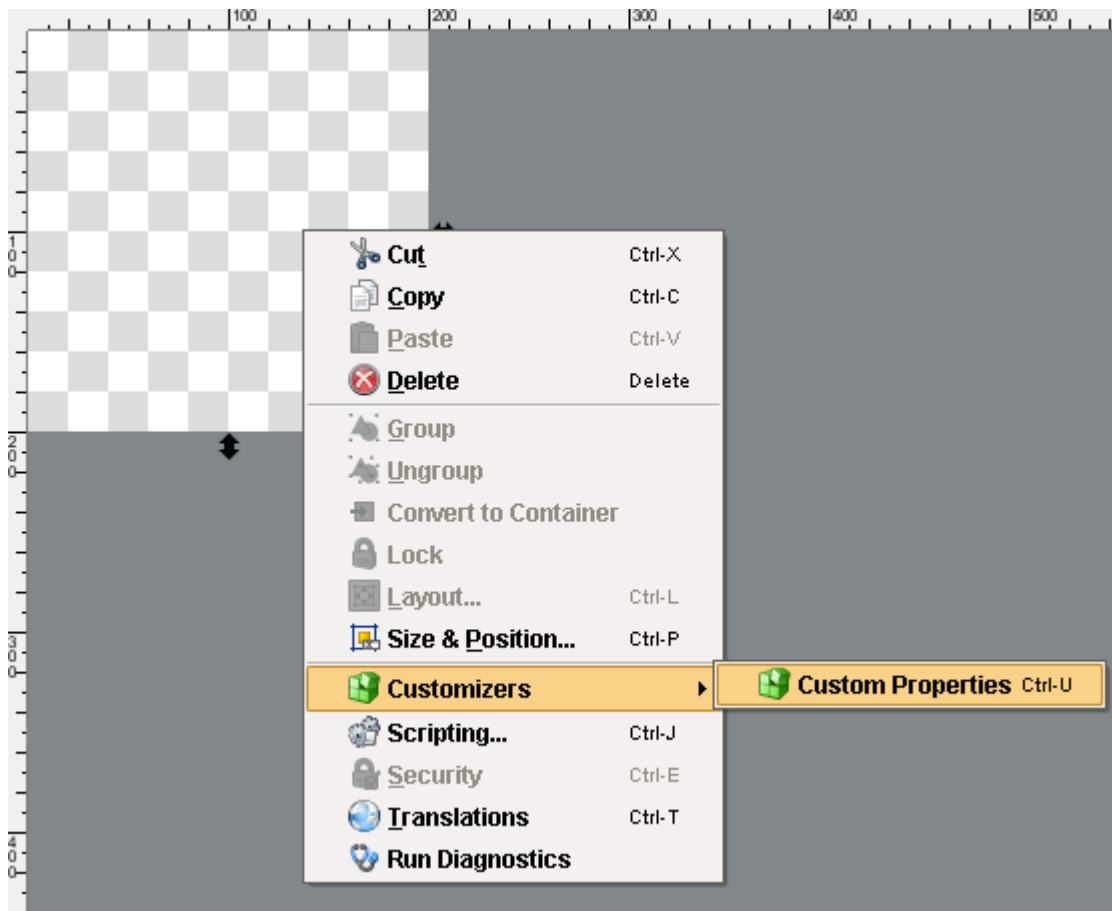


A checkered box is displayed in the design space where you design your template. The checkered box means that the template is completely transparent. You can set a background if you want.

2. Right-click on **New Template** and click **Rename** to change its name to something else, let's say **Motor**.



3. Right-click on the background and select **Customizers > Custom Properties**.



The **Custom Properties** window is displayed. Here you can set both the **Template Parameters** and **Internal Properties**. Template Parameters appear on each template instance, allowing each instance to be configured differently, like an indirection variable. Internal properties show up when designing the template master, but they do not show up on the template instances.

The template needs at least one Template Parameter property so that it can exist on every instance you put on the screens. Then each instance can bind the parameter property to a tag.

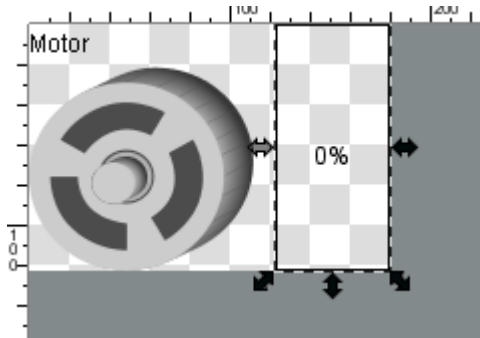
4. In the **Custom Properties** window, click the + icon in the Template Parameter area to add the values.
5. Enter the following and click **OK**.

Name: **value**

Type: **Integer**

Drop Target: **select the check box**

6. Go to the checkered area of your template, from the Symbol Factory and Component Palette drag one or a few components such as a motor, a label, and a Level Indicator.
7. Resize the checkered area to fit all your components, and not larger, to remove all the unusable space.



Now we need to link one of the components, lets say the Level Indicator, to the Template Parameter **value** we created earlier.

8. Select the Level Indicator and go to the Property Editor, click on the binding icon for the **Value** property. The Property Binding window is displayed.
9. Click the **Property** button, choose the **value** property, and click **OK**.

You now have configured a motor template, with a parameter property called value with an integer data type. Use this template to create many instances of it in your project.

Using the Template

Once you have a template, here are some of the things you can do with it.

Renaming the Template

You can rename your template by right clicking on it in the project browser and selecting Rename, or changing the name in the Property Editor.

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 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.

Binding Template 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.

Editing 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 a 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.

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

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

Custom Input Template



INDUCTIVE
UNIVERSITY

Custom Input Template

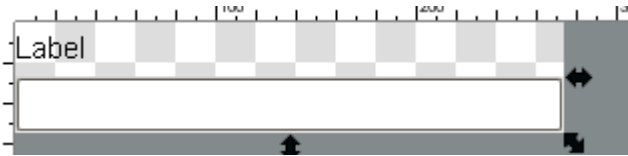
[Watch the video](#)

Sometimes you may need to have text fields in your project for the user to input data. Rather than copying and pasting these text fields into each window, you can create a template for them that includes a label and a text field. A template can contain scripting to validate the data in the text field.

To Create a Custom Input Template

1. In the Project Browser, right-click **Templates** and select **New Template**.
2. Right-click on **New Template** and click **Rename** to change its name it to something, lets say **Text Entry**.
3. To add a parameter, right-click the checkered area and select **Customizers > Custom Properties**.
4. In the **Custom Properties** window, add two **Template Parameters** by clicking the + icon twice and entering the following:
 - i. 1st Parameter
 - Name: **display**
 - Type: **String**
 - ii. 2nd Parameter
 - Name: **text**
 - Type: **String**
5. Click **OK**.

- From the Component Palette, drag a **Text Field** and a **Label** to the checkered area of the template, resizing so that the label and the text box occupy the majority of the space.



- Select the **Label** component, go to **Property Editor**, and click on the binding icon of the **Text** property. The Property Binding window is displayed.
- In the **Property Binding** window, click on **Property**, choose the **display** property. Make sure the **Bidirectional** Box is not selected, and click **OK**. Don't worry if the label on the template disappears. Its is simply displaying the value of the display custom property which is currently blank.
- Now select the **Text Field** component, go to **Property Editor**, and click on the binding icon of the **Text** property.
- In the **Property Binding** window, click on **Property**, choose the **text** property. Make sure the **Bidirectional** Box is selected, and click **OK**.

Suppose you want data validation on the Text Field. We can have the background color of the Text Field change depending on whether the user has entered a text value or not. We can set the field to red when its empty, and white when it has a value.

- While the **Text Field** component is still selected, go to **Property Editor**, and click on the binding icon of the **Background** property.
- In the **Property Binding** window, click on **Expression**, enter the following expression:

```
//Inside the if statement is the len and the trim functions that are
//available when doing expression bindings.
//The trim function will trim the blank spaces from the text therefore
//validating that there is actual text rather than spaces.
//The len function will count the lenth of the recently trimmed text.
//The if statements asks the question: Is the length of the trimmed text
//greater than zero?
if(len(trim({textBox.Text
Field.text}))>0,color(255,255,255),color(255,0,0))
```

- Click **OK**.

- Close the template, click **Save** on the prompt window, and click **OK**.

12. To set data validation, drag a few the **Text Entry** templates from **Project Browser** to the screen or window. Notice that the text field is red.
13. In the **Property Editor**, go to the **display** custom property and enter something like First Name. Do something similar for the other templates. Notice how the label gets filled in with the data we enter.

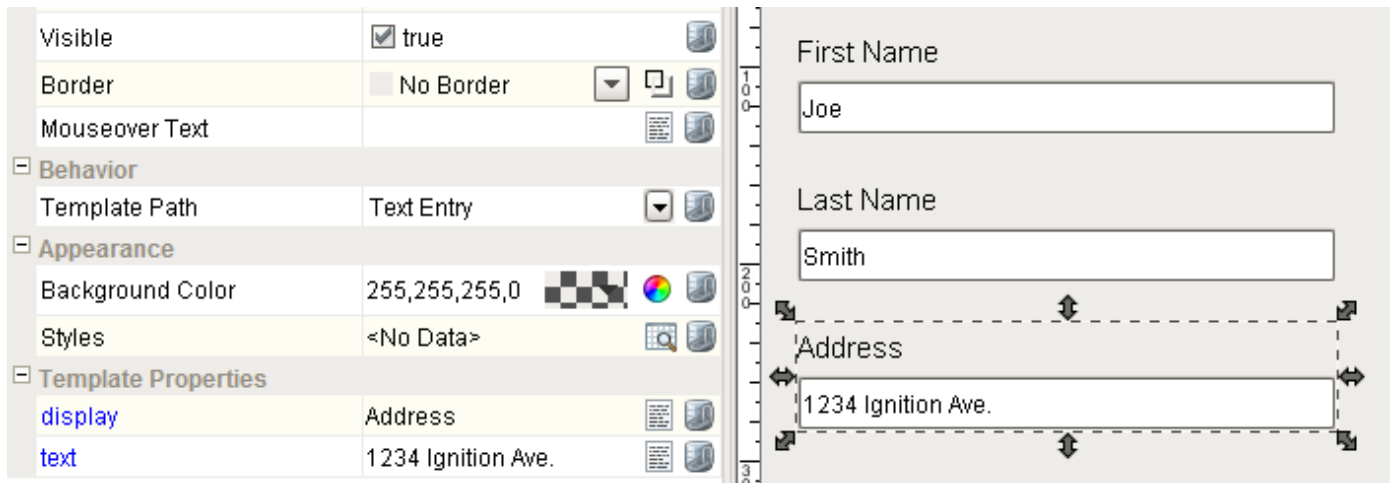


First Name

Last Name

Address

14. Put the designer into Preview Mode, and enter in some values into the text fields. Notice that the background color changed to white, and the information that was entered is now in the **text** property that can be bidirectionally bound so that it writes to a tag or is used in a script.



Visible	☒ true
Border	No Border
Mouseover Text	
Behavior	
Template Path	Text Entry
Appearance	
Background Color	255,255,255,0
Styles	<No Data>
Template Properties	
display	Address
text	1234 Ignition Ave.

First Name

Joe

Last Name

Smith

Address

1234 Ignition Ave.

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

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

Template - Resizing and Enable Layout



INDUCTIVE
UNIVERSITY

Template - Resizing and
Enable Layout

[Watch the video](#)

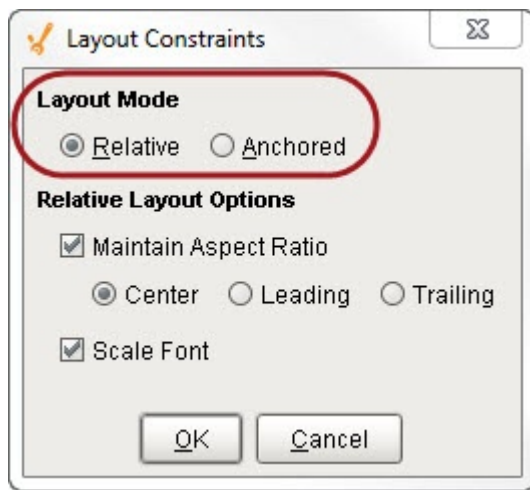
Resizing Templates

You can configure the layout of each template so it resizes properly.

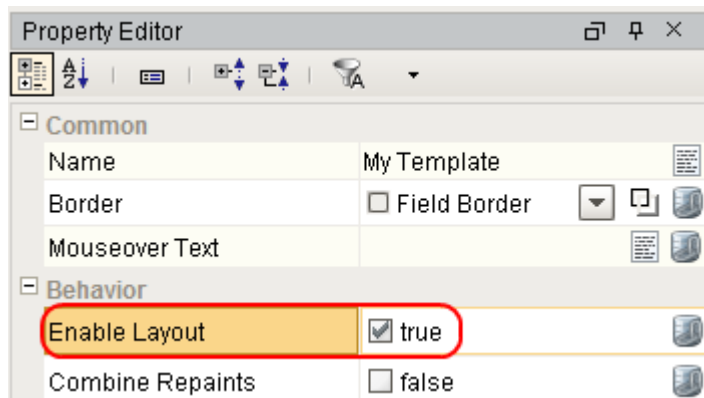
By default, when you drag a template into the window from the Project Browser, the size of the instance is exactly the same size as the master template. You can make the size larger or smaller. To go back to the same size as the master template, right-click on the instance and choose **Revert to Master size**.

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.



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

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

Template - Indirect Binding



INDUCTIVE
UNIVERSITY

Template - Indirect Binding

[Watch the video](#)

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 Motor UDTs that all have the same kind and number of tags. The only thing that is different is the Motor number.

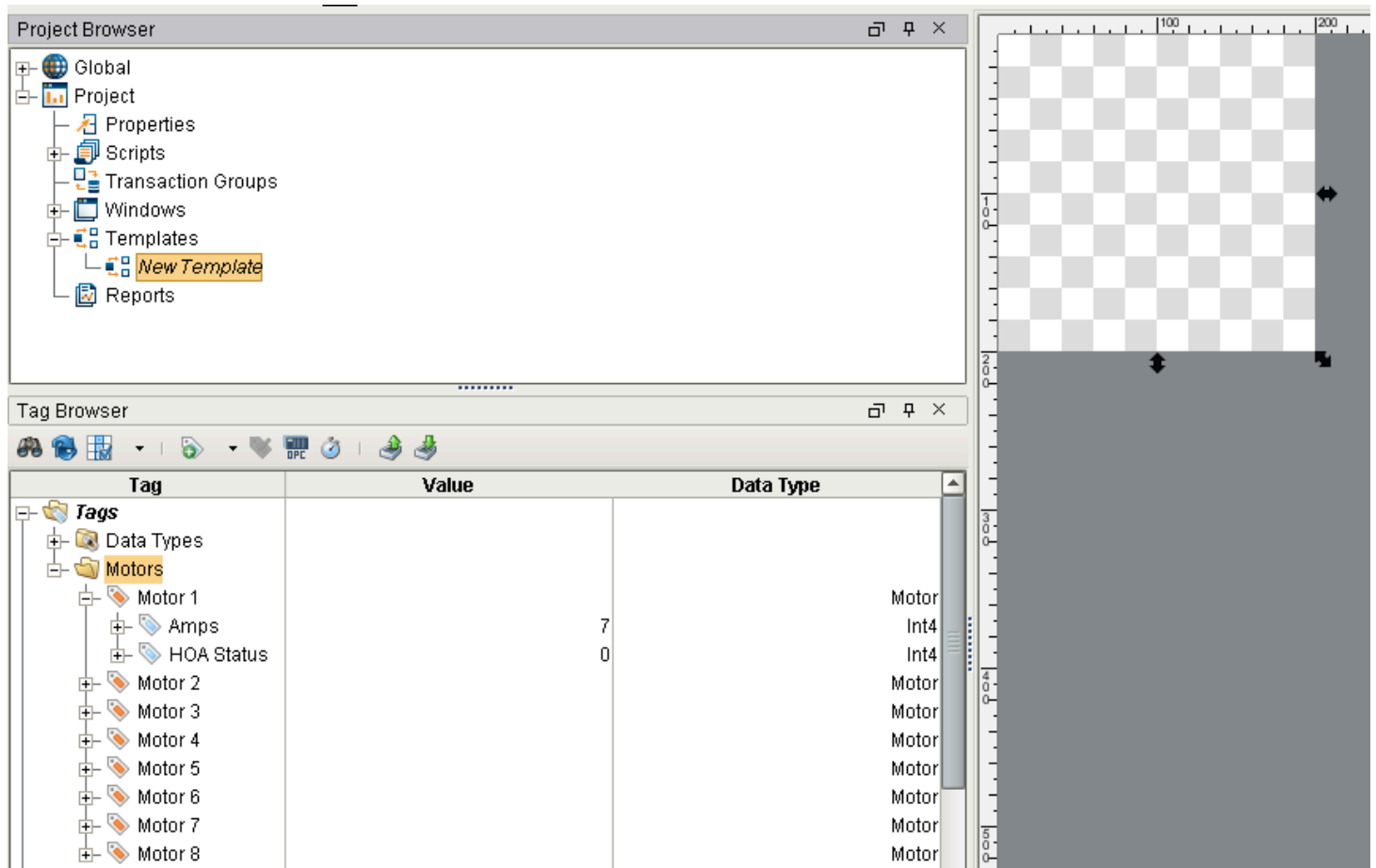
If each tag inside the UDT has a tag path that looks like this:

Motors/Motor 1/AMPS

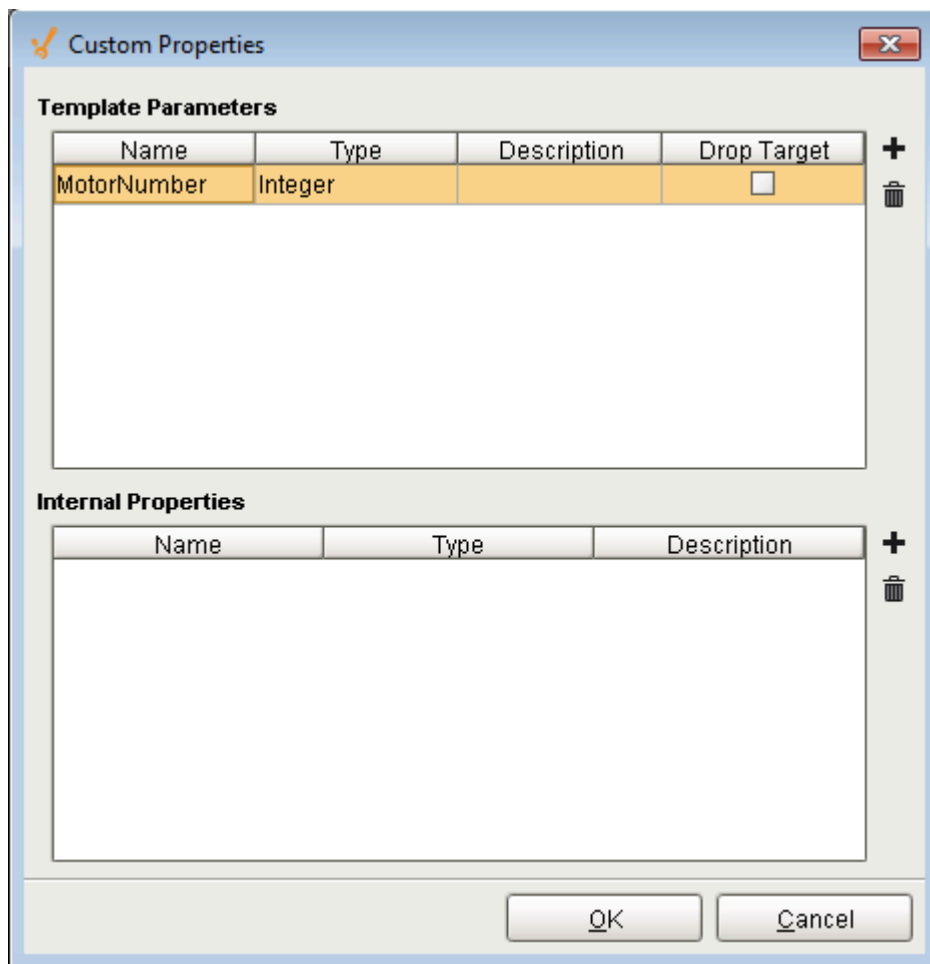
You can create a Template Parameter on the Custom Properties window called `motorNumber`, and use it to make indirect tag bindings. For every component within the template the tag path can be bound in such a way that the Custom property of `motorNumber` substitutes the motor number therefore resulting in a valid tag path that points to the desired motor.

Example - Indirect Template

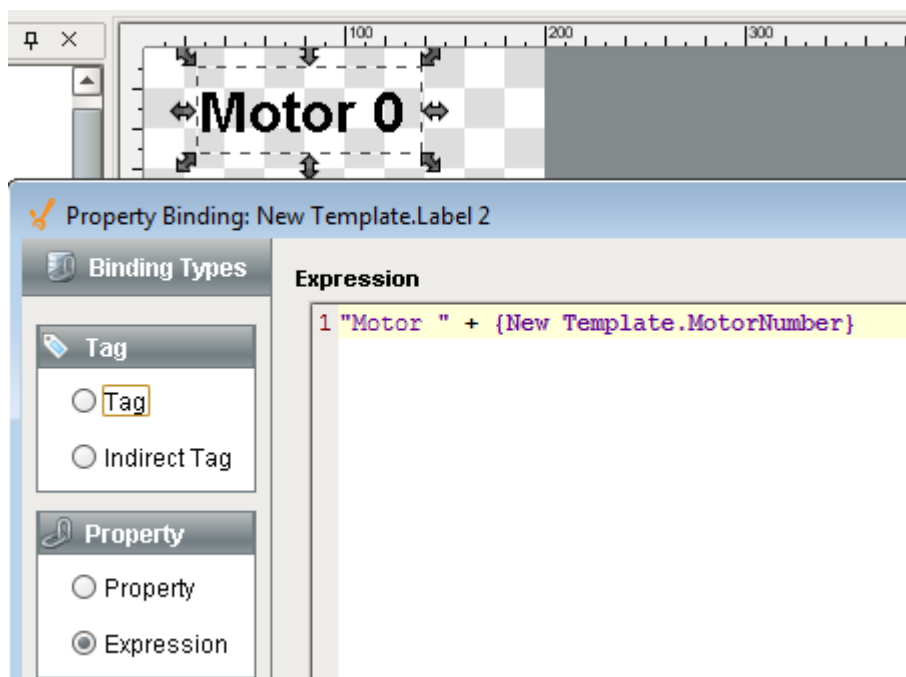
Let's take a look at how we can set something like this up. First, we need to make sure we have a fresh template and some similarly named tags, like I have done below.



We can then right click on the background of the template, and select Customizers > Custom Properties, where we will add our Template Parameter. We only need to add one, and name it something that makes sense. In this instance, the template will be of some values from a motor and the parameter that will be passed to it will be the number of the motor, so I called mine MotorNumber and made it of type integer.

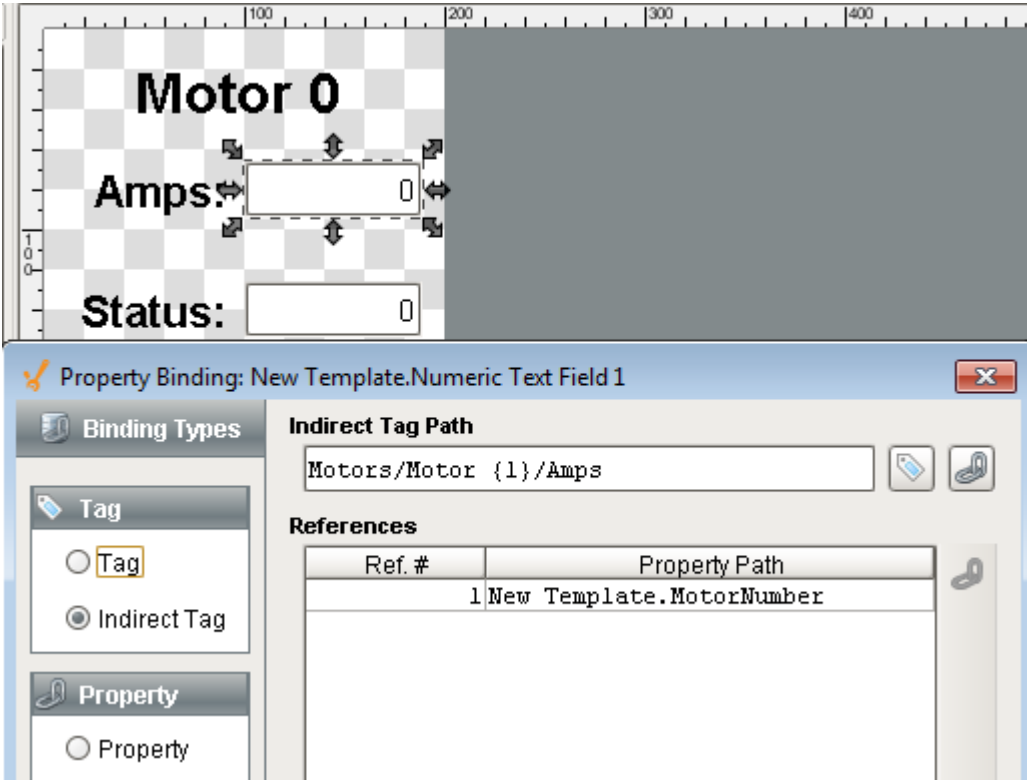


Next, we want to add some components to our template, being sure to utilize the Template Parameter MotorNumber that we created earlier to make them indirect. A Label component at the top with a simple expression binding will clearly display what motor is being shown in the template.

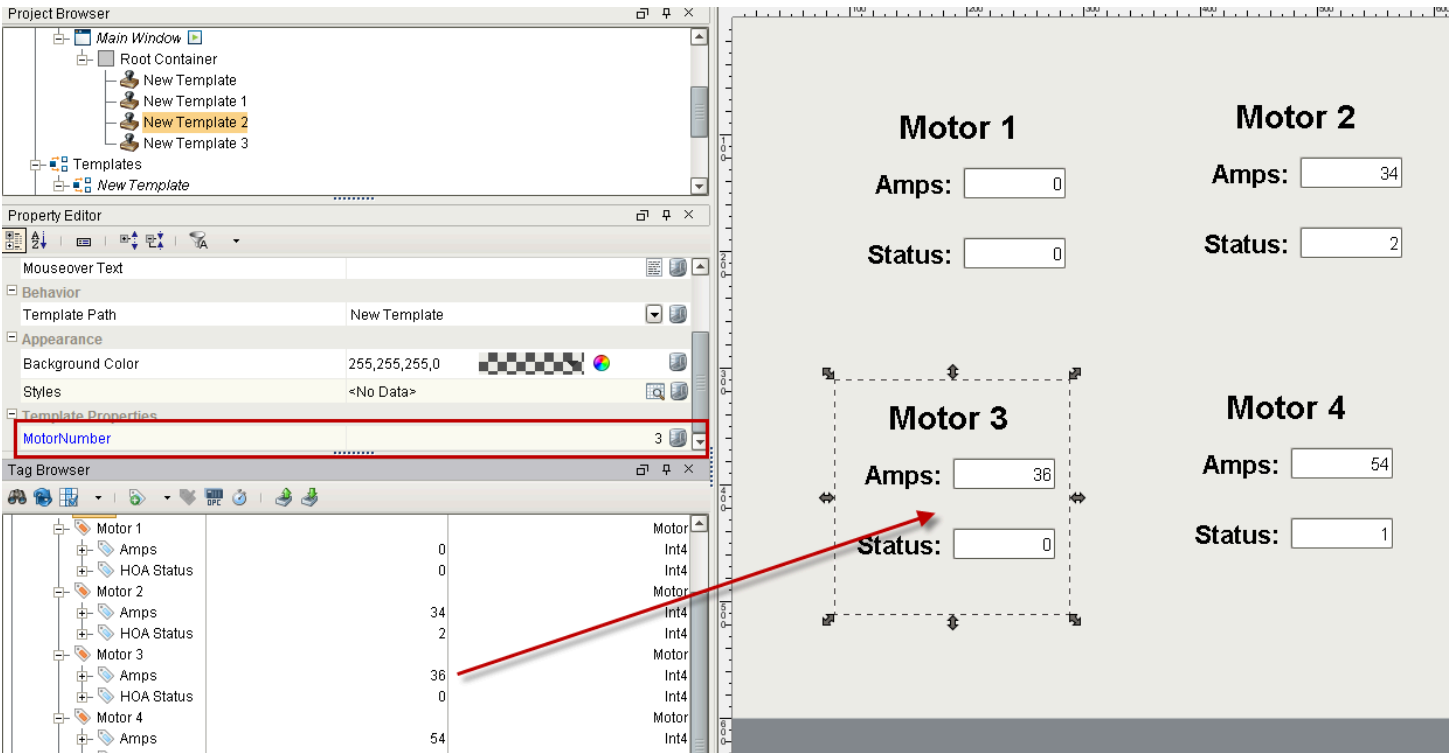


Then we can add two Numeric Text Fields, one for the Amps tag in the Motor UDT and one for the Status tag. I have also added a pair of labels to make it clear what each Numeric Text Field is displaying. We can then add an Indirect Tag Binding to the Value (Integer) property of the Numeric Text Field that will display our Amps tag. We want to bind it to an Amps tag and then in the tag path,

just take out the number of the motor as a reference and attach the MotorNumber Template Parameter that we created earlier. We will want to repeat this on the second Numeric Text Field, except we will indirectly bind it to the HOA Status tag.



The only thing left to do is drag a few instances of the Template onto a window. We can then enter a different value into our MotorNumber 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 MotorNumber property to something, allowing you to easily change what motor the template is displaying in the client during runtime.



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

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

Template - UDT Parameter



INDUCTIVE
UNIVERSITY

Template - UDT Parameter

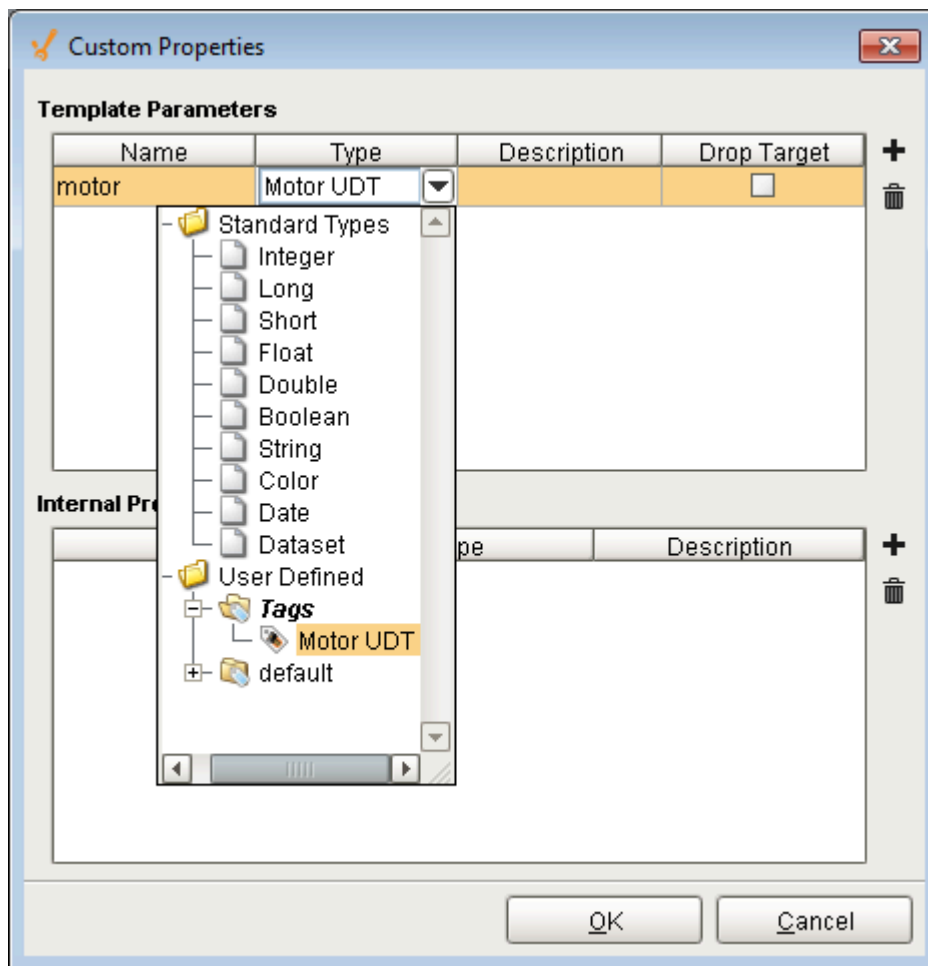
[Watch the video](#)

UDTs in Templates

User Defined Types (UDT) parameters are properties that you can bind the template components to. Since the UDT parameter is a Custom property that encapsulates the UDT tags and some meta properties you can access all this information through a custom property.

To Add a Component to the Template and Bind it to Parts of UDT

1. Start with a blank template.
2. Add components to the template.
3. Create a **Custom** property on the template's Root Container, and make the Custom property's **Type** a **User Defined** type instead of a **Standard** data type and click **OK**.



4. Select a component on the template, in the component's **Property Editor**, click the binding icon next to the property that you want to influence.
5. Select the **Property** binding and choose the tag from the UDT that you would like the binding to occur and click **OK**.

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

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

Template Composition



INDUCTIVE
UNIVERSITY

Template Composition

[Watch the video](#)

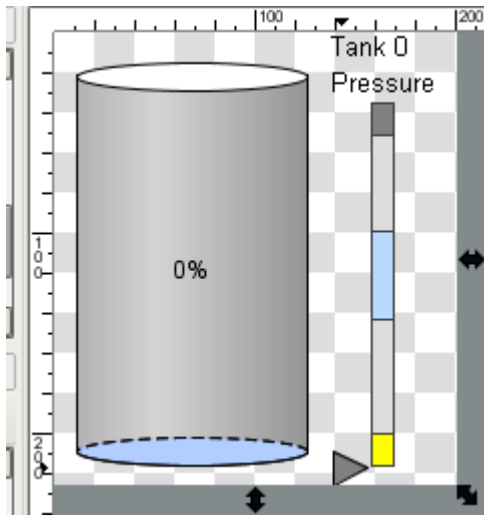
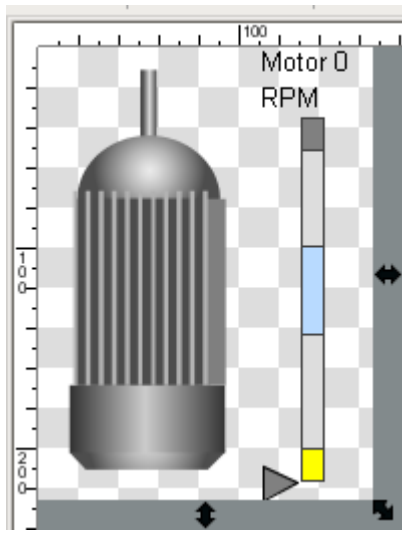
Templates Within Templates

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 guage, a motor with a guage, and a compressor with a guage, it might instead be better to first build a simple guage template that can then be added to each of three templates so that it already is setup correctly.

To Nest a Template Inside Another Template

1. Drag an existing template from the **Project Browser** into the template that you are currently working on.
2. Bind the nested template properties to the properties located on the parent template's **Custom** properties such as UDT parameters.

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.



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

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

Advanced Template Features

Some of the advanced template features in the Designer are: changing Template Path using Template Repeater and [Ignition Exchange](#).

Changing Template Path



INDUCTIVE
UNIVERSITY

Changing Template Path

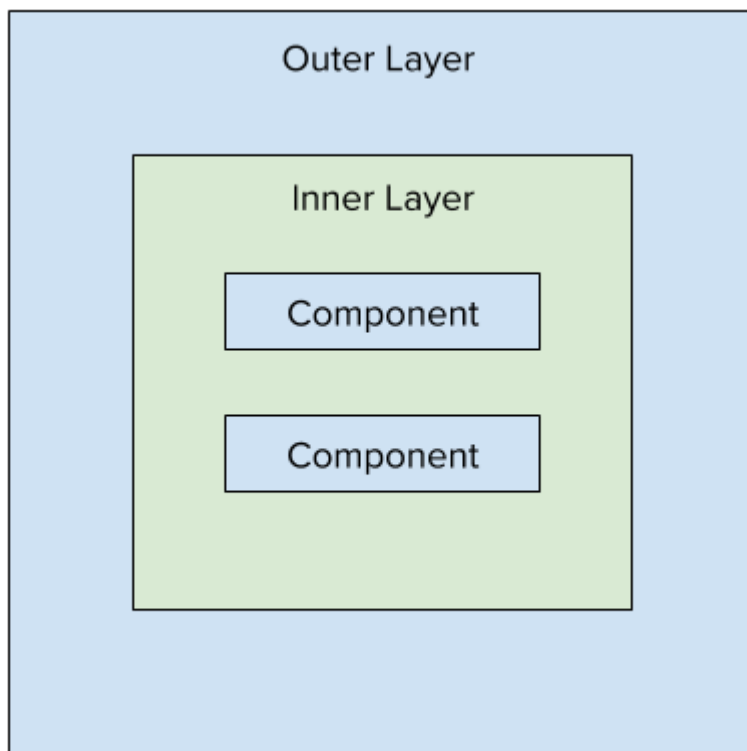
[Watch the video](#)

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 template's property to any string reference that says Tanks/Tank A or Tanks/Tank B.

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
```

Template Repeater



INDUCTIVE
UNIVERSITY

Template Repeater

[Watch the video](#)

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.

! INFO

Additional information on the Template Repeater can be found on the [Template Repeater Component](#) page in the appendix.