Perspective

Perspective is the next generation visualization system for industrial applications, optimized specifically for mobile devices. Perspective puts the power of your plant floor in the palm of your hand by empowering you to create beautiful, mobile-responsive industrial applications that run natively on any mobile device and web browser. The Perspective module has full HMI and SCADA capabilities and marks the beginning of truly mobile-optimized, touch-responsive, easily accessible applications for monitoring, control, analysis and data gathering in industrial systems.

Perspective sits within the existing Ignition architecture so it can take advantage of our years of development and work flawlessly with external databases, PLCs, reports, and so forth. It was designed from the ground up as a means to deliver first-class mobile responsive applications and offer a browser-based design environment.

The following are some key features of the Perspective module:

- **Responsive Design** - Perspective is mobile responsive so it responds to changes in screen size and orientation, giving users a personalized view into their processes that are automatically optimized for whatever device they are on.
- **Browser-based** - Perspective lives in a browser instead of a Java client. No mobile module is required.
- **Device Compatibility** - Designers can create applications that run on any device that can support a modern web browser using any major operating system: Windows, macOS, and Linux. The native App runs on both iOS and Android devices.
- **Designed for Touch** - Perspective employs multi-touch technology to work with touchpad and touch-screen interfaces allowing users to utilize commonly used gestures such as pinching, panning, zooming, and scrolling.
- **Sensor Information** - Perspective is able to take advantage of cameras and GPS information that are natively provided by mobile devices.
- **Cascading Style Sheets HTML5/CSS3 technology** - Enables users to control their application on any device type such as smartphones, tablets, touchscreens, laptops, and desktop computers.
- **Transforms** - Let you easily take the value coming into a binding, manipulate it even further, and then *transform* it to the output of your choice.

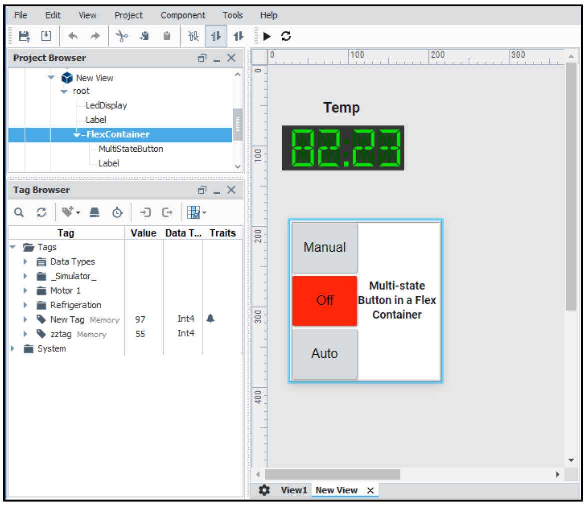
On this page ...

Perspective Project Elements

Watch the Video

Views and Containers

Views and Containers are an integral part of the Perspective design experience because they work together to create your HMI screens, which are the windows into your application. The View is the primary unit of design and the Container provides a way of laying out and organizing child components wit hin a View.



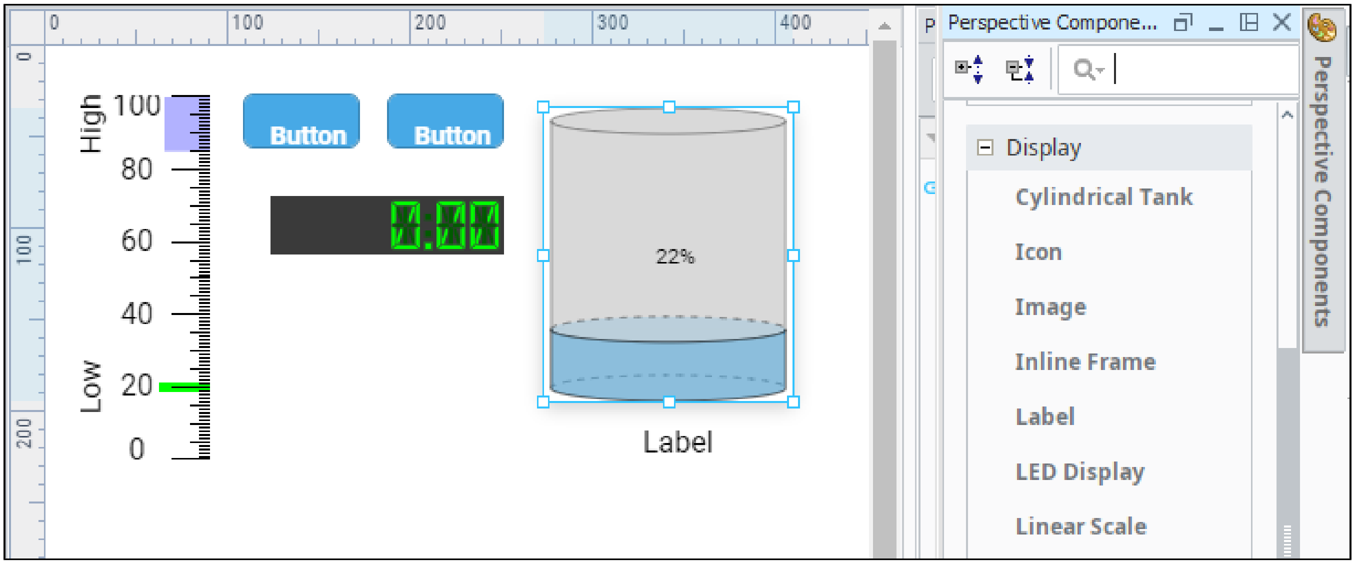
Tag	Value	Data T...	Traits
Tags			
Data Types			
Simulator			
Motor 1			
Refrigeration			
New Tag Memory	97	Int4	🔔
zztag Memory	55	Int4	
System			

Anatomy of a View

Watch the Video

Components

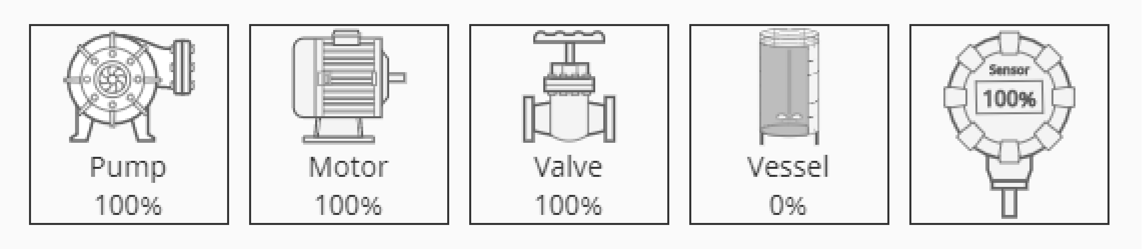
Components are what give you flexibility in designing HMI and SCADA that reflect your company's design and your site's layout. Components are the widgets you deal with every day: buttons, text areas, dropdowns, charts, gauges, linear displays, and so on. The Perspective module comes with a host of built-in components that you can select from for use in your project. There are many ways to manipulate and arrange components when working in the Designer.



The following feature is new in Ignition version **8.1.0**
Click here to check out the other new features

Perspective Symbols

There is a category of components called Perspective Symbols found in the Component Palette. These HMI components are easy to configure using the built-in properties. You have the option of choosing any one of three appearances for the component. Each symbol has a number of states that apply different visual effects. There are properties that allow you to change the size, orientation, appearance, and even select an Auto setting which takes its value from the Session props.

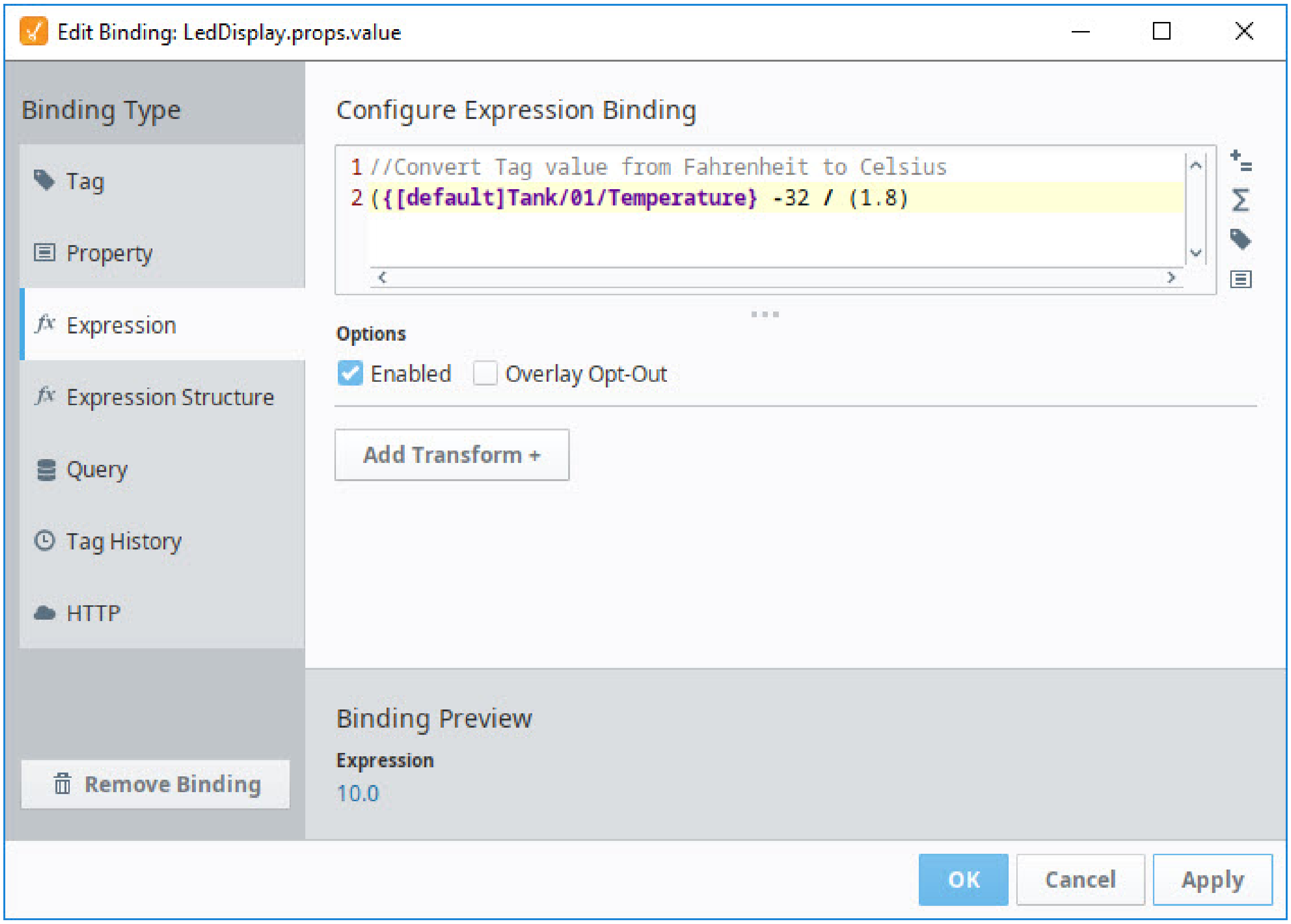


Bindings

A binding is a mechanism that allows a property on a component to change based on a change to a value elsewhere in Ignition. For example, with binding, the liquid level displayed in a tank graphic can be bound to the realtime liquid level in a tank. The value of a Tag could be bound to a linear scale, a meter, or a label on your window. The power of bindings comes from the variety of binding types.

Click on the following links for complete information about binding types:

- **Tag** - Binds a property directly to a Tag, which sets up a Tag subscription for that Tag.
- **Property** - Binds one property to another.
- **Expression** - A powerful type of property binding that uses simple expression language to calculate a value.
- **Expression Structure** - A property binding that uses the property structure to pass data.
- **Query** - A polling binding type that runs a structured Query against database connections.
- **Tag History** - Used for dataset type properties. It runs a query against the Tag Historian.
- **HTTP** - Used for passing data directly to and from a URL link.



Configure Expression Binding

```
//Convert Tag value from Fahrenheit to Celsius
({[default]Tank/01/Temperature} -32 / (1.8)
```

Options

☑ Enabled □ Overlay Opt-Out

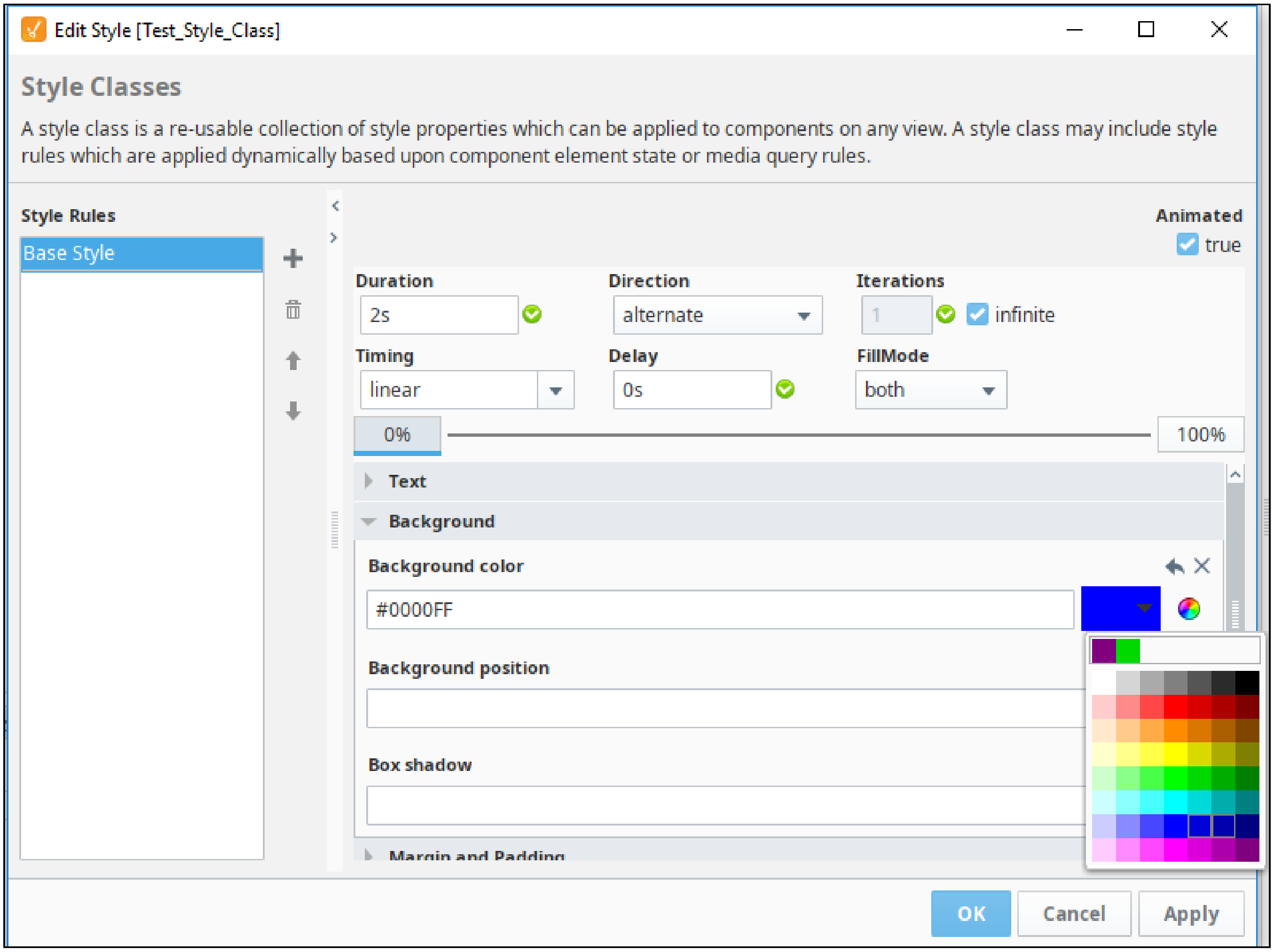
Binding Preview

Expression

10.0

Styles and Style Classes

Perspective gives you the power to style your project in just about any way and easily edit styles across your entire project instantly. You can use powerful and flexible CSS3 styles to change the appearance and position of anything in your application. By combining styles into themes, you'll be able to apply and edit styles across multiple applications in an instant.



Edit Style [Test_Style_Class]

Style Classes

A style class is a re-usable collection of style properties which can be applied to components on any view. A style class may include style rules which are applied dynamically based upon component element state or media query rules.

Style Rules: Base Style

Animated: ☑ true

Duration: 2s

Direction: alternate

Iterations: 1 ☑ infinite

Timing: linear

Delay: 0s

FillMode: both

Background color: #0000FF

Background position

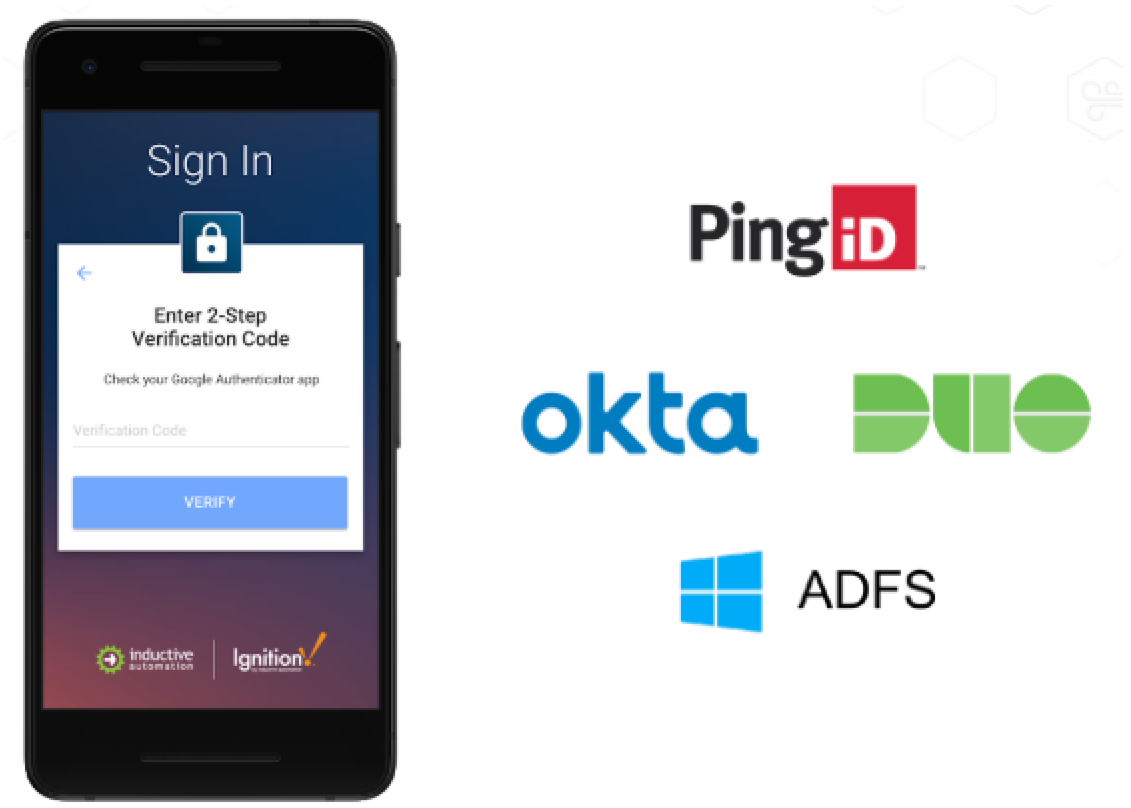
Box shadow

Security in Perspective

Perspective's approach to security covers a wide array of topics, including authentication, authorization and permission modeling, and transport layer s ecurity (TLS).

- Uses single sign-on (SSO) with existing corporate credentials to get access to all of your assigned accounts and applications in one place.
- Clients launched from the Perspective module are secured using cutting-edge encrypting technologies and communication protocols to provide the most secure web-based data transfer.
- Strongly enforced "guest mode" access to prevent against unauthorized writes on the Gateway.
- Integrates with existing corporate identity infrastructure that uses two-factor authentication to verify the identity of users, thus adding extra protection against phishing and brute-force attacks.
- Ignition now has security Levels (instead of/similar to Roles) that are assigned to users. This makes defining permissions simple. The hierarchy of security levels can be used to simplify the security settings, because users with more specific security levels also "inherit" the more general security levels (i.e., a user granted the security level of "Operator / LineB" also has the securitylevel of "Operator").
- Ignition can now use popular authentication methods in addition to Active Directory. It uses trusted federated identity technologies such as SA ML, OAuth, OpenID, and others.

For more information, see Security in Perspective.



Sign In

Enter 2-Step Verification Code

Check your Google Authenticator app

Perspective Co-Branding

The following feature is new in Ignition version **8.1.20**
Click here to check out the other new features

Perspective supports customization, color, and branding options, allowing you to make your Perspective project feel personalized and unique. You can read more about Perspective Co-Branding here.

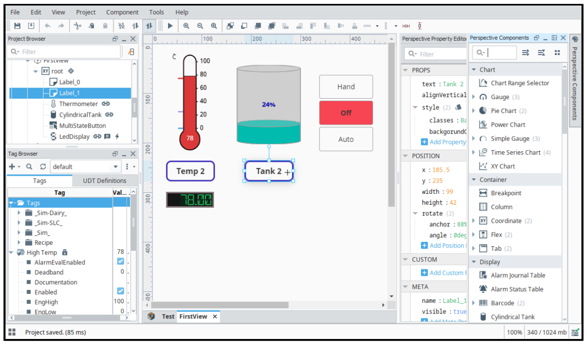
In This Section ...

Perspective Designer Interface

The Perspective Designer Interface includes a number of panels and menus containing functionality that allow you to design and build your project. The Perspective Designer Interface has several basic panels that are used for specific objects such as the Project Browser and Tag Browser. The Project Browser panel allows you to view the different Designer Spaces and their component hierarchies at a glance as you design and build your project, while the Tag Browser panel allows you to browse Tags in the Designer and OPC server as well as create new Tags. Some of the menus, like the File Menu and Help Menu, are shared throughout the Designer, and some are specific to certain objects that will only be displayed when an object of that type is selected.

There are many dockable and draggable panels that surround the workspace, as well as the familiar menu bars and toolbars. The default panels include: Component Palette, Property Editor, Project Browser, and Tag Browser. The dockable panels can be rearranged as you wish and will snap into place as you move them around the screen. Each workspace remembers its layout, which is the docking arrangement of the panels around it.

If you closed a panel and want to get it back, re-enable it from the **View > Panels** submenu. To reset the user interface back to its default arrangement, go to **View > Reset Panels** or click the **Panel Chooser** icon in the lower left corner of the window and select **Reset Panels**.



INDUCTIVE UNIVERSITY

The Designer User Interface

Watch the Video

Perspective Menubar

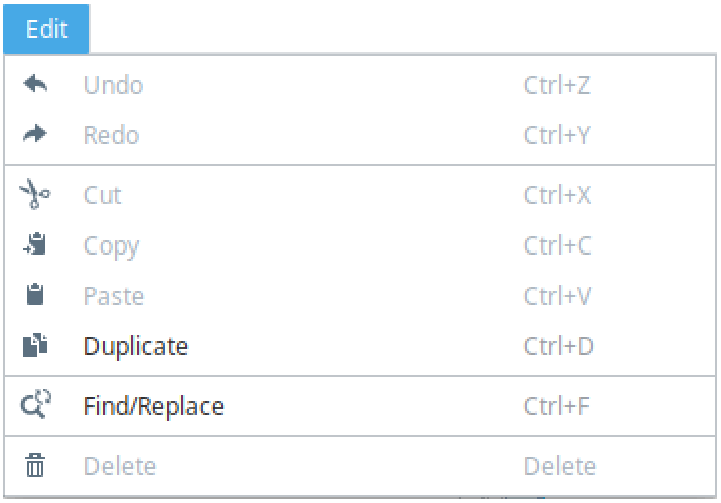
There is a menubar at the top of the Designer that provides functionality that you can interact with when working in the Perspective workspace. Each menu has a host of functions as it relates to that menu. The other menus that are shared between Vision and Perspective are discussed in the General Designer Interface.

File Menu

See General Designer Interface.

Edit Menu

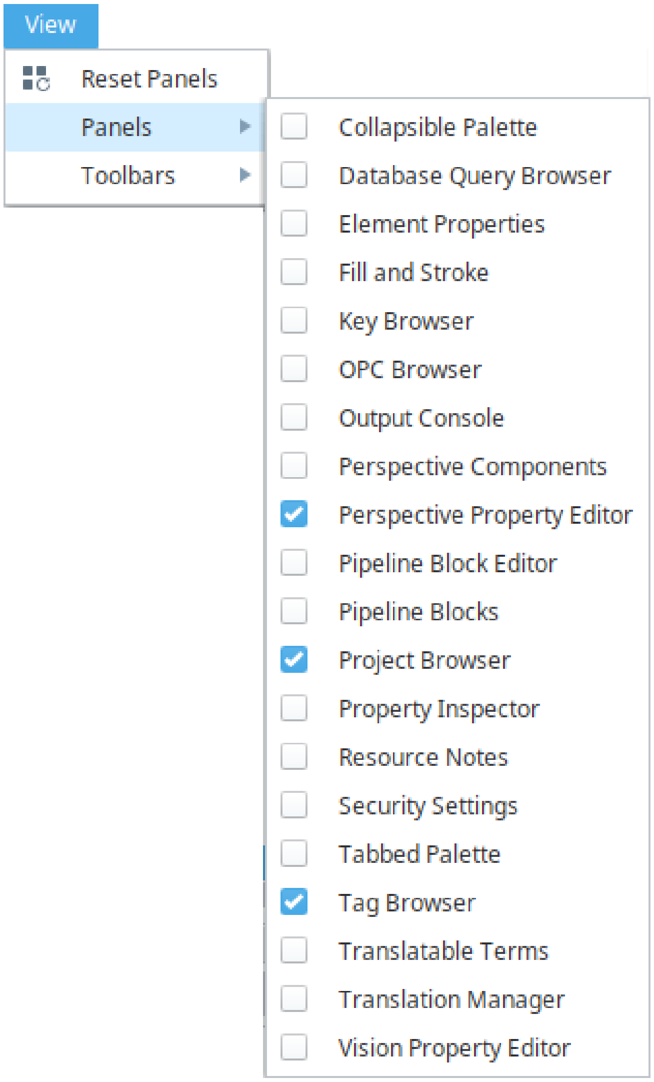
The **Edit Menu** is similar to other applications edit menus in that it provides much of the basic copy/paste functionality. You can also right-click on an item in the browser to access this menu.

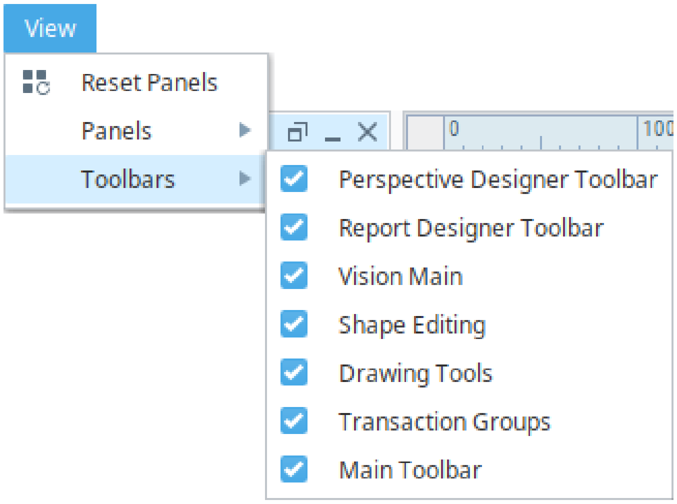


Function	Description
Undo and Redo	Can be used to revert to the previous state, essentially removing the last change, or redoing it again after having been removed. This has a large queue that can be traversed, but does not include every change (i.e., Tag edits cannot be undone).
Cut/Copy /Paste /Duplicate	These functions perform similarly to most software applications. • Cut: Removes the selected item but keeps a copy on the clipboard. • Copy: Puts a copy of the selected item on the clipboard. • Paste: Pastes the current contents of the clipboard. • Duplicate: Duplicates the selected item (essentially a fast copy and paste action). Most things in the Designer can be copied and pasted elsewhere, from individual components to entire Views.
Find /Replace	Brings up the Find and Replace interface to allow you to find specific objects within the project. See Find and Replace for more information.
Delete	Deletes the currently selected component. This can also be done using the delete key.

View Menu

The **View Menu** enables you to control the display of panels and toolbars in the Designer.





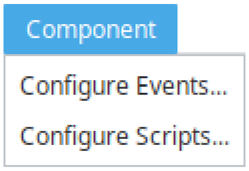
Function	Description
Reset Panels	Resets the Panels in the Designer so that they are in the default configuration.
Panels	A list of all the available Panels for the Designer in Perspective. Select the checkbox next to the Panel name to display that panel. Panel options include the Perspective Property Editor, Project Browser, Tag Browser, and Perspective Components, among others.
Toolbars	A list of all the available Toolbars for the Designer in Perspective. Select the checkbox next to the Toolbar name to display that toolbar. Toolbar options include the Perspective Designer Toolbar, Drawing Tools, Shape Editing, and Main Toolbar, among others.

Project Menu

See General Designer Interface.

Component Menu

The **Component Menu** provides links to the Event Configuration and Script Configuration screens where you can add events, actions, and message handlers to individual components.



Function	Description
Configure Events...	Displays the Event Configuration screen. For more information, see Perspective Events and Actions.
Configure Scripts...	Displays the Script Configuration screen. For more information, see Component Message Handlers.

Tools Menu

Perspective's workspace makes use of the general Tools Menu options. However the **Launch Project** item holds several unique sub options.

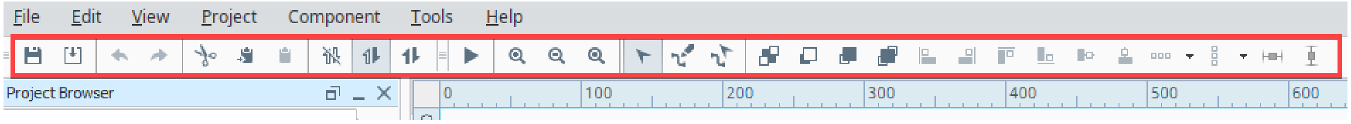
Function	Description
Launch Session	Launches the current project, using the system's default web browser.
External Debugger	Opens Dev Tools, via the designer's JxBrowser, inspecting whichever view was currently being viewed in the designer. **Note:** In cases where the gateway is on the same system as the designer, the external debugger may fail to connect when selecting this option. This issue can be avoided by using the **Copy Debug URL** option, and replacing instances of "localhost" with "127.0.0.1", which would look something like the following: `http://127.0.0.1:9222/devtools/inspector.html?ws=127.0.0.1:9222/devtools/page /####################` **This feature was removed from Ignition in version** **8.1.37** As of version 8.1.37, the option to copy the debug URL has been removed.
Copy Debug URL	Copy's the URL used by the External Debugger. This can be useful in cases where Dev Tools is having trouble starting via the External Debugger option.

Help Menu

See General Designer Interface.

Perspective Toolbar

The Perspective Toolbar contains shortcuts to options from the menubar as well as options to set the z-order for components.



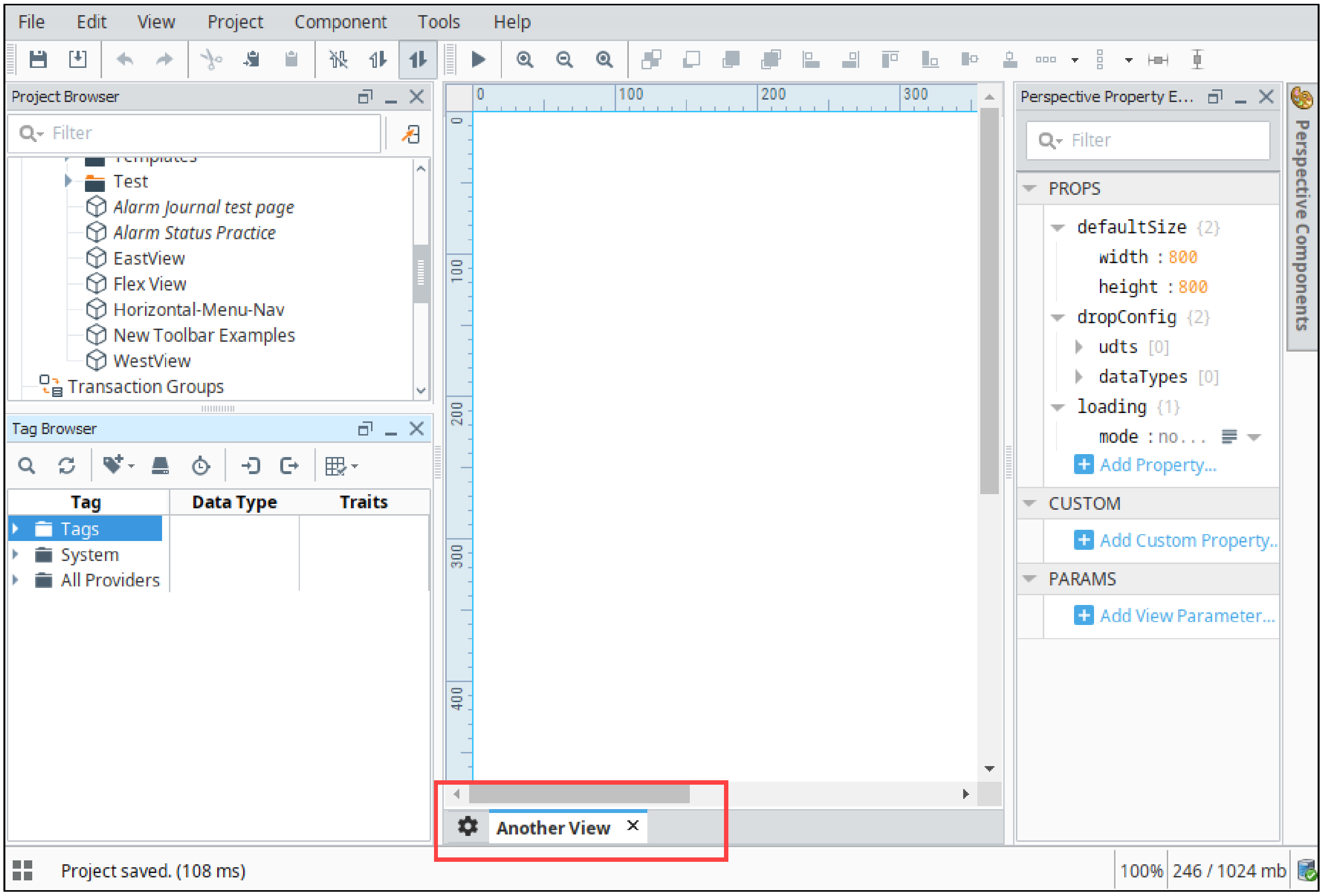
Icon	Function/Description
💾	Save all outstanding project changes in Ignition Gateway.
📥	Merges any new changes on the Gateway into the open project.
↶	Undo the last action.

↷	Redo the last undo action.
✂	Cuts the current selection into the clipboard.
⎘	Copies the current selection into the clipboard.
📋	Pastes the current selection into the clipboard.
⇅	Gateway communication such as queries and Tag subscriptions disabled.
⇅	Read-only communication operations such as SELECT queries and Tag values allowed.
⇅	Full read-write Gateway communication allowed
▶ ■	Toggle the active view between Preview mode and Design mode.
🔍	Zoom into the currently open window.
🔍	Zoom out of the currently open window.
🔍	Zoom reset to 100%.
➤	The following feature is new in Ignition version **8.1.10** Click here to check out the other new features Toggles the component selection tool, allowing the mouse to select and interact with components. Disabled when a container other than a coordinate container is deeply selected.
✎	The following feature is new in Ignition version **8.1.10** Click here to check out the other new features Toggles the Pipe Draw Tool, allowing the mouse to draw pipe connections. Disabled when a container other than a coordinate container is deeply selected.
➤	The following feature is new in Ignition version **8.1.10** Click here to check out the other new features Toggles the Pipe Move Tool, allowing the mouse to move existing pipes. Disabled when a container other than a coordinate container is deeply selected.
⧉	Move the selected components to the back of the z-order.
⧉	Move the selected components backward in the z-order relative to the overlapping components.
⧉	Move the selected components forward in the z-order relative to the overlapping components.
⧉	Move the selected components to the front of the z-order.

The remainder of the icons on the Perspective toolbar are for aligning components. See aligning for more information.

Resource Tabs

The Resource Tabs allow you to change which resource is being edited in the workspace, as well as navigate to the **Settings** ⚙ area of the Perspective Workspace.

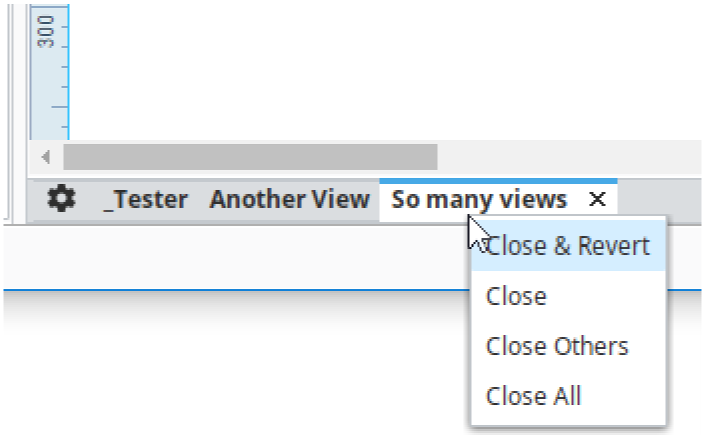


defaultSize {2}
width : 800
height : 800
dropConfig {2}
udts [0]
dataTypes [0]
loading {1}
mode : no...

Project saved. (108 ms)

Right-Click Menu

The right-click menu provides multiple ways to close one or more views.

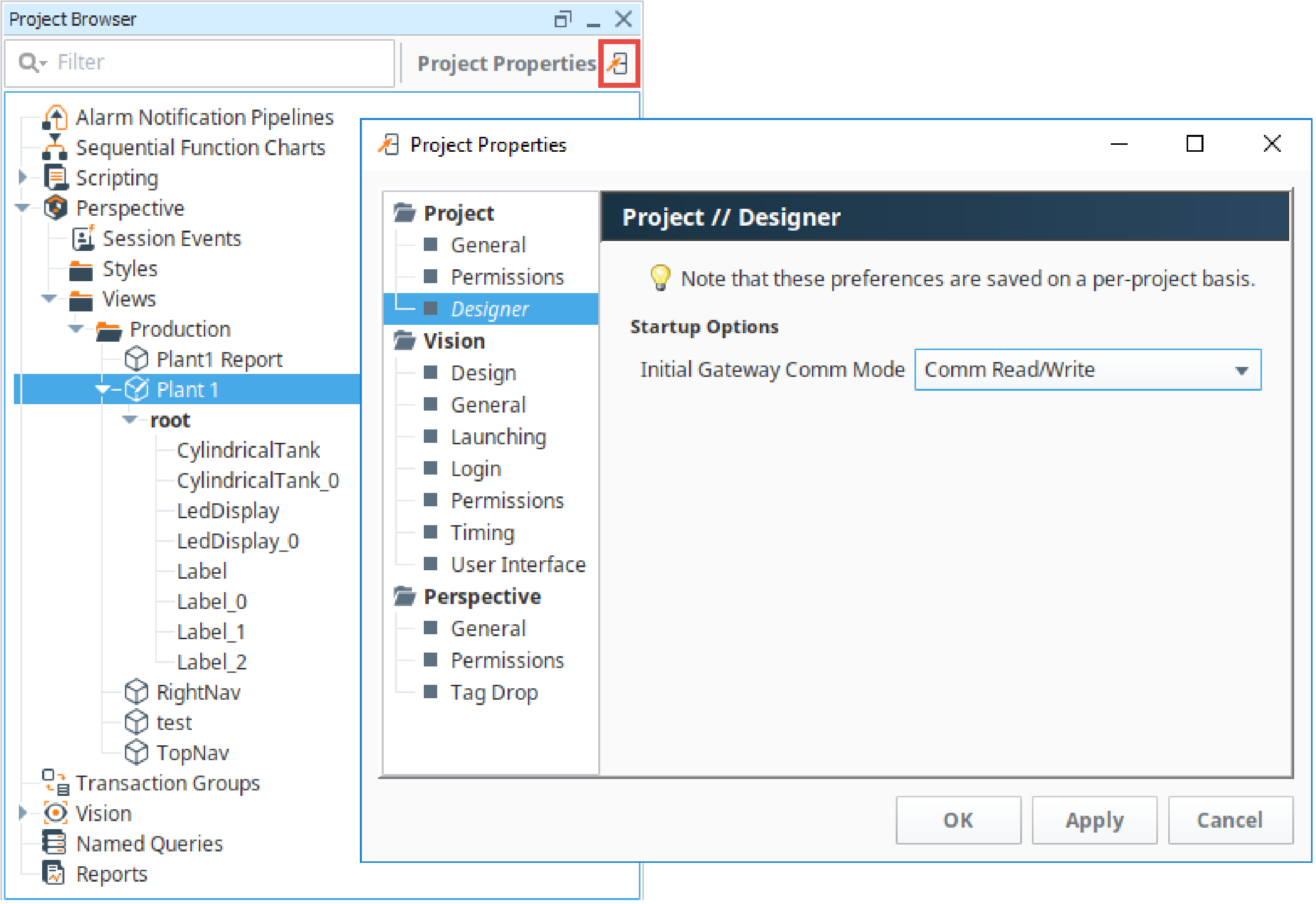


Option	Description
Close & Revert	Closes the selected view, and reverts any changes that have been made, but not yet saved. This option should only be used when you wish to lose any changes made to the view.

Close	Closes the selected view, and commits changes (meaning changes will be retained in the designer session).
Close Others	Closes all views except for the view selected. Changes made to the views are committed.
Close All	Closes all views. Changes made to the views are committed.

Project Browser

The Project Browser panel allows you to view the different Designer Spaces and their component hierarchies. You can expand folders and navigate down through each folder to see elements of a project such as windows, views, containers, and components. The Project Browser shows the entire tree structure from the project level folder down to the component level. You can view or change many of the project properties settings by clicking the **Project Properties** icon. The Project Browser is also discussed on the General Designer Interface page.



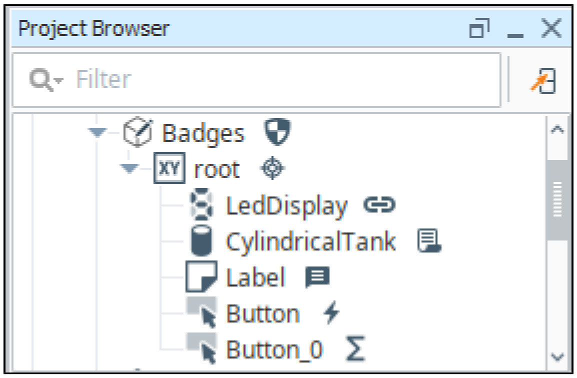
Note that these preferences are saved on a per-project basis.

Startup Options

Initial Gateway Comm Mode: Comm Read/Write

Badge Icons

Perspective has host of useful icons in the Project Browser that can show extra configurations on a component, such as scripting, security, deep selection, message handler, etc. These icons, as shown in the image below, are extremely useful when trying to navigate through a view to find components with extra configurations. Here is a list of the Perspective Badge icons.



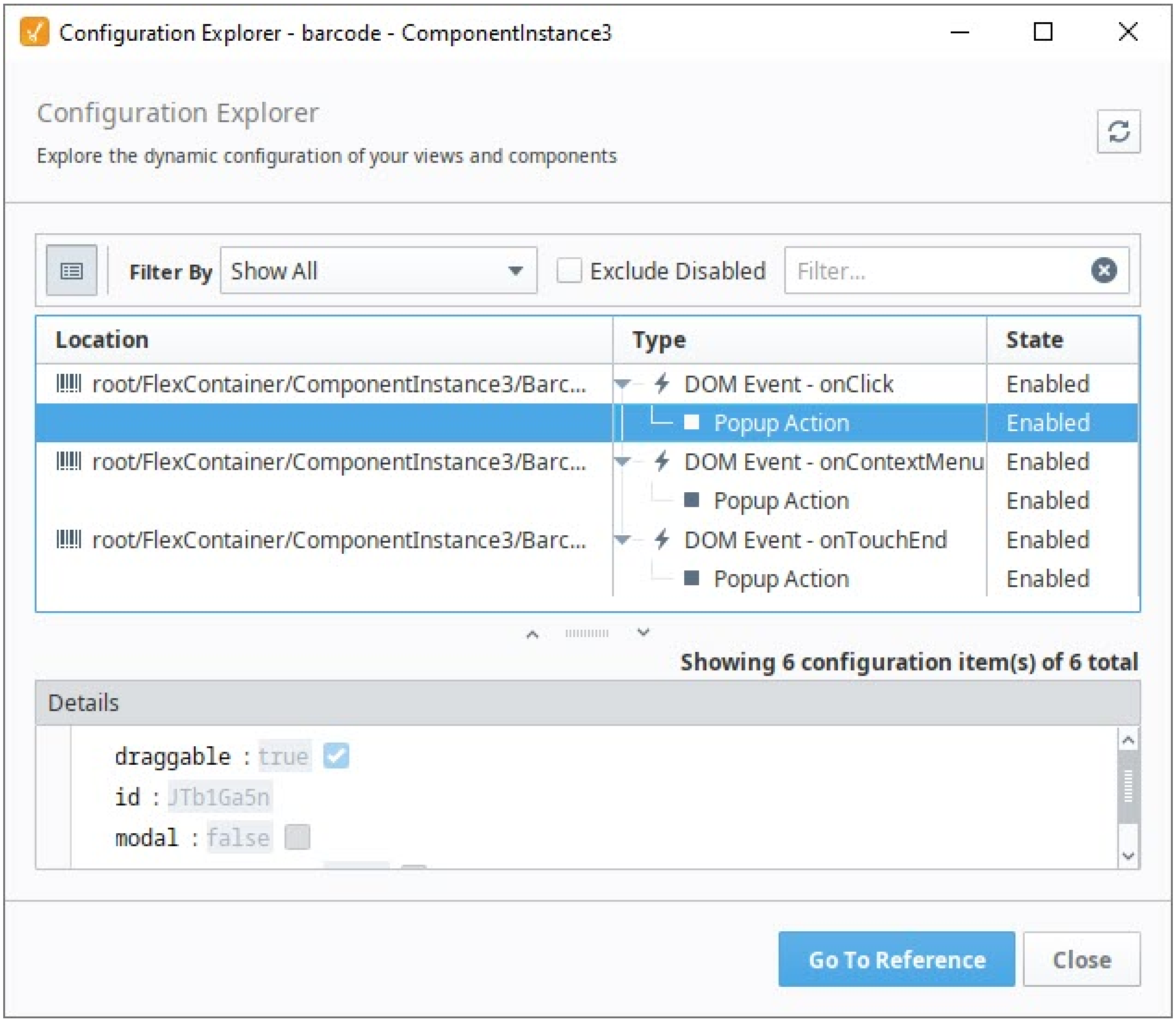
Badge Icon	Name	Description
🔗	Binding	Indicates the component has a binding script. Appears next to the name of the component.
Σ	Custom Method	Indicates the component has a custom method. Appears next to the name of the component.
⌖	Deep Select	Shows a selected component within a selected container. Indicates you have deeply selected into the component and not just selected the component itself. Appears next to the name of the selected component.
⚡	Event Action	Indicates that the component has its own event actions which are related directly to the functionality of the component. Appears next to the name of the component.
💬	Message Handler	Shows that the component contains a user-created script that listens for a particular message. Appears next to the name of the component.
📜	Script	Identifies that the component has a property change script. Appears next to the name of the component.
🛡	Security	Indicates that the component has security permissions applied to the view. Appears next to the name of the component.

Configuration Explorer

The following feature is new in Ignition version **8.1.22**
Click here to check out the other new features

The Configuration Explorer allows you to view the dynamic configuration of all of your views and components. Right-click on a view or component in the Project Browser and select **Configuration Explorer** to examine the following configurations:

- Bindings and Transforms
- Custom Methods
- Embedded Views
- Events and Actions
- Extension Functions
- Message Handlers
- Property Change Scripts


Configuration Explorer - barcode - ComponentInstance3

Configuration Explorer

Explore the dynamic configuration of your views and components

Filter By Show All □ Exclude Disabled Filter...

Location	Type	State
root/FlexContainer/ComponentInstance3/Barc...	DOM Event - onClick	Enabled
	Popup Action	Enabled
root/FlexContainer/ComponentInstance3/Barc...	DOM Event - onContextMenu	Enabled
	Popup Action	Enabled
root/FlexContainer/ComponentInstance3/Barc...	DOM Event - onTouchEnd	Enabled
	Popup Action	Enabled

Showing 6 configuration item(s) of 6 total

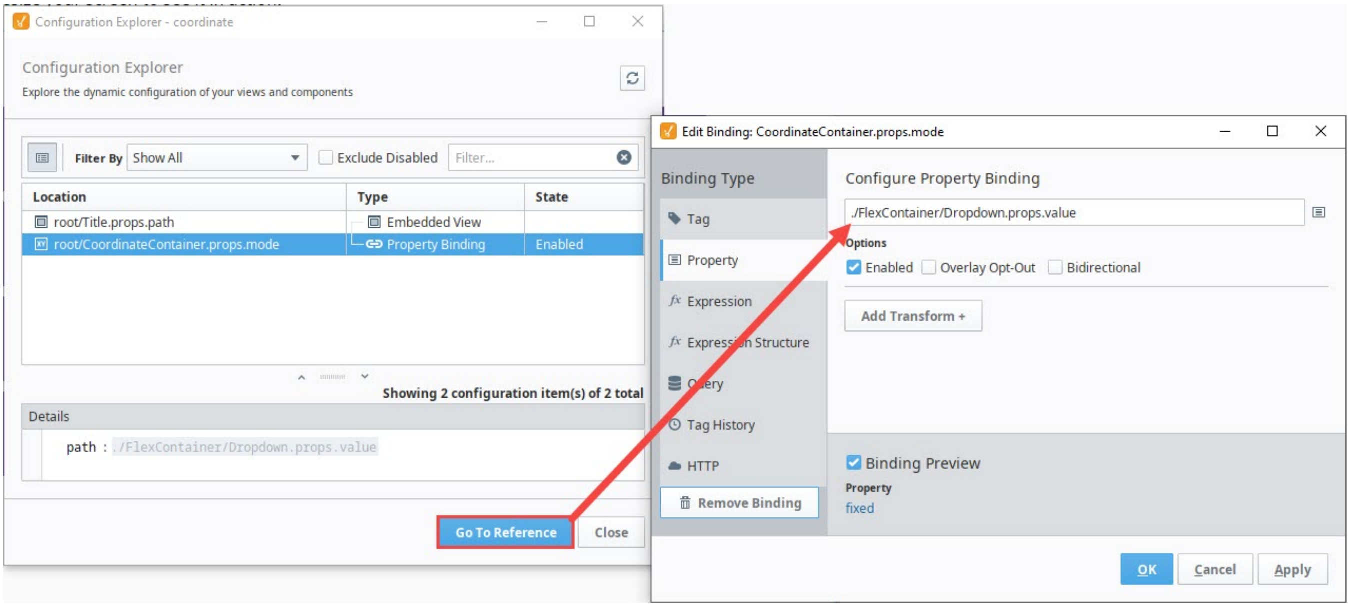
Details

draggable : true ☑

id : JTb1Ga5n

modal : false □

Select a configuration and click **Go To Reference** to highlight the relevant view or component in the Project Browser and, if applicable, open an Edit window to view details about the selected configuration.


Configuration Explorer - coordinate

Configuration Explorer

Explore the dynamic configuration of your views and components

Filter By Show All □ Exclude Disabled Filter...

Location	Type	State
root/Title.props.path	Embedded View	
root/CoordinateContainer.props.mode	Property Binding	Enabled

Showing 2 configuration item(s) of 2 total

Details

path : ./FlexContainer/Dropdown.props.value

Edit Binding: CoordinateContainer.props.mode

Configure Property Binding

./FlexContainer/Dropdown.props.value

Options

☑ Enabled □ Overlay Opt-Out □ Bidirectional

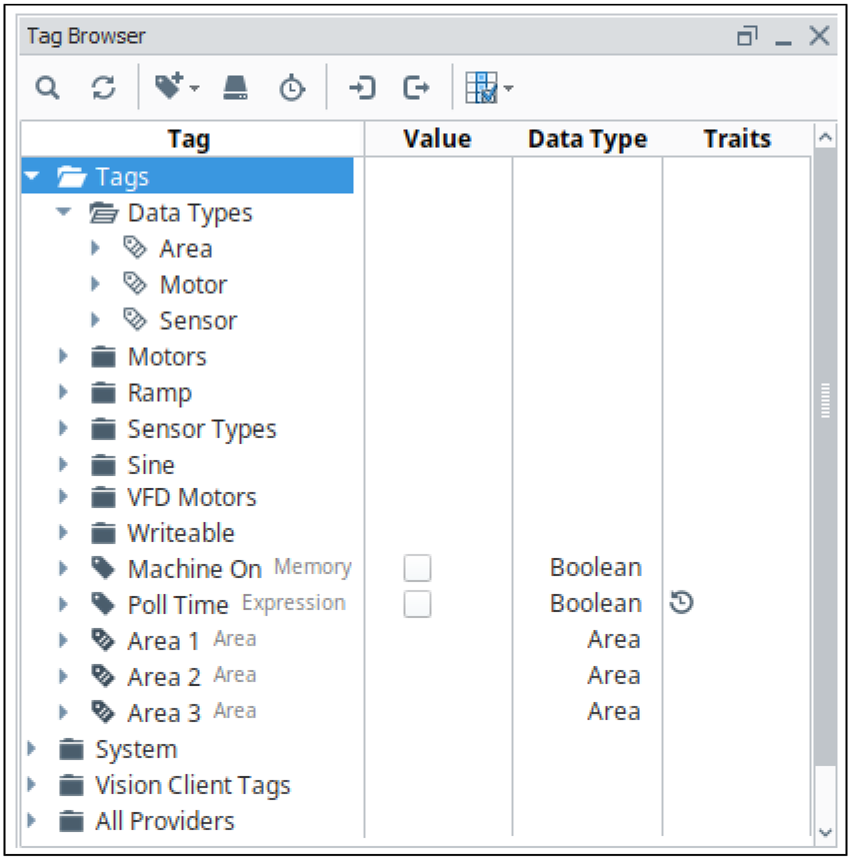
☑ Binding Preview

Property

fixed

Tag Browser

The Tag Browser allows you browse Tags in the Designer and OPC servers. In addition, Tags can be created, edited, exported and imported directly from the Tag Browser. For more information on Tags, the different Tag types, and how they work, see Tag Browser.



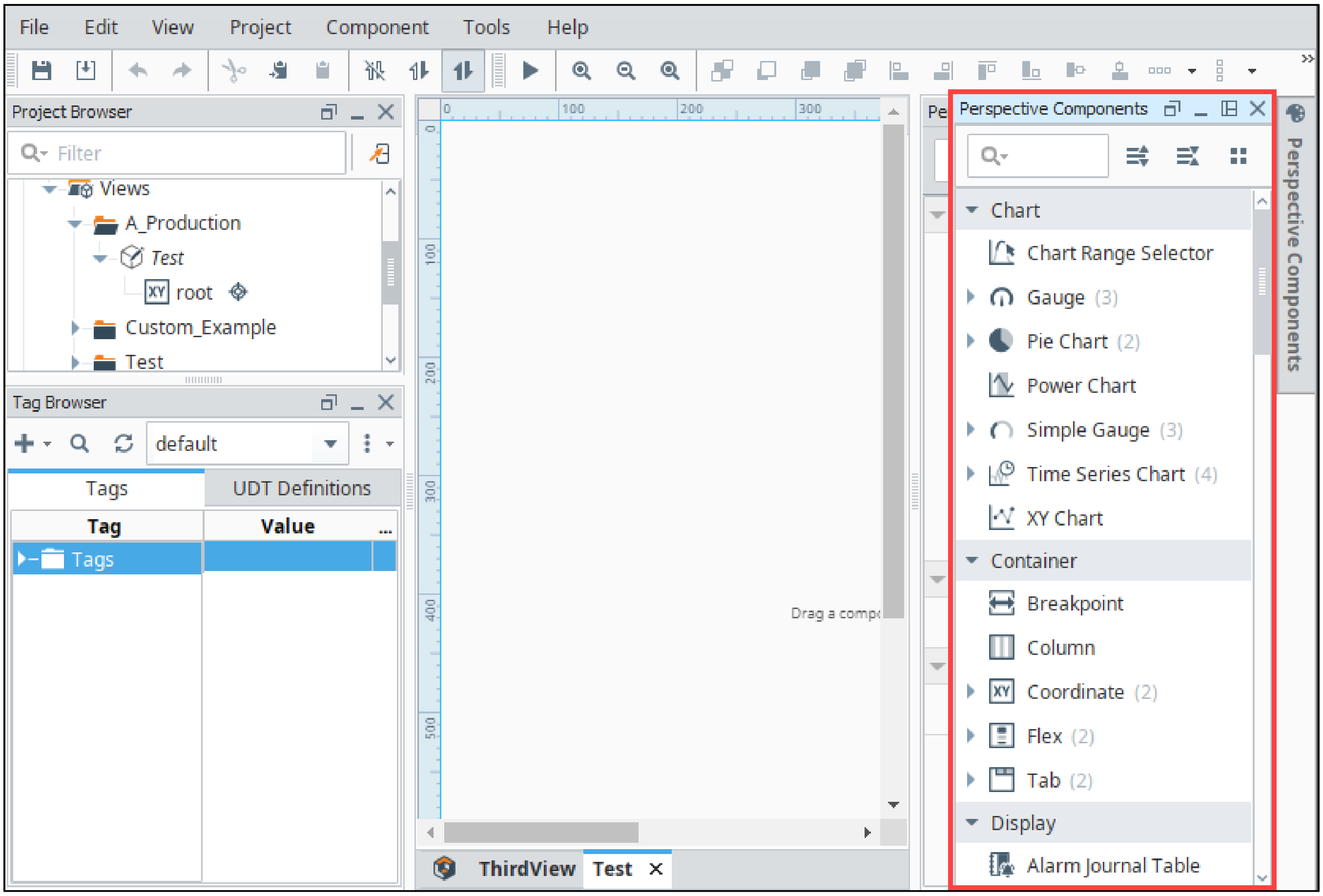
Tag Browser

Tag	Value	Data Type	Traits
Tags			
Data Types			
Area			
Motor			
Sensor			
Motors			
Ramp			
Sensor Types			
Sine			
VFD Motors			
Writeable			
Machine On Memory	☐	Boolean	
Poll Time Expression	☐	Boolean	⟲
Area 1 Area		Area	
Area 2 Area		Area	
Area 3 Area		Area	
System			
Vision Client Tags			
All Providers			

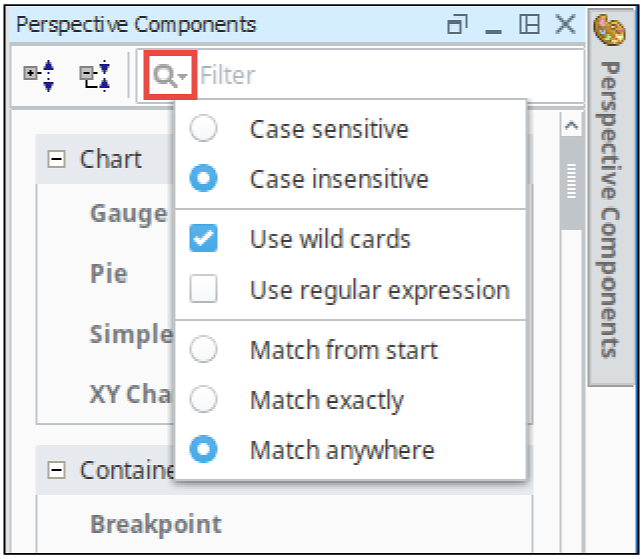
Perspective Component Palette

The Perspective module contains numerous components, such as buttons, labels, and charts. Perspective utilizes modern web technologies so many of the built-in components may look reminiscent of components you may have seen on your favorite websites. Components are created by dragging the component from the Component Palette and dropping it onto a view. A complete list of components is found on the Perspective Components page.

The Component Palette is located in the upper right side of the Perspective Designer Interface. If the component palette is not visible, click Perspective Components to open. This panel defaults to auto-hide itself. Components are grouped into different categories based on functionality. Each category can be individually expanded to show all components in that category, or collapsed to hide the components in that category.



In the Component Palette search bar, you can search for components. Start typing in the search bar and the component list will update based on the text you entered. You can also set filter options by clicking on the **Search** 🔍 icon to expose the filter selection to make the search case sensitive, or use wildcards, regular expressions, match from start, match exactly, and match anywhere.



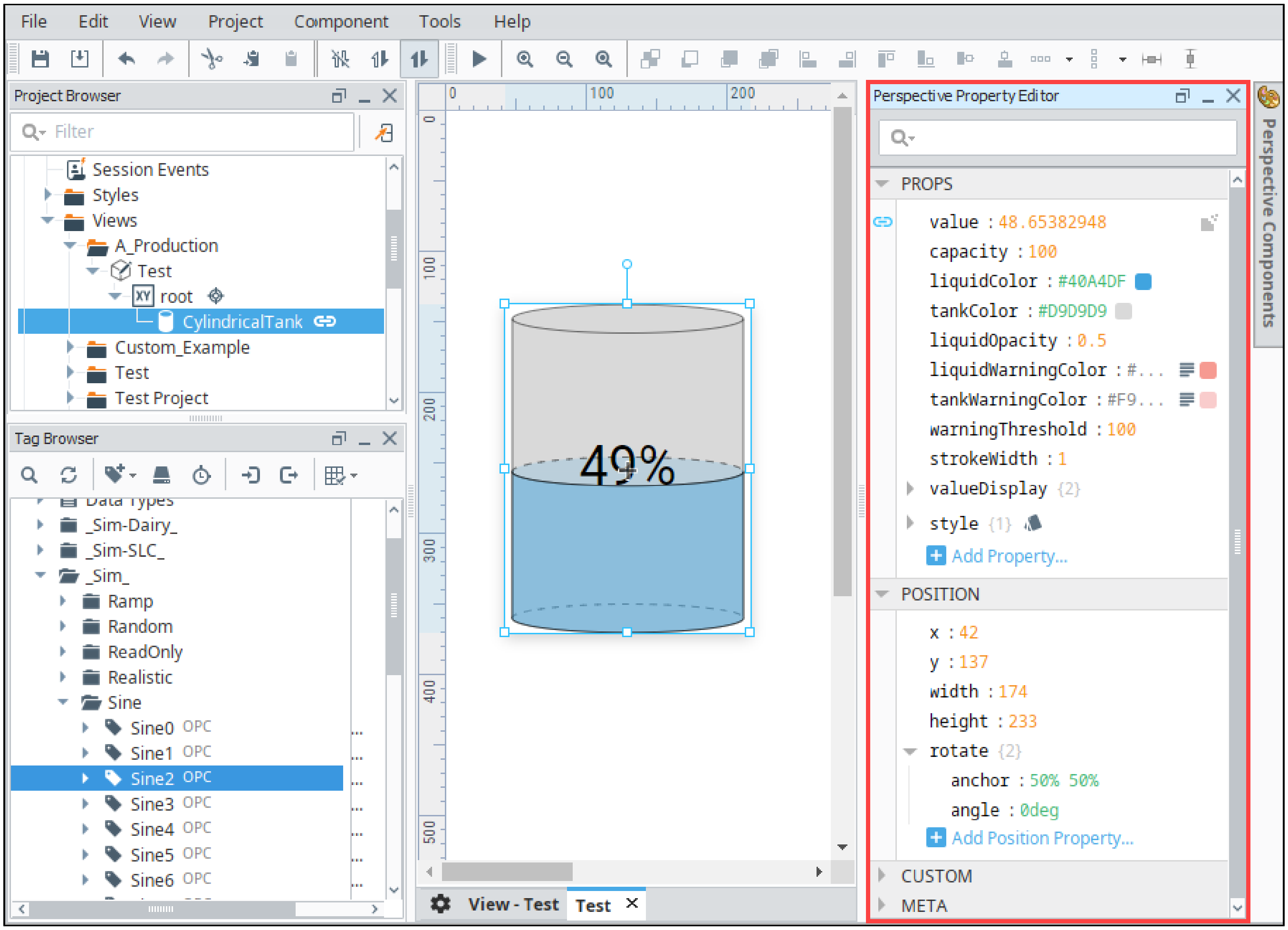
○ Case sensitive
● Case insensitive
☑ Use wild cards
□ Use regular expression
○ Match from start
○ Match exactly
● Match anywhere

Perspective Property Editor

As with other areas of Ignition, the Perspective module has a dedicated Property Editor panel that allows for fine-tuning of individual objects. The Property Editor is contextual, meaning that the properties available are dependent on which object or component is selected.

The Property Editor contains a search field, allowing you to filter properties based on the text typed into the field. The image below shows the properties that are set on a Cylindrical Tank component. There are **Binding** icons to the left of each of the properties that appear when you mouse over them. If you have a Cylindrical Tank on your view and click the Binding icon for the Value property, it will open the Binding window and you can set what the Cylindrical Tank component is bound to. In following example, the Value property of the Tank is bound to a Sine2 OPC Tag.

In the Property Editor you can also set Position properties and add Custom properties to components, as well as create Params for passing values from one view to another. To learn more about each of the categories in the Property Editor, refer to the Perspective Component Properties page.



PROPS

value : 48.65382948

capacity : 100

liquidColor : #40A4DF

tankColor : #D9D9D9

liquidOpacity : 0.5

liquidWarningColor : #...

tankWarningColor : #F9...

warningThreshold : 100

strokeWidth : 1

valueDisplay {2}

style {1}

Add Property...

POSITION

x : 42

y : 137

width : 174

height : 233

rotate {2}

anchor : 50% 50%

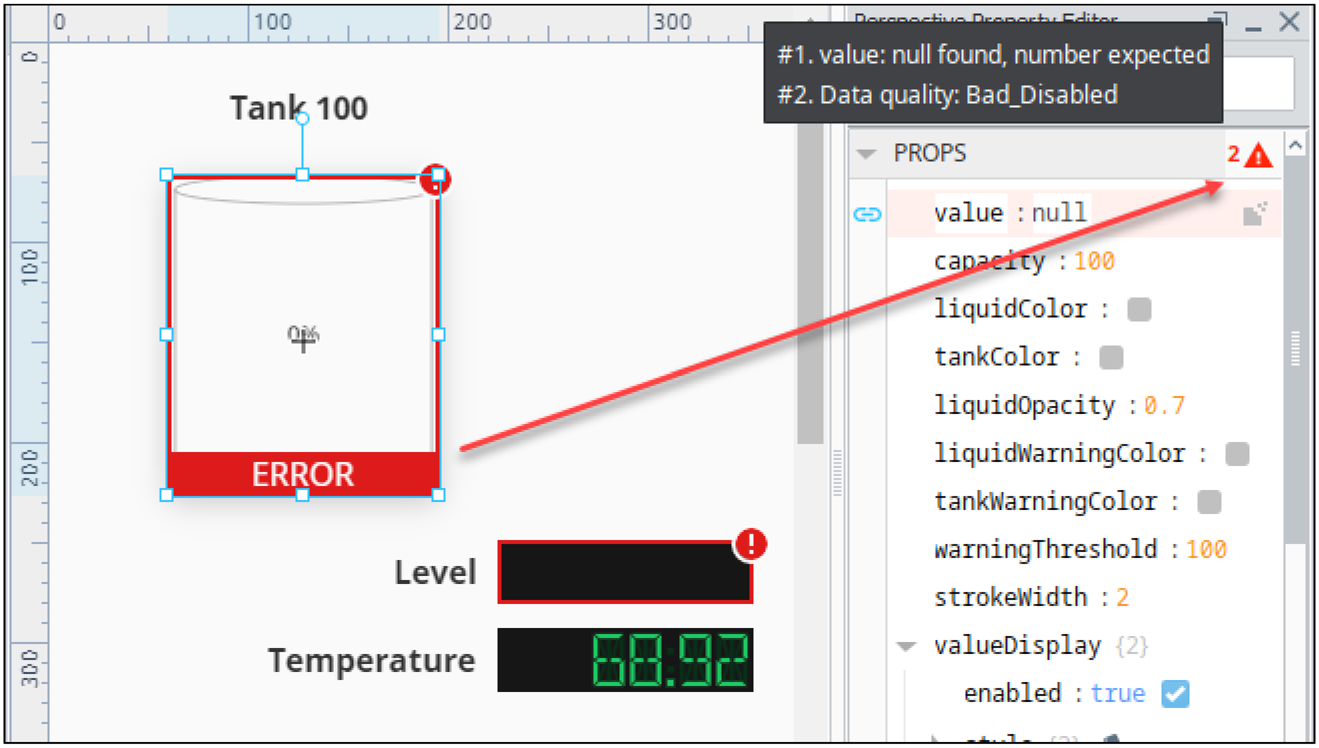
angle : 0deg

Add Position Property...

CUSTOM

META

The Property Editor will allows show warnings if there an an issue with the selected component. This is commonly used to show when the quality on a property is non-good, or if there is some other configuration issue with a property. Clicking the red triangle in the upper right corner of the property category will display additional information.



#1. value: null found, number expected
#2. Data quality: Bad_Disabled

```
value : null
capacity : 100
liquidColor :
tankColor :
liquidOpacity : 0.7
liquidWarningColor :
tankWarningColor :
warningThreshold : 100
strokeWidth : 2
valueDisplay {2}
    enabled : true ☑
```

Page Configuration

Pages are how you navigate within a Perspective project. Each page represents a collection of views that are displayed in a single space. Just like a single tab of a web browser, this represents a single page (at a time). Before you begin to configure a page, it's important to understand the page layout. It has several specific regions and each behave differently: Top Dock, Bottom Dock, Left Dock, Right Dock, and a Primary View. The type of content you create and the design strategy you use for your views will determine where you place them on the page layout. For example, you may want a staff schedule to be available on every page, but not displayed all the time on the page. What you could do is configure the view on a docked window, thus making it available on demand when someone wants to see it by clicking on a tab in a sesson to view it, and clicking the tab again to hide it. Pages in Perspective describes in detail about page layout, page configuration, configuring docked views, and more.

Open the Page Configuration window by clicking on the **Settings** ⚙ icon in the lower left corner. This is where you'll configure your pages in Perspective.

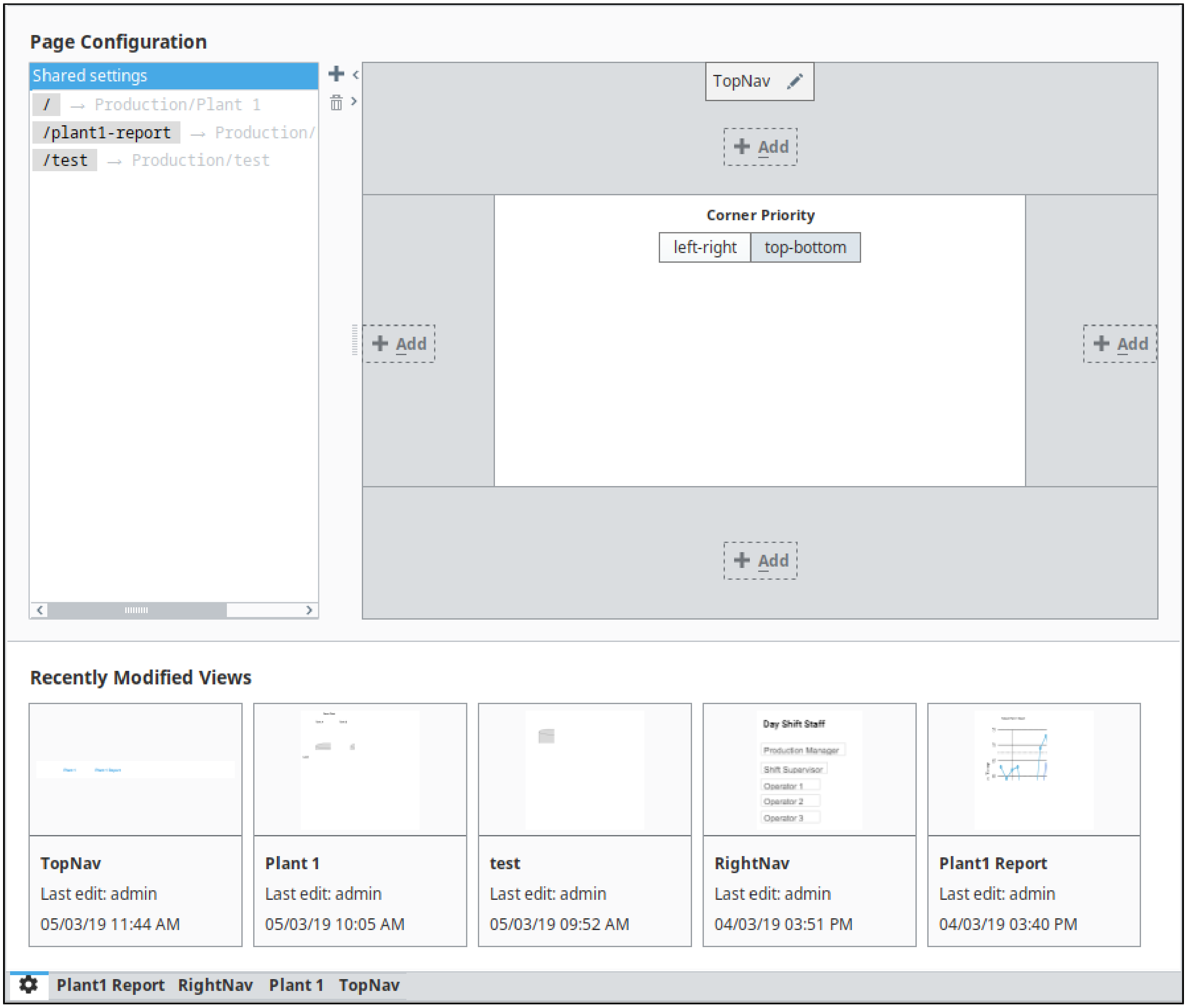
Initially, **Shared Settings** allow you to apply configuration changes to all pages (such as adding an omnipresent docked view), as well as specify the **C orner Priority**.

To the right of the Page Configuration column, the following buttons are present:

- **Add** - Creates a new Page Configuration.
- **Trash** - Removes the selected Page Configuration.

Recently Modified Views

You'll also notice in the Page Configuration image below, there is a Recent Modified Views list. These are your most recent modified views along with a timestamp denoting when the last edit to the view was made and who made the modifications. If the project does not contain any views, the listing will be empty.



Keyboard Shortcuts

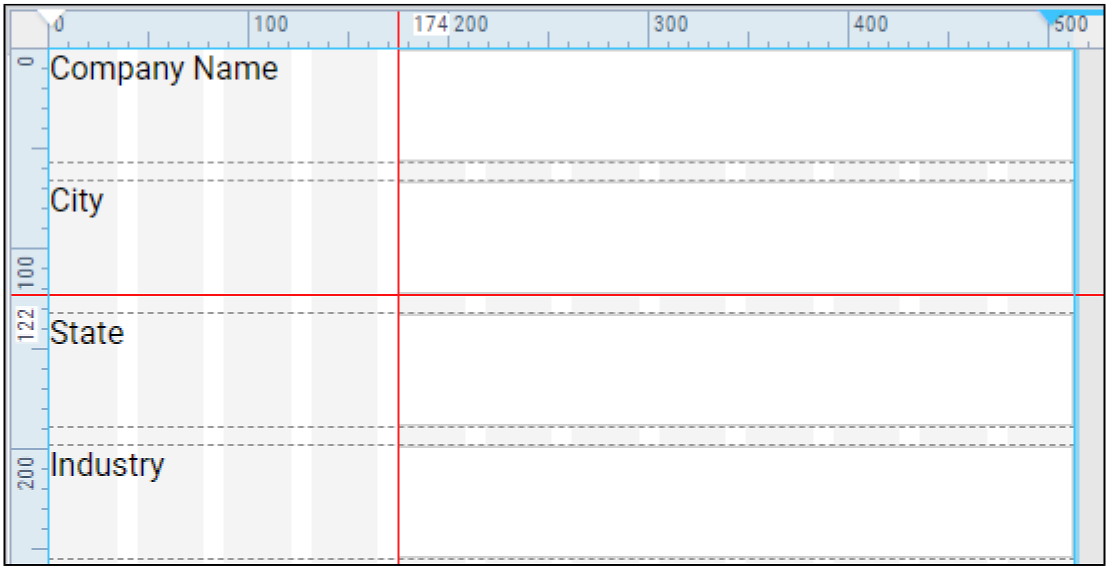
There are a lot of ways to speed up development once you are familiar with how Ignition works. There are many keyboard shortcuts in Perspective Designer that are listed throughout the Designer interface alongside menu options. To learn about keyboard shortcuts, go to the Windows, Linux, and Mac Keyboard Shortcuts page.

Vertical and Horizontal Guides

In the Designer workspace, you can set vertical and horizontal guides to help you align components. To set a vertical guide, slide your cursor along the top horizonal ruler and click it where you want it. The number of pixels will be displayed in the top ruler and a red vertical line will appear the length of your workspace. To remove the guide, click on the guide and drag it to the left into the vertical ruler and it will disappear.

To set a horizontal guide, slide your cursor in the vertical ruler and click it where you want it. The number of pixels will be displayed in the left ruler and a red horizontal line will appear on the width of your workspace. To remove the guide, click on the guide and drag it to the top into the horizontal ruler and it will dsappear.

You can add multiple vertical and horizontal guides.



The Perspective Designer toolbar also has options for aligning components.

Related Topics ...

- Perspective Components
- Perspective Component Properties
- General Designer Interface
- Pages in Perspective

Perspective Sessions

A Perspective **Session** represents one instance of a project running on a web browser or mobile app. The term "session" might be new to some users, where "client" or "runtime" may sound more familiar. So anytime we refer to a session, we are talking about a runtime application in Perspective.

Now that you know that a session is effectively running an instance of the application, it can have multiple **Pages** in the same web browser. Perspective Sessions work the same way as web pages do on your favorite website on the internet. You can log into your favorite shopping website and then open several new tabs, where each tab is aware of your user credentials without having to retype them.

A session appears fairly simplistic, but there is actually a rich hierarchy of elements to be aware of. This page details the various moving pieces that comprise a session.

On this page ...

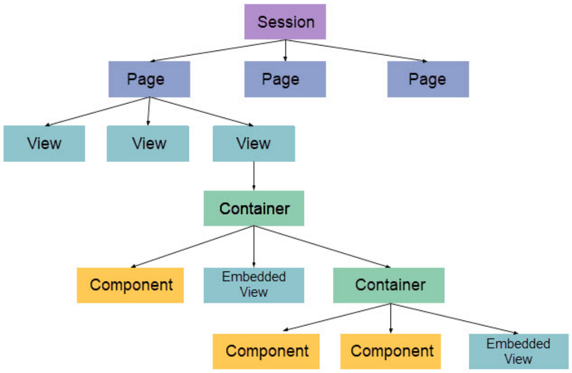
Perspective Project Elements

Watch the Video

Session Anatomy

A Perspective Session contains a hierarchy of logical elements, as shown in the example below.

A session can have any number of pages at a time, pages can have any number of views, and containers can have any number of components, including other containers as children, which can also be nested to support multiple layout strategies. In this section, we discuss each building block of this hierarchy, starting at the bottom and working our way up. When describing each element, it helps to understand the smallest piece first which is a component, and then move up the hierarchy simply because that's where the design process begins in Perspective.



Component

Components are elements that you can select from to design your project. Perspective has a host of built-in components such as displays, buttons, charts, and other elements that display information to the user viewing a session. There is also an Embedded View component that allows you to include a view in the place of any component in the above structure.

Container

Containers are elements that contain components. You can nest one container inside of another container. It is important to note that containers do more than simply contain components, they also define a layout strategy of how components inside a container resize and reposition. Because of this, there are several different types of containers and each behave differently, and support different layout strategies.

View

Views are the primary unit of design in Perspective. Views are unique in that they can act as both a top level screen (a whole page in your session) or a component (embedded in another view). Each View is a project resource, which are named and organized into folders in the Ignition Designer's Project Browser tree. These folders/paths are important not only for organization and referencing, but also because these paths uniquely identify each view, and are used in the session (runtime) for navigation. Each View has a container type that decides how the components inside it will behave. Multiple Views can be present within the same page, and views can also be nested. Parameters can be passed into a view from an external source.

Page

A Page represents a collection of views that are displayed in a single space such as a browser window or tab, and is the main navigation element in a session. Each page consists of a primary view, but multiple views can be defined from within a single page. Some views can be docked to specific edges of the page, or “popups” floating on top of the other views. Multiple pages can be open as part of the same session, but can show different information. A URL is defined for a page, which means the Forward and Back buttons in a web browser can be used to navigate to pages that have already been visited.

Properties

A session has its own properties that can be customized. For information, see Session Properties.

A Simple Example

If you are just starting to explore Perspective, you will most likely create a single view, add a few components (no containers), and then launch a session. In this case, our structure seems much simpler than the diagram above, but your session has one page, which is displaying one view, which has a few components. As you add to your project, it will become more complex and start to resemble the image above.

Launching a Perspective Session

There are three ways to launch a Perspective Session:

- From the Designer
- From the Gateway Webpage
- By entering the web address of the project in your web browser

Launching a Session from the Designer

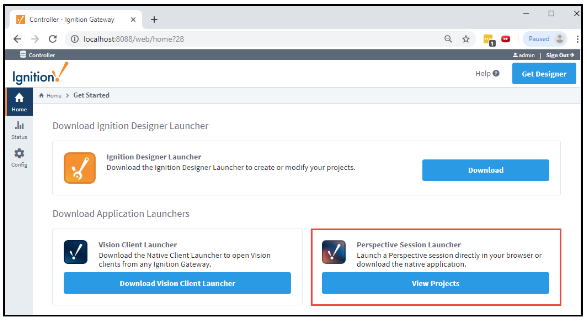
Launching a Perspective Session from the Designer is a great method if you're testing your changes. When you're in the Designer, simply click the **Tools > Launch Perspective > Launch Session** from the top menubar. This opens a session for the project you are working on directly from the Designer.

Launching a Session

Watch the Video

Launching a Session from the Gateway Webpage

On the Gateway Webpage, you'll notice the **Get Started** page is the home page. Under the Perspective logo, click **View Projects**.


Controller - Ignition Gateway

localhost:8088/web/home?28

Download Ignition Designer Launcher

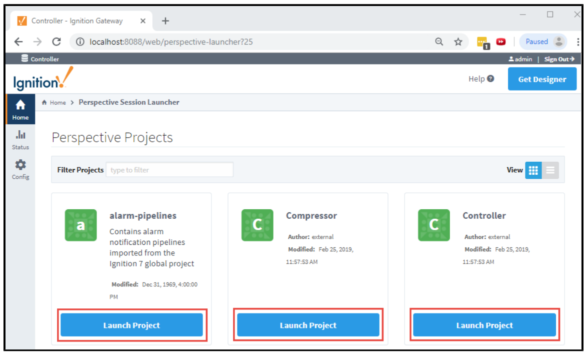
Ignition Designer Launcher
Download the Ignition Designer Launcher to create or modify your projects.

Download Application Launchers

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

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

The window will refresh with a list of Perspective projects. Find your project and click the **Launch Project** button. This opens your project in a browser tab.


Controller - Ignition Gateway

localhost:8088/web/perspective-launcher?25

Perspective Projects

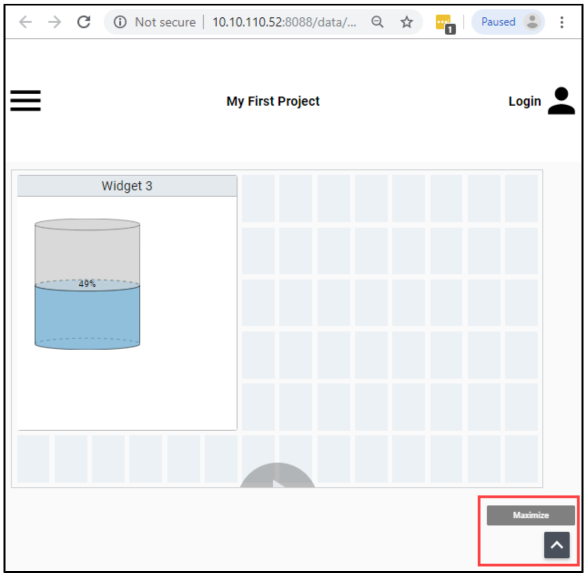
alarm-pipelines
Contains alarm notification pipelines imported from the Ignition 7 global project
Modified: Dec 31, 1969, 4:00:00 PM

Compressor
Author: external
Modified: Feb 25, 2019, 11:57:53 AM

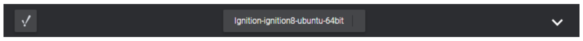
Controller
Author: external
Modified: Feb 25, 2019, 11:57:53 AM

Session App Bar

At the bottom of every Perspective Session, there is an app bar with a few options that provide information about Ignition and the current session. To display the app bar, click the **Maximize** ^ icon in the lower right corner of the screen.



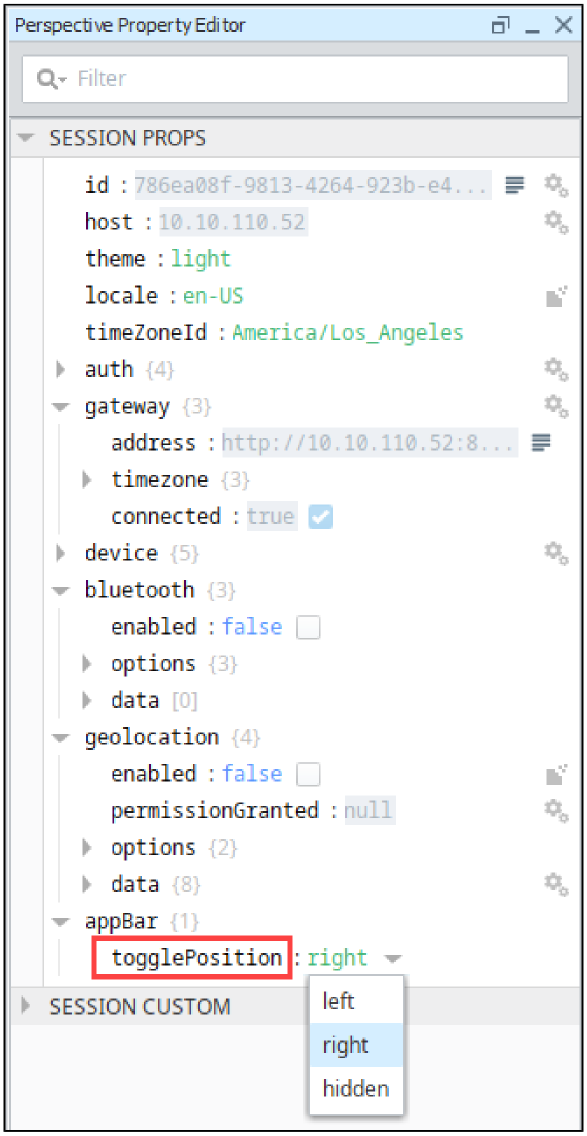
This brings up the app bar at the bottom of the window.



You can change the location of the Maximize/Minimize icon on the app bar in the Property Editor. Scroll down the list of properties in the Perspective Property Editor and expand the **appBar** property. Right-click on the **togglePosition** property and select one of the three options: **left**, **right** or **hidden**.

Session App Bar

Watch the Video



Perspective Property Editor

Filter

SESSION PROPS

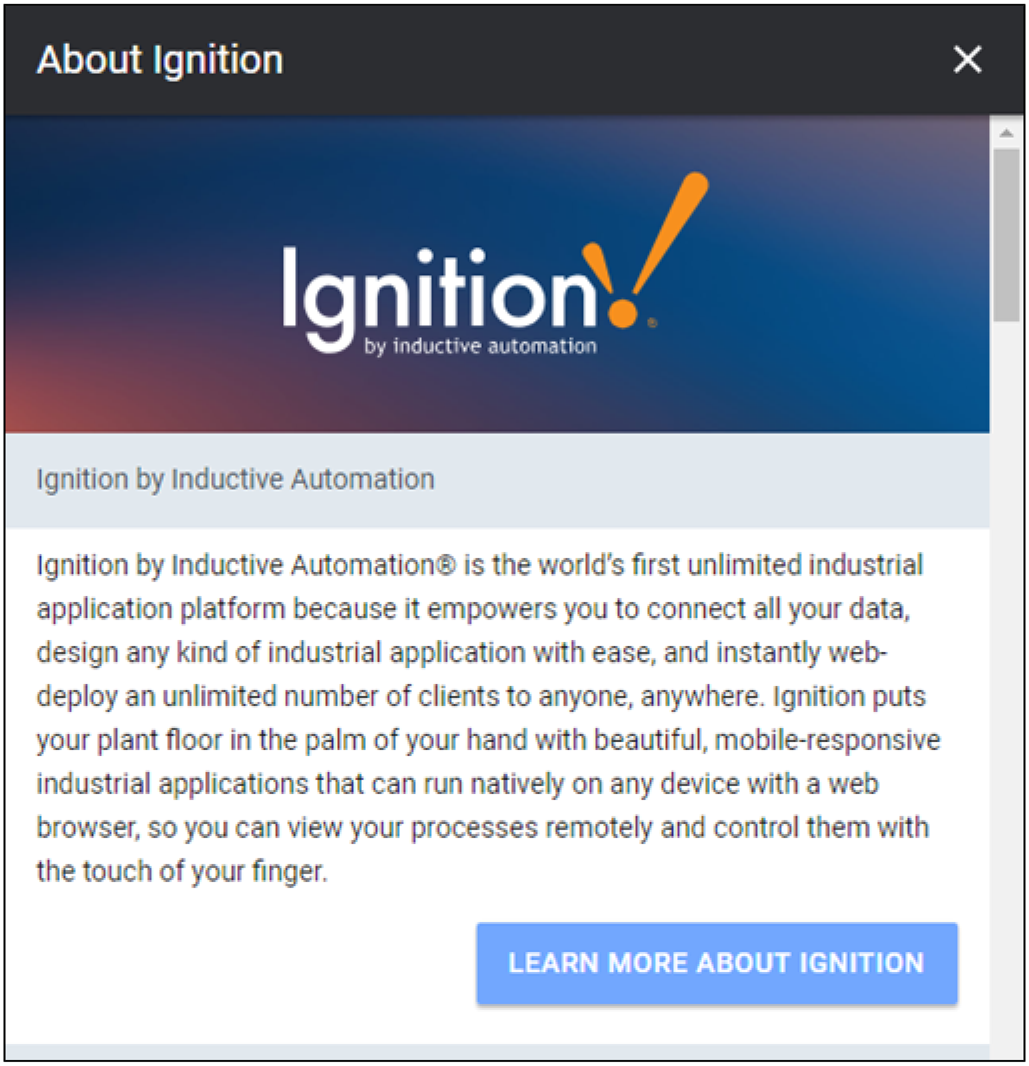
```
id : 786ea08f-9813-4264-923b-e4...
host : 10.10.110.52
theme : light
locale : en-US
timeZoneId : America/Los_Angeles
auth {4}
gateway {3}
    address : http://10.10.110.52:8...
    timezone {3}
    connected : true ☑
device {5}
bluetooth {3}
    enabled : false □
    options {3}
    data [0]
geolocation {4}
    enabled : false □
    permissionGranted : null
    options {2}
    data {8}
appBar {1}
    togglePosition : right
```

left
right
hidden

SESSION CUSTOM

About Ignition Information Popup

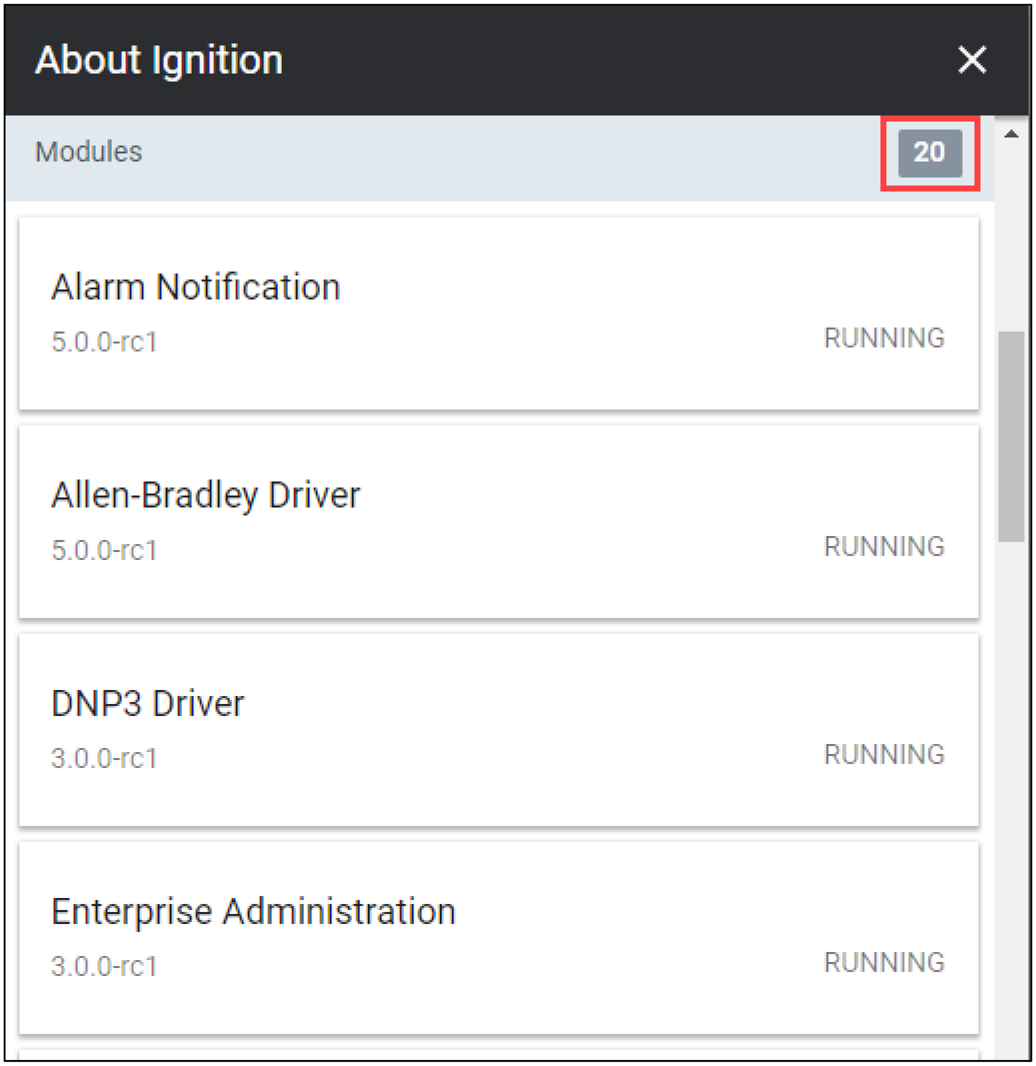
On the lower left of the app bar, click the **Ignition** ✓ icon to open the **About Ignition** information popup. This screen has a short introduction to Ignition and a link to the Inductive Automation website. Use the scroll bar on the right to scroll down.



Ignition by Inductive Automation

Ignition by Inductive Automation® is the world's first unlimited industrial application platform because it empowers you to connect all your data, design any kind of industrial application with ease, and instantly web-deploy an unlimited number of clients to anyone, anywhere. Ignition puts your plant floor in the palm of your hand with beautiful, mobile-responsive industrial applications that can run natively on any device with a web browser, so you can view your processes remotely and control them with the touch of your finger.

Here you'll see the number of modules installed on the Gateway, along with a list of those a modules, their version numbers, and whether they are currently running.



Alarm Notification
5.0.0-rc1 RUNNING

Allen-Bradley Driver
5.0.0-rc1 RUNNING

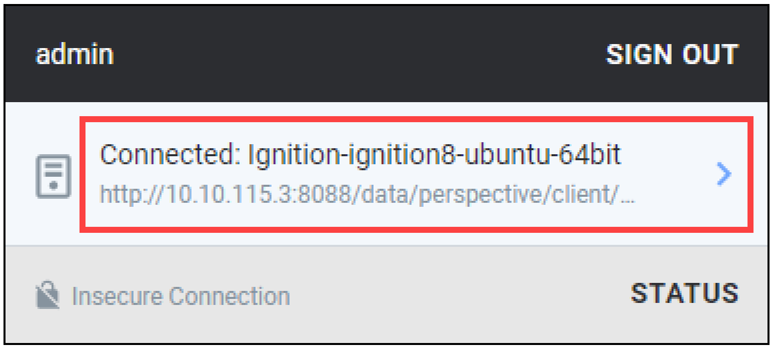
DNP3 Driver
3.0.0-rc1 RUNNING

Enterprise Administration
3.0.0-rc1 RUNNING

Session Status Popup

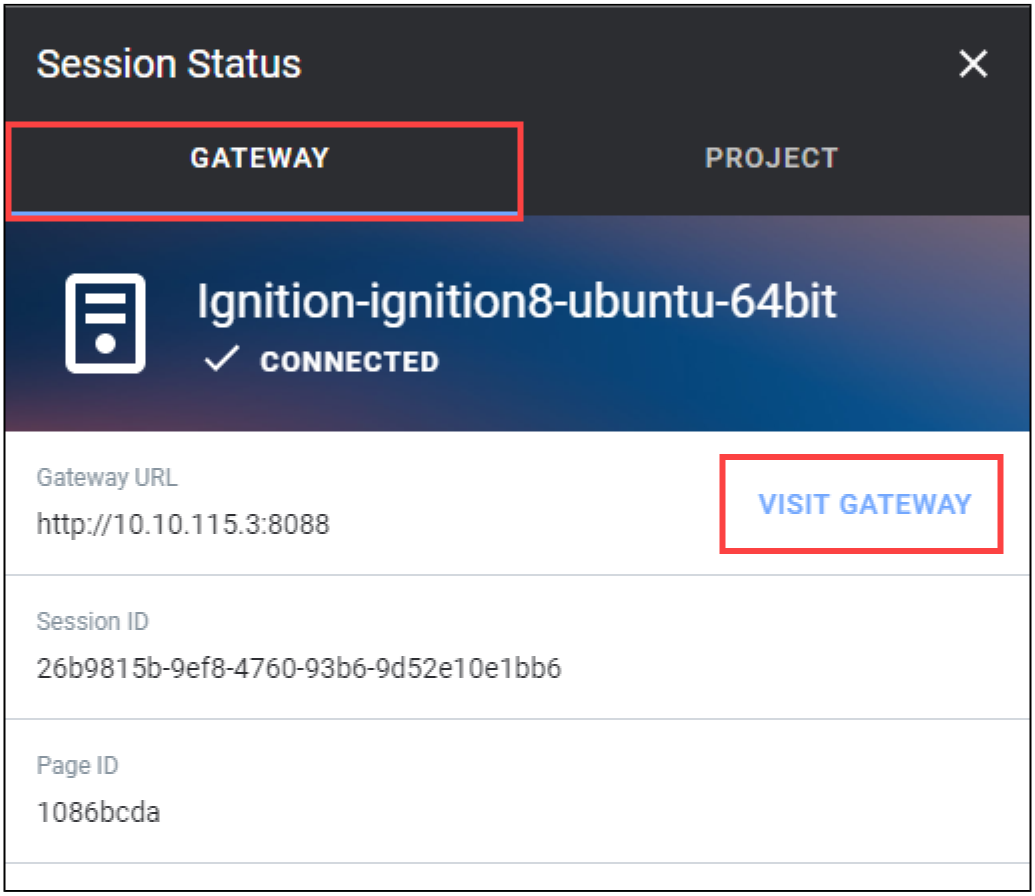
In the middle of the app bar at the bottom of the window, the current Gateway ID is displayed. Click on this for additional information. The pop-up screen shows the current user and connected Gateway. If no user is signed in, a Sign In button is displayed in the upper right corner.

Click on the **right arrow or anywhere on the Gateway name to open the Session Status popup.**



Connected: Ignition-ignition8-ubuntu-64bit
http://10.10.115.3:8088/data/perspective/client/...

The Session Status popup has two tabs: **Gateway** and **Project**. Under the **Gateway** tab, you can see the connection status, Gateway URL, Session ID, and Page ID. The **Visit Gateway** button is a shortcut to the Gateway Homepage.



Ignition-ignition8-ubuntu-64bit

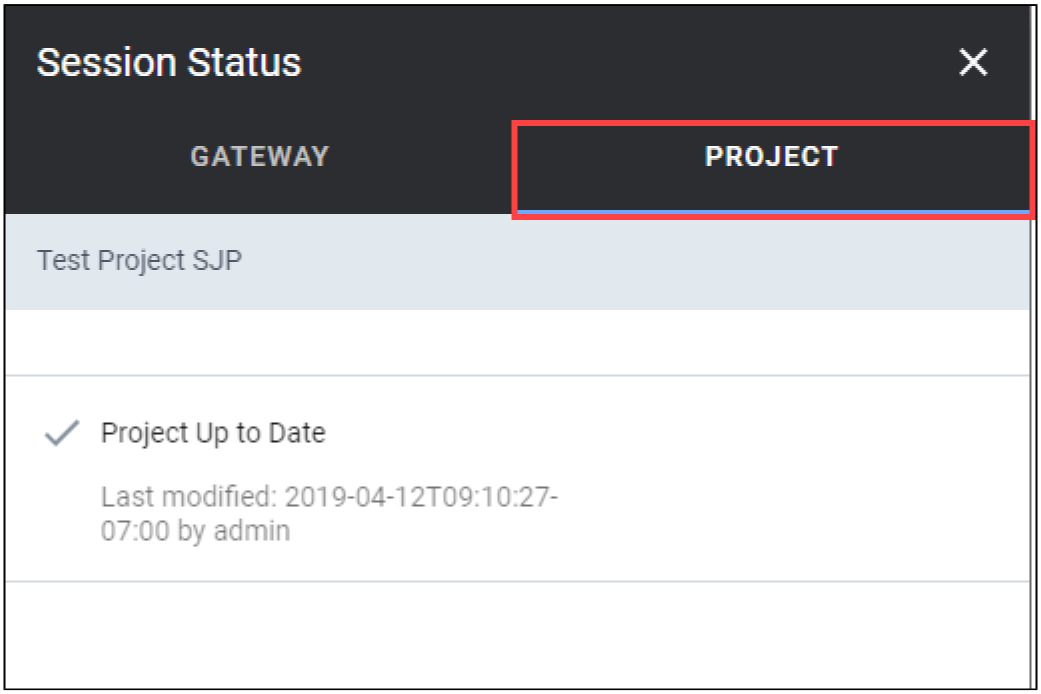
✓ CONNECTED

Gateway URL
http://10.10.115.3:8088

Session ID
26b9815b-9ef8-4760-93b6-9d52e10e1bb6

Page ID
1086bcda

Under the **Project** tab, you see the name of the project, the last modified date for the project, and whether the project is up to date.



Test Project SJP

✓ Project Up to Date

Last modified: 2019-04-12T09:10:27-07:00 by admin

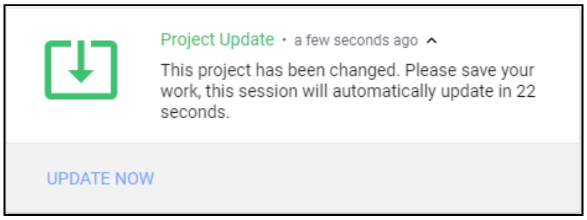
Project Updates

INDUCTIVE UNIVERSITY

Session Update Notifications

Watch the Video

If you have a Perspective Session open and a change was made in the Designer that was saved and published, one of two things may happen. Either the project will silently update, or an Update Notification window will appear in your session. Your session will automatically update in 30 seconds or you can click **Update Now**.

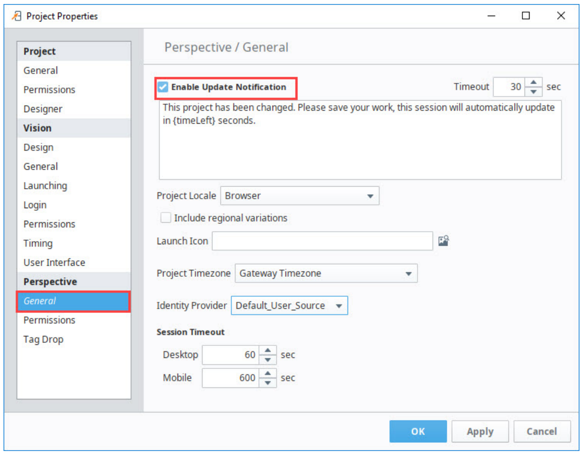


Project Update • a few seconds ago

This project has been changed. Please save your work, this session will automatically update in 22 seconds.

UPDATE NOW

In order to receive update notifications in a Perspective Session the Update Notification option must be enabled. To access the Project Properties, in the Designer, click on **Project** tab on the menu bar. Then select **Project Properties**. Scroll down to **Perspective > General** and then check **Enable Update Notification**.



Perspective / General

☑ Enable Update Notification

Timeout 30 sec

This project has been changed. Please save your work, this session will automatically update in {timeLeft} seconds.

Project Locale: Browser

□ Include regional variations

Launch Icon:

Project Timezone: Gateway Timezone

Identity Provider: Default_User_Source

Session Timeout

Desktop 60 sec

Mobile 600 sec

Page URLs

Because Perspective is designed to operate in a web browser, each page must be “mounted” at a given URL. The URL setting of the page will be reflected in the browser’s URL bar if the session is running in a consumer web browser. To learn more, refer to the section on Page URLs in Pages in Perspective.

Pages can be mounted at URLs that also include parameters. These parameters are used to allow a page to be mounted at a dynamic URL, allowing information in the URL to be interpreted as input parameters to the page’s primary view. Go to the section on Passing Parameters to get more detailed information on passing URL parameters.

Browser Version Requirements

Browser version requirements are hard to define since each major or minor browser versions may introduce new CSS specifications which are leveraged by individual features in Perspective. For example, a container component built around a relatively new and exciting CSS layout system will likely require a more recent browser version. In general however, the supported browser listed below do a good job at staying up to date with implementation and support of these new specifications.

Our general policy is to only use features that have been supported for some time, whenever possible. We are currently using features that have been standardized for a few years before Perspective was even released. Overall, we will always recommend that you keep your browser up to date to take advantage of browser features, fixes, and most importantly for security updates.

Ignition (as well as browsers) is constantly evolving and leveraging new APIs to provide you with the best experience possible, so it's difficult to pin down minimum versions for 100% of Perspective's functionality. We do take care to ensure Perspective sessions will always work on the browsers and versions listed below. Perspective sessions may very likely work for some browsers that are not even listed below, but we do not test against them, and thus cannot guarantee their behavior.

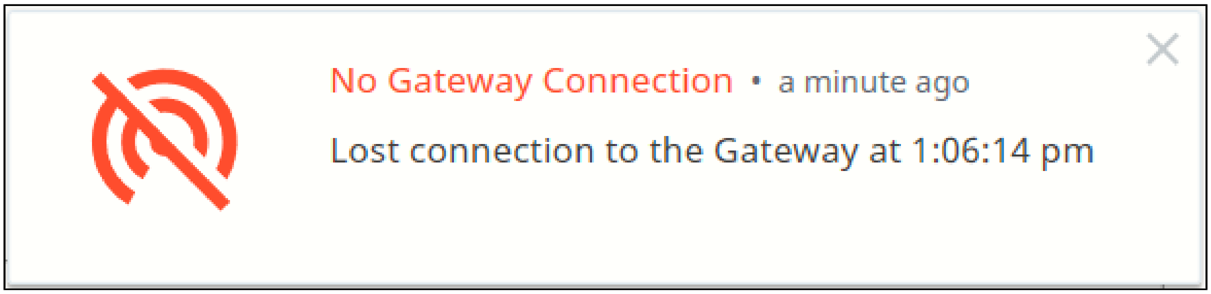
Some browsers that are not listed below may also work, but we cannot guarantee them.

Basic Session Functionality Requirements for Ignition 8.1.0	
Browser	**Minimum Version Required**

Chrome	57
Firefox	52
Safari	11
Edge	16

Session Disconnections

If a Perspective Session loses its connection to the Ignition Gateway, a message will immediately display on the upper right-hand corner of the Session to inform a user of the lost connection status and the time of connection loss.



No Gateway Connection • a minute ago

Lost connection to the Gateway at 1:06:14 pm

The Session itself will continuously attempt to reconnect to the Gateway. When the connection between the Session and the Gateway is re-established, your Session will resume its normal operation. If the project associated with the Session is somehow deleted, the Session will fail to reconnect because its source project is missing. A project can be manually deleted or overwritten by a Gateway restore.

Related Topics ...

- Pages in Perspective

In This Section ...

Ignition Perspective App

On this page ...

INDUCTIVE UNIVERSITY

Perspective App

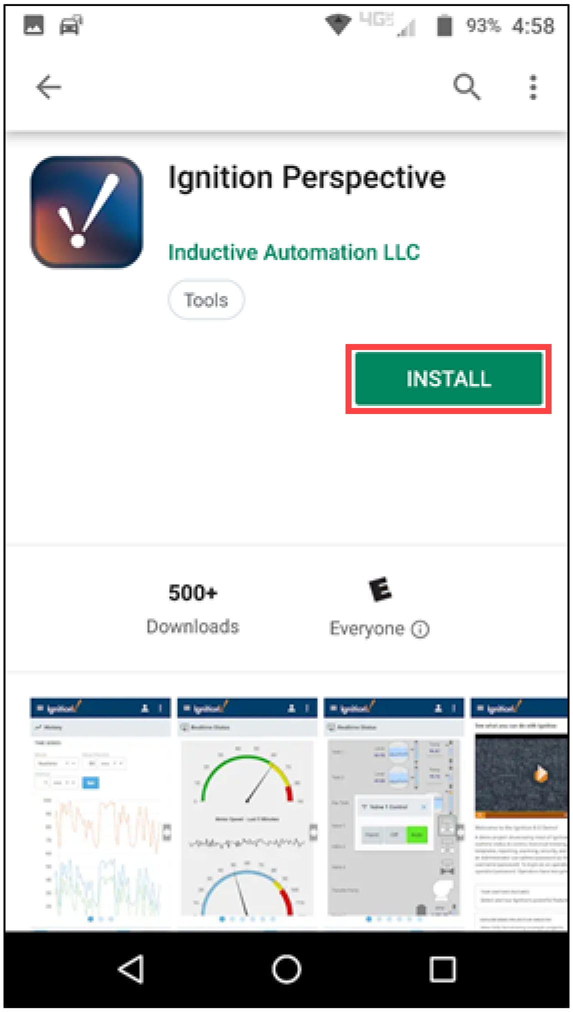
Watch the Video

The Ignition Perspective App is a native mobile application that can be downloaded and installed on Android and iOS mobile devices. It provides users with a single location to access all of their Perspective projects. While a Perspective Session can be launched from a mobile device's web browser, the Perspective App enables the use of many built in pieces of hardware within your mobile device, such as the camera, the accelerometer, or the NFC Scanner. Data from these tools can be pulled into the Perspective Session and used in various ways: a barcode can be scanned and the data inserted into a database, the accelerometer can be used to remote control a device with motion rather than buttons, or an NFC Tag can provide vital information about a system. For information about setting up native app actions in Perspective, see Perspective Events and Actions .

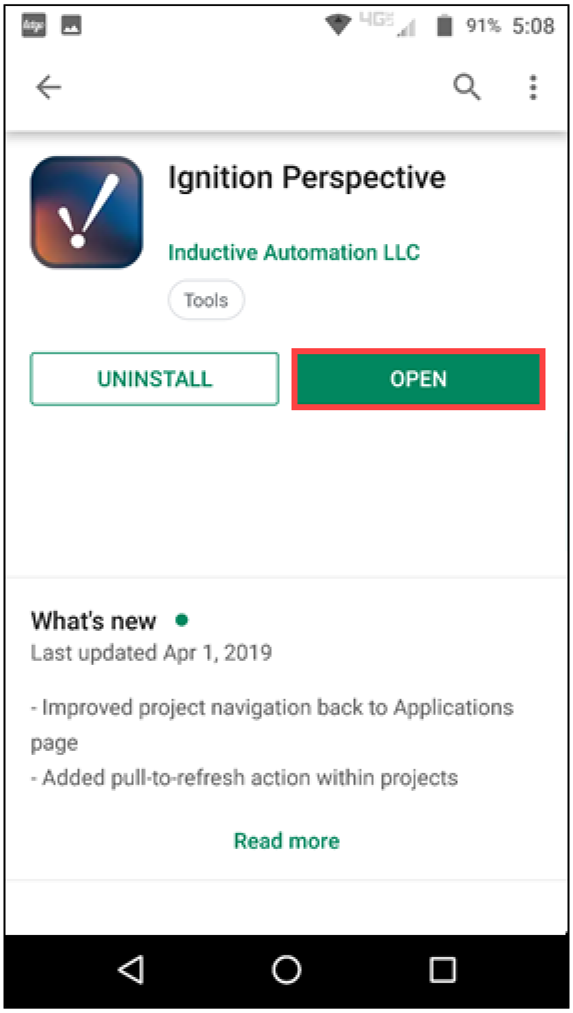
Download the App

You can download the Ignition Perspective App from a local online app store, such as the Google Play Store for Android or the App Store for iOS.

1. In your app store, search for Ignition Perspective.
2. Click the **Install** button.



3. Once installation is complete, click the **Open** button, or click on the **Perspective** icon on your device's home screen.



Ignition Perspective

Inductive Automation LLC

Tools

UNINSTALL

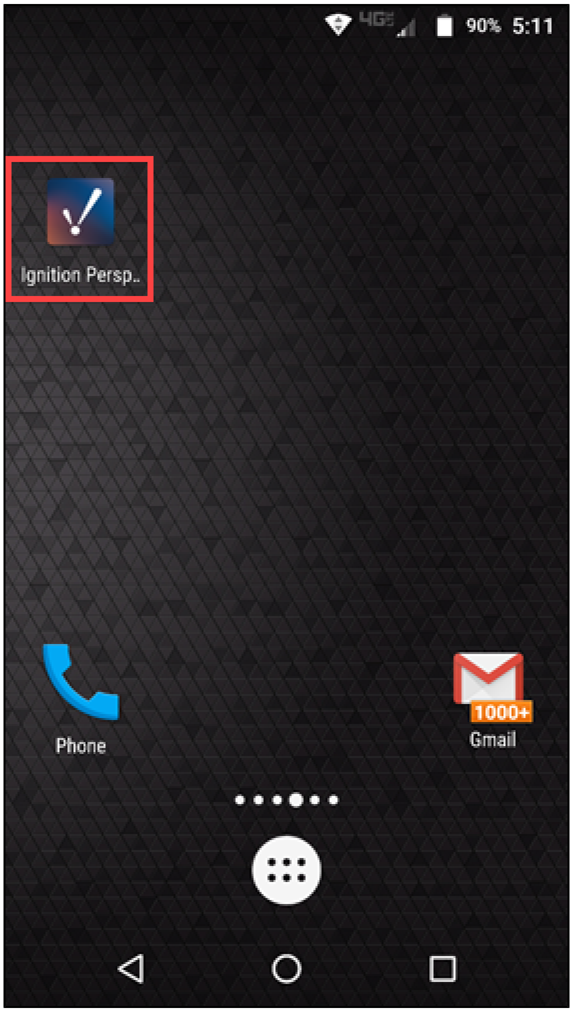
OPEN

What's new ●

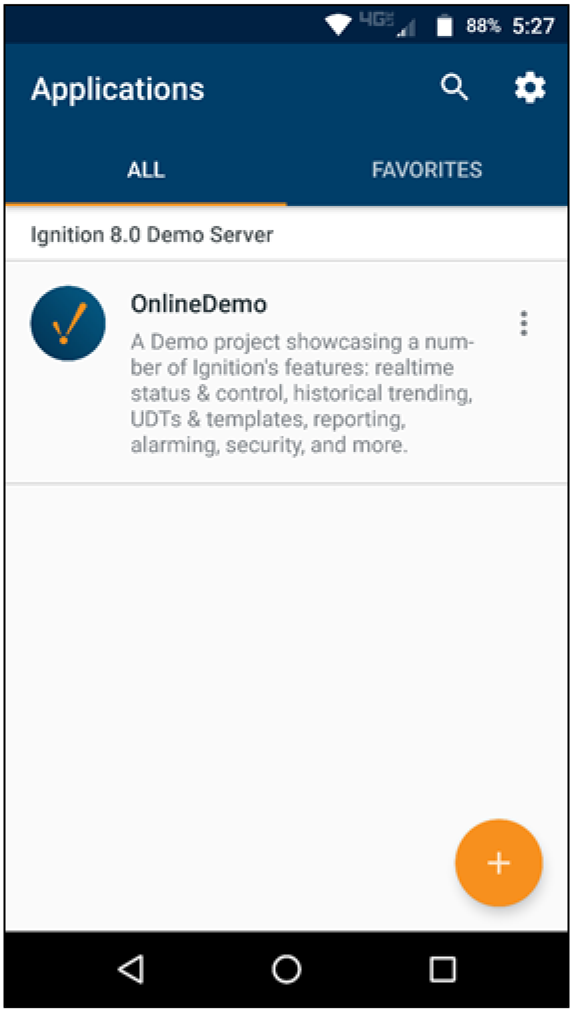
Last updated Apr 1, 2019

- Improved project navigation back to Applications page
- Added pull-to-refresh action within projects

Read more


4G
90%
5:11
Ignition Persp..
Phone
1000+
Gmail

4. The homepage is displayed. This will show a list of your projects. Initially, just the OnlineDemo is listed.



Ignition 8.0 Demo Server

OnlineDemo

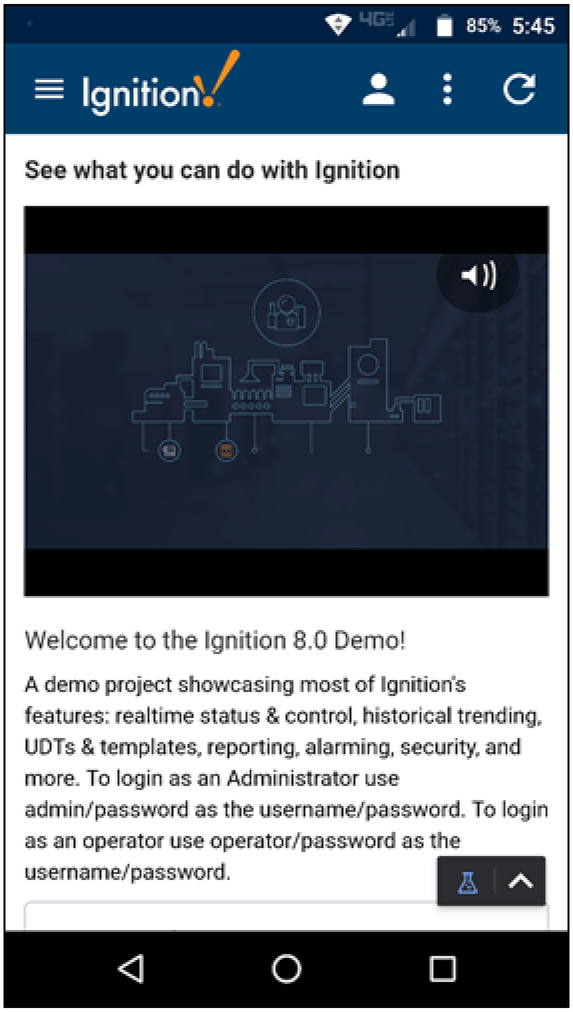
A Demo project showcasing a number of Ignition's features: realtime status & control, historical trending, UDTs & templates, reporting, alarming, security, and more.

Launch a Perspective Session

On the home screen, you can launch the Online Demo, which is on the Ignition 8.0 Demo Server or launch a project on a different Gateway.

Launch the Online Demo

To launch the demo, just click the **Perspective** icon or click on the demo description.



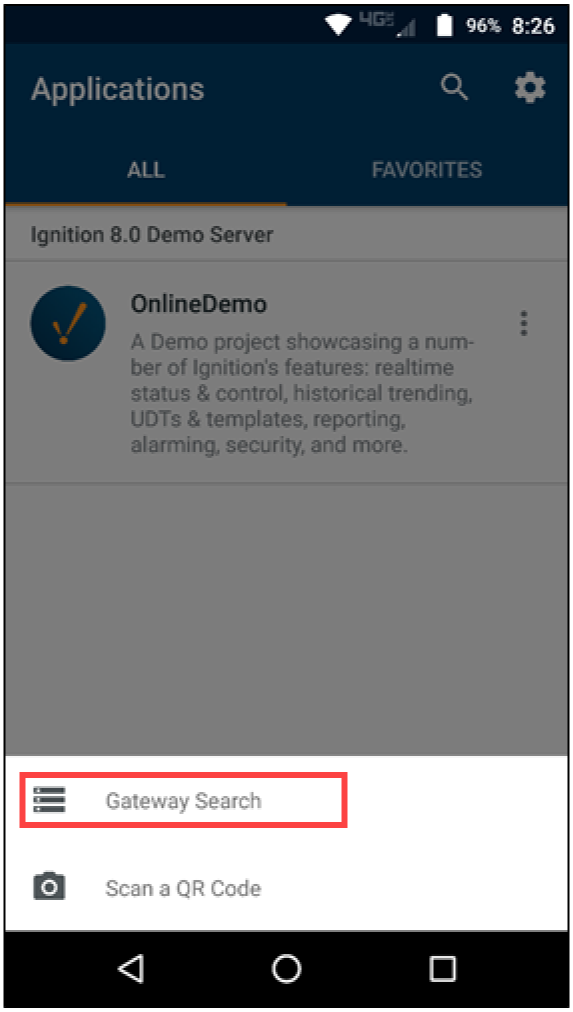
See what you can do with Ignition

Welcome to the Ignition 8.0 Demo!

A demo project showcasing most of Ignition's features: realtime status & control, historical trending, UDTs & templates, reporting, alarming, security, and more. To login as an Administrator use admin/password as the username/password. To login as an operator use operator/password as the username/password.

Launch a Project

1. To launch a different project, click the **Plus** + icon in the lower right corner of your screen.
2. You then have two options:
   - Scan a QR Code
   - Search for an available Gateway with Perspective projects.
3. For this example, we'll search for available Gateways. Click on **Gateway Search**.

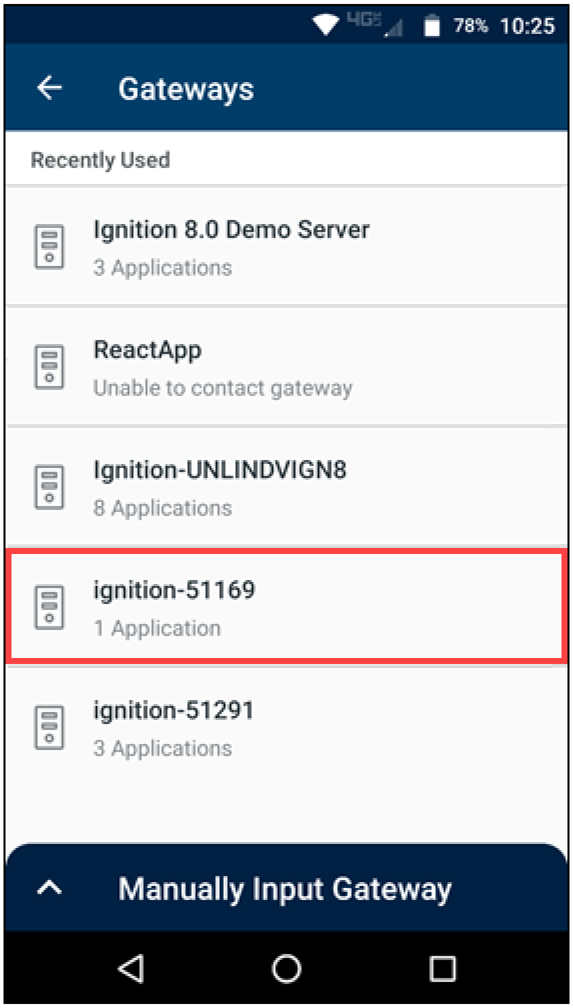


Ignition 8.0 Demo Server

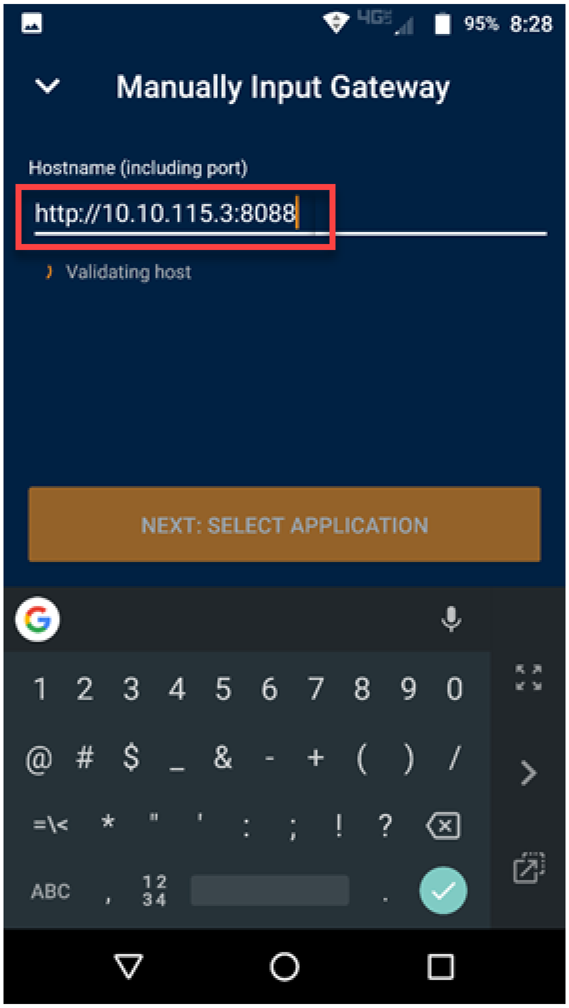
OnlineDemo

A Demo project showcasing a number of Ignition's features: realtime status & control, historical trending, UDTs & templates, reporting, alarming, security, and more.

4. Select the Gateway you want to connect to.



a. If the Perspective App doesn't find the Gateway automatically, you can manually input a Gateway host name and port. Click on **Manually Input Gateway**.
b. Type in the host name including the port, for example: http://10.10.115.3:8088 (or https://10.10.115.3:8043 if using SSL).



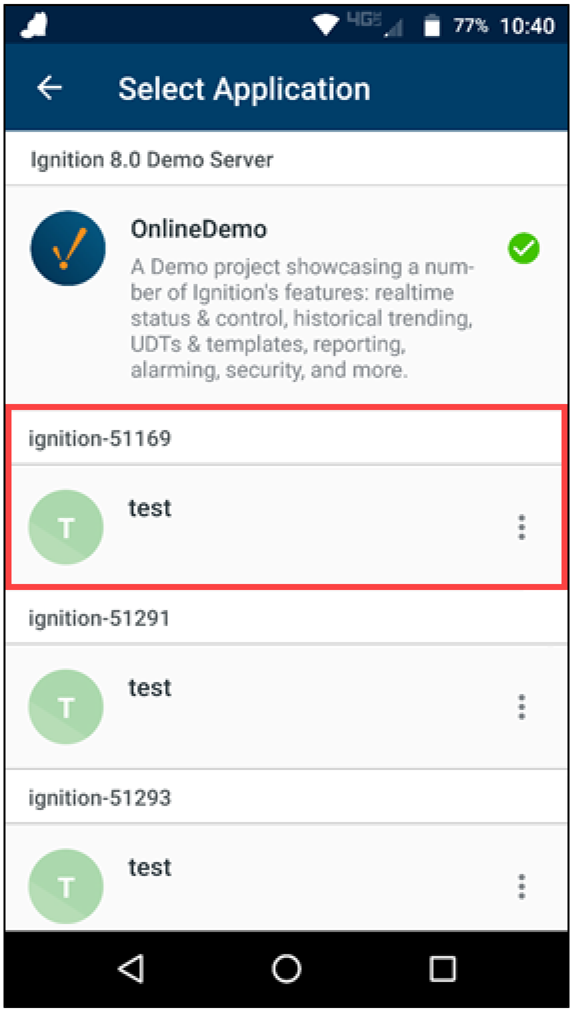
Manually Input Gateway

Hostname (including port)

http://10.10.115.3:8088

Validating host

5. Click on the project you want to open.



77% 10:40

Select Application

Ignition 8.0 Demo Server

OnlineDemo

A Demo project showcasing a number of Ignition's features: realtime status & control, historical trending, UDTs & templates, reporting, alarming, security, and more.

ignition-51169

test

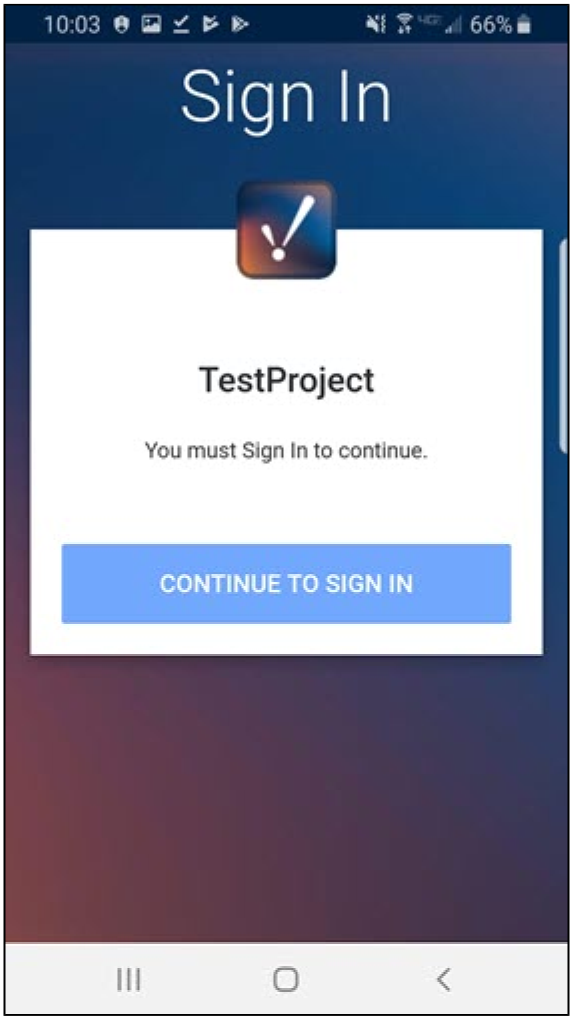
ignition-51291

test

ignition-51293

test

6. If security is set up on the project, you'll be prompted to sign in with your credentials.



Sign In

TestProject

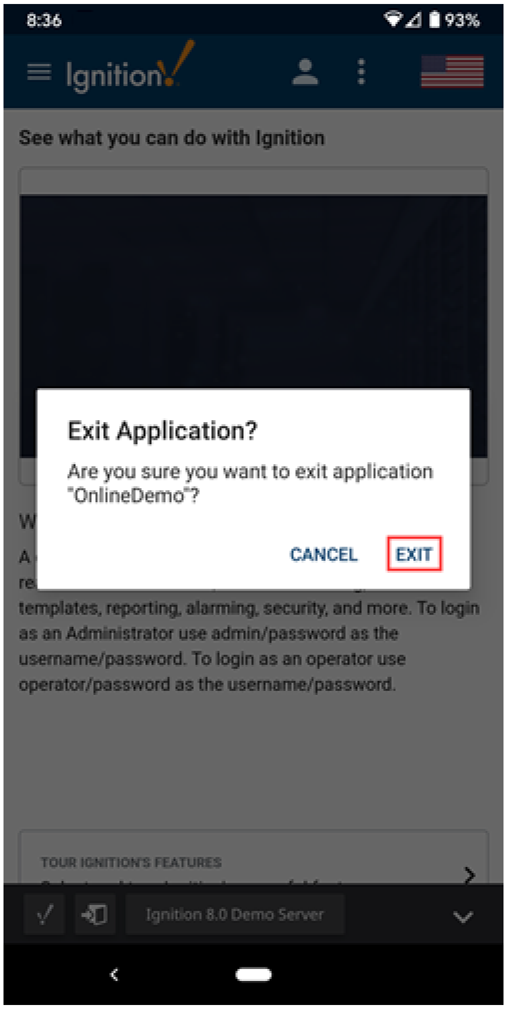
You must Sign In to continue.

Switch to a Different Application

When you are running a project or the demo app, you can switch to a different application, as follows:

1. Click the **App Bar** ⌃ icon in the lower right corner.
2. Click the **Exit** ⇥ icon.

3. At the confirmation prompt, click **Exit**.



Exit Application?

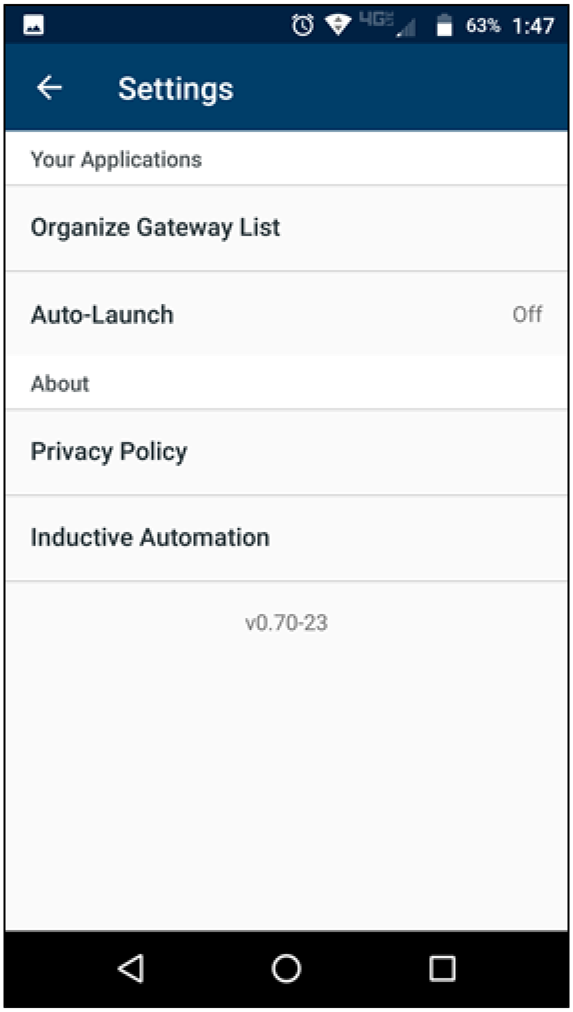
Are you sure you want to exit application "OnlineDemo"?

templates, reporting, alarming, security, and more. To login as an Administrator use admin/password as the username/password. To login as an operator use operator/password as the username/password.

Additional Options

Click on the **Search** 🔍 icon at the top of the screen to filter the project list.

Click on the **Settings** ⚙ icon at the top of the screen to view the settings for the Perspective app. Here you can organize your list of available Gateways. You can also turn the Auto-Launch option on and select a project that will be automatically launched when the Perspective App is opened on this device.



Settings

Your Applications

Organize Gateway List

Auto-Launch Off

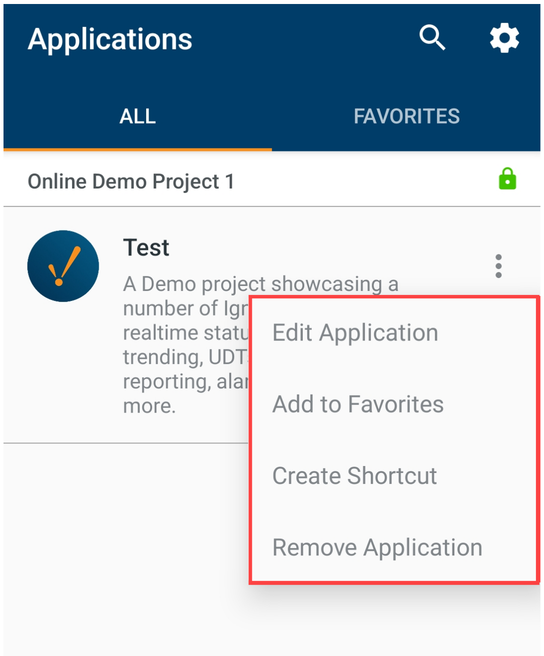
About

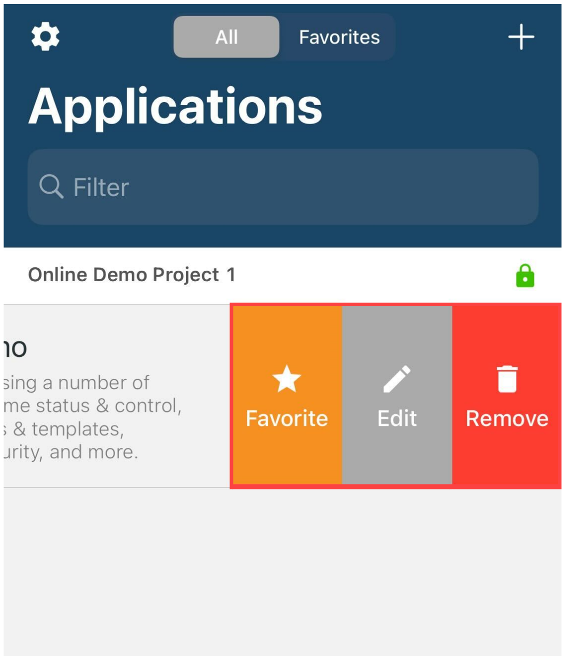
Privacy Policy

Inductive Automation

v0.70-23

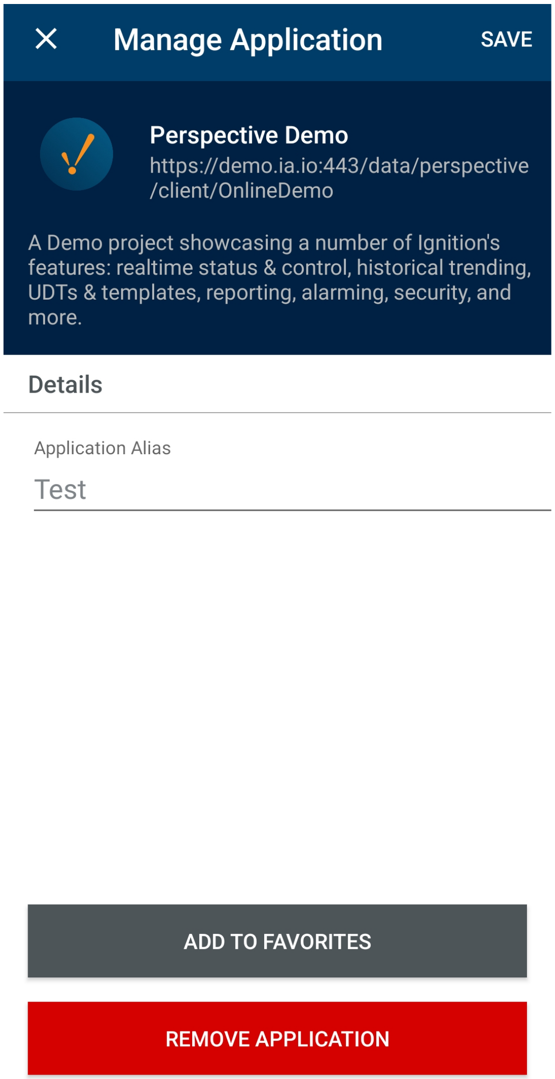
Individual projects also have additional options that can be seen by tapping the **Three Dots** ⋮ icon for the project on an Android device or swiping to expand the options on an iPhone device. The additional project options include making the project a Favorite, editing the project, and removing the project. Android devices also have the option to Create Shortcuts. Tapping **Add to Favorites** or **Favorite**, on an Android or iPhone device respectively, means that project will appear on the **Favorites** tab in the app. If you have multiple projects listed on your **All** tab, marking certain projects as Favorites can give you a quick and easy way to access a tailored list of projects.





Selecting **Edit** will access the **Manage Application** screen, where users can:

- Add an Application Alias
- Add the project to Favorites
- Remove the project
- Save any changes made on the Manage Application screen
- Return to the previous screen by tapping the **X** icon or **Cancel**, on an Android or iPhone device respectively.



✕ Manage Application SAVE

Perspective Demo
https://demo.ia.io:443/data/perspective/client/OnlineDemo

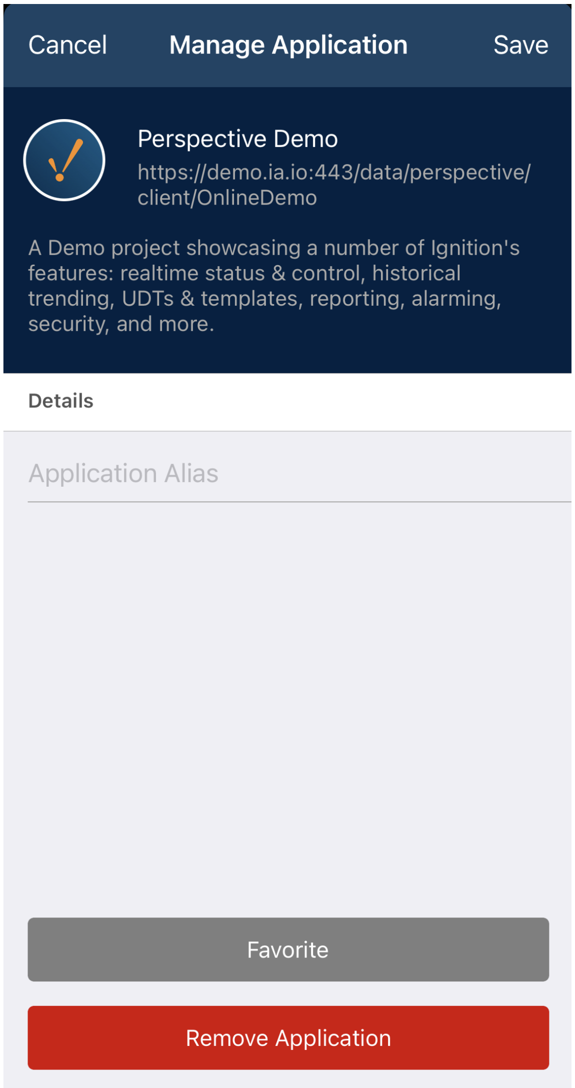
A Demo project showcasing a number of Ignition's features: realtime status & control, historical trending, UDTs & templates, reporting, alarming, security, and more.

Details

Application Alias
Test

ADD TO FAVORITES

REMOVE APPLICATION



Cancel Manage Application Save

Perspective Demo
https://demo.ia.io:443/data/perspective/client/OnlineDemo

A Demo project showcasing a number of Ignition's features: realtime status & control, historical trending, UDTs & templates, reporting, alarming, security, and more.

Details

Application Alias

Favorite

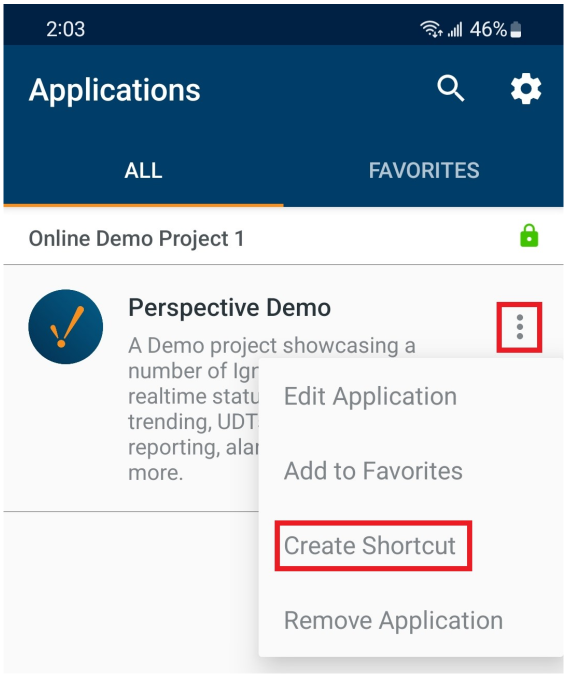
Remove Application

Users can alternatively remove a project by simply tapping the **Remove Application** or **Remove** option, on an Android or iPhone device respectively, from the homepage.

Create Project Shortcuts

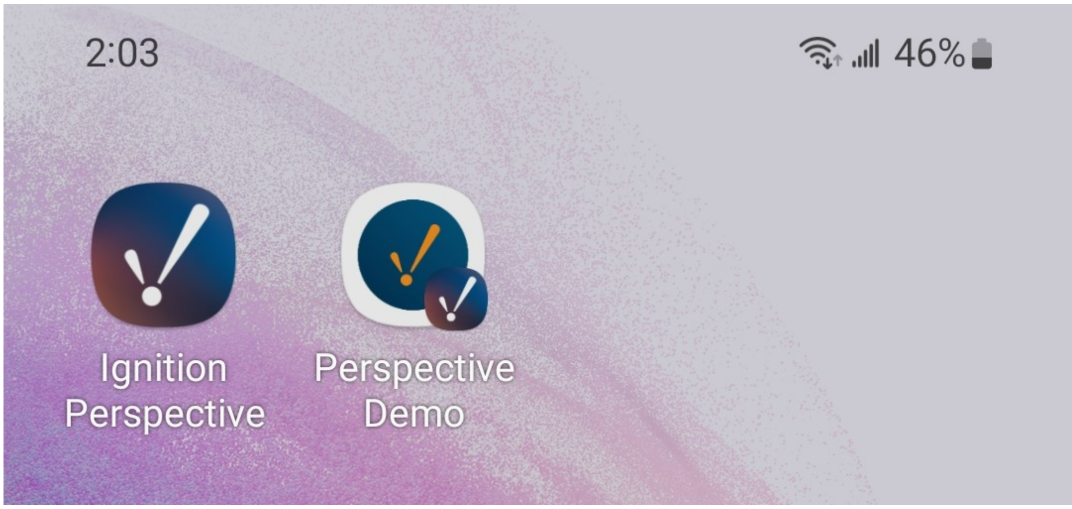
If the Ignition Perspective App is downloaded on an Android mobile device, you can create a project shortcut to open a project directly from your Home screen. Although iOS users are not able to create project shortcuts currently, the Long Press can still be used to select and open a project without launching the Ignition Perspective App. Long Press instructions are covered below under the second method for creating project shortcuts.

1. To create the shortcut, open the Ignition Perspective App.
2. Push the **Three Dots** ⋮ icon for your project.
3. Select **Create Shortcut**.



4. Select **Add**.

A shortcut is now added to your Home screen and will open a project as if you were launching from within the Ignition Perspective App without needing to use the Ignition Perspective App.

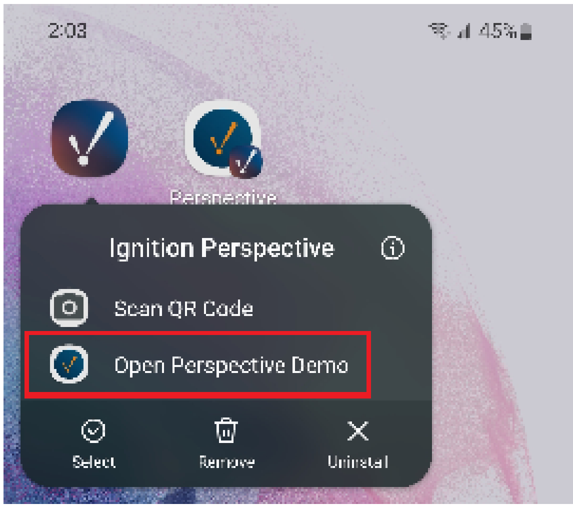


The shortcut icon and App Name will match what is displayed in the Ignition Perspective App project list.

- The shortcut icon can be edited in the Designer **Project Properties > Perspective > General > Launch Icon**.
- The App Name can be edited by selecting the **Three Dots icon > Edit Application** and entering an Application Alias. If the Alias is changed after creating the shortcut, make sure to re-launch the project inside the Ignition Perspective App. This will trigger the shortcut name to update on your Home screen.

A second method to create a Home screen shortcut is to use the Long Press (press and hold) feature.

1. Long Press the Ignition Perspective App on your Home screen until the Information popup appears.



2. Locate the project name (or Alias) from the popup. Note that a project has to be opened at least once from inside the Ignition Perspective App before it will appear in this popup.
3. Drag and drop the project to your Home screen to create a shortcut in the same manner as detailed above.

Managed Configurations

The Ignition Perspective App can be configured via iOS and Android enterprise services , allowing third-party tools to install the app with some initial configurations. The actual configurations are pushed out by the Enterprise Mobile Management (EMM) solution, so there are no configurations to be made on the Ignition side.

For Android devices, EMMs pull the keys from the APK and manages the configuration from a console in the EMM software.

For iOS devices, you will need to provide the EMM software with a sample PLIST. See the Ignition Perspective iOS App - AppConfig XML on our Extra Materials page.

The following keys will be made available to the EMM solution.

<table>
<tr><th>Key</th><th>Description</th></tr>
<tr><td>auto_launch</td><td>Contains keys that control the auto-launch capability of the app, allowing you to determine if the app should immediately launch into a project or not. Contains the following keys:
<table>
<tr><th>Key</th><th>Description</th></tr>
<tr><td>auto_launch_locked</td><td>If true, prevents user from disabling auto-launch or changing the auto-launch URL within the app.</td></tr>
<tr><td>prevent_exit</td><td>If true, prevents the user from exiting the auto launched project</td></tr>
<tr><td>auto_launch_url</td><td>If set with a project URL, auto launch is enabled and the project specified in the URL will automatically launch upon startup of the app.</td></tr>
</table></td></tr>
<tr><td>hide_demo</td><td>If true, the built-in demo project will become hidden.</td></tr>
<tr><td>initial_applications</td><td>Allows you to add projects to the main app page. Each application has the following keys:
<table>
<tr><th>Key</th><th>Description</th></tr>
<tr><td>project_url</td><td>A URL leading directly to the application, for example:
<pre>
http://www.someurl.com:8088/data/perspective/client/myproj

OR

perspective://www.someurl.com:8088/myproj
</pre></td></tr>
<tr><td>alias</td><td>The name for the application, as it will appear in the project list.</td></tr>
<tr><td>is_favorite</td><td>Sets this project as a favorite.</td></tr>
</table></td></tr>
</table>

<table>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>initial_gateways</td>
<td>Allows you to add Gateways to the Recent Gateways list. Each gateway has the following key:
<table>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
<tr>
<td>gateway_url</td>
<td>A URL to the Gateway you'd like to add to the list, for example:
<code>http://www.someurl.com:8088</code></td>
</tr>
</table>
</td>
</tr>
</table>

Session Properties

Session properties are available for use throughout Perspective Sessions. Sessions of a given project will have the same list of properties, however, the actual values are unique and independent for each running Session.

Each Session creates its own instance of these properties. This makes them very useful as in-project variables for passing information between views or browser tabs, and between other parts of the Session, such as scripting.

The Property Editor displays Session properties when viewed from the Perspective Start screen. Each Session contains a series of properties with unique values. These properties provide some useful information about where the Session is running. Additionally, custom properties may be added, providing a way for a Session to store additional values that can be used in bindings and scripts, and are also important for passing parameters from one view to another.

On this page ...

- Restrictions from System Properties
- Session Properties Table

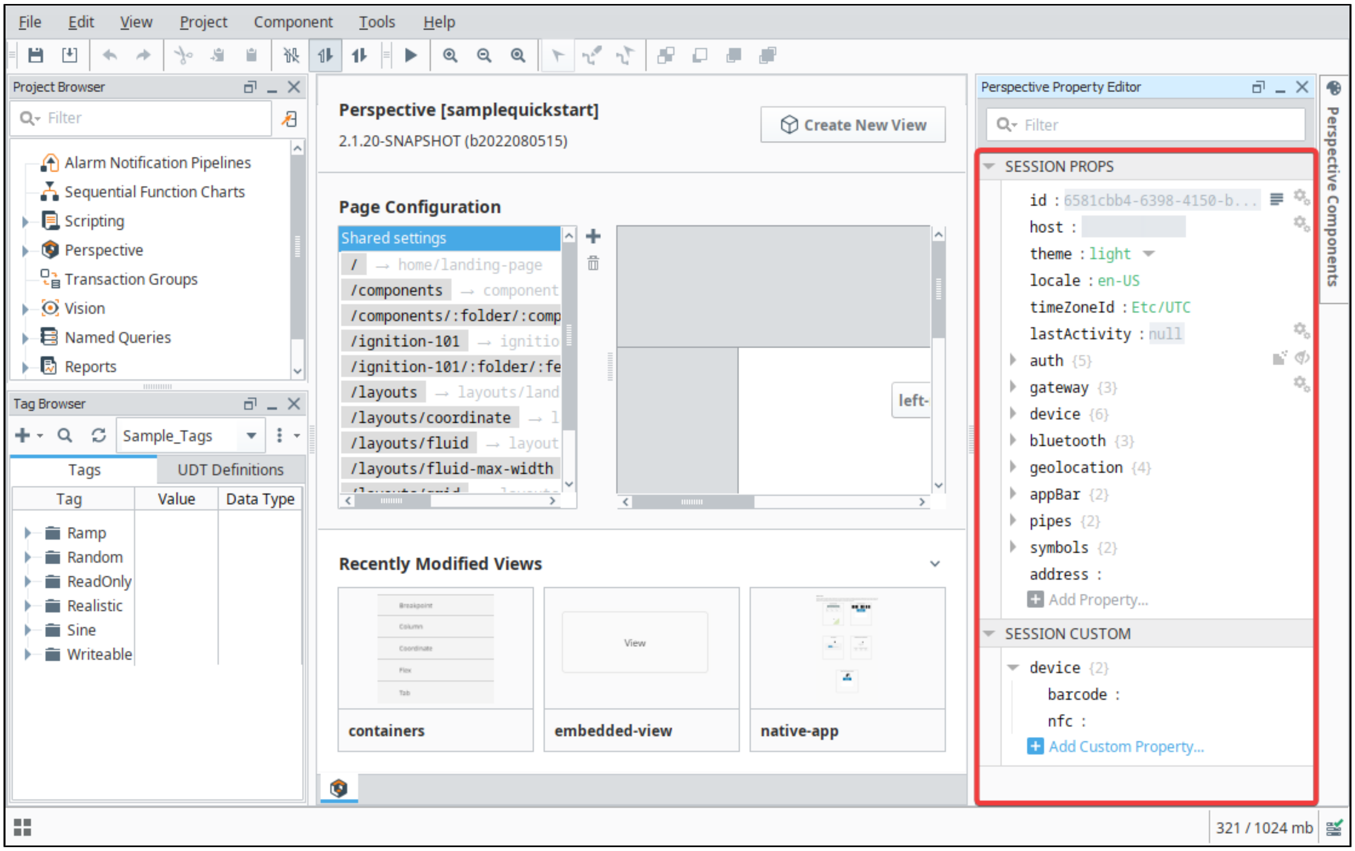
INDUCTIVE UNIVERSITY

Session Props

Watch the Video

Restrictions from System Properties

Some Session props are intentionally restricted with a System property and cannot be changed or removed. These properties have a **System** ⚙ icon displayed next to them in the Property Editor.



SESSION PROPS

- id : 6581cbb4-6398-4150-b...
- host :
- theme : light
- locale : en-US
- timeZoneId : Etc/UTC
- lastActivity : null
- auth {5}
- gateway {3}
- device {6}
- bluetooth {3}
- geolocation {4}
- appBar {2}
- pipes {2}
- symbols {2}
- address :
- Add Property...

SESSION CUSTOM

- device {2}
  - barcode :
  - nfc :
- Add Custom Property...

Session Properties Table

Name	Description
id	Unique Session identifier.

<table>
<tr><td></td><td></td></tr>
<tr><td>host</td><td>Reflects the connecting system's IP address or hostname.

This feature was changed in Ignition version <b>8.1.10</b>:

The value of this property is impacted by the gateway's Resolve Client Hostname setting. While <b>Resolve Client Hostname</b> is enabled, the ga attempt a reverse DNS hostname lookup. Whatever is returned by this lookup will be set as the value for <b>host</b>.</td></tr>
<tr><td>theme</td><td>The theme to use in the Session. The default theme is <code>light</code>. Writing a theme name to this property will change the theme for the Session.</td></tr>
<tr><td>locale</td><td>The current locale of this Session.</td></tr>
<tr><td>timeZoneId</td><td>Timezone identification code, for example America/Los_Angeles.

This sessionProps.timeZoneId is the root property. The other timezone.id properties you may see are gateway.timezone.id and device.timezo both read-only representations of what the Gateway and user agent (mobile device, browser, workstation, etc.) report back to the session. Yo of these properties to override the sessionProps.timeZoneId by setting the project property Timezone Behavior for Perspective to Gateway or update the timeZoneId session property on session startup to the reported value.

Beyond manually updating in Session Props or binding the property, you can also update the timeZoneId session property so Perspective will dynamically.</td></tr>
<tr><td>lastActivity</td><td>The following feature is new in Ignition version <b>8.1.8</b>
Click here to check out the other new features

A readonly that represents the last time the session was interacted with, such as a user clicking within the session. The time reported by this off of the gateway's time.</td></tr>
<tr><td>auth</td><td>Represents the user's authentication and authorization for this Session.
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>authenticated</td><td>True if the user is authenticated. False if the user is unauthenticated. Null if the user's authentication status is unknown.</td></tr>
<tr><td>user</td><td>Contains information about the user, if they are authenticated.
<table>
<tr><th>Name</th><th>Description</th><th>Property Type</th></tr>
<tr><td>id</td><td>The IdP's unique identifier for this user. Null if the user is not authenticated.</td><td>value: string</td></tr>
<tr><td>userName</td><td>The user's username. Null if the user is not authenticated.</td><td>value: string</td></tr>
<tr><td>firstName</td><td>The user's first name. Null if the user is not authenticated, if the IdP did not provide this attribute, or if no mapping was configured for this attribute.</td><td>value: string</td></tr>
<tr><td>lastName</td><td>The user's last name. Null if the user is not authenticated, if the IdP did not provide this attribute, or if no mapping was configured for this attribute.</td><td>value: string</td></tr>
<tr><td>email</td><td>The user's email address. Null if the user is not authenticated, if the IdP did not provide this attribute, or if no mapping was configured for this attribute.</td><td>value: string</td></tr>
<tr><td>roles</td><td>The roles that the IdP assigned this user. Null if the user is not authenticated, if the IdP did not provide this attribute, or if no mapping was configured for this attribute.</td><td>value: string</td></tr>
<tr><td>timestamp</td><td>A timestamp representing the last time the current user authenticated against the Identity Provider.</td><td>value: timestamp</td></tr>
</table></td></tr>
<tr><td>securityLevels</td><td>The deepest security levels in the tree granted to the current user, starting with the children of the Public security level. The Publi security level is never shown since all Sessions include Public.
<table>
<tr><th>Name</th><th>Description</th><th>Property Type</th></tr>
<tr><td>name</td><td>The name for this security level. Must be unique among its siblings.</td><td>value: string</td></tr>
<tr><td>children</td><td>Security levels which descend from this security level.</td><td>array</td></tr>
</table></td></tr>
</table></td></tr>
</table>

idpId	The identity provider's ID. Replaced by the `idp` property
idp	The name of the Identity Provider configuration set on the project.
idpAttributes	Represents the JSON object returned by the identity provider after logging in. The structure of this object will match that of the JSON provided by the Test Login Identity Provider page. **Note:** The Designer does not authenticate against identity providers in Ignition 8.0, so this object will always appear empty in the designer. Use the Test Login page to determine the shape of this property, or use a simple binding to something visual (i.e. a label) display and parse the results while developing your project. **Note:** Ignition currently supports the Ignition and OpenID Connect (OIDC) Response Documents, but not SAML. Attempting to use a SAML IdP will result in the idpAttributes property containing an empty string. This note will be updated once support for SAML Response Documents has been implemented. ***Editor notes are only visible to logged in users*** **The note directly above these Editor Notes is from this request in the Training channel. This should be updated once IGN-4637 has been completed.**

gateway

Properties for the Gateway that this Session is running on. The value of this property will not be saved with the view. Does not persist by defa

Name	Description
address	Remote host address of the connected Gateway.
timezone	Document providing time zone information.
connected	True when connected to a websocket. All tabs in the Session must have a connected websocket to become true. If a Session disconnects but is still open in the browser, the property will not change, as the property write from the Gateway can't write to the disconnected Session.

timezone:

Name	Description	Property Type
id	Time zone identification code, for example America/Los_Angeles.	value: string
name	Name of the timezone.	value: string
utcOffset	Offset of the current timezone relative to UTC, in hours.	value: numeric

device

Properties for the device that is running the Session.

Name	Description
type	Type of device that created this Session. Read only. Options are ios, android, designer, browser, and workstation. Empty string if device is unknown during loading.
identifier	Unique ID representing this device. This is a convenience property not intended/suited for security purposes. May change via device/application re-installs or browser cache clears.
timezone	Document providing time zone information.
userAgent	User agent string of the connected device.
settings	Array of settings for the device.

timezone:

Name	Description	Property Type
id	Time zone identification code, for example America/Los_Angeles.	value: string
utcOffset	Offset of the current timezone relative to UTC, in hours.	value: numeric

settings:

Name	Description	Property

		Type
pullToRefresh	The following feature is new in Ignition version **8.1.5** Click here to check out the other new features If true, swiping down from the top of page and holding for two seconds will refresh the project in the mobile Perspective App. If false, swiping down will scroll the screen. Default is true.	value: boolean
preventSleep	Prevents the device from sleeping while viewing project in the mobile Perspective App. Default is false.	value: boolean

accelerometer

When continuous read mode is active, represents values retrieved from the accelerometer.

Name	Description	Property Type
timestamp	Timestamp represented as standard 'milliseconds since unix epoch'.	value: string
x	Acceleration force (in m/s^2) along the x axis (including gravity).	value: numeric
y	Acceleration force (in m/s^2) along the y axis (including gravity).	value: numeric
z	Acceleration force (in m/s^2) along the z axis (including gravity).	value: numeric

bluetooth

Options and data provided by device Bluetooth services.

Name	Description
enabled	If true, enables bluetooth capability.
options	Bluetooth options.

Name	Description	Property Type
updateInterval	How often should the Session check for new data packets. Duration in ms to buffer Bluetooth data before sending to Perspective.	value: numeric
limit	Maximum number of packets to display. The order of packets is strongest RSSI (Received Signal Strength Indicator) to weakest.	value: numeric
filter	Bluetooth filtering options.	

Name	Description	Property Type
enabled	If true, will enable filtering on the packets.	value: boolean
minimumRSSI	Minimum strength of RSSI to return. Enter 0 to ignore.	value: numeric
altBeacon	AltBeacon format.	object

Name	Description	Property Type
exclusive	Exclude other beacon types that are not altBeacon.	value: boolean
uuid	The 16 byte beacon identifier. Ignores packets that don't match the value specified.	value: string

Name	Description	Property Type
eddystone	Eddystone open beacon format.	object

Name	Description	Property

		Type
exclusive	Exclude other beacon types that are not eddystone.	value: boolean
nameSpaceID	Namespace identifier. Ignores packets that don't match the value specified.	value: string

iBeacon	iBeacon format.	object

Name	Description	Property Type
exclusive	Exclude other beacon types.	value: boolean
uuid	16 byte proximity uuid of iBeacon. On iOS this must be specified in order to receive iBeacon data.	value: string

data	Will populate with most recent packets from any detected beacons.

geolocation	Options and data provided by web or native device geolocation services.

Name	Description
enabled	If true, will attempt to populate location data into the 'data' property.
permissionGranted	If geolocation is enabled and a geolocation permission prompt is requested, this field populates true if the user allowed permissio Otherwise, it is false. Read only.
options	

Name	Description	Property Type
accuracy	Indicates the mode of accuracy the application uses to receive results: max, balanced, and low.	value: boolean

Name	Description	Property Type
max	Maximum accuracy (and highest battery use). Accurate to the level allowed by the environment/device.	value: string
balanced	Balanced accuracy - accuracy resolves ~100m (about a city block) using a more efficient poll rate and supplementing with device data. Balanced is the default value.	value: string
low	Low accuracy typically does not use a GPS sensor, but relies on environmental meta data (such as cell tower information, Wi-Fi connectivity, etc.). Most efficient, accurate to approximately town/3 kilometers.	value: string

maximumAge	A positive long value indicating the maximum age in milliseconds of a possible cached position that is acceptable to return. If set to 0, it means that the device cannot use a cached position and must attempt to retrieve the real current position. If set to infinity, the device must return a cached position regardless of its age. Default is 0.	value: numeric

data	If geolocation is enabled and the device can provide geolocation data, this will hold information about location.

Name	Description	Property Type
latitude	A floating point value representing the position's latitude in decimal degrees. Null if location is disabled.	value: float
longitude	A floating point value representing the position's longitude in decimal degrees. Null if location is disabled.	value: float
altitude	A double representing the position's altitude in meters, relative to sea level. This value can be null if the implementation cannot provide the data.	value: double
accuracy	A double representing the accuracy of the latitude and longitude properties, expressed in meters.	value:

		double
altitudeAccuracy	A double representing the accuracy of the altitude expressed in meters. May be null if device fails to provide or if geolocation is disabled.	value: double
heading	Returns a double representing the direction in which the device is traveling. This value, specified in degrees, indicates how far off the device is from heading true north. 0 degrees represents true north, and the direction is determined clockwise (which means that east is 90 degrees and west is 270 degrees). If speed is 0, heading is NaN. If the device is unable to provide heading information, this value is null.	value: double
speed	Returns a double representing the velocity of the device in meters per second. This value can be null.	value: double
timestamp	Time the last location update was received.	value: string

appBar

Settings relevant to the bottom-docked App Bar, which lists Gateway information.

<table>
<tr><th>Name</th><th>Description</th><th>Property Type</th></tr>
<tr><td>togglePosition</td><td>The position of the overlaid toggle button that shows the app bar: <b>right</b>, <b>left</b> or <b>hidden</b>.</td><td>value: string</td></tr>
<tr><td>about</td><td>The following feature is new in Ignition version <b>8.1.20</b>
Click here to check out the other new features

Settings allowing for more customization of the bottom-docked App Bar.
<table>
<tr><th>Name</th><th>Description</th><th>Property Type</th></tr>
<tr><td>show</td><td>Determine if a custom about page should be shown.</td><td>value: boolean</td></tr>
<tr><td>icon</td><td>The path of the about button icon.</td><td>value: string</td></tr>
<tr><td>path</td><td>Path of the view to display in the about modal.</td><td>value: string</td></tr>
<tr><td>title</td><td>The title of the about modal.</td><td>value: string</td></tr>
</table></td><td>object</td></tr>
</table>

pipes

The following feature is new in Ignition version **8.1.10**
Click here to check out the other new features

An object containing properties that pertain to pipes.

Name	Description
autoAppearance	The styling to use when a set of pipes is set to an "auto" style.
overlapGap	The following feature is new in Ignition version **8.1.18** Click here to check out the other new features The width of the gap to draw when P&ID pipes overlap. When this property is set to 0 or a negative number, no overlap is rendered. This may result in a rendering performance boost when using complex P&ID pipes. Default is 4.

symbols

The following feature is new in Ignition version **8.1.0**
Click here to check out the other new features

Settings relevant to the components on the Perspective Symbols Palette.

<table>
<tr><td></td><td>
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>autoAnimationSpeed</td><td>Sets the animation speed for any Perspective symbol components that have their animationSpeed property set to auto.</td></tr>
<tr><td>autoAppearance</td><td>Sets the appearance for any Perspective symbol components who have their appearance property set to auto. Options are auto, p&id, mimic, and simple.</td></tr>
</table>
</td></tr>
<tr><td>googleMapsApiKey</td><td>The following feature is new in Ignition version <b>8.1.33</b>
Click here to check out the other new features
Google Maps API Key. This key is required to use a functional version of the Google Map component.</td></tr>
<tr><td>address</td><td>The following feature is new in Ignition version <b>8.1.10</b>
Click here to check out the other new features
Represents the IP address of the Session as the gateway sees it.</td></tr>
</table>

Perspective Session Proxy Considerations

In cases where a reverse proxy or other middleware is located between the Gateway and a running Perspective session, then the proxy server will need to whitelist several custom headers. Doing so insures that the proxy allows requests from the session to pass through to the gateway.

Prior to 8.1.17

- client_timezone
- device_id
- device_type
- version_code
- perspective-session-id
- designer-session-id

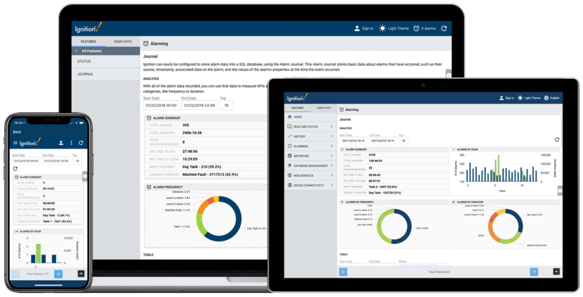
8.1.17+

- client-timezone
- device-id
- device-type
- version-code
- perspective-session-id
- designer-session-id

Perspective Design Tips

There are many things to consider when designing for mobile-responsive applications that can run in a web browser. Designs must work everywhere, on any device from a small smartphone to an oversized desktop, in an optimized format, using one design. Unfortunately, there is no one design strategy that fits all projects, but you can have a single design for a project that works well across many different screen sizes based on content and context.

On this page ...



With mobile devices ranging from smartphones to tablets, understanding how your website will display on the variety of formats is critical. This page provides some design principles to consider when designing in Perspective, but it only scratches the surface. You can augment these design principles with the design processes from your company.

Before designing your first project in Perspective, understanding terms like mobile optimized, responsive design, and user experience are sure to come up in conversation when defining the requirements for your project.

- **Responsive Design** - is a method of developing web pages that are completely flexible and renders on any device type. Responsive designed websites "respond" to the screen size of the device being used from smartphones to industrial monitors.
- **Mobile Optimized** - means the website will reformat itself for mobile devices such as smartphones and tablets.
- **User Experience** - can separate a good a successful app from an unsuccessful one. You want users to have a quality experience using your app such as fast loading time, easy to use, and navigation for starters.

Building a good mobile-responsive app starts with defining specific goals and objectives to clearly identify the problem you're trying to solve whether your starting a new project or redesigning an exising one. Equally important is understanding the needs of the users and integrators. Some things to ask your users about is, what types of devices they plan to use to access the project, what type of data they want to see, are they viewing data, entering data, or controlling equipment. What do they want the components on the screen to do. Do they need to work offline.

It's extremely important to capture and document all the user requirements. When doing requirements gathering, it's also important to audit the workflow process. This way you get to see firsthand what the users are doing so you can define requirements prior to entering the design phase. This doesn't mean that you can't refine requirements as you progress through the design process. You most certainly can!

Designing for Mobile-Responsive Applications

You have choices when it comes to designing in Perspective. You can design for desktop or mobile devices, or both. Ignition Perspective allows you to design your apps to work well across many different screen sizes, whether a user is seated in the control room, grabbing a tablet to walk into the field, or receiving a notification to respond to a crisis after hours.

One of the first considerations when tackling a design project is to determine the types of devices your users are going to use to access the project. This often dictates whether you design for a desktop, mobile devices, or both. Something to keep in mind, is designing for multiple screen sizes at the start of your project will be far easier, less time consuming, and less costly than adding it on later. This way you have one design that adapts to many screen sizes.

Think of a responsive design as a layout strategy, allowing your apps to work well across many screen sizes. It's a single design that reorganizes and responds to the available browser space to display the same content in a more usable way whether your using a smartphone, table or desktop. You also get a consistent UI and functionality across various screens, and it works everywhere. If you open a responsive site on the desktop and then change the size of the browser window, the content will move dynamically and arrange itself optimally for the browser window. On mobile devices, this process is automatic; the site checks for the available space and the reformats itself in the size matching the device while optimizing the content for an ideal arrangement. This is responsive design!

Design Principles

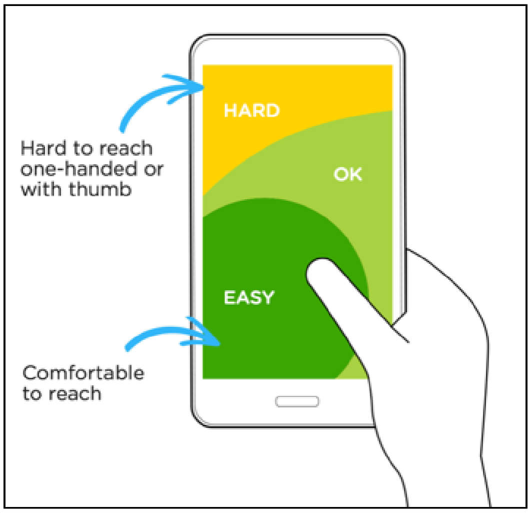
We all know what good design looks like in the desktop world. Now let's expand our knowledge to mobile devices using the following design principles.

Touch and Ergonomics

Good design begins with touch and ergonomics. On mobile devices all input is touch driven and have smaller touch targets. The smaller the target, the harder they are to use, and the more chance for errors. For example, turning on/off an industrial motor or some other critical process can lead to serious repercussions in the event the wrong button is inadvertently pressed.

When designing for mobile devices like smartphones and tablets, it's important to make touch targets big enough so they are easy for users to tap. As a good rule of thumb for sizing touch targets; 25 pixels is considered touchable, 40 pixels is optimal, and at a minimum, 10 pixels between elements.

Next, let's talk about the ergonomics of mobile devices. Designing for mobile devices isn't only about making targets big enough, it's also about considering the way we hold our devices. There is a comfortable area for touch on a screen called the 'thumb zone'. A touch device like a smartphone, the bottom of the screen is the best area for your most important actions because it's easy and most comfortable to reach with your thumb. You might consider placing destructive actions in hard to reach areas such as the top of the screen if you don't want them pressed accidentally. Also, keep in mind that the bigger the display area is on a smartphone, more of the screen is less easily accessible.



Be sure to test your applications on different operating systems and devices, including all sizes of smartphones and tablets. Even better, is to have some of your users do some testing. Nothing works better than getting some real feedback from your users.

Mobile-First Design Approach

The Mobile-First design approach prioritizes the app design on smaller devices instead of designing a desktop app and then forcing it to fit into a mobile box. A mobile-first approach is exactly as it sounds: designing for the smallest screen and working your way up. It is one of the best strategies to create either a responsive or adaptive design. This design strategy focuses on progressive enhancement of features and content as browser size and available space increases. Mobile-first design can lower development costs over the lifetime of your applications because single data model is maintained instead of maintaining several specific solutions for specific devices.

Fluid Content

Fluid design offers users a consistent experience across multiple devices and screen sizes. In responsive design, you want to design your app with a fluid layout that works good on small and large screens. You don't want them to look too busy on small screens and too empty on big ones. Think of web content as a box. In a responsive app, these boxes are going to move and change depending on their container. Perspective containers provide a way of laying out and organizing components within a view. This is really important because containers and views are an integral part of the Perspective design experience and play a major role in creating fluid content. Not only does Perspective containers and views allow you to create fluid content, but they also provide reusability throughout your existing project and other projects.

Content as UI

You can't talk about content without talking about design. Together, they both create a great user experience. By including design techniques in your discussions about content, you truly discover what is important and how to communicate to the user. We are all familiar with how various interface elements behave on a screen and how to directly interact with the content, but each design is different.

As a designer, the content UI focuses on what interface elements the user needs to understand to easily access content, so you want to be consistent and predictable. There are many different types of interface elements such as input controls, navigational and informational elements to name a few. It's important to know your users so you make the right choices so here are a few things to keep in mind:

- Keep the content UI simple.
- Be purposeful when designing your page layout.
- Create consistency and use common UI elements.
- Provide visual clues about behavior before actions are taken.

Declutter

Good UI design is about delivering relevant information and avoiding irrelevant information. By cluttering your interface with elements and content, you overwhelm users with too much information. Every element you add makes the screen more complicated, and if looks complicated on a desktop, it's even worse on mobile devices where there's not a lot of real estate. It's imperative to remove anything in the UI that isn't absolutely necessary. Decluttering the screen will improve the user's comprehension. A good rule of thumb is "less is more."

With limited screen space, you have to create focused content. Here's a couple things to remember:

- Keep content to a minimum and present the user with only what they need to know.
- Keep interface elements to a minimum. A simple design will make your UI more intuitive and ultimately more productive.

Understanding Views and Containers

We covered some of the basic design principles for designing mobile responsive apps, but we must also stress that views and containers are an integral part of the Perspective design experience because they work together to create your HMI screens, the windows into your application. The View is the primary unit of design and the Container provides a way of laying out and organizing child components within a View. Every view and container in Perspective has an associated layout, simply put, it is a way of defining and describing the way that elements inside the container interact.

A container is also a component that contains other components. They also indicate what layout strategy should be used to control the size and shape of any housed components. The layout strategy defines how the container displays each of its child components in the view. There are a variety of container types that support different layout strategies.

It's also important to understand how each of the container types behave when they are big and small. This dictates what container types you should use for your design. There is a bit of learning curve so you will need to experiment with each container type and physically test them out.

To learn more about each container type, refer to the section on Containers in Perspective.

Pages in Perspective

On this page ...

Pages are the main method of navigation in Perspective. When talking about pages, there are two distinct objects to be aware of.

- **Page** - A collection of views that are displayed within a single tab of a browser during runtime. A page consists of at least one view (called a "primary view"), any number of additional views that can be docked along the edges (called "docked views"), and a partial URL associated with the page,
- **Page Configuration** - A collection of settings that dictate how the page will behave during runtime, including which views are present on a page, as well as rules for docked views in the page.

Thus, Page Configurations are used to define pages that can be opened in the session, while Pages are running instances that are defined by the Page Configuration.

Methods of Navigation

Users will traverse the various views in a project by navigating to different pages. There are many ways to navigate to a page in Perspective.

- The Navigation Component Action is the easiest way to switch to a different page based on some user action.
- The system.perspective.navigate scripting function can be used as an alternative to the component action if more logic is desired.
- Some components have built-in navigational functionality, such as the Link and Menu Tree components.

Furthermore, all of the methods mentioned above can be used to navigate to arbitrary web addresses, allowing you to add links to popular sites that might be useful to your users.

INDUCTIVE UNIVERSITY

Creating Pages

Watch the Video

Page Configuration

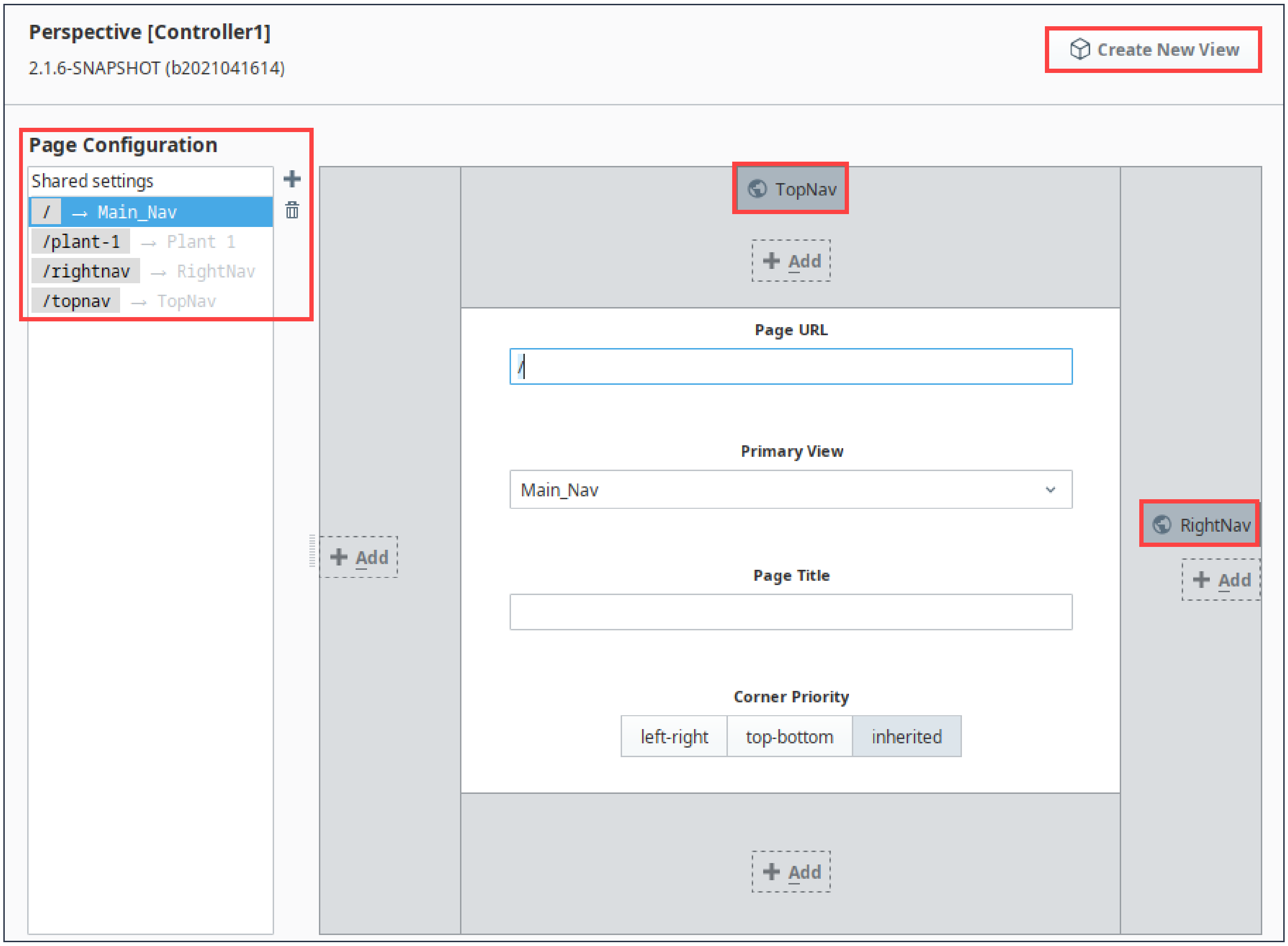
All page configuration happens on the Perspective Page Settings tab. Any number of pages can be created for a project, each with their own Primary View and Docked Views. When configuring a docked view in Page Configuration you can choose to dock the view on a specific Page URL with a set primary view, or you can choose to share the view across all pages under Shared Settings. Each project can configure a **Shared Settings** that will be “inherited” by all pages in that project, allowing multiple pages to use the same docked views and corner priority settings.

You can open the Page Configuration from any view in the Perspective workspace by clicking on the **Perspective** icon at the bottom of the Designer window. Here you can assign the Page URLs to your primary views and add docked views to your pages. You can even create a new view without going to the Project Browser.

Below is an image of a Page Configuration screen. The Page Configuration column on the left side shows all the Page URLs assigned to views in your project. You can click on the **Shared Settings** to see all docked views that are shared across all pages of your project. To see docked views that are specific to a Page, navigate through each page of your project. In the following example, you'll notice that the **TopNav** view is a shared view across all pages, and the Plant 1 view has one docked view called, **RightNav**, which is specific to this page and only visible when this page is open in your browser.

The following feature is new in Ignition version **8.1.6**
Click here to check out the other new features

As of 8.1.6 the Page Configuration page includes a new setting called Page Title, where you can set a title for the page being edited. If the field is left empty, the browser's tab for the page will be the project name. If there is text in the field, the browser's tab for the page will be the exact content of the field.



Perspective [Controller1]

2.1.6-SNAPSHOT (b2021041614)

Page Configuration

Shared settings	
/	→ Main_Nav
/plant-1	→ Plant 1
/rightnav	→ RightNav
/topnav	→ TopNav

Page URL: /

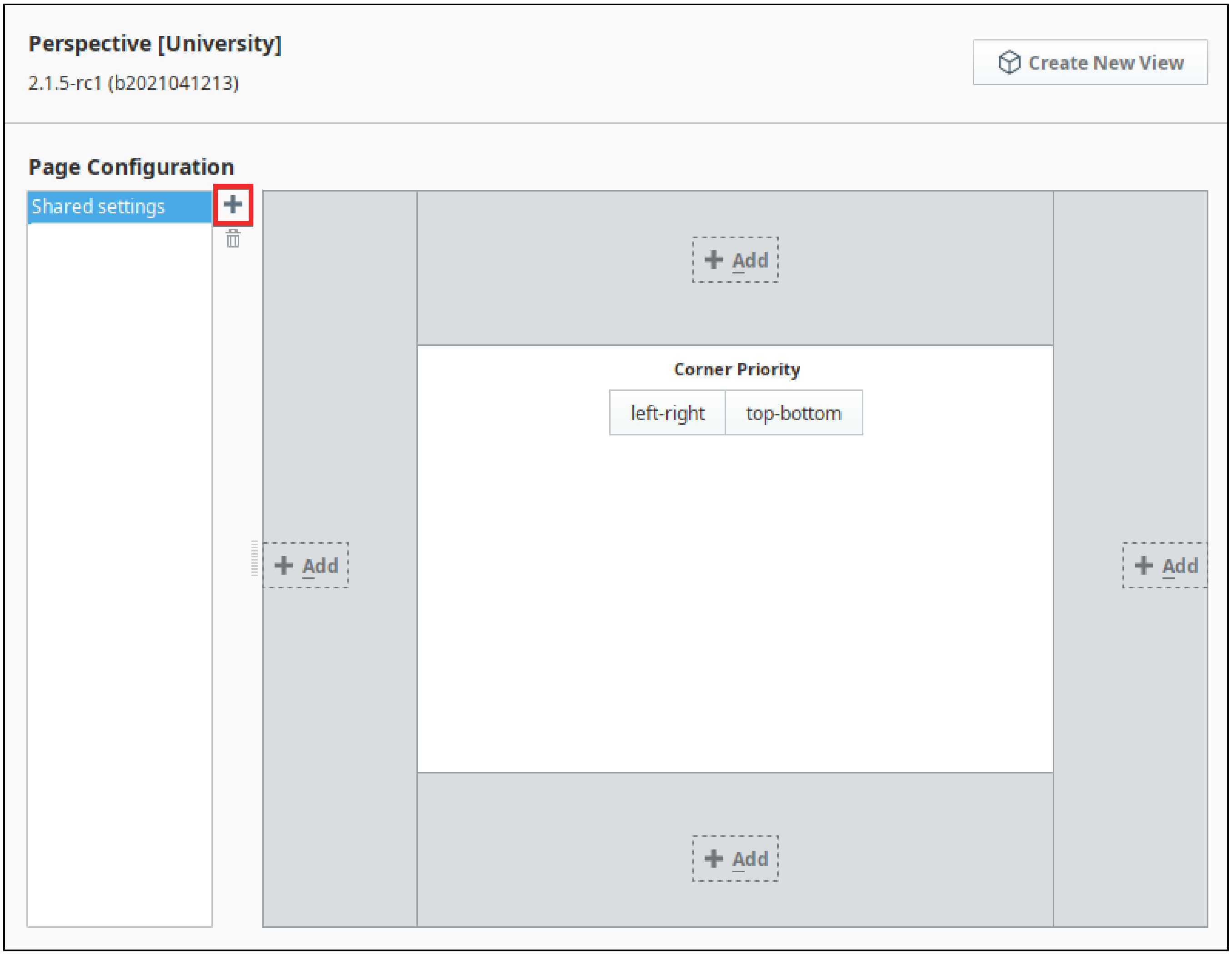
Primary View: Main_Nav

Page Title:

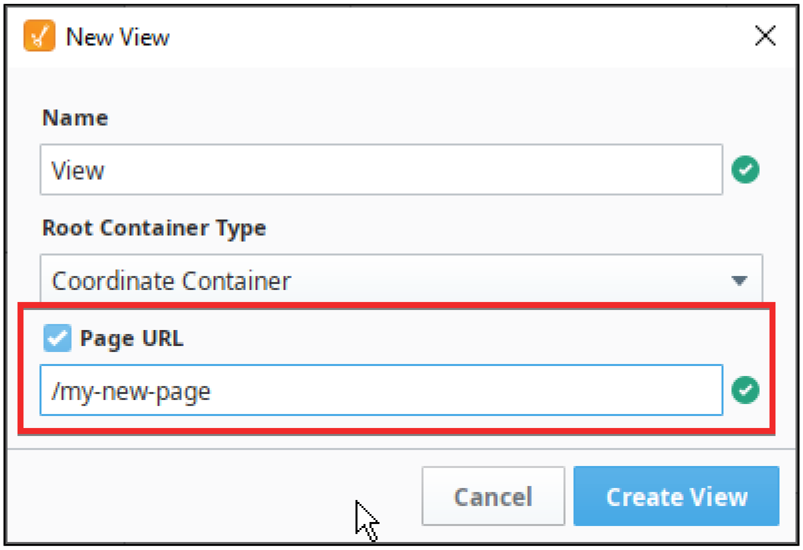
Corner Priority: left-right | top-bottom | inherited

Creating a Page Configuration

Pages can be manually created by pressing the **Add** ✚ icon under the Page Configuration area of the Perspective Workspace. Pages created this way will need to have a primary view specified.



Alternatively, whenever a new view is added to a project, a page can be created automatically by opting into the **Page URL** setting and typing value for the Page URL. Note that all Page URLs should start with a forward slash character ("/"). Creating a page through this method will automatically set the newly created view as the primary view on the new page.



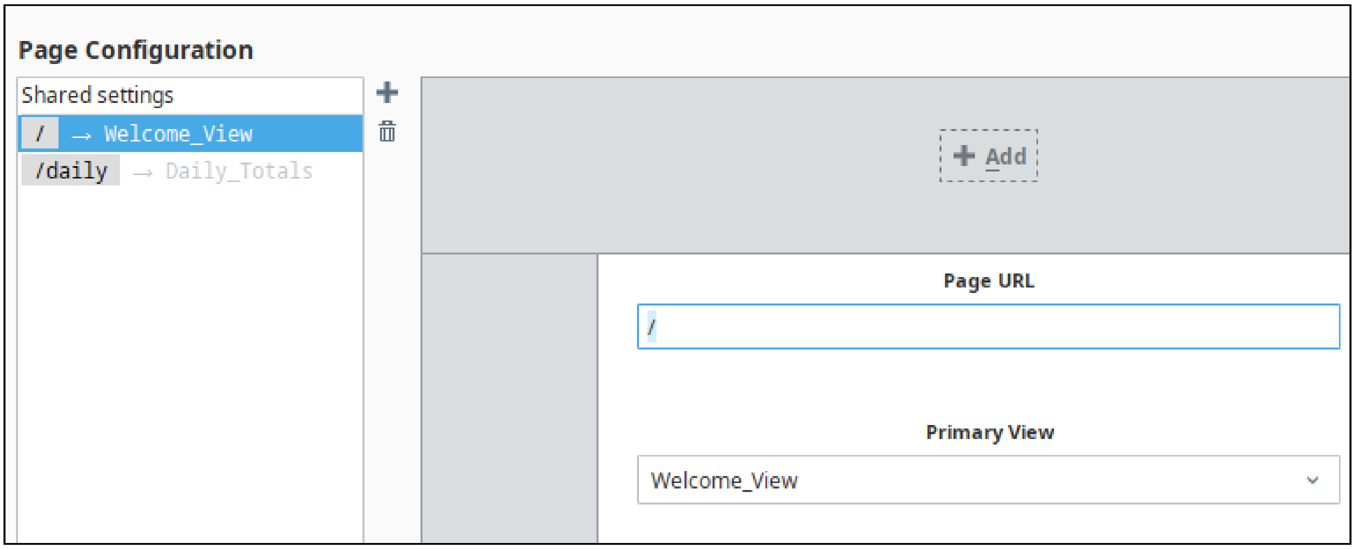
Name: View

Root Container Type: Coordinate Container

☑ Page URL: /my-new-page

Initial Page URL

Perspective assumes the initial page within a project will be located at page URL "/". When a session is launched, the session will start at the "/" page. It's recommended that the "home" or starting view for your project uses a page URL of just "/".

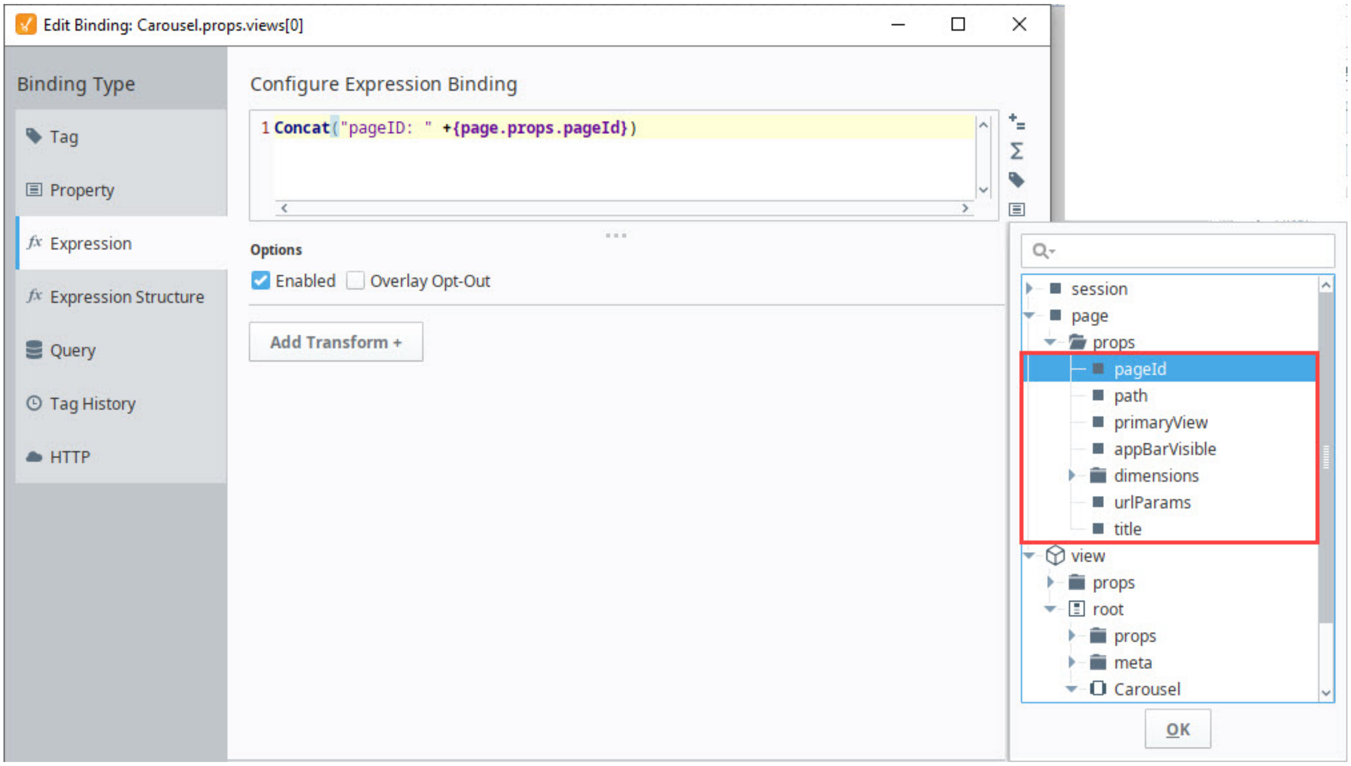


Page URL: /

Primary View: Welcome_View

Page Properties

Like most objects in Perspective, pages have their own properties. However, since pages aren't selectable in the Designer, page properties can't appear in the Perspective Property Editor. However, pages do have properties that can be accessed via component property bindings and scripts.



```
Concat("pageID: " +{page.props.pageId})
```

☑ Enabled ☐ Overlay Opt-Out

Property	Description	Property Type
pageId	The unique identifier associated with the page. Can be used with other features to identify the page, such as the system .perspective.navigate function. Note, that the value of of this property in the Designer will match the name of the active view. However, during a session the value will return a uniquely generated string, such as "2bf737f8".	value: string

<table>
<tr><td>path</td><td>Represents a string "path" to the page. Effectively the same as the Page URL.</td><td>value: string</td></tr>
<tr><td>primaryView</td><td>Represents a path leading to the page's primary view. The "path" is based off of the view's location in the Project Browser, starting under the Views item, and including folders.</td><td>value: string</td></tr>
<tr><td>appBarVisible</td><td>A Boolean value representing whether or not the app bar is visible on the current page. Writing to this value will hide or show the app bar.</td><td>value: boolean</td></tr>
<tr><td>dimensions</td><td>
<p>The following feature is new in Ignition version <b>8.1.6</b>
Click here to check out the other new features</p>
<p>Settings for the dimensions on a page.</p>
<table>
<tr><th>Property</th><th>Description</th><th>Property Type</th></tr>
<tr><td>enabled</td><td><p>The page dimensions in this object will update accordingly if true. While false, the dimension values will not update.</p><p>The remaining dimension properties in this section are read-only.</p></td><td>value: boolean</td></tr>
<tr><td>viewport</td><td>
<p>Current settings for the "inner" boundaries of the browser, i.e., the dimensions of the page (read-only).</p>
<table>
<tr><th>Property</th><th>Description</th><th>Property Type</th></tr>
<tr><td>width</td><td>Width of the inner boundary.</td><td>value: numeric</td></tr>
<tr><td>height</td><td>Height of the inner boundary.</td><td>value: numeric</td></tr>
</table>
</td><td>object</td></tr>
<tr><td>screen</td><td>
<p>Settings for the dimensions of the screen the page is on. Note that this only updates on browser resize and page startup (read-only).</p>
<table>
<tr><th>Property</th><th>Description</th><th>Property Type</th></tr>
<tr><td>width</td><td>Width of the screen the page is on.</td><td>value: numeric</td></tr>
<tr><td>height</td><td>Height of the screen the page is on.</td><td>value: numeric</td></tr>
</table>
</td><td>object</td></tr>
<tr><td>primaryView</td><td>
<p>Settings for the dimensions of the primary view in use for the page, and how much scroll is currently applied to that view (read-only).</p>
<table>
<tr><th>Property</th><th>Description</th><th>Property Type</th></tr>
<tr><td>width</td><td>Width of the primary view.</td><td>value: numeric</td></tr>
<tr><td>height</td><td>Height of the primary view.</td><td>value: numeric</td></tr>
<tr><td>scrollLeft</td><td>Distance in pixels that the view scrolled to the left.</td><td>value: numeric</td></tr>
<tr><td>scrollTop</td><td>Distance in pixels the view is scrolled from the top.</td><td>value: numeric</td></tr>
</table>
</td><td>object</td></tr>
</table>
</td><td>object</td></tr>
<tr><td>urlParams</td><td>
<p>The following feature is new in Ignition version <b>8.1.13</b>
Click here to check out the other new features</p>
<p>URL query parameters. Perspective supports the use of query parameters, such as those passed by an API call. When Perspective detects a ? in a URL string, any key:value pair supplied after the ? is mapped as a key:value pair within this object. This property is read-only.</p>
<p><b>Note:</b> All values passed in a URL are passed as Strings.</p>
</td><td>object</td></tr>
</table>

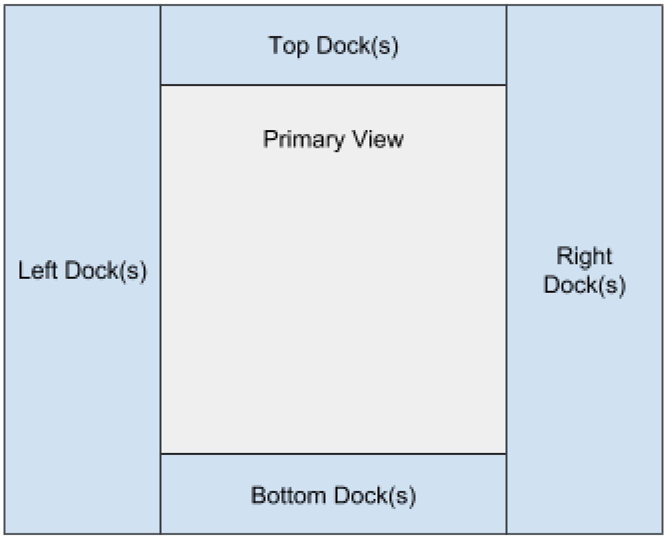
title	The title shown in the browser's tab. The value of this property defaults to the project's title if there is one. The property falls back to the projects name if a title isn't configured. The property is writeable, so scripts and bindings can be used to create a unique title.	value: string

Page Layout

Page layout has specific UI regions where you can place instances of your views. Depending on screen size and orientation, the UI regions have different behavior. There are six primary content regions: Center area, Top Dock, Bottom Dock, Left Dock, and Right Dock. There is also the Popup region that floats on top of the other content regions.

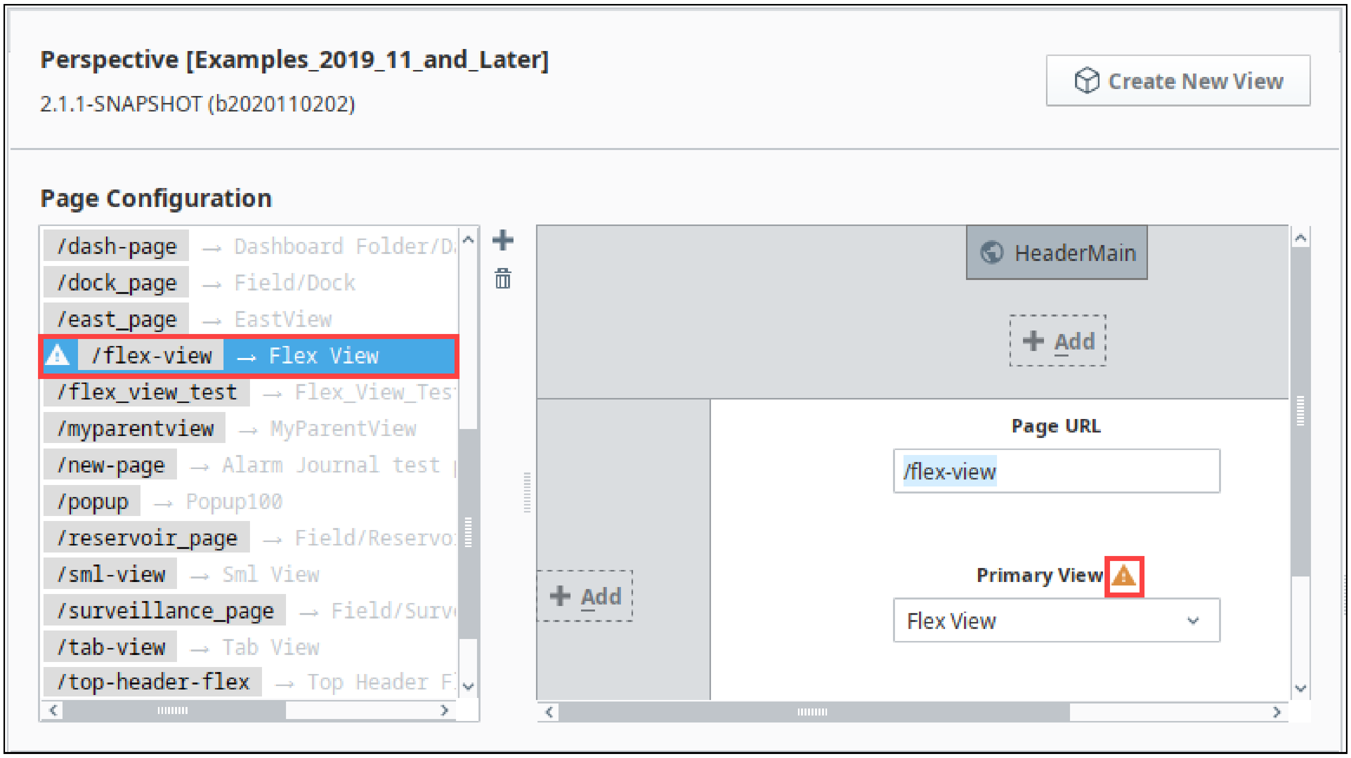
The Primary View is in the center of each page taking all available space in the browser window. Each page must have a single view configured to be its primary view.

Docked views, "docked" or positioned towards the edges of the Primary View. In a session, docked views can appear permanently along the edges or the browser, or be opened an dismissed by the user.



Page URLs

Perspective is designed to operate in a web browsing environment. A Page is the main navigational element in a Perspective Session, so when a Perspective Session starts, it typically begins at the page mounted to the Page URL "/". Although users can directly navigate to any page in a project if you know the URL.



Perspective [Examples_2019_11_and_Later]

2.1.1-SNAPSHOT (b2020110202)

Page Configuration

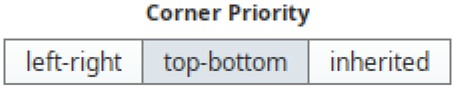
```
/dash-page → Dashboard Folder/D
/dock_page → Field/Dock
/east_page → EastView
/flex-view → Flex View
/flex_view_test → Flex_View_Tes
/myparentview → MyParentView
/new-page → Alarm Journal test
/popup → Popup100
/reservoir_page → Field/Reservo
/sml-view → Sml View
/surveillance_page → Field/Surv
/tab-view → Tab View
/top-header-flex → Top Header F
```

Page URL: /flex-view

Primary View: Flex View

Corner Priority

The Corner Priority setting determines which docked sides push all the way to the corners when the user navigates to that page: whether the top and bottom docked views get the full width of the page or whether the left and right docked views get the full height of the page. Whichever sides have priority, those docked views will extend on those sides to the edges of the page, thus shrinking the opposing sides down to fit within the page. If the Inherited option is selected, then the page will inherit the Corner Priority setting from the Shared Settings.



Corner Priority

left-right | top-bottom | inherited

Passing Parameters (URL Parameters)

Pages can be mounted at URLs that include parameters. These parameters are used to allow a page to be mounted at a dynamic URL, allowing information in the URL to be interpreted as input parameters to the page's primary view . For example, suppose we had a page that displayed information about a Tower site and our system had many Tower sites. Each Tower has an ID number that uniquely identifies it.

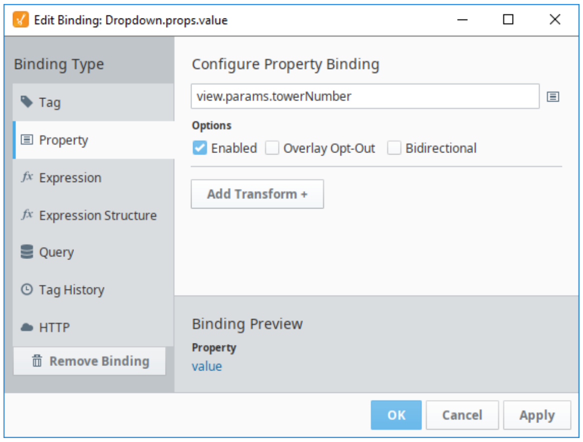
Our Tower site view, called Towers, needs an input parameter called **towerNumber** that is used in indirect bindings throughout the view configuration to allow this view to correctly display information about any Tower.

> ⚠ Certain characters can cause issues when passed as part of a parameter string in a URL. The following characters are invalid and should not be used in page parameters: / ? # %

INDUCTIVE UNIVERSIT

Passing Parameters to Pages

Watch the Video



Edit Binding: Dropdown.props.value

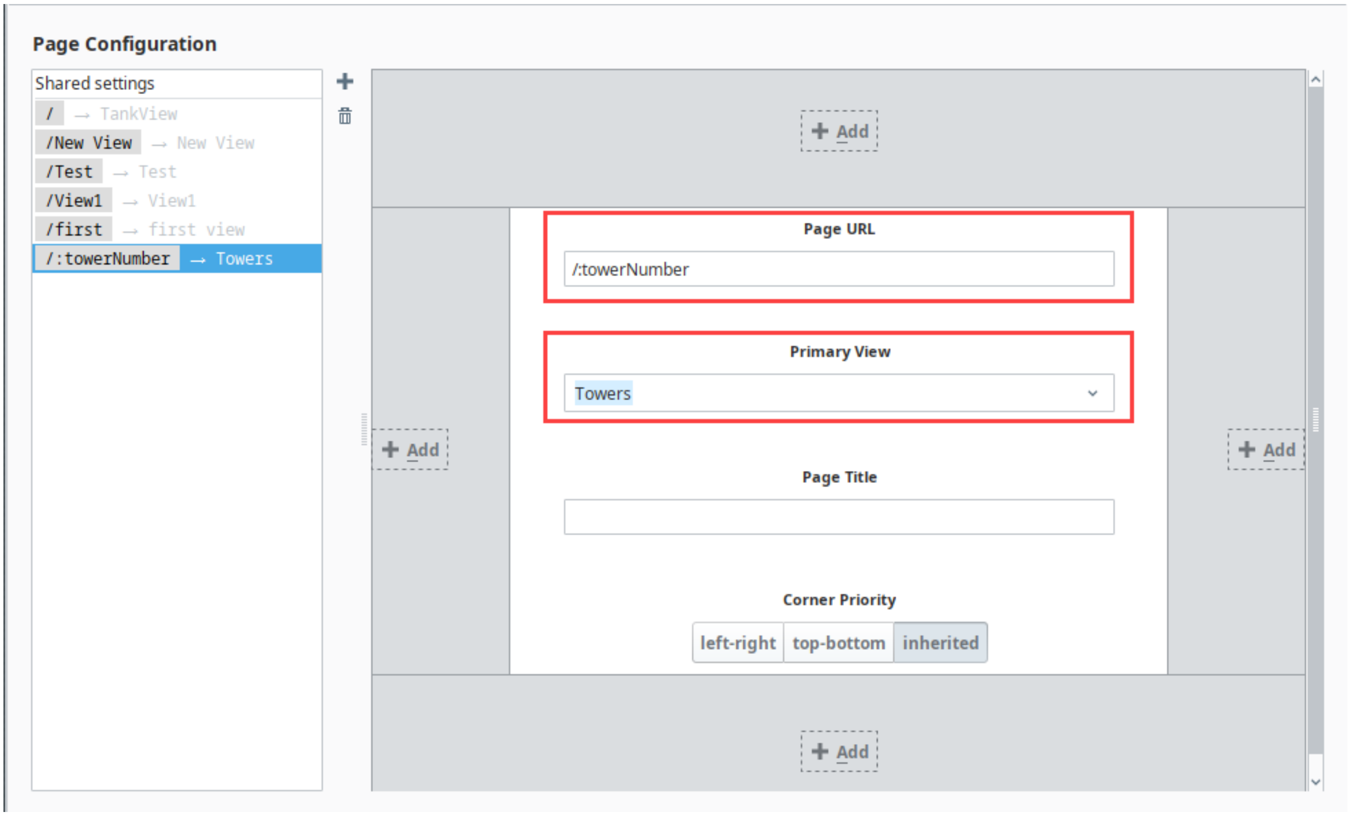
Binding Type: Property

Configure Property Binding: view.params.towerNumber

Options: ☑ Enabled □ Overlay Opt-Out □ Bidirectional

Binding Preview
Property
value

We configured our project to have a page, and set the page's primary view to **Towers**. We now need to configure the page's URL so that a user can navigate to any Tower using their browser's URL bar. To do this, we mount the page at a special URL using a parameter replacement syntax for the Page URL: **/:towerNumber** . Our dynamic URL mounting uses a colon to signify that a portion of the URL is meant to by dynamic and map to an input parameter on the page's primary view.



Page Configuration

Shared settings

```
/            → TankView
/New View    → New View
/Test        → Test
/View1       → View1
/first       → first view
/:towerNumber → Towers
```

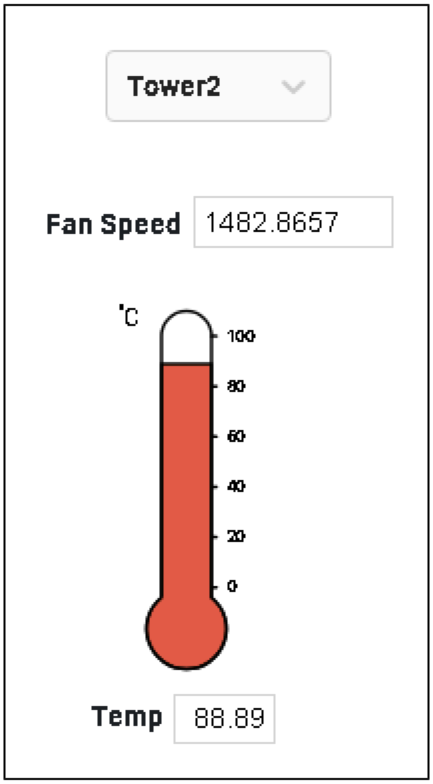
Page URL: /:towerNumber

Primary View: Towers

Page Title:

Corner Priority: left-right | top-bottom | inherited

Perspective understands this URL format. Therefore if the user were to navigate to the <URL> /Tower2, Perspective would render "**/**" as the primary view with an input parameter mapping of "towerNumber = Tower2", displaying the Tower2 information.



Adding parameter passing to an existing set of pages follows a strict formula. The table below assumes your project is named: MyProject.

Original Page URL Property	Parameter Name	New Page URL	Sample URL in Browser
/	`param`	`/:param`	http://localhost:8088/data/perspective/client/MyProject/value
/new-page	`param`	`/new-page/:param`	http://localhost:8088/data/perspective/client/MyProject/new-page/value
/new-page	areaNumber and lineNumber	`/new-page/:areaNumber/:lineNumber`	http://localhost:8088/data/perspective/client/MyProject/new-page/100/101

URL Query Parameters

The following feature is new in Ignition version **8.1.13**
Click here to check out the other new features

Perspective supports the use of query parameters, such as those passed by an API call. When Perspective detects a ? in a URL string, any text after the ? is mapped to the urlParams property. This property is read-only.

You may pass query parameters to a Perspective URL by manipulating the URL in your browser, or by a URL navigation call using system.perspective.navigate() and specifying the URL argument.

Note: Page navigation will not work for passing query parameters, since page navigation is used exclusively for navigating to pages within the Perspective session.

URL Example

```
system.perspective.navigate(url= "http://localhost:8088/data/perspective/client/MyProject/view?<key>=<value>")
```

Original Page URL Property	Parameter Name	New Page URL	Sample URL in Browser

/view	key	/view?key=value	http://localhost:8088/data/perspective/client/MyProject/view?key=8

Views in Perspective

Perspective views are unique in that they can act as both a top level screen (taking up a whole page in your session) or a component (embedded in another view). Each view is a project resource, which are named and organized into folders in the Ignition Designer’s Project Browser tree. These folders/paths are important not only for organization and referencing, but also because these paths uniquely identify each view, and are used in the session (runtime) for navigation. Each view has a Container type that decides how the components inside it will behave.

While views themselves are not strictly components, there are components that can display an instance of a view, such as nesting views inside of other views. There are also repeater components that may dynamically create view instances at runtime.

Views can be mounted in a Page as a primary, docked, or popup views. For more information on how pages work, see Pages in Perspective.

On this page ...

INDUCTIVE UNIVERSITY

Anatomy of a View

Watch the Video

View Properties

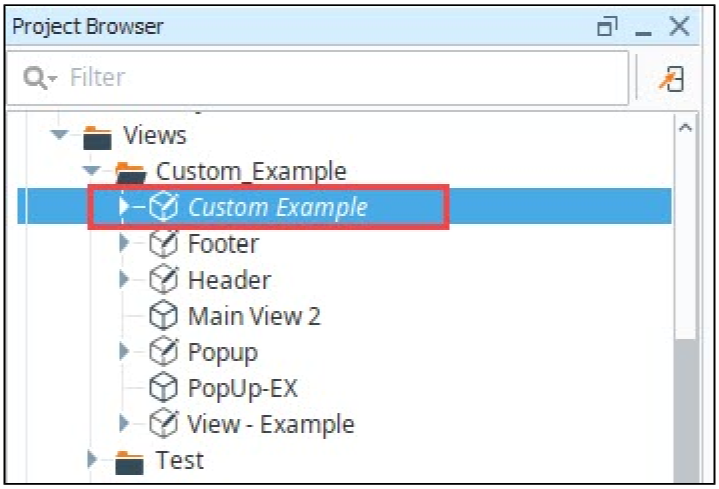
Views, like components, have properties. They are organized into a few types: **props**, **params**, and **custom**. Custom properties can be defined for views. They act just like custom properties of a component and are internal to the view, and can be referenced by all child components and containers in that view.

Each view contains exactly one “root” level container, which may be any of the available container types. Therefore, the design experience of a view is simply the design of the selected container type.

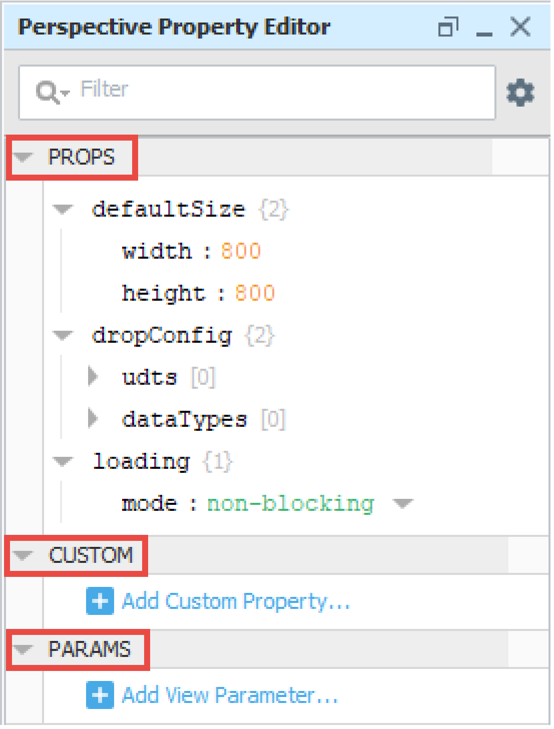
The view properties have three categories:

- **Props** - Properties used to configure the component's visual appearance, behavior, and data.
- **Custom** - Properties defined by the user. They have no direct effect on the component, but are used as variables for the application designer's convenience.
- **Params** - Properties only found on views. They define the parameters that may be passed in or out of that view.

For complete description of each of the view properties, see Perspective - View Object. To access the properties for a view, first select the view in the Project Browser.



The property types will now be displayed in the Perspective Property Editor.



```
defaultSize {2}
  width : 800
  height : 800
dropConfig {2}
  udts [0]
  dataTypes [0]
loading {1}
  mode : non-blocking
```

Input/Output Parameters

The properties inside of the **params** collection define parameters for the view. This is how views interchange information with other entities, such as a parent view or the page. Parameters must be defined as "input", "output" or "in/out". The default direction will be "input".

- An **input parameter** is not bindable from within the configuration of a view. The purpose of an input parameter is to receive information from an external entity. For example, when a view is opened, it may receive parameter values which will become the values of its input parameters. Or, if a view is placed inside another container, that instance of the view will show the input properties, and they will be bindable.
- An **output parameter** is the opposite. These parameters are bindable from within the configuration of the view. If an instance of that view is then placed inside another container, the output values will appear as properties, but not be bindable; they will be read-only from the outside.
- An **in/out parameter** combines the features of both input and output. An in/out parameter is bindable from both inside the definition of a view, and from the outside. This can be useful when making a view that acts as a decorator around an input, for example. Suppose you had a view that contained a Text Field component and an Image component, and the image displayed whether or not the text in the Text Field meets some criteria. You would use an in/out parameter to mirror the text across the view boundary.

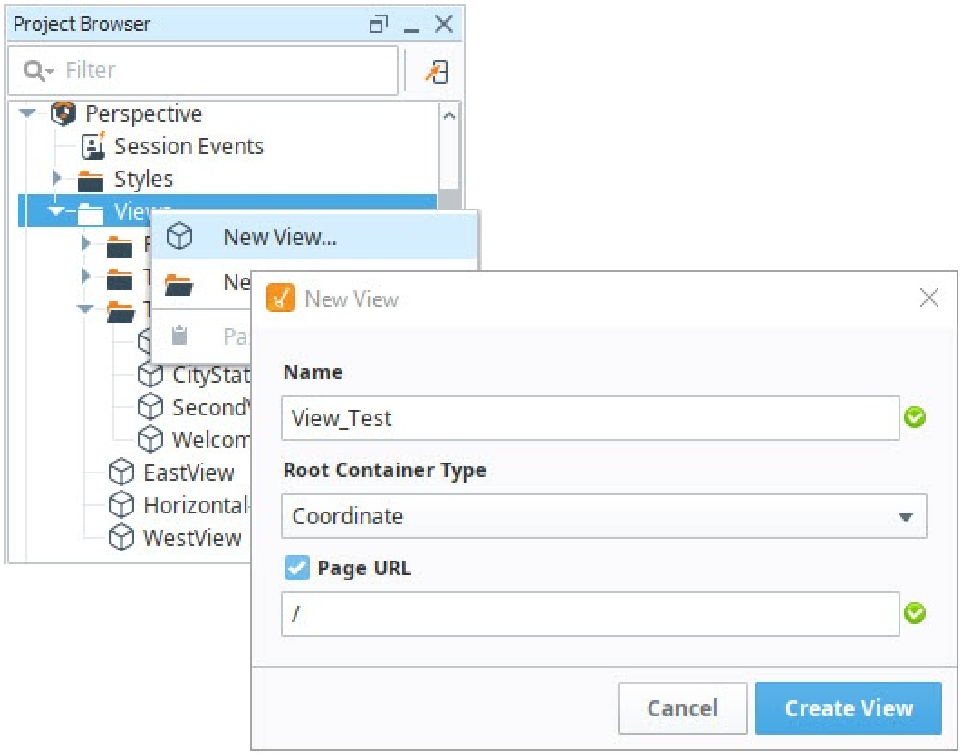
Configuring Views

Views can be configured in many different ways. They can be displayed as the entire browser window if it is configured to be the primary view of a page. Views can also be displayed across the top, bottom, or sides of the browser window if it is configured to be a docked view. A view can also be displayed floating on top of the page as a popup view. A view can also be embedded within another view in various ways using a variety of components that are able to display embedded views. Pages in Perspective talks more about page layout and how views are configured on a page.

When you start designing your project in Perspective, the first thing you'll do is create a view and how you want to configure it. You can configure it as a primary view attached to a Page URL, or as a docked view on a specific Page URL with a primary view, or you can choose your docked view to be shared across all pages in your project. You also have the option of creating folders for your views or simply leaving all your views in the Views folder.

1. In the Project Browser, right click on the Views folder and click **New View.**
   a. In the New View window, enter a name for your view.
   b. Select a root container type.
   c. Check the **Page URL** checkbox if you want your view attached to the Page URL. Perspective will match the Page URL with the view's folder structure upon creation.

d. Click **Create View**.



New View

Name

View_Test

Root Container Type

Coordinate

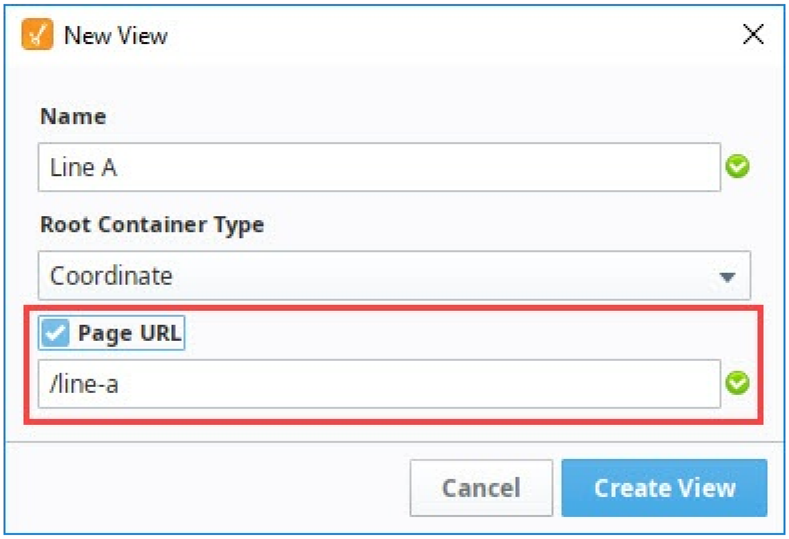
☑ Page URL

/

Cancel

Create View

When creating additional views, the Page URL will automatically fill in if the Page URL option is checked. If you use a space in your view name, the Page URL will replace the space with a hyphen. For example, a View named **Line A** will have a page URL of **/line-a**.



New View

Name

Line A

Root Container Type

Coordinate

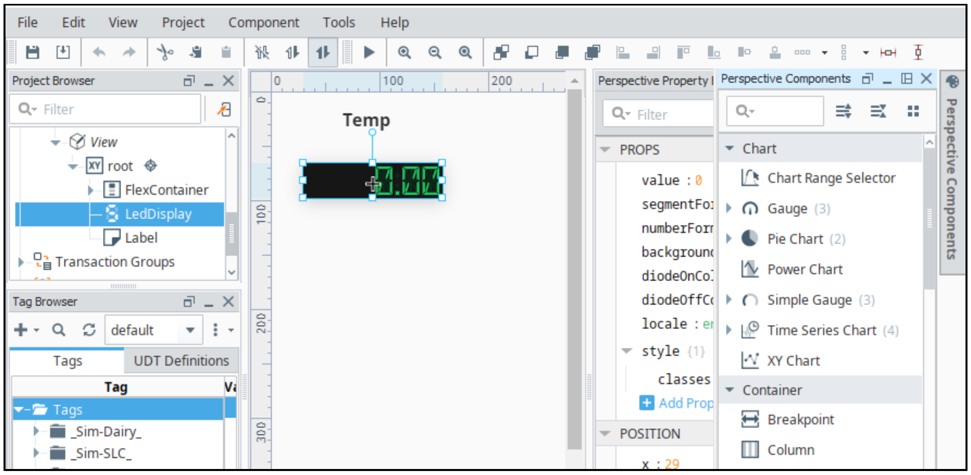
☑ Page URL

/line-a

Cancel

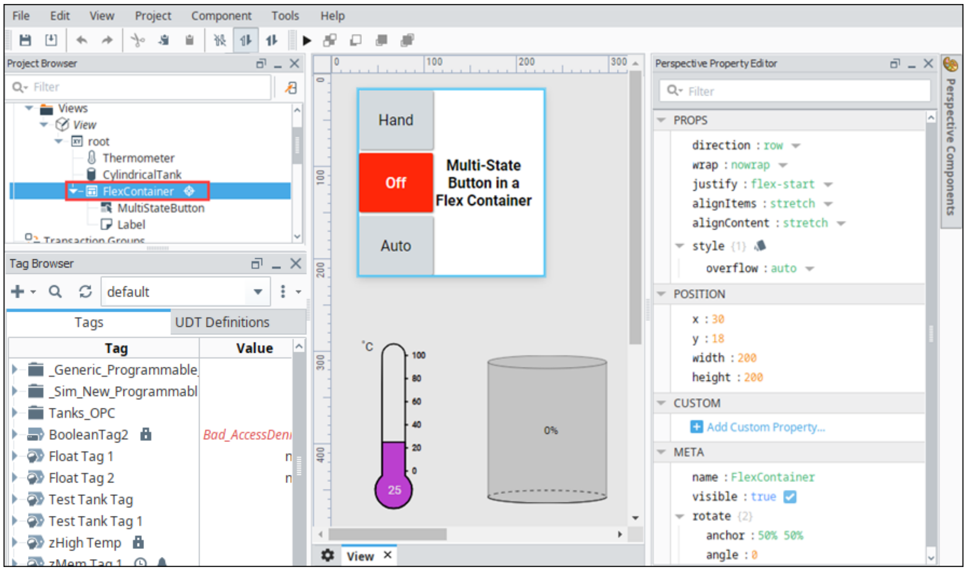
Create View

2. Once you create your view, the view will open in the Designer workspace, and you can begin adding and configuring components.



3. If you want to add additional containers, simply select another container type and drag it to your workspace. The container type added in this example is a Flex container.

> **Note:** To add components to this container, deep select it and drag components into the container. Notice that the properties are now in the Flex container.



Once you create your view, the next step is to set up your navigation using Pages.

Docked Views

Views can be anchored to an edge of a session. This configuration is called a docked view, as the view is "docked" to an edge in the session.

Docks are created in the Page Configuration interface.

On this page ...

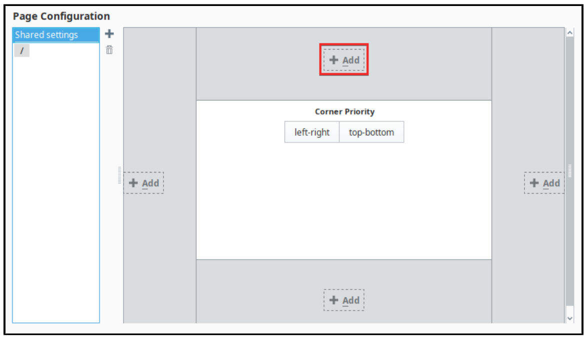
Docked Views in Perspective

Watch the Video

Configuring a Docked View

Docked views can be added to a page configuration in the following way:

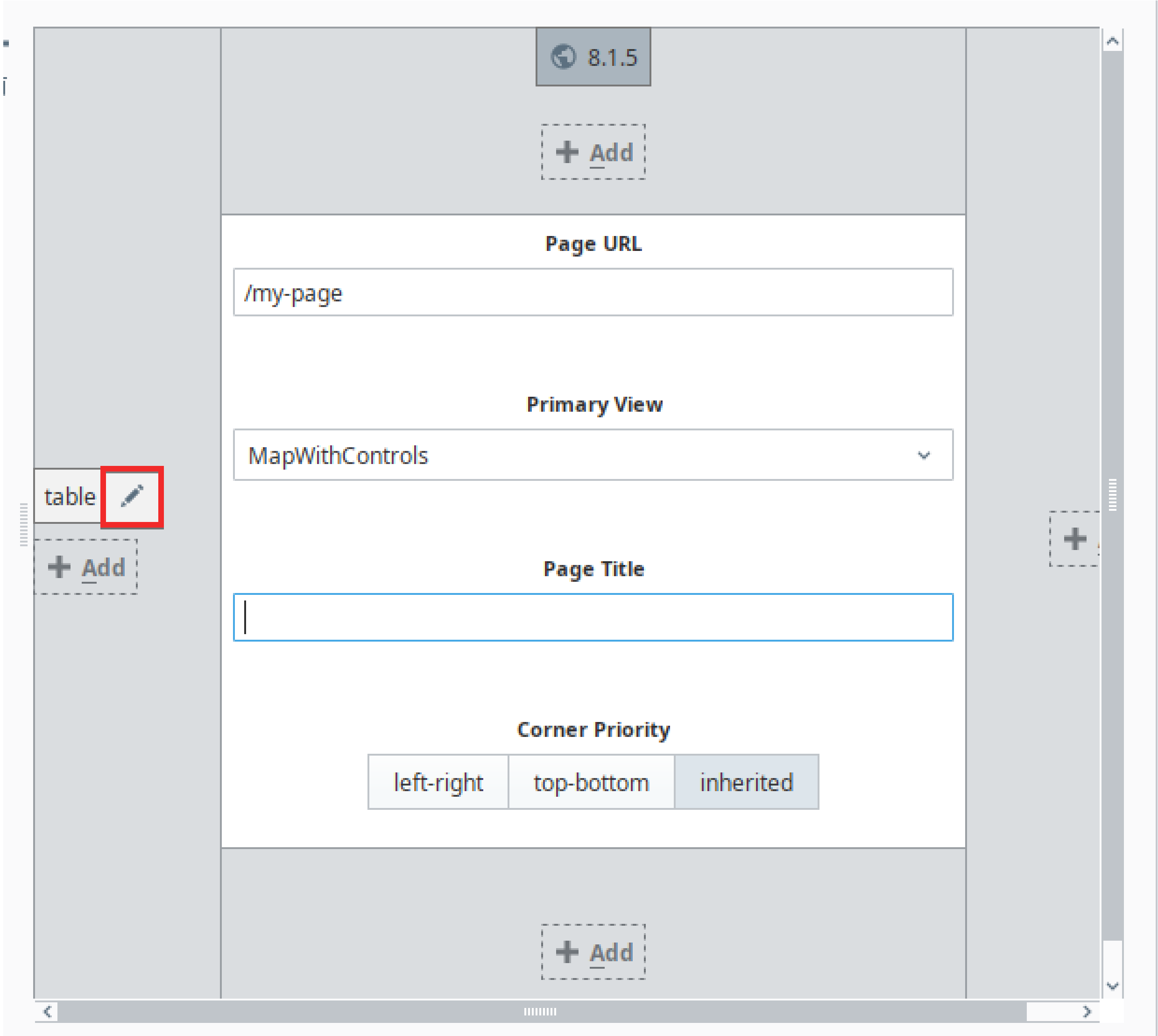
1. Select the configuration you wish to add the dock to
2. and then clicking the **Add** button on the desired edge. In this example, we'll add a north docked view by clicking at **Add** button at the top of the diagram.



3. You'll see a popup tree showing all views in the project. At this point you'll select the view that will be placed in the docked. Select the desired view and click **OK**. You'll find your new dock has been added.

Editing an Existing Docked View

Once a view has been docked, it can be further customized via the **Configure Docked View** popup by left-clicking on the docked view's **Edit** ✏ icon on the docked view configuration.



Page URL

/my-page

Primary View

MapWithControls

Page Title

Corner Priority

left-right | top-bottom | inherited

Docked View Properties

<table>
<tr><th>Property Name</th><th>Description</th></tr>
<tr><td>View</td><td>The currently selected view. Changing this will change which view is mounted to this position.</td></tr>
<tr><td>Display</td><td>This property allows you to show or hide the docked view. Options are:
<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>visible</td><td>The docked view is always expanded/displayed.</td></tr>
<tr><td>onDemand</td><td>The docked view is collapsed, but allows the user to display the view by clicking on the docked view's handle.</td></tr>
<tr><td>auto</td><td>Automatically shows or hides the docked view depending on how much space is available in the session: showing the view if the page is wider than the width specified in the auto-breakpoint setting. (Works in conjunction with the Auto Breakpoint property).</td></tr>
</table></td></tr>
<tr><td>Resizable?</td><td>Determines whether the docked view may be resized or not.</td></tr>
<tr><td>Modal?</td><td>Determines if the view should be modal, meaning users will not be able to directly interact with other views while the modal view is present. This property is only enabled when the <b>Display</b> property is set to <b>onDemand</b>.</td></tr>
<tr><td>Content</td><td>Determines how the docked view interacts with other views on the page.</td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>push</td><td>Opening or closing the docked view causes the content in the center to resize: the center view will be 'pushed' out of the way.</td></tr>
<tr><td>cover</td><td>When opening the dock, it slides in front of the center view, obscuring part of the center view: the dock will 'cover' part of the center view.</td></tr>
<tr><td>auto</td><td>Acts like the cover option when the viewport is smaller than the Auto Breakpoint value. Acts like the push option when the viewport is larger than the Auto Breakpoint value.</td></tr>
</table>
</td></tr>
<tr><td>Anchor</td><td>Allows you to make a view always visible while scrolling. Only available on North docked view configurations.
<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>fixed</td><td>The docked view will remain in a fixed position, relative to the page. Useful when a north-docked view should stay at the top of a page. Select this option if a docked view is acting as a header that should always be present.</td></tr>
<tr><td>scrollable</td><td>The docked view will not stay in a fixed position as the user scrolls down in the page. Select this option if the north dock should move along with the page as the user scroll down.</td></tr>
</table>
</td></tr>
<tr><td>Size</td><td>Determines the size, in pixels, of the view.
<ul>
<li>If the view is docked to the North or South edge, then size determines the height.</li>
<li>If the view is docked to the East or West edge, then size determines the width.</li>
</ul>
</td></tr>
<tr><td>Auto Breakpoint</td><td>Controls the minimum page width for Auto docked views to be visible. When the session is smaller than this width, these views will be hidden and able to be displayed on demand.

This property is enabled when the <b>Display</b> property is set to <b>auto</b>.</td></tr>
<tr><td>Dock ID</td><td>An optional arbitrary string that can be used to reference a docked view through other parts of Perspective such as in an action or as a scripting call.</td></tr>
<tr><td>Handle</td><td>Allows you to show or hide a handle for users to expand/collapse the view.
<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>Show</td><td>Show handle at all times.</td></tr>
<tr><td>Hide</td><td>Hide handle at all times.</td></tr>
<tr><td>AutoHide</td><td>Hide handle when page is not active.</td></tr>
</table>
</td></tr>
<tr><td>Handle Icon</td><td>Path to an icon used to identify the view when the view is hidden.</td></tr>
<tr><td>View Parameters</td><td>Allows specific parameter values to be passed to the docked view when navigating to the page.</td></tr>
<tr><td>Remove</td><td>Deletes the view from the page.</td></tr>
</table>

Embedded Views

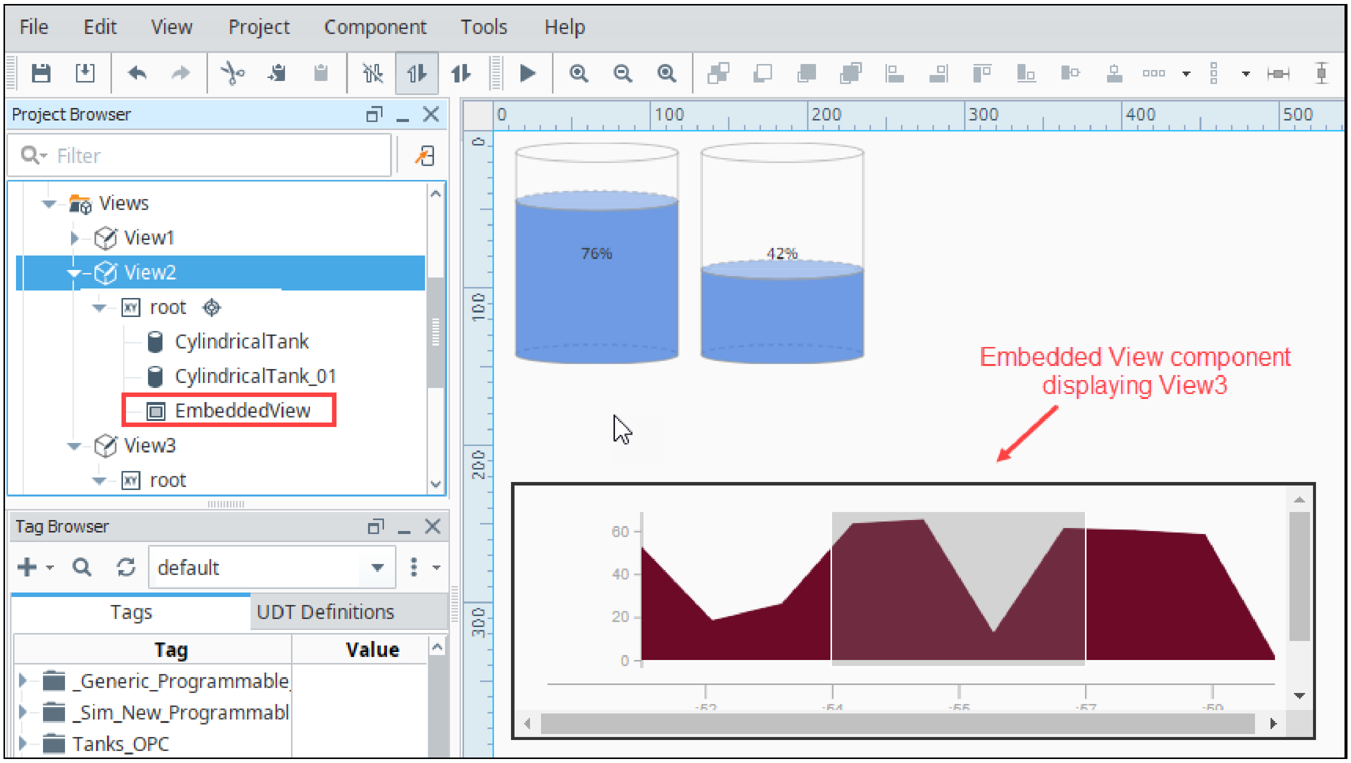
An embedded view is an instance of a view that is used as a component within another view. Similar to how containers can be added inside another container. An embedded view in Perspective is actually created using the Embedded View component, which internally opens the view. You'll notice when you nest a view inside another that the Embedded View component has some properties that are distinct from those of the view you're nesting:

- **path** is the path to the view you wish to embed.
- **params** is an object in which you'll put any parameters you wish to pass through to the embedded view. When passing parameters into the embedded view, the names must match the parameters on that view.
- **useDefaultViewWidth** and **useDefaultViewHeight** are two very critical properties when nesting a view. They control whether a view's default configured dimensions are carried into the embedded view (using scrollbars if the defaults are too large for the embedded setting), or whether the view is scaled from its defaults to *match* its new setting. Typically if you're finding unwanted scrollbars, unchecking these properties often achieves the desired result.

Embedded View

Watch the Video

When designing your project, views need not be specified as embedded. It is possible that a single view may end up being instantiated as a regular view and an embedded view, even simultaneously in the same session. For example, you may have a view that shows 10 tanks using a view repeater as well as a popup that shows the details for just tank_1. This is common when the properties of a view are used to pass in parameters.



Embedded View component displaying View3

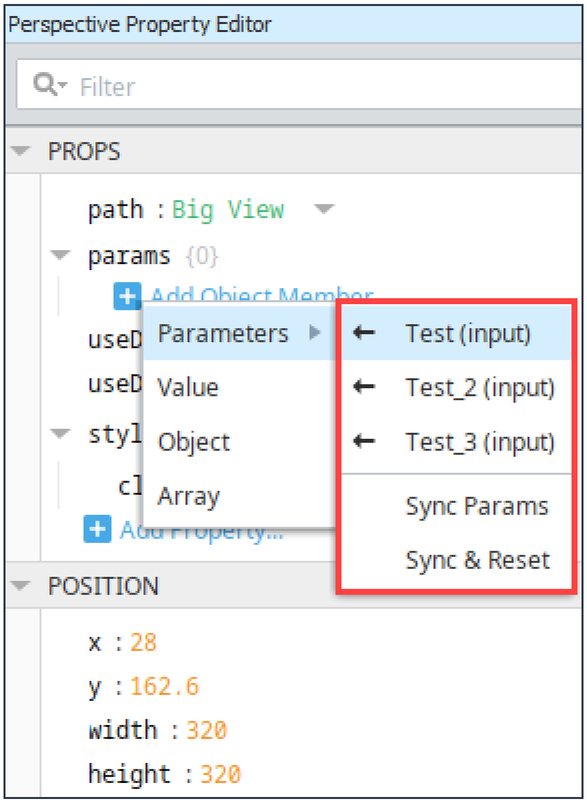
The following feature is new in Ignition version **8.1.4**
Click here to check out the other new features

As of release 8.1.4, the view parameters available from the embedded view, will appear on a dropdown list of parameters in the Property Editor. You can select any or all of the parameters to add. Two additional operations are available:

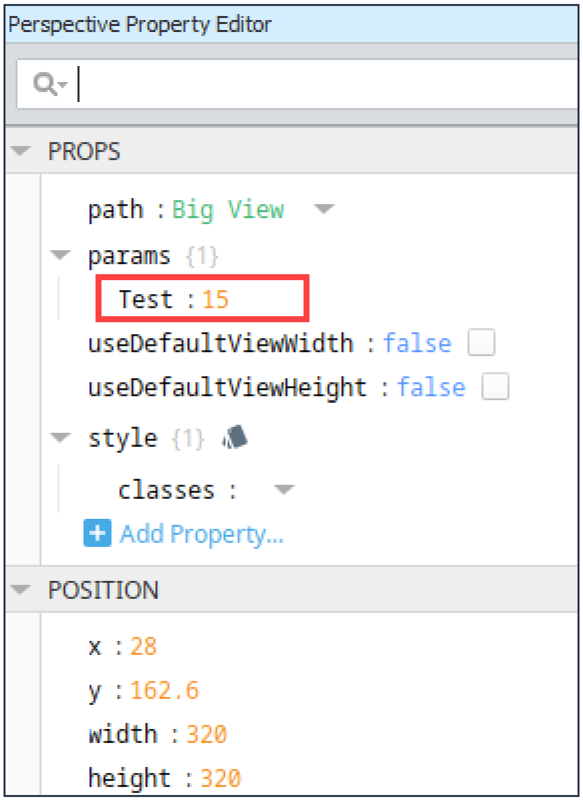
- **Sync Params** - Adds all the embedded view's parameters and removes any parameters that don't exist on the embedded view.
- **Sync & Reset** - Resets the parameter values to their default.

The following is an example of passing parameters from a view called Big View into an embedded view.

1. Click the **Add Object Member** ➕ icon under props.params. A dropdown list appears with all available parameters from the view that is embedded.



2. Click the **Left Arrow** ← icon next to the parameter you want to add.
3. The parameter now appears in the the params list.



Popup Views

A Popup View typically floats on top of the primary view in a Perspective Session, and it can be resized and moved around at the user's discretion. Popup Views are great for displaying additional information about an item on the primary view. Popup Views are often opened by components such as a Button on another view. When a user doesn't need to have the additional information displayed on the screen, it can be closed by clicking the Button again or simply closing the view.

On this page ...

INDUCTIVE UNIVERSITY

Popup Views

Watch the Video

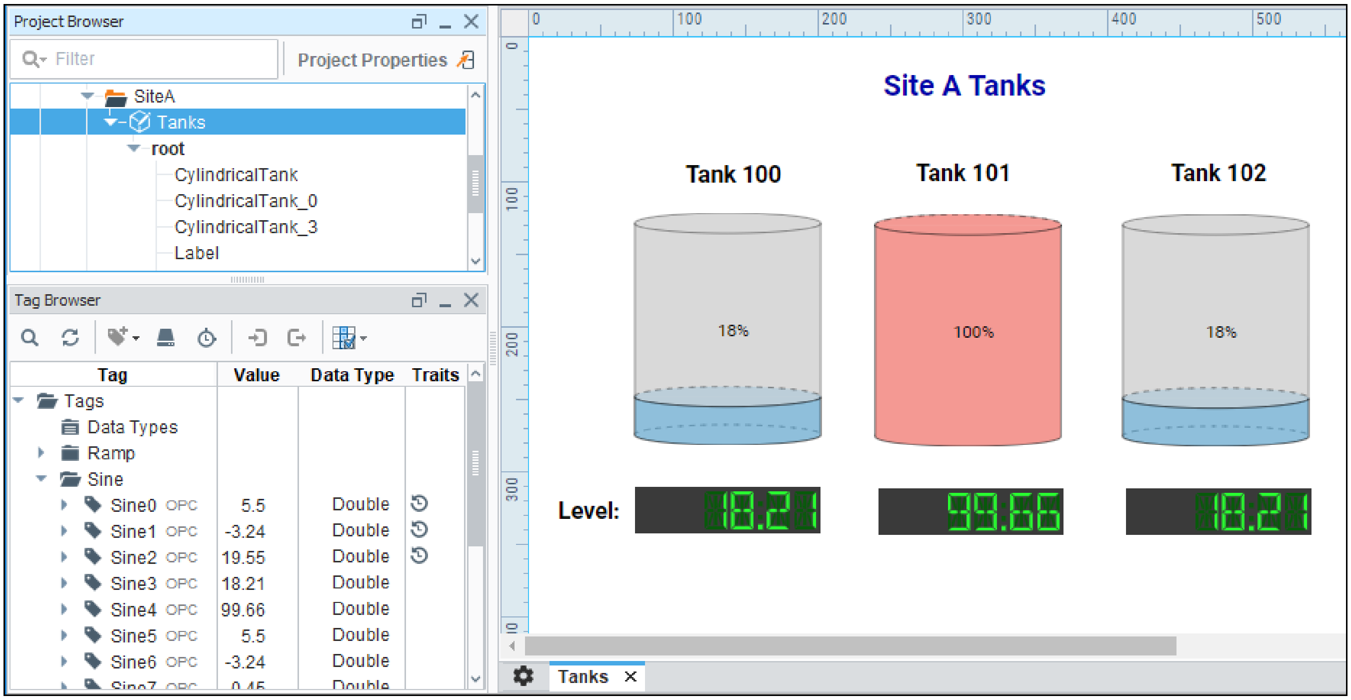
Title Bars on Popups

When called as a popup, a built-in title bar will be applied to the view if any of the following conditions have been met via the Popup Action configuration, or the corresponding parameters have been set on the appropriate system function, such as system.perspective.togglePopup:

- A non-empty string title is provided (the `title` parameter on scripting functions)
- The popup is marked as with a "close" icon, meaning the "show close Icon" setting is enabled (the `showCloseIcon` scripting parameter)
- The popup is marked as "Draggable", (the `draggable` scripting parameter).

Configuring a Popup View

Let's assume you already have a primary view that contains some Tanks. At a glance, an operator can view some basic information on all the Tanks at a particular site, but they cannot see the additional information that is being collected that is unique to each Tank. The perfect way to display that unique information is to use a Popup View for each Tank.

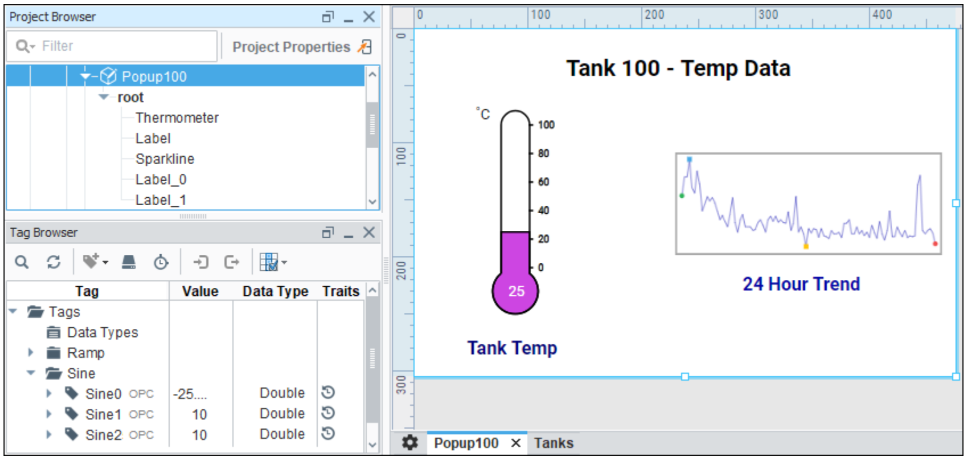


Tag	Value	Data Type	Traits
Tags			
Data Types			
Ramp			
Sine			
Sine0 OPC	5.5	Double	⟲
Sine1 OPC	-3.24	Double	⟲
Sine2 OPC	19.55	Double	⟲
Sine3 OPC	18.21	Double	
Sine4 OPC	99.66	Double	
Sine5 OPC	5.5	Double	
Sine6 OPC	-3.24	Double	
Sine7 OPC	0.45	Double	

Here is an example of how to use a Button component to set up a popup view for displaying the current tank temperature and history for the last 24 hours of the selected tank.

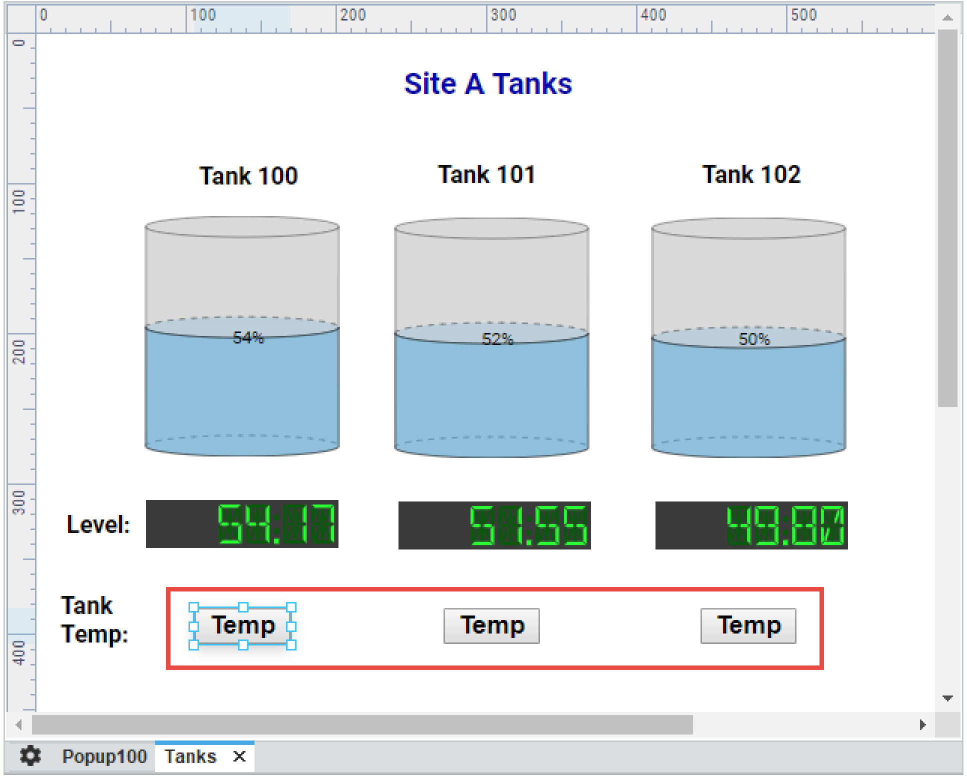
1. First create a new view, and give it a name. You many want to assign a name that you can easily recognize as a Popup View and the Tank ID number. This example uses the name Popup100.

2. Next add some components to the Popup View. In this example, a Thermometer, Sparkline, and Label components were added.

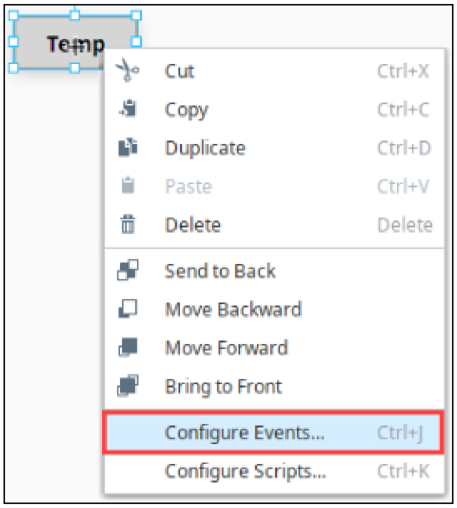


> **Note:** To display the additional data for all Tanks, you will need to create a Popup View for each Tank. You can use the Flex Repeater component to easily create multiple instances of components for display in another view each having the same look, feel, and functionality of the original components .

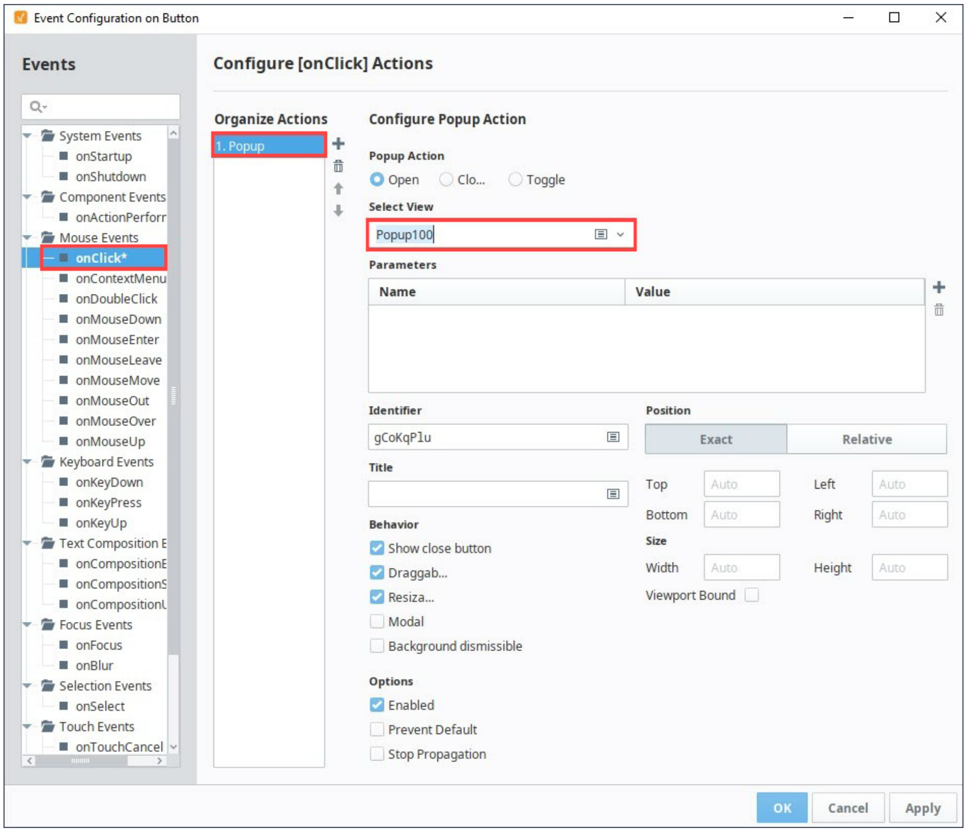
3. Now, let's go back to the primary view and add a Button component for each of the Tanks and label them **'Temp'**.



4. Righ- click on the Temp button for Tank 100, and select **Configure Events**.



5. The Event Configuration window opens. Under Events, select **onClick**.
6. Under Organize Actions, click the **Add** ✚ icon to choose the appropriate action for the Temp button. Since we are opening a Popup View, select **Popup** from the dropdown list.
7. You have several Popup Actions to choose from.
   - **Open** - Opens the Popup View.
   - **Close** - Closes the Popup View.
   - **Toggle** - Opens and closes the Popup View. Select **Toggle** so you can use the Temp button to both open and close the Popup View for the specified tank.
8. In the Configure Popup Action area under Select View, select your Popup View that you created in Step 1. This example uses Tank 100 and the Popup View is **'Popup100'**.
9. You also have some additional settings for customizing the behavior and appearance of your popup. Some of these options are set by default, and others you can customize. When you're finished, click **OK**.
10. Repeat Steps 4 through 9 for Tank 101 and Tank 102.



1. Popup

Popup Action: ● Open ○ Clo... ○ Toggle

Select View: Popup100

Identifier: gCoKqPlu

Title:

Position: Exact | Relative

Top: Auto Bottom: Auto Left: Auto Right: Auto

Size

Width: Auto Height: Auto

Viewport Bound □

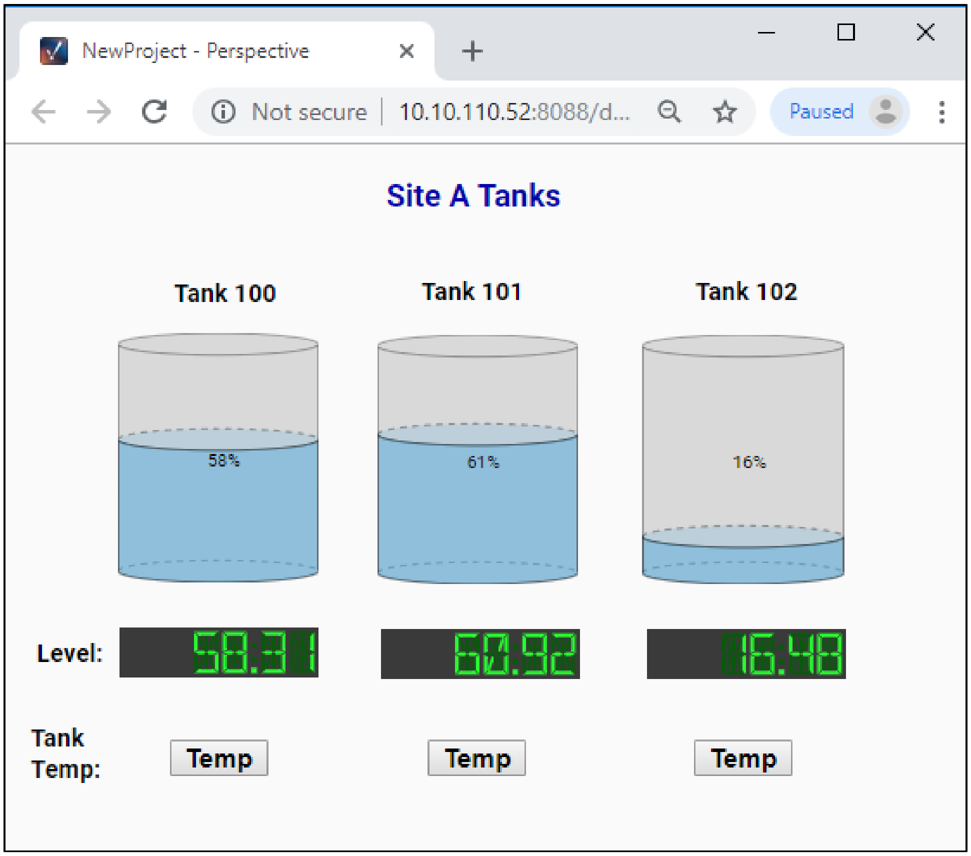
Behavior

☑ Show close button
☑ Draggab...
☑ Resiza...
□ Modal
□ Background dismissible

Options

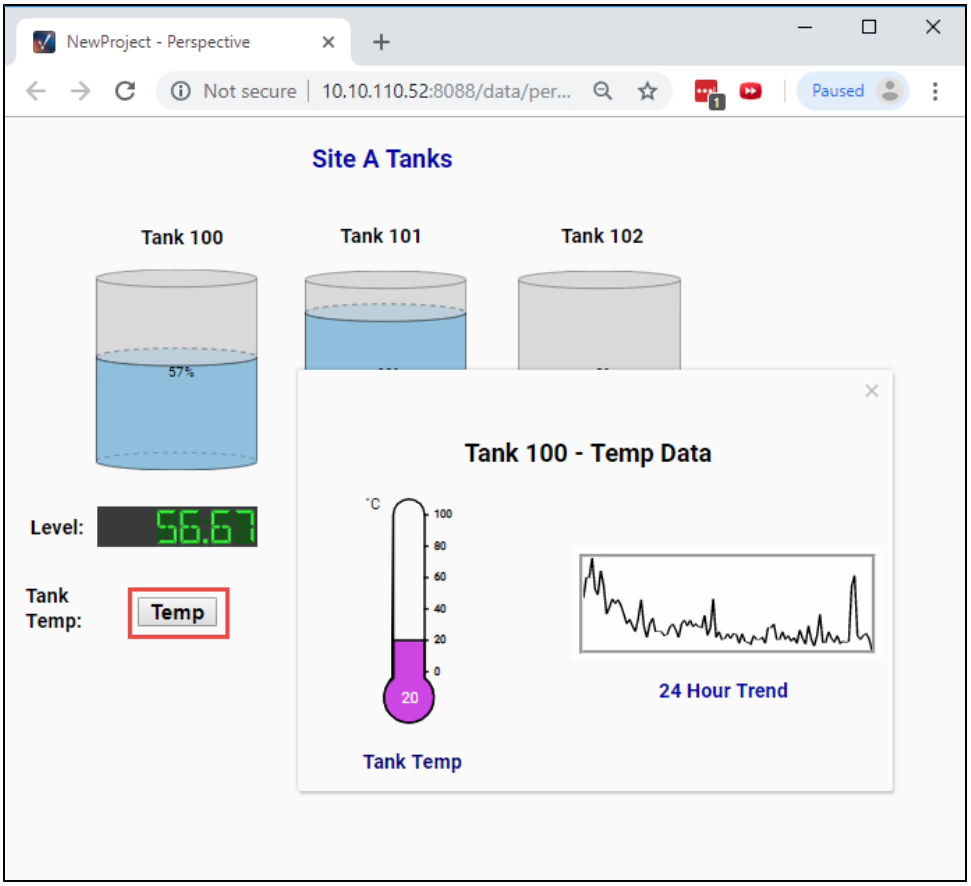
☑ Enabled
□ Prevent Default
□ Stop Propagation

11. **Save** your project.
12. Now let's see how your Popup View works in a Perspective Session. While in the Designer, go to the top menubar and click **Tools > Launch Perspective > Launch Session.**



13. Click on the **Temp** button for **Tank 100.** You'll notice Tank 100's Popup View will open displaying its Temp Data. You can drag the Popup View around the screen or even change its size.

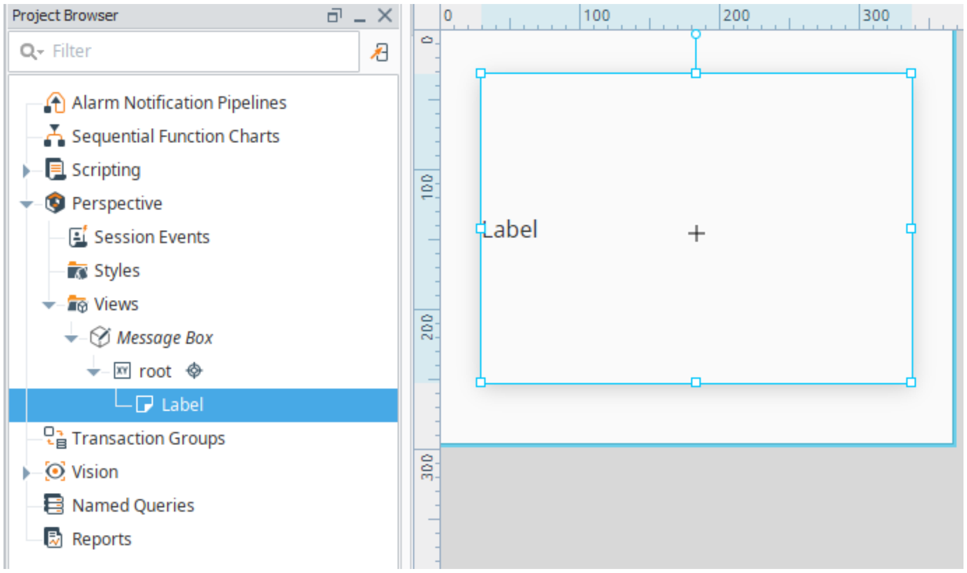
14. To close the Popup View, click either the Temp button, or click the **Close** ✕ icon in the upper right corner of the Popup view.



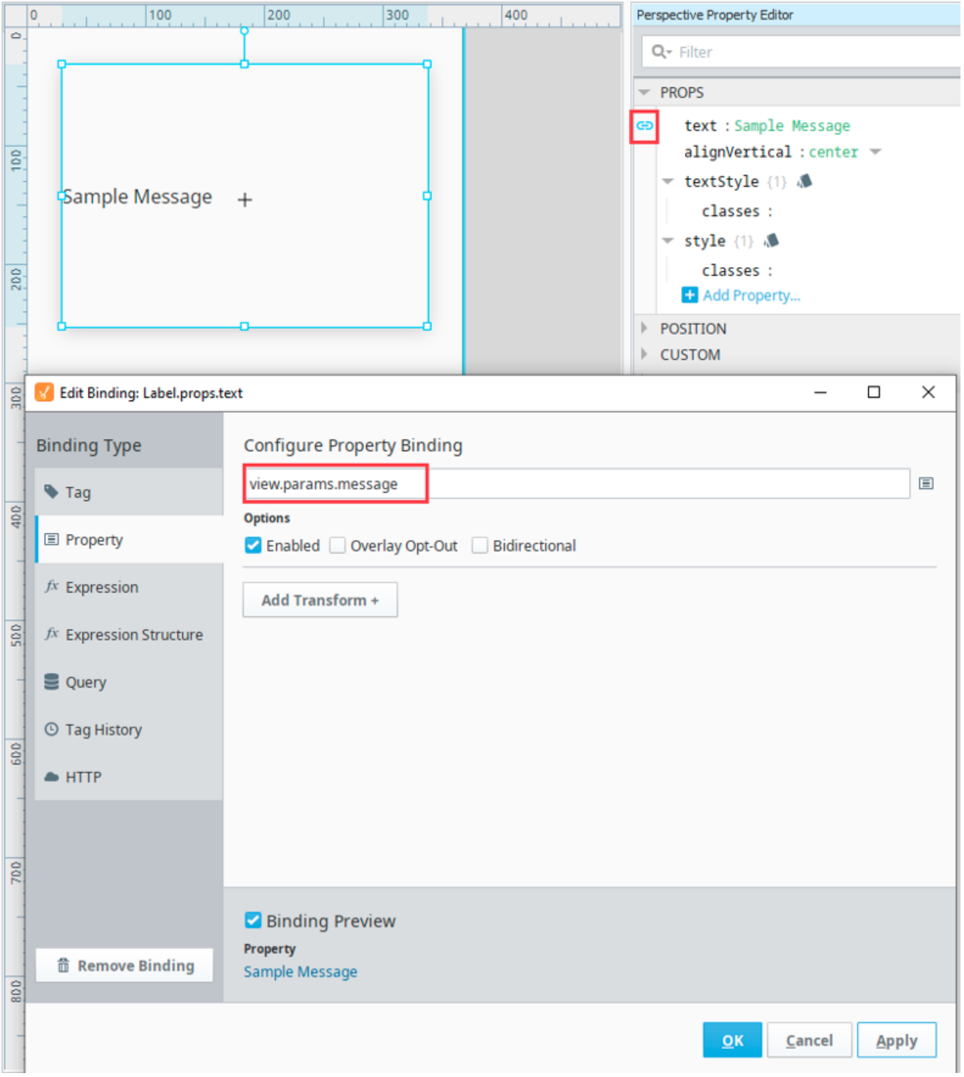
Configuring a Parameterized Popup View

It is often useful to pass information to a popup view as it is being opened. The popup can use this information in its design, usually to make it dynamic in some way. For this example, we will build a dynamic message box. We will have a message box view which will accept a parameter value, the message to display.

1. Create a new popup view, and name it Message Box. We can use the Coordinate root container type for this example. No page URL is required.
2. Add a Label component to this View.



3. Select the Message Box view from your Project Browser, and from the Property Editor, define a new **value** parameter named "message" with a default value of "Sample Message."
4. Bind the Label component to our newly created view parameter.



text : Sample Message
alignVertical : center

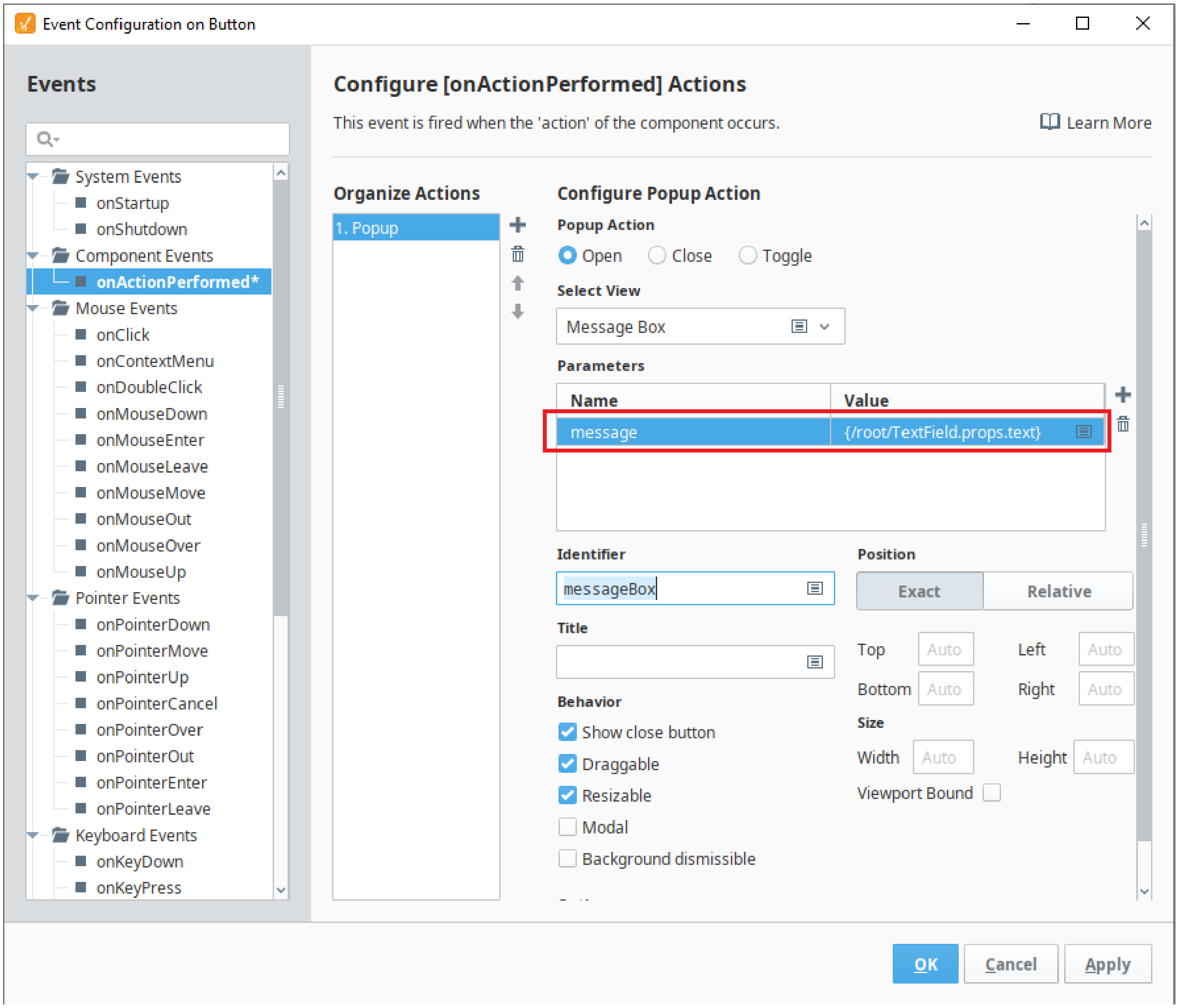
view.params.message

Options
☑ Enabled □ Overlay Opt-Out □ Bidirectional

☑ Binding Preview
Property
Sample Message

Our popup is now ready to be used.

Next we have to trigger this popup to open, passing the appropriate parameter value. You can use a Popup Action from any Component Event like in the previous example. The only difference will be that now you are passing in a value for the Message Box view's **message** parameter. Pressing the **+** button on the upper right-hand corner of the Parameters table will allow you to pass in a value for the Message Box's **message** parameter. In this example, we are passing the popup text property value of a simple Text Field component.



Configure [onActionPerformed] Actions

This event is fired when the 'action' of the component occurs.

Configure Popup Action

Popup Action: ● Open ○ Close ○ Toggle

Select View: Message Box

Parameters

Name	Value
message	{/root/TextField.props.text}

Identifier: messageBox

Title:

Behavior

☑ Show close button
☑ Draggable
☑ Resizable
□ Modal
□ Background dismissible

Position: Exact | Relative

Top: Auto Left: Auto
Bottom: Auto Right: Auto

Size

Width: Auto Height: Auto

Viewport Bound □

Note
The "messageBox" identifier is the string that specifies a unique popup identity. If you want to close this Message Box popup from a popup action, you'll need to supply the identifier that was used to open it.

It is also possible to trigger the opening of the popup using scripting through the use of the system.perspective.openPopup()

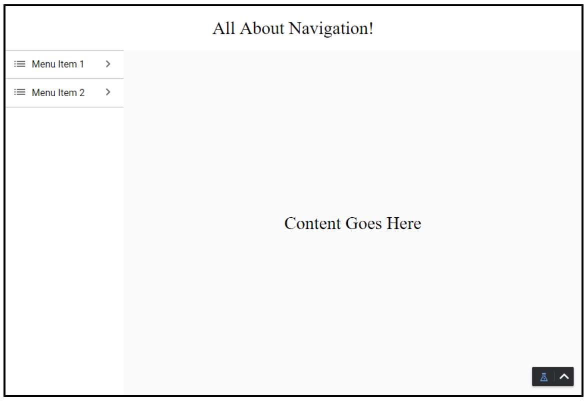
```
system.perspective.openPopup("messageBox",'Popups/MessageBox', params = {'message':self.getSibling
("TextField").props.text})
```

Self-Hiding Navigation Drawer

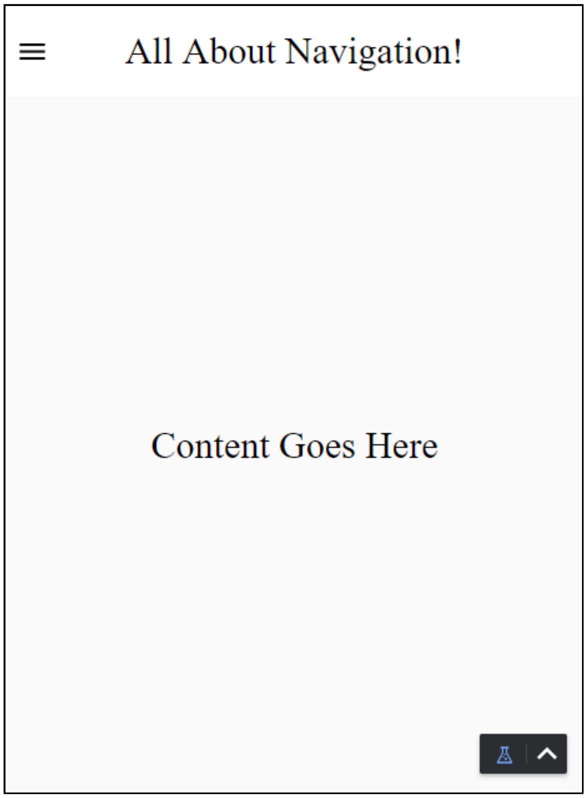
Navigation Drawer

A navigation drawer is a special type of docked menu, usually appearing on the left side of a session. What makes a navigation drawer special is its responsive design. On smaller devices, this docked menu can hide itself and pop out when the user needs it. These drawers have become ubiquitous in User Interface (UI) design, particularly in apps.

Here's what one might look like on a computer monitor:



Here's what one might look like on a mobile device:



As the screen becomes smaller, the menu is hidden and an icon appears in the top left to allow us to toggle its visibility. This particular navigation drawer will probably need about 200 pixels horizontally, which on a desktop is fine, but on a mobile device takes up too much of the screen.

Note: This guide assumes a bit of knowledge about how views and components work. Please see those sections of this manual for more information as needed.

On this page ...

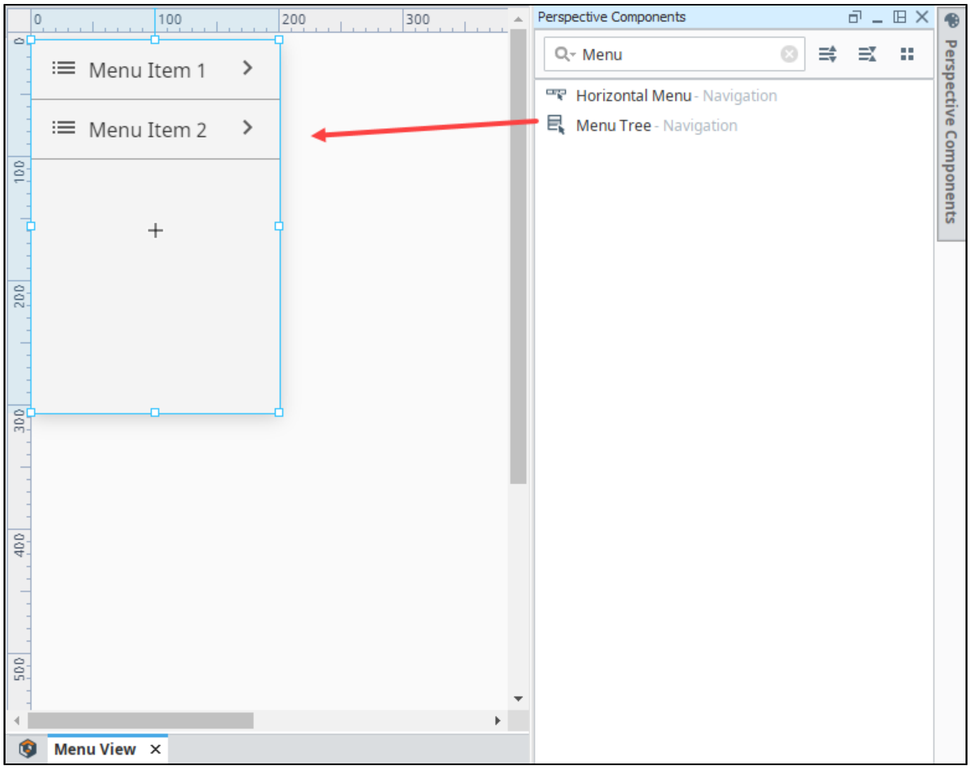
Configuring a Navigation Drawer

The following example walks through how to configure this self-hiding navigation drawer. There is no mention of setting up the Menu Tree or any content pages, it is strictly a guide to show you the layout type.

Navigation View

1. In the Project Browser, right click on **Views** and click **New View**.
2. Name the view **Menu View** and set the Root Container Type to **Coordinate**.
3. Click **Create** View**.**
4. Set the width of the view to **200 pixels**.
5. Drag a MenuTree component onto the Menu View. Configure the component as you would if you were using a standard docked view.

> **Note:** You can set the root of the Menu View to use the Percent Mode. This way it is easy to make the Menu Tree fill all the space.

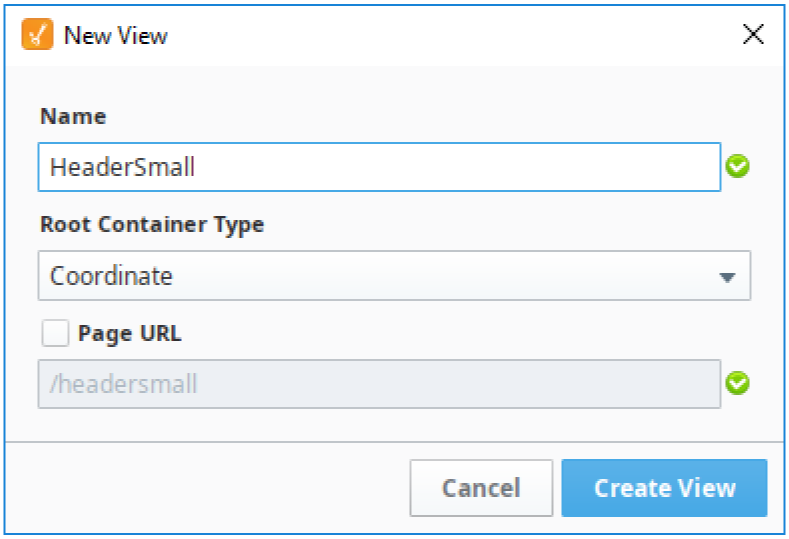


Header Views

Next, we will create our header views. The two views will be set up in a **breakpoint** container, which will swap between a small header with an icon, and a big header without one.

1. Create a small header view for our mobile UI.
    a. In the Project Browser, right click on **Views** and click **New View**.
    b. Name the new view **HeaderSmall**, and set the Root container Type to **Coordinate**.

c. Click **Create View.**



New View

Name

HeaderSmall

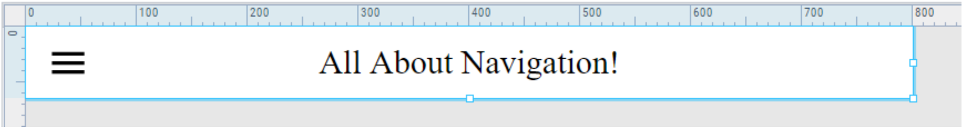
Root Container Type

Coordinate

□ Page URL

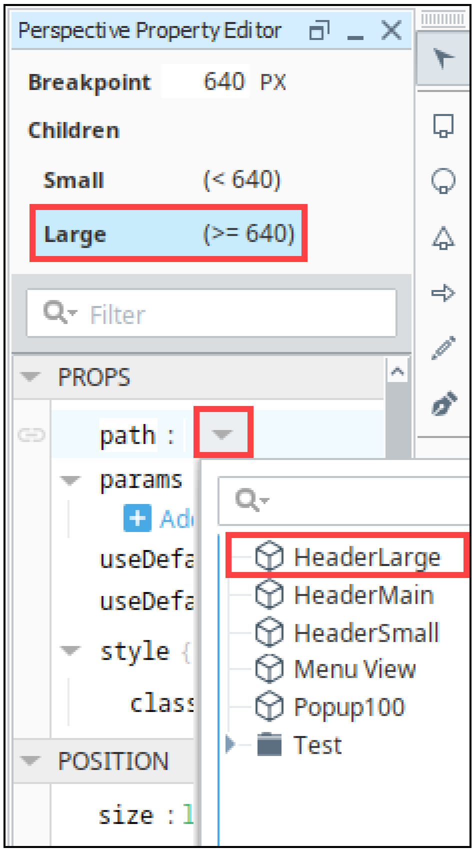
/headersmall

d. Set the height of the view to **75** pixels.
e. Drag an Icon component onto the upper left side of the view, then click on the component to select it.
f. In the Property Editor, set the path property to **material/menu**. You can of course use whatever icon you'd like; a list of all the icons in the material folder can be found here.
g. Add a title for the header. We used a Label component with the text "All About Navigation!"



2. Create a large header view for our desktop UI
   a. Create another view called **HeaderLarge**, again with a layout of **Coordinate**.
   b. Set the height to **75 pixels**.
   c. Don't add the icon to this one, but add the same Label as above.

3. Finally, create a Breakpoint view to toggle between them.
   a. Create a view called **HeaderMain** with a layout of **Breakpoint**.
   b. Set the height to **75 pixels**.
   c. In the Property Editor, click on **Large** (under Children).
   d. Drag an Embedded View component on the HeaderMaster view.
   e. Under PROPS, click on the **Expand** ▼ icon next to the path property then select **HeaderLarge**.

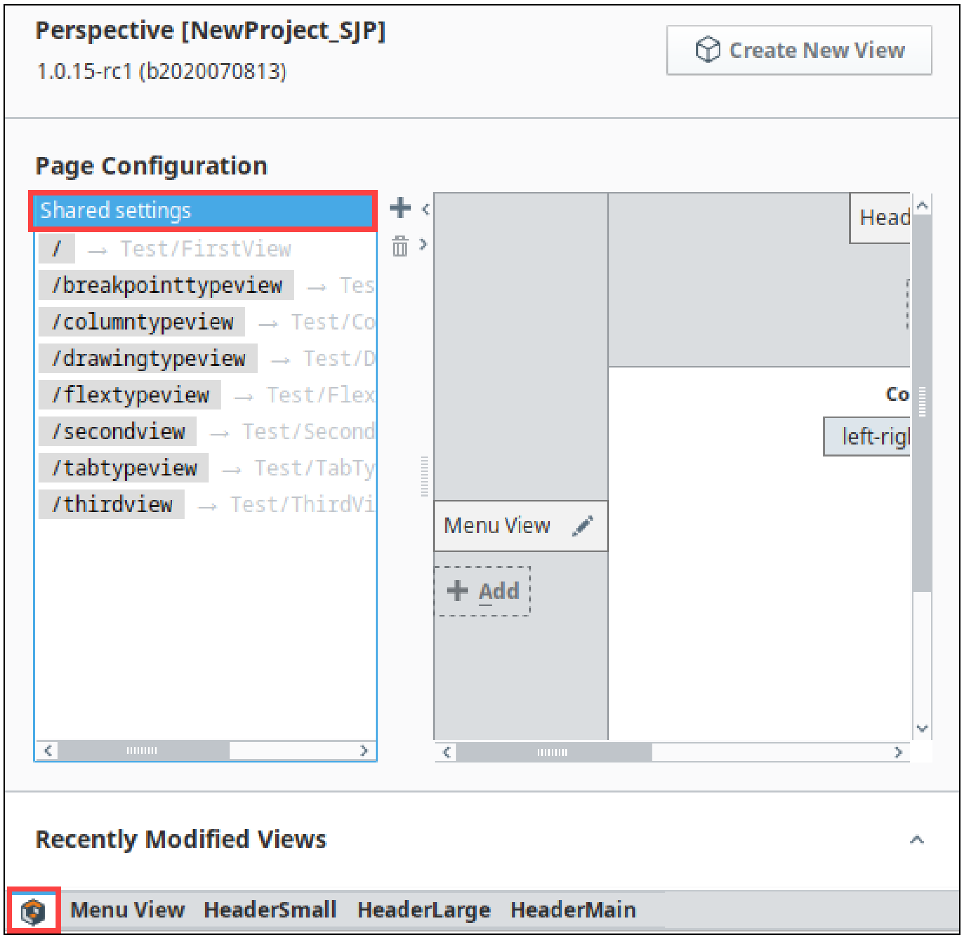


f. Now we'll do the same for the small child of this Breakpoint container. In the Property Editor, click on **Small** (under Children).
g. Drag an Embedded View component on the HeaderMain view.
h. In the Property Editor, click on the **Expand** ▼ icon next to the **path** property then select **HeaderSmall**.

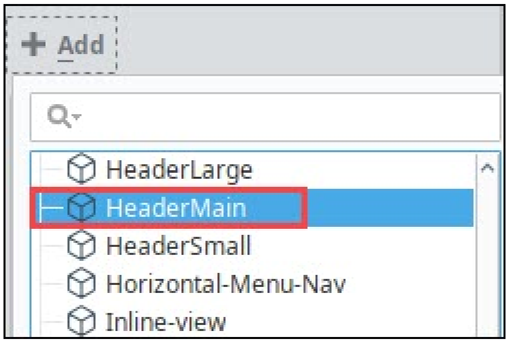
Page Setup

Now that we have our header and menu views, we need to set up our pages to display the views properly.

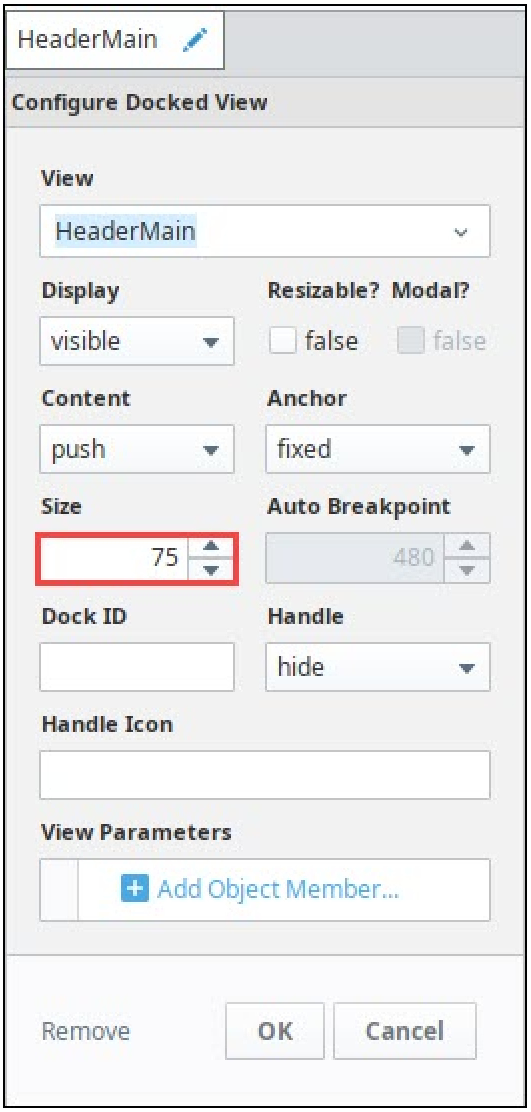
1. Click on the **Perspective** icon in the bottom left of the Designer to access the Page Configuration menu. Select **Shared settings**. We're going to add our two docked views here, so they show up on every page.


Perspective [NewProject_SJP]
1.0.15-rc1 (b2020070813)
Create New View
Page Configuration
Shared settings
/ → Test/FirstView
/breakpointtypeview → Tes
/columntypeview → Test/Co
/drawingtypeview → Test/D
/flextypeview → Test/Flex
/secondview → Test/Second
/tabtypeview → Test/TabTy
/thirdview → Test/ThirdVi
Head
Co
left-rigl
Menu View
Add
Recently Modified Views
Menu View
HeaderSmall
HeaderLarge
HeaderMain

2. On the top dock, click the **Add** ✚ icon. Select the **HeaderMain** view and click **OK**.



3. Click the **Edit** ✏ icon next to the HeaderMain docked view.
4. In the Configure Docked View menu, set the size to **75**.



HeaderMain

Configure Docked View

View: HeaderMain

Display: visible

Resizable? □ false

Modal? □ false

Content: push

Anchor: fixed

Size: 75

Auto Breakpoint: 480

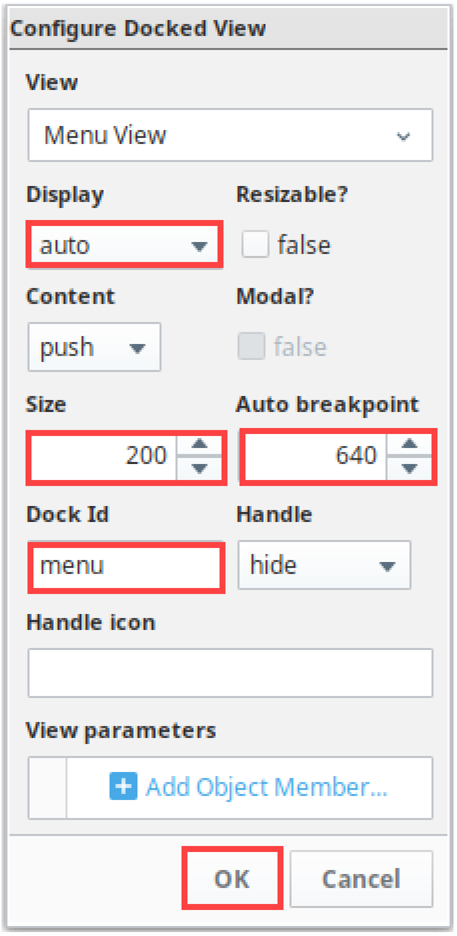
Dock ID:

Handle: hide

Handle Icon:

View Parameters

5. On the left dock, click the **Add** + icon. Add the **Menu View** and set the following:
   - Size: **200**
   - Display: **Auto** (This enables us to configure a breakpoint below)
   - Autobreakpoint: **640** (This is the same width that the breakpoint container on the HeaderMaster view is using)
   - Dock Id: **menu** (This will be used in our dock action on the menu icon to toggle the menu.)



Configure Docked View

View: Menu View

Display: auto

Resizable? □ false

Content: push

Modal? □ false

Size: 200

Auto breakpoint: 640

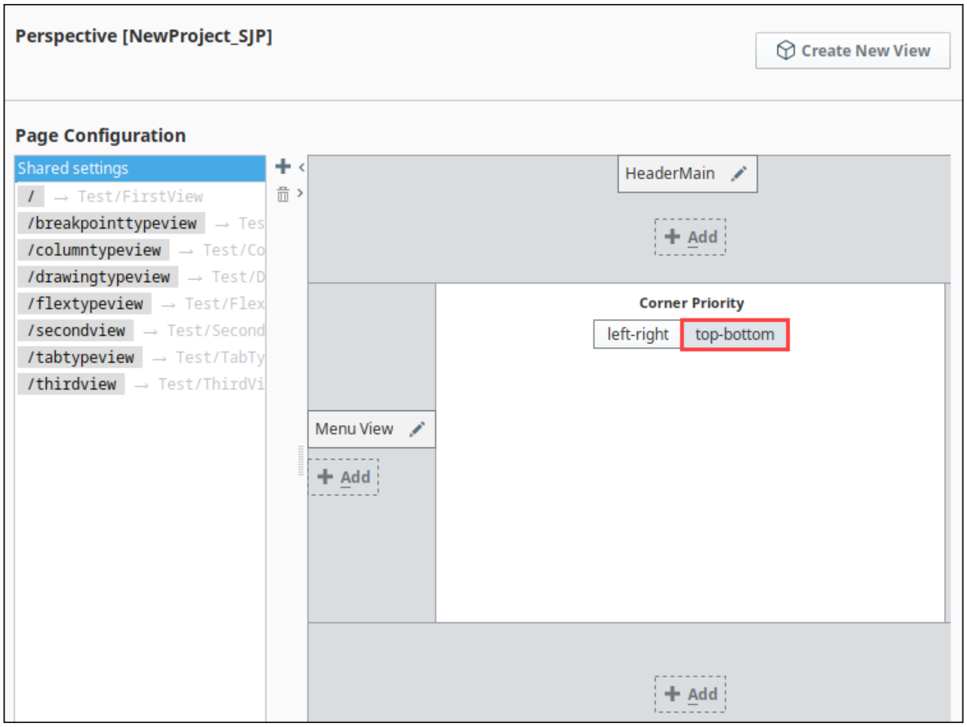
Dock Id: menu

Handle: hide

Handle icon:

View parameters

6. In the center of the Page Configuration menu, set the Corner Priority to **top-bottom**. The other option won't look quite right.



Perspective [NewProject_SJP]

Page Configuration

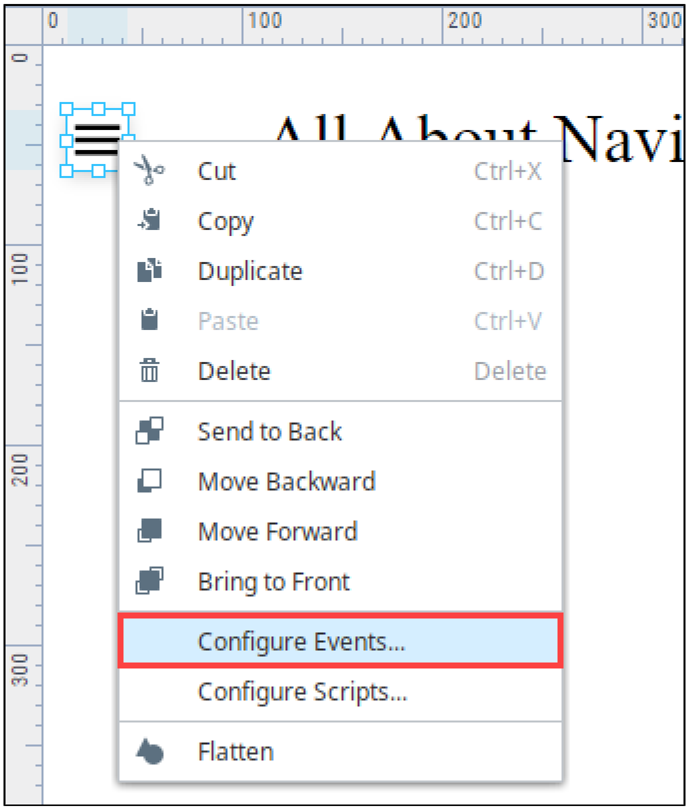
Shared settings
/ → Test/FirstView
/breakpointtypeview → Tes
/columntypeview → Test/Co
/drawingtypeview → Test/D
/flextypeview → Test/Flex
/secondview → Test/Second
/tabtypeview → Test/TabTy
/thirdview → Test/ThirdVi

Corner Priority: left-right [✓ top-bottom]

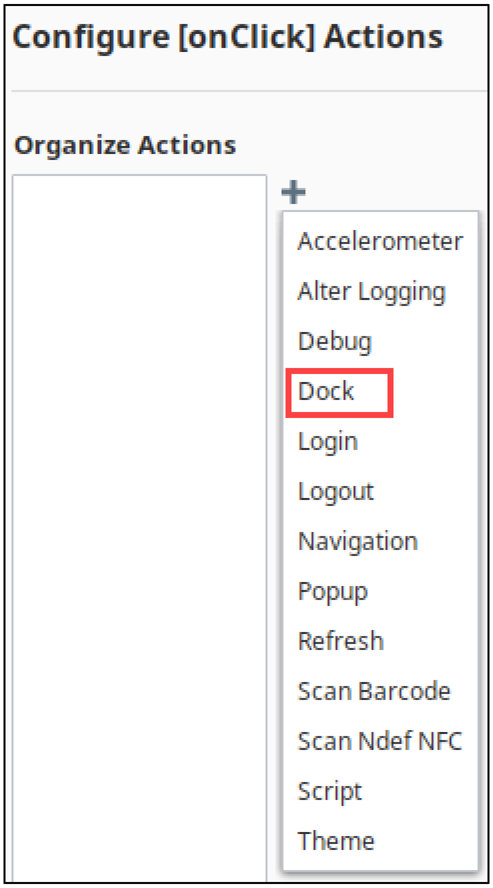
Configure Menu Icon

Now we need to configure the menu icon we created on HeaderSmall to pull up **Menu View.**

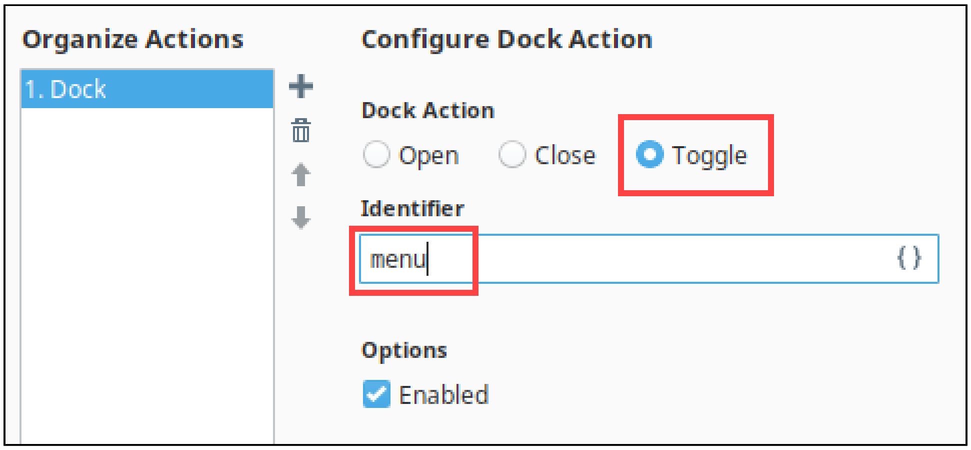
1. Open the HeaderSmall View.
2. Right-click on the Icon component, and select **Configure Events...**



3. Under Mouse Events, select **onClick.**
4. Next click the **Add** ✚ icon to add an action. Select the **Dock** action.



5. This action needs the identifier we created. Set the Dock Action to **Toggle**, and the Identifier to **menu**. Click **OK**.



Configure Dock Action

Dock Action

○ Open ○ Close ● Toggle

Identifier

menu

Options

☑ Enabled

Now go test it out! It's easiest to open a browser on your desktop and change the size to toggle between the different views.

Using this strategy, you can configure a navigation drawer in combination with any basic navigation component or method.

Working with Perspective Components

Components give you flexibility in designing HMI and SCADA that reflect your company's design and your site's layout. Components represent widgets you deal with every day: buttons, text areas, dropdowns, charts, gauges, linear displays, and so on. The Perspective Module comes with a host of built-in components that you can select from for use in your project. There are many ways to manipulate and arrange components when working in the Designer.

This section introduces you to how to work with components so you can learn how to quickly select, move, resize, duplicate, and customize components during the design process. Properties and specifics for individual components are covered in the Appendix.

Component Categories

Components are separated into the following categories in Perspective:

- Chart - Charts allow you to display and show off your data in a graphical way.
- Container - Containers provide a way of laying out and organizing components within a view.
- Display - Display components display static and dynamic information.
- Embedding - Embedding components can be embedded in multiple views of a project.
- Input - Input components enable users to enter data, or select data, and even control a device.
- Navigation - Navigation components provide you with design strategy options to navigate within a Perspective session.

Component Properties

Properties on Perspective components are separated into categories.

- **Props** - Properties that control the configuration and provide the runtime data for the component. See individual Perspective components for a list of the properties and their descriptions.
- **Position** - Properties defined by the component's parent container which control the location of the component. The available properties listed under this category depend entirely the container type that the component is placed in.
- **Custom** - Custom properties defined by the designer for each component instance.
- **Meta** - Properties defined by the Perspective Module itself for common things like the component's name. See Meta Properties on Perspective Components.

On this page ...

INDUCTIVE UNIVERSIT

Component Overview

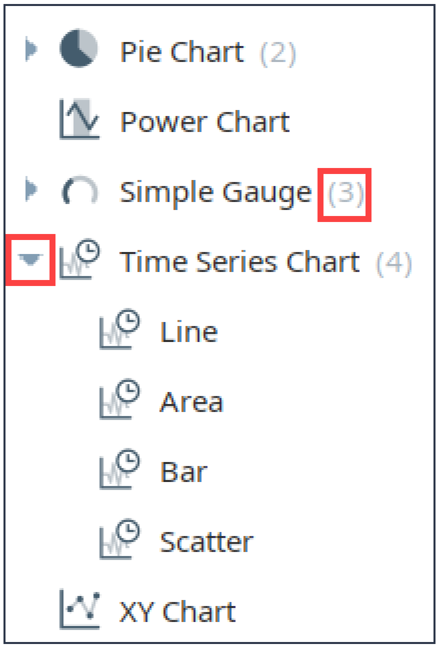
Watch the Video

The following feature is new in Ignition version **8.1.2**
Click here to check out the other new features

Variants

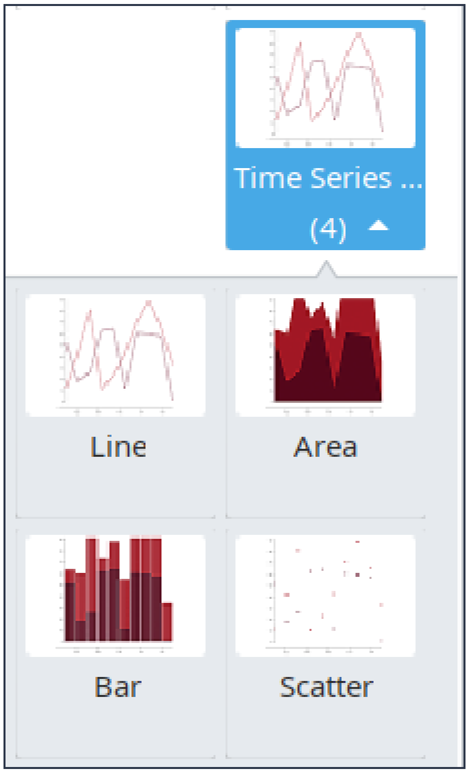
Some Perspective components have multiple pre-configurations, called variants, based on options a user might commonly use. For example, the Time Series Chart components has variants that set the chart up as a Line chart, Area chart, Bar chart, or Scatter chart. Using a variant can save you time. Variants have a group of properties pre-set to affect the appearance and functionality of the component.

When a component has variants, a number appears next to its name or thumbnail image. Click the arrow to the left of the component to view a dropdown menu of variants. Refer to Perspective Components for information about individual components.



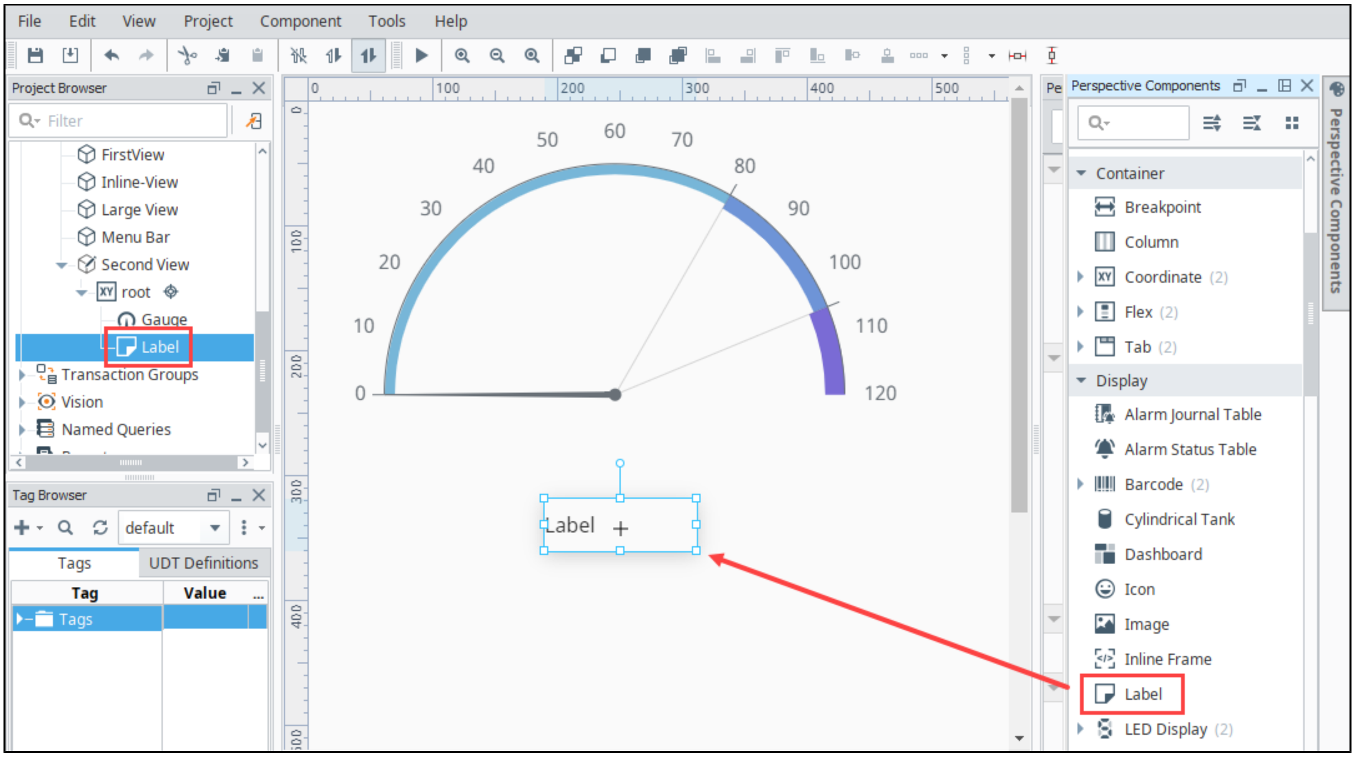
List View vs Tile View

As of release 8.1.2 there are two options for viewing Perspective Components in the Designer. Click on the **List View** ≡ icon to see components listed just by category and name. Click on the **Tile View** ⠿ icon to see the components listed by category and thumbnail icon. The following image shows the Tile View of the variants for the Time Series Chart component.



Adding Components

To add a component, open the Perspective Components palette. Click on the component you want (or click on a variant of the component) and drag it onto your view. In this example, we put a Label component under the Gauge component.

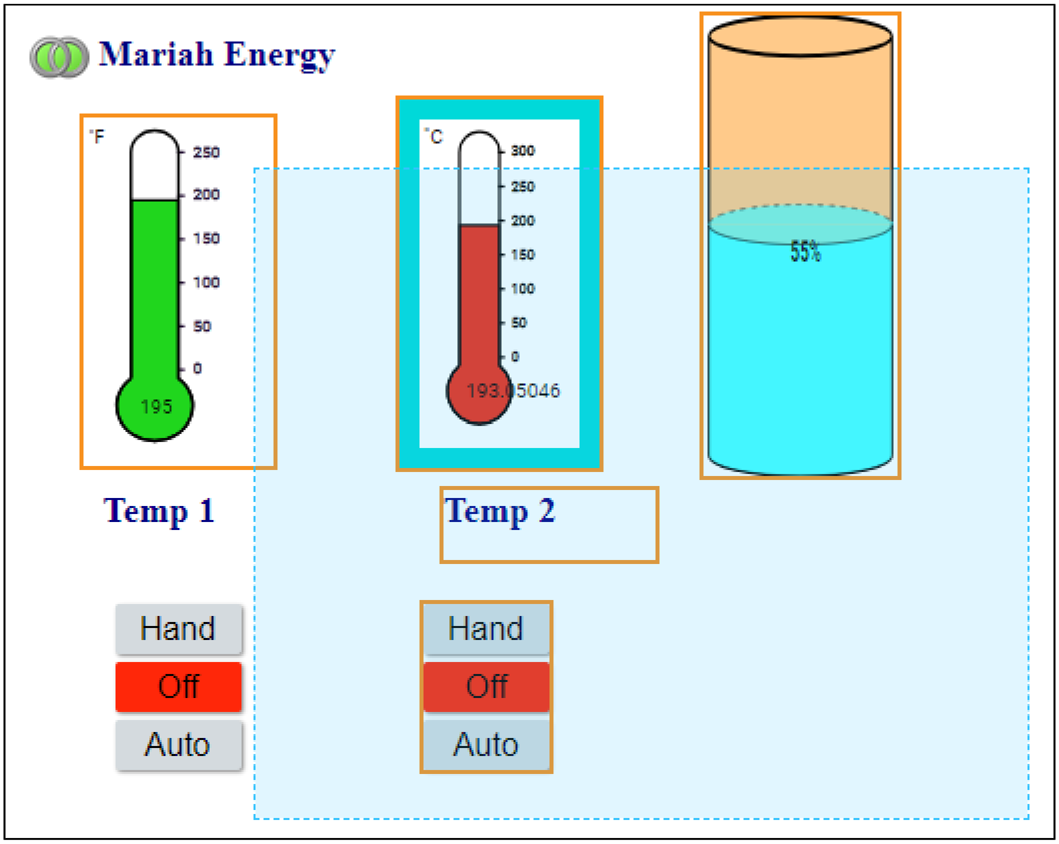


Selecting Components

Mouse Selection

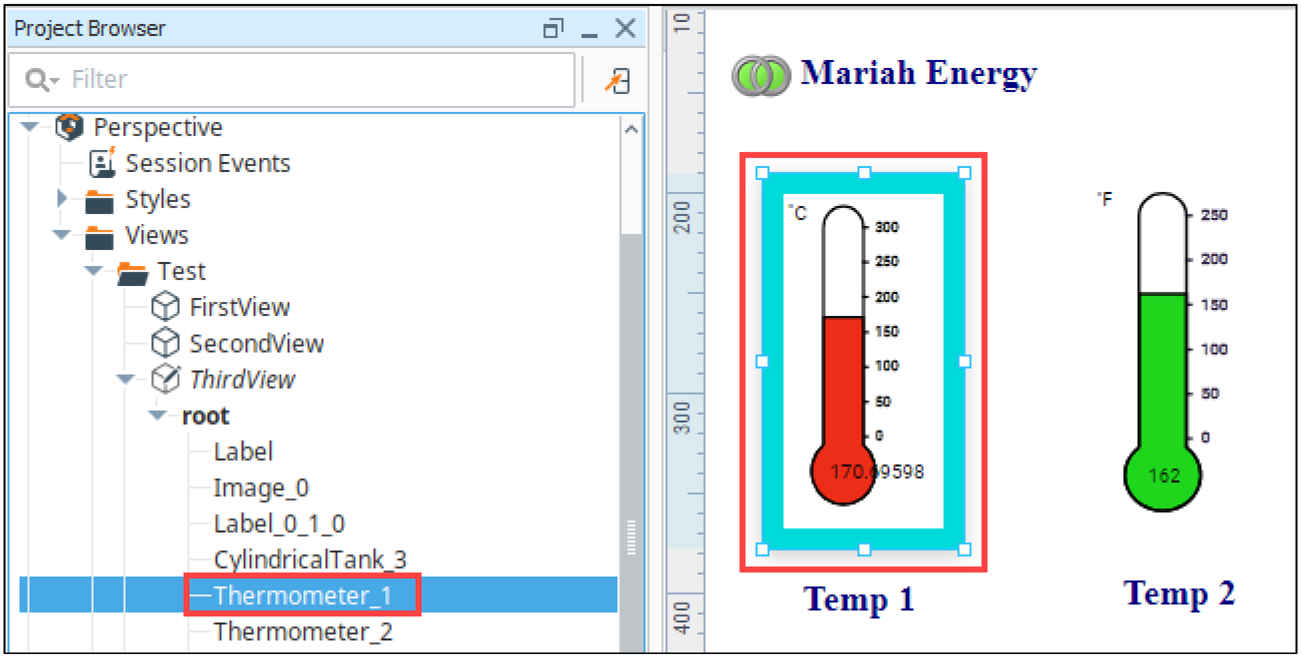
Using the mouse is the most common way to select components, where a single click on a component selects it and double clicking Deep Selects it.

You can also select components by clicking and dragging the mouse 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 view, the root for the container, and any components on the view. This is also the only way to select components that are invisible.



Deep Selection

Perspective makes great use of its many containers to help create a layout and design to fit any scenario on any device. To accommodate this, components sometimes need to be nested within a series of containers, which are themselves nested in other types of containers. With this comes a true tree of components, and selecting components nested inside containers works differently than components on the root. Deep Selection allows you to select into a container and select a component within.

To deep select a component, **double-click** on it. Three things happen to indicate you have deeply selected into the component and not just selected the component itself:

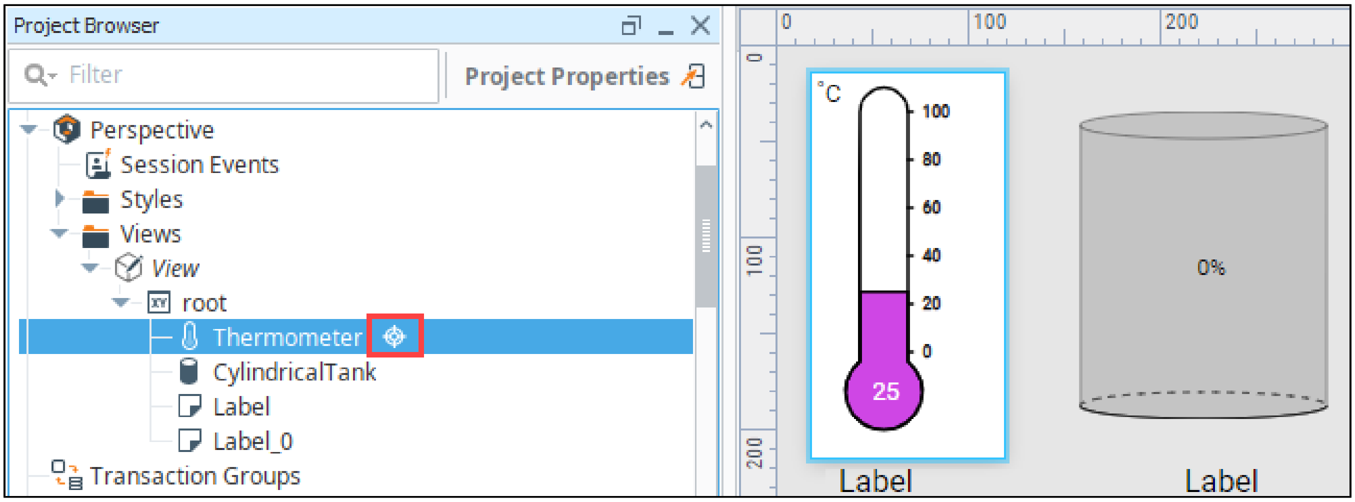
- The border of the component changes to a thick solid line that can't be manipulated.
- The surrounding area darkens.
- In the Project Browser, a **Deep Selection** ⌖ icon appears next to the component.

Once inside this new deep selection space, you can then select a component much like before, either by clicking or dragging to select components, or by deep selecting into another container inside the original container.

The following feature is new in Ignition version **8.1.26**
Click here to check out the other new features

In addition to double-clicking a component, you can use a single-click while holding the Alt and Shift keys to enable deep selection. For the Button, Label, Checkbox, and Toggle Switch components deep selection also enables inline text editing.

To cancel the deep selection, click anywhere outside of the component.



Select Through

When working with nested containers, **Alt + left clicking** on a component in a container will select the clicked component, and automatically deeply select the component's parent container. This can save time compared to first deeply selecting the container, and then selecting the component.

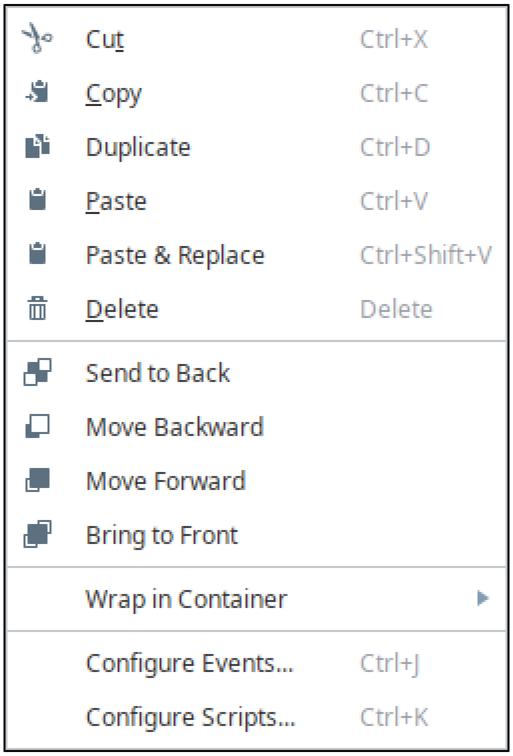
Select View

The following feature is new in Ignition version **8.1.37**
Click here to check out the other new features

Starting in Ignition version 8.1.37, you can click the grey design area in the Designer's workspace to directly select the top level view when a nested child component is already selected.

Right-Click Menu

When working with components in the Designer, you can right-click to get quick access to options.



The top portion of the menu is similar to an edit menu in that it provides much of the basic copy/paste functionality

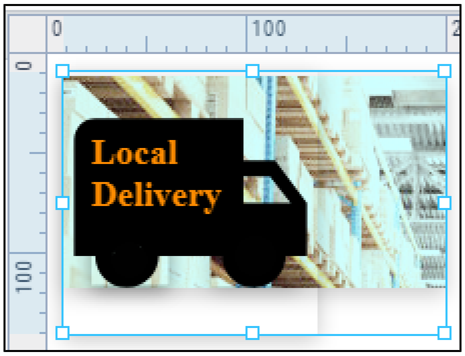
Function	Icon	Description
Cut	✂	Removes the selected component but keeps a copy on the clipboard. You can also use the **Ctrl-X** shortcut to quickly cut a component.
Copy	📋	Copies the selected component to the clipboard. You can also use the **Ctrl-C** shortcut to quickly copy a component.
Paste	📋	Pastes the current contents of the clipboard. Does not paste over and replace components. You can also use the **Ctrl-V** shortcut to paste.
Duplicate	⧉	Duplicates the selected item (essentially a fast copy and paste action). Components can also be duplicated by dragging them and holding down the **Ctrl** key. You can also use the **Ctrl-D** shortcut to quickly duplicate a component in place.
Delete	🗑	Deletes the currently selected component. This can also be done using the delete key.
Paste & Replace	📋	Pastes the current contents of the clipboard over the currently selected component and replaces it. You can also use the **Ctrl-Shift-V** shortcut to paste over.

Z-order

The next section of the right-click menu contains options or setting the z-order of components. The z-order is the order in which two-dimensional objects are stacked, for example shapes in a graphic that overlap each other. In Perspective, z-order defines relative order of components when they overlap. There are four z-order icons and actions that will reorder any selected components.

Function	Icon	Action
Send to Back	⧉	Move the selected components to the back of the z-order.
Move Backward	◻	Move the selected components backward in the z-order relative to any overlapping components.
Move Forward	◼	Move the selected components forward in the z-order relative to any overlapping components.
Bring to Front	⧉	Move the selected components to the front of the z-order.

In the following example, we have an image component with a photo of a warehouse, an icon, and a label. We placed an icon (the truck) and a label "Local Delivery" on the view as well, and then set the z-order so the label is on top, the truck icon is in the middle, and the image in the background.

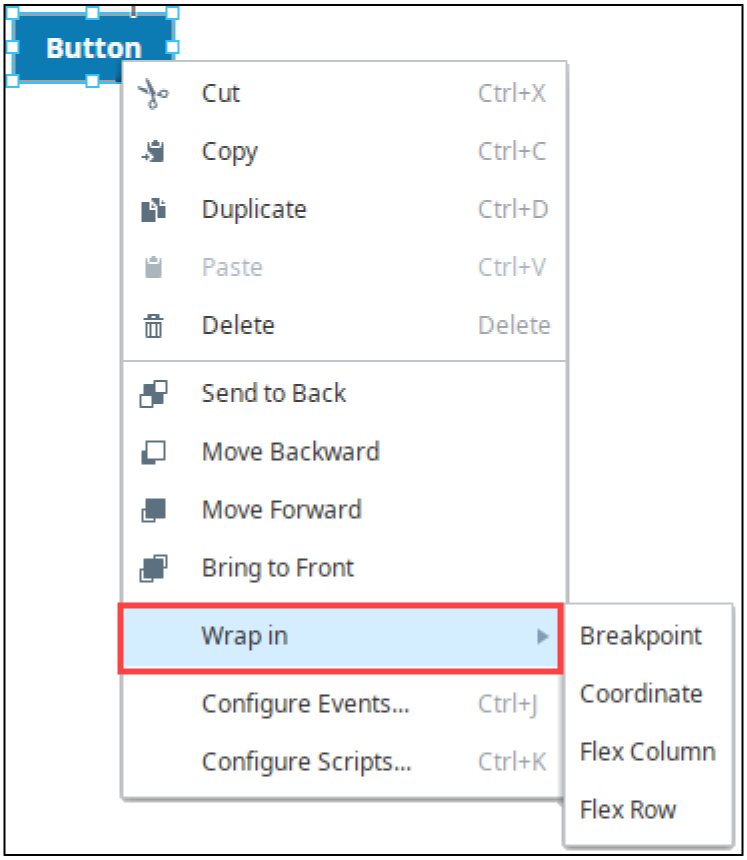


Wrap in Container

The following feature is new in Ignition version **8.1.0**
Click here to check out the other new features

Perspective features a one-step wrapping option. The Wrap In Container action, will "wrap" a component into a new container within the existing container. This is functionally similar to creating a new container, and copy-pasting the component into the new container.

Wrap in Container is available on the right-click menu from any Perspective component.



Modify Source View

The following feature is new in Ignition version **8.1.6**
Click here to check out the other new features

This option only appears in the list after right clicking on an embedded view component or flex repeater component. Selecting this option will take you to the source view's configuration in the designer.

Configure Events

The Configure Events option takes you to the Event Configuration page where you can set up actions for the component based on events such as a mouse click or key press. For complete information, see Component Events and Actions.

Configure Scripts

The last option is Configure Scripts, which takes you to a Script Configuration page where you can set up Custom Methods Custom Methods and Message Handlers on the component.

Manipulating Components

Manipulating components can be done with both the mouse and the keyboard. You can move components around, resize them, and rotate them.

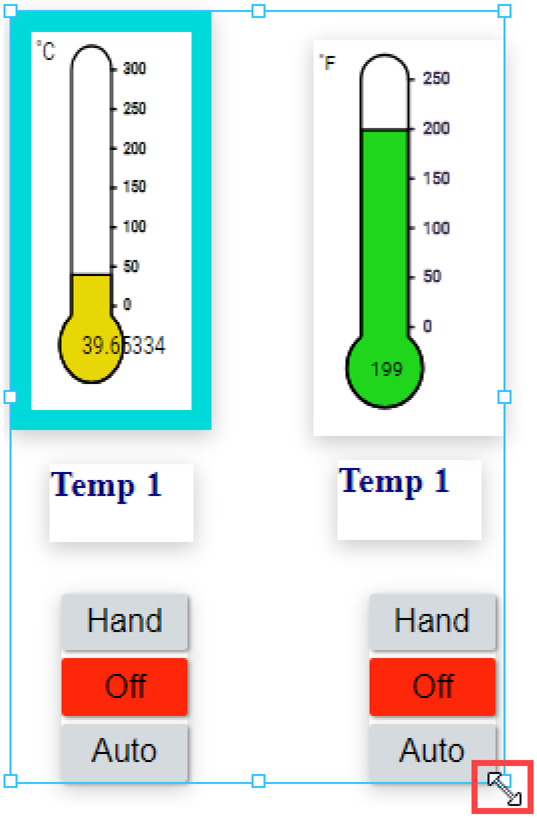
Moving

To move the component, click on it once then drag it anywhere within the container's bounds.

Resizing

When you click on the component you want to resize, you'll see eight handles displayed around the edge of the selection. When you click on a handle, the mouse cursor will change to a two-way arrow. Use the mouse to drag the handle and change the size of the component. You can also select multiple components and resize them together. To resize around the center of the current selection, hold down **Shift**.

You can also resize the current selection using the keyboard. To nudge the right or bottom edge of the selection in or out, use **Shift** combined with the arrow keys, which resizes by the nudge distance, which defaults to one pixel at a time. To nudge the top or left edge of the selection, use **Ctrl-Shift** combined with arrow keys.



Aligning

New alignment tools are available in the Perspective Designer Toolbar. These tools allow easy alignment of selected components within a Coordinate container including align top, bottom, left, right, as a row, and as a stack. Align as row, and align as stack include a normalize version, that adjusts the size of the selected components to match the component that was selected first. Rotated components being aligned will correctly align along the top-most, bottom-most, left-most, right-most point of the rotated component. If a rotated component is being normalized is within the group that is being aligned, normalizing the component will adjust the pre-rotated dimensions.

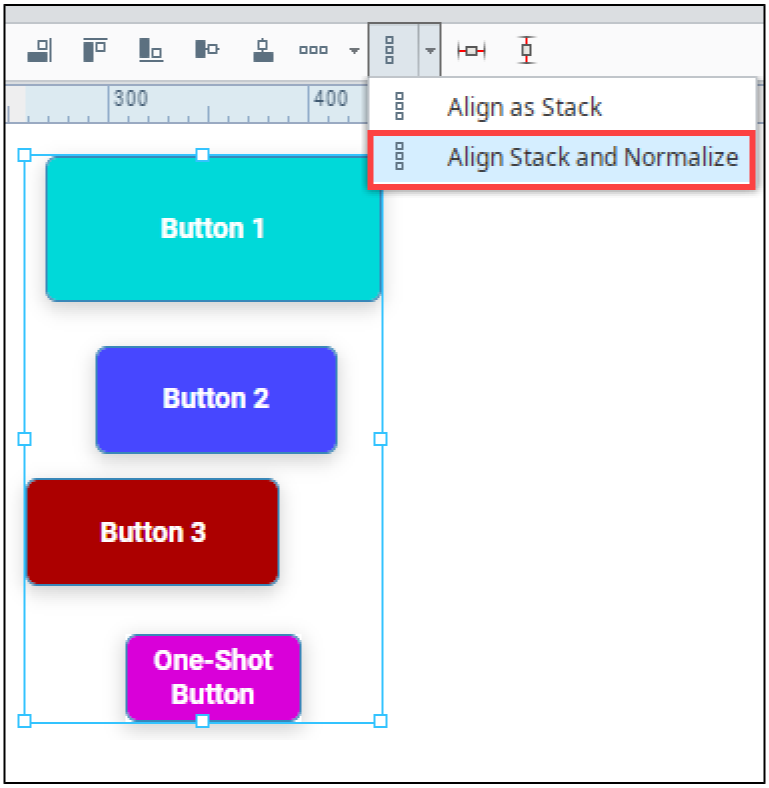
Note: These tools can only be used with components in a Coordinate container.

Icon	Function	Description
	Align Left	Align the left edges of a group of components.
	Align Right	Align the right edges of a group of components.
	Align Top	Align the top edges of a group of components.
	Align Bottom	Align the bottom edges of a group of components.
	Align Centers Horizontal	Aligns all of the selected components horizontally on their centers.
	Align Centers Vertical	Aligns all of the selected components vertically on their centers.
	Align as Row	Aligns all of the components on their centers as a row, and will add padding between them that you can select.
	Align as Row and Normalize	Aligns all of the components on their centers as a row, and changes the size of all of the components to the first selected component. If a rotated component is within the group that is being aligned, normalizing the component will adjust the pre-rotated dimensions.
	Align as Stack	Aligns all of the components on their centers as a stack, and will add padding between them that you can select.
	Align as Stack and Normalize	Aligns all of the components on their centers as a stack, and changes the size of all of the components to the first selected component. If a rotated component is within the group that is being aligned, normalizing the component will adjust the pre-rotated dimensions.
	Center Horizontally	Centers the currently selected components horizontally.
	Center Vertically	Centers the currently selected components vertically.

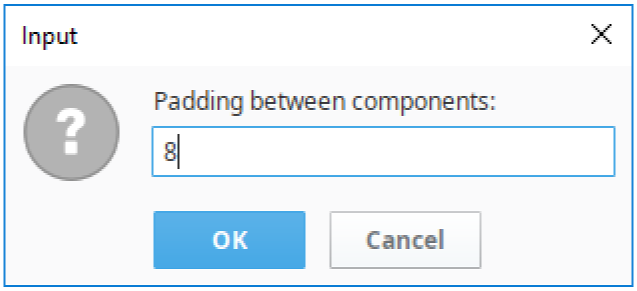
Align and Normalize Example

In the following example, we have a stack of buttons that are various sizes. We want to align them, stack them with equal space in between them, and make them all the same size.

1. Select the buttons, then choose Align as Stack and Normalize.

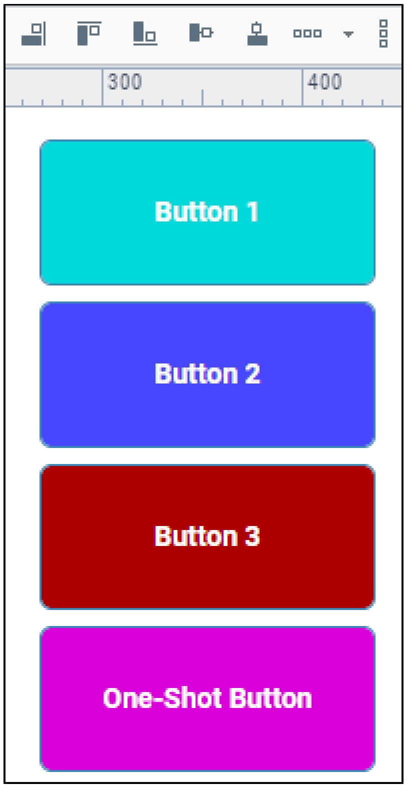


2. Enter a padding distance in pixels. We use 8px for this example.



Padding between components: 8

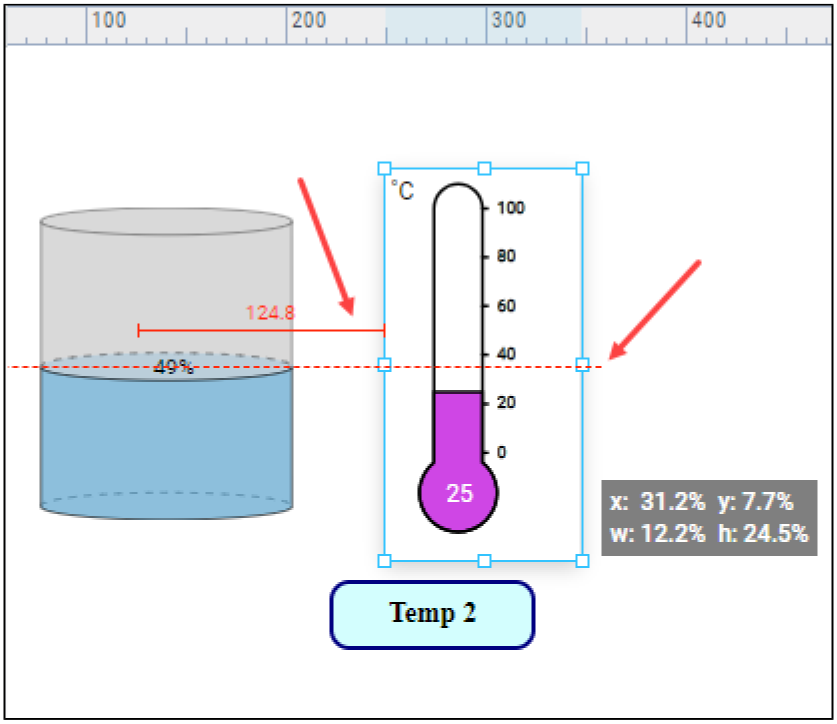
3. Click **OK**. The buttons will be stacked and normalized.

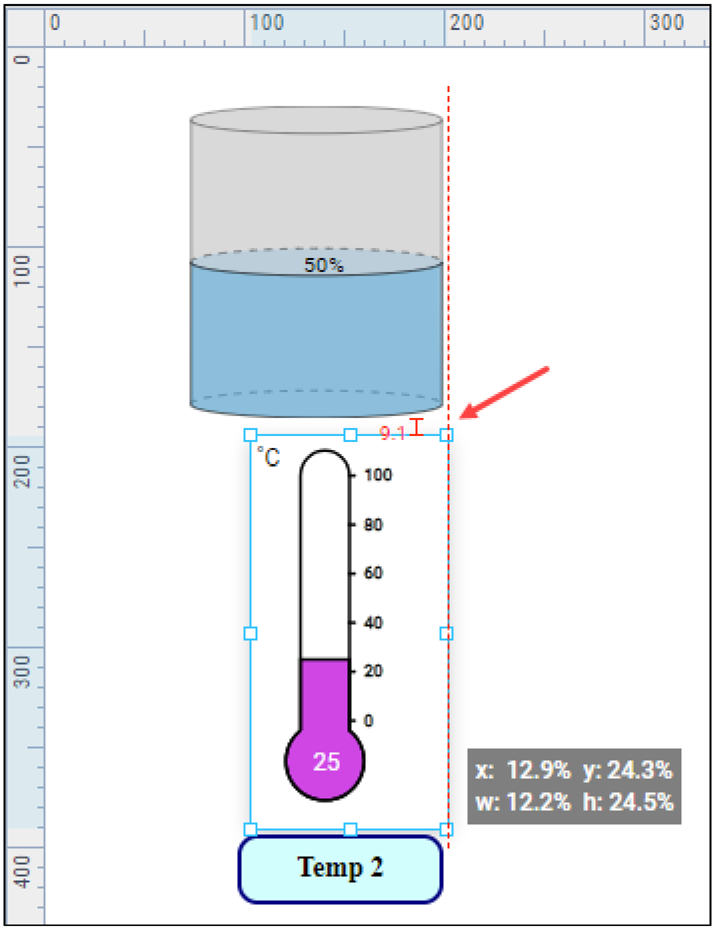


Alignment Guides

When you drag a component near another component in a container, Alignment Guidelines appear to help you better align your elements.

- A dashed red line will appear to guide alignment to the center of the components or to the right or left edge, depending on where you drag.
- A solid red line indicates distance, in pixels, between components.



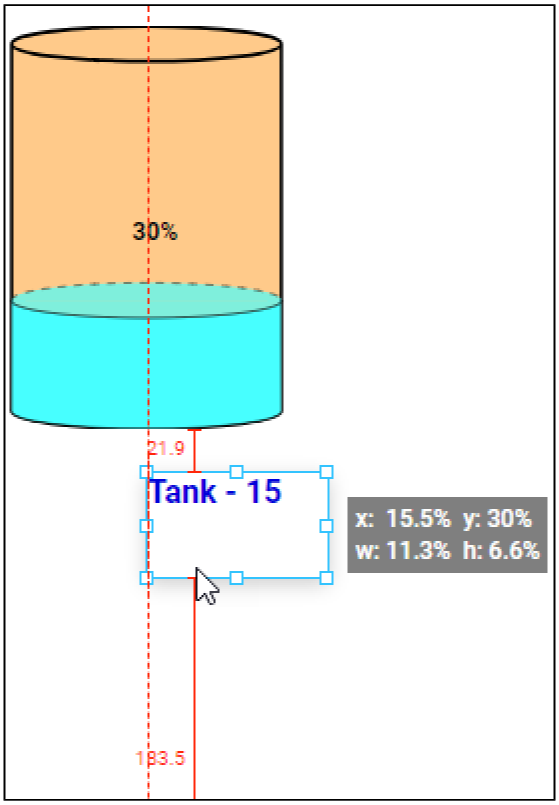


The Designer also has Vertical and Horizontal guide that can be set up to help you align components. For more information, see Vertical and Horizontal Guides.

Alignment Guides Example

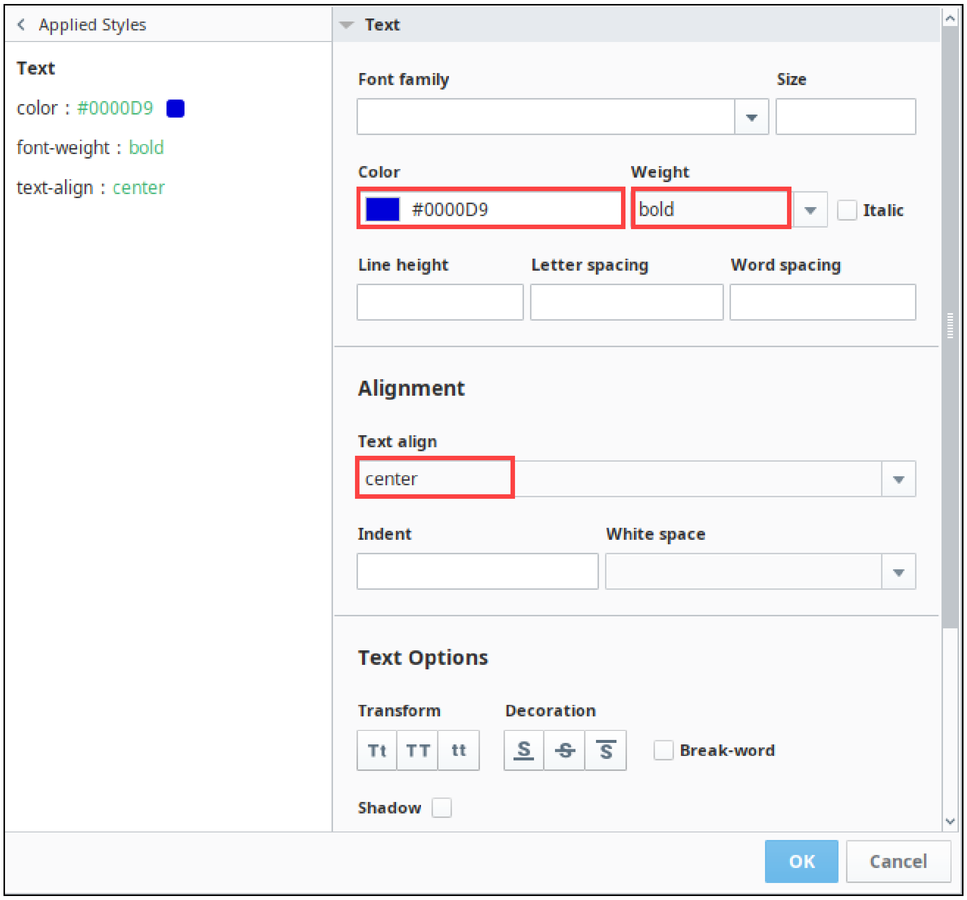
In the following example we've added a Label component beneath a Cylindrical Tank component. I want to align the label so that it is centered with the center line of the tank.

1. Drag the Label component underneath the Cylindrical Tank component. As the Label component approaches another component, you'll see Alignment Guides appear.



   Now the starting edge of the Label is aligned with the center of the Cylindrical Tank. But we want the text to be centered under the tank.

2. Select the Label component and click the **Modify Style** icon next to the **style** property.
3. Expand the Text settings. Set the color to blue, the font weight to bold, and the text align to center.
4. Click **OK** to save the style settings.



Applied Styles

Text

color : #0000D9

font-weight : bold

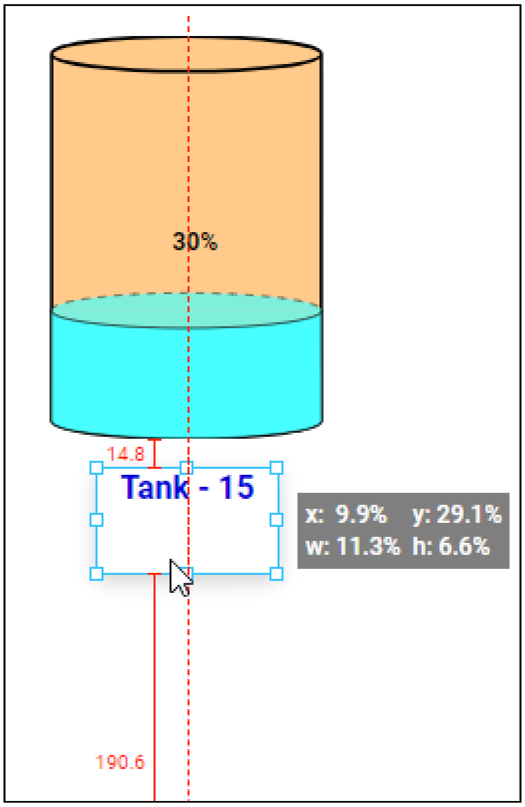
text-align : center

Color: #0000D9

Weight: bold

Text align: center

5. Now the text is centered in the Label component. Drag the Label component underneath the Cylindrical Tank component again. You'll see Alignment Guides appear.
6. Stop dragging when the Label is centered under the Cylindrical Tank.



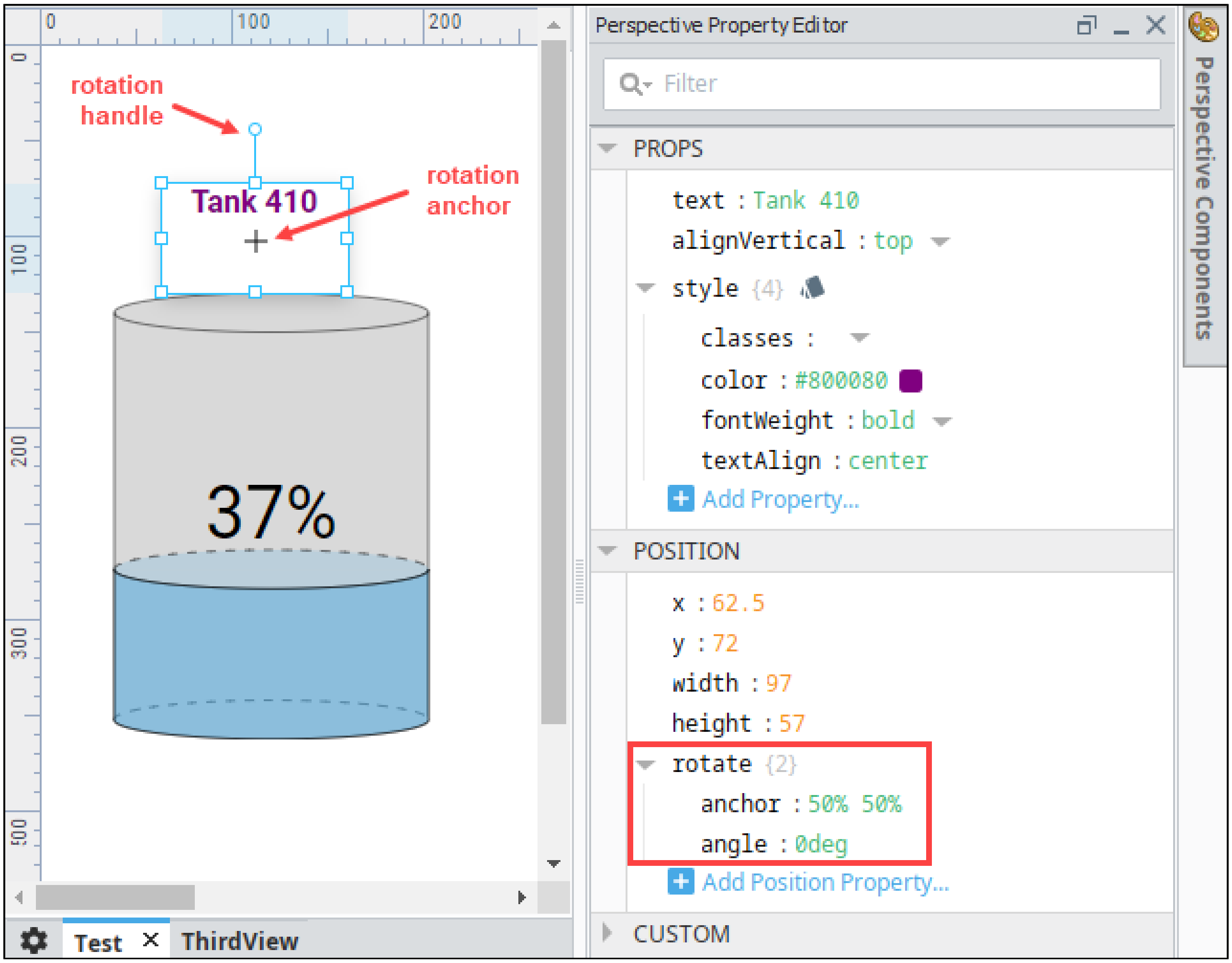
Rotating

Components place on coordinate containers can be rotated. The Rotate property is located on Position Properties section of the Perspective Property Editor. For a definition of all the position properties see the individual container pages. Rotated components being aligned to other components will correctly align along the top-most, bottom-most, left-most, right-most point of the rotated component.

There are two ways to rotate a component:

- Enter a rotation angle in the props.rotate.angle property.
- Grab the rotation handle on the component and drag it with your mouse until the component is rotates as you'd like.

The rotation anchor sets the point of rotation around which the component will be rotated.



PROPS

text : Tank 410
alignVertical : top
style {4}
classes :
color : #800080
fontWeight : bold
textAlign : center
Add Property...

POSITION

x : 62.5
y : 72
width : 97
height : 57
rotate {2}
anchor : 50% 50%
angle : 0deg
Add Position Property...

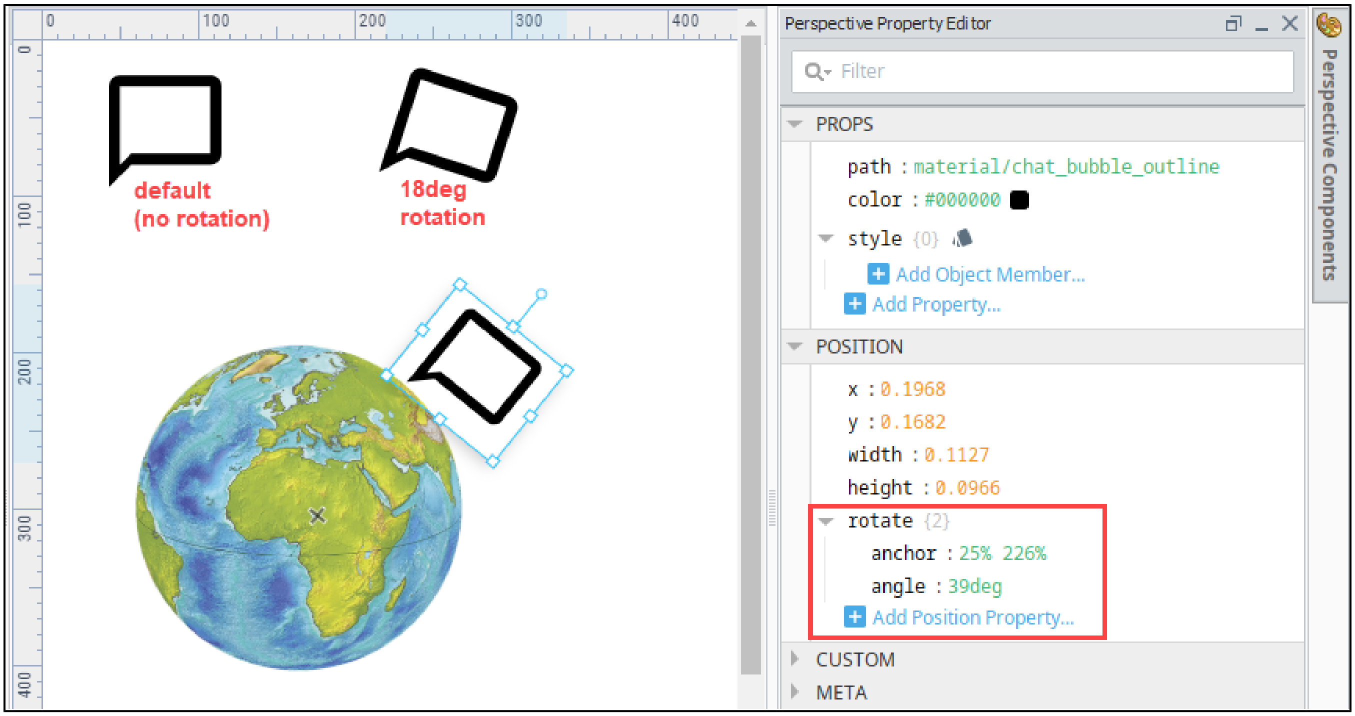
CUSTOM

Note: If a component is smaller than 28px by 28px, the rotation handle and anchor + symbol are not displayed. Use the property editor to set rotation and anchor properties in this case.

Rotation Examples

Example 1

The following example shows icon component at default (no rotation), at 18 degrees rotation, and then at 39deg rotation with the rotation anchor (point of rotation) placed over the earth image. In the latter, the icon has a better angle in relation to the earth graphic.



Perspective Property Editor

Filter

PROPS

path : material/chat_bubble_outline
color : #000000 ■
style {0}
Add Object Member...
Add Property...

POSITION

x : 0.1968
y : 0.1682
width : 0.1127
height : 0.0966
rotate {2}
anchor : 25% 226%
angle : 39deg
Add Position Property...

CUSTOM

META

Example 2

If a rotated component is within the group that is being aligned, normalizing the component will adjust the pre-rotated dimensions. The component on the left of the following image are not aligned but were resized at some point. The components on the right show what they look like after align stack and normalize.

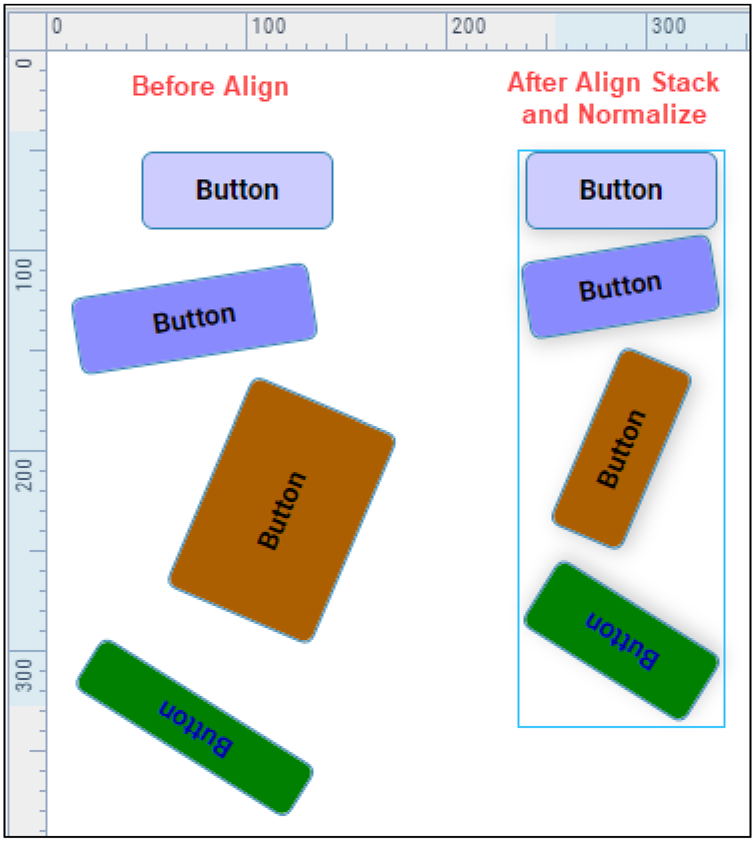
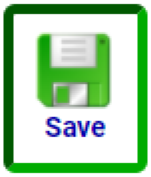


Image Source

Many components in Perspective contain a `image.source` property, that allows you to show an image on the component. The property expects a URL to the image, which can either be on the internet or something that is stored on the Gateway. For example, here is a Button component with an image:



Web Address

Simply enter the wed-address for the image you wish to display. Both raster images and SVGs can be displayed via this method.

Image Management

Images stored in Image Management are available at http://{your gateway's IP address}:{your gateway's port}`/system/images/{path to your image`}.

```
http://gatewayIpAddress:8088/system/images/Builtin/icons/24/lightbulb_on.png
```

For additional information on using images in Perspective, see Images and Icons in Perspective.

Perspective Component Properties

Each Perspective component has a unique set of properties that can be set and modified within the Perspective Property Editor. A component property is simply a named variable with a distinct type that affects something about the component's behavior or appearance, such as size, color, name, visibility. You can also create your own custom properties on a component which act like variables that can store any information that you want on the component.

The available properties for each Perspective component are described individually in the Perspective Components section. In the default panel settings, the Property Editor appears on the right side of the Designer screen and contains all the properties that can be configured on a component, including custom properties.

The image below shows the properties that are set on the selected Button component.

Property Categories

Each Perspective component has a list of available properties. Each property is placed into one of several categories, and each property category groups the properties by some commonality. The property categories are described below.

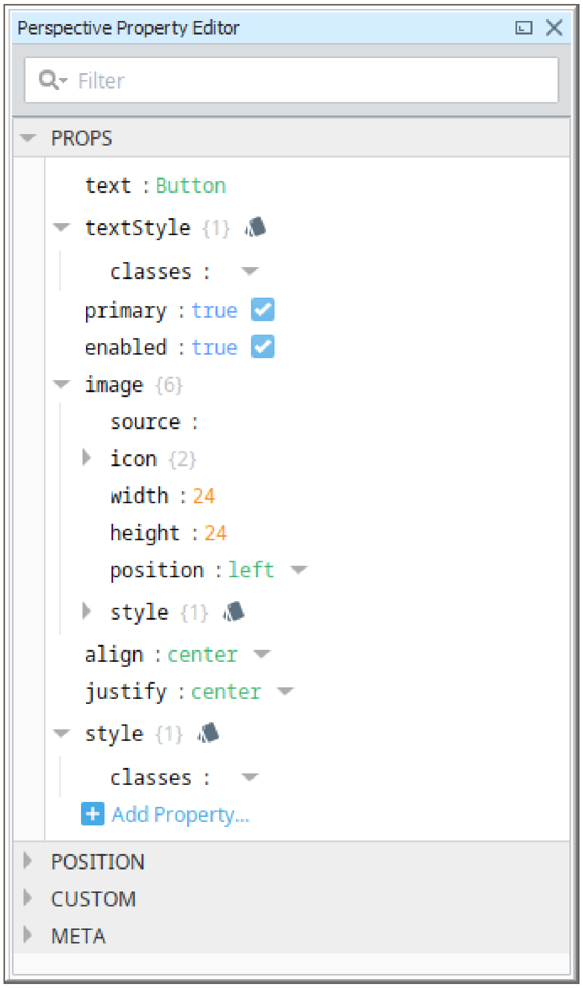
- **Props** - Properties that control the component's configuration and provides the runtime data for how the property appears and behaves in a session. See individual Perspective Components for a list of the properties and their descriptions.
- **Position** - Properties defined by the component’s parent container control where the component is located inside the container. The available properties listed under this category depend entirely on the container type that the component is placed in. For more information, see the pages for each type of container.
- **Custom** - The Custom category was designed as an ideal location to add user created properties.
- **Meta** - Properties defined by the Perspective Module itself for common things like the component’s name, and if the component is visible.
- **Params** - Only available on Views. This category of properties is used when passing parameters from one view to another view via navigation, or the Perspective - Embedded View component

The following image shows an example of the Property Editor for a Button component. Here we see Props is expanded, while the other categories are collapsed.

On this page ...

Component Properties

Watch the Video



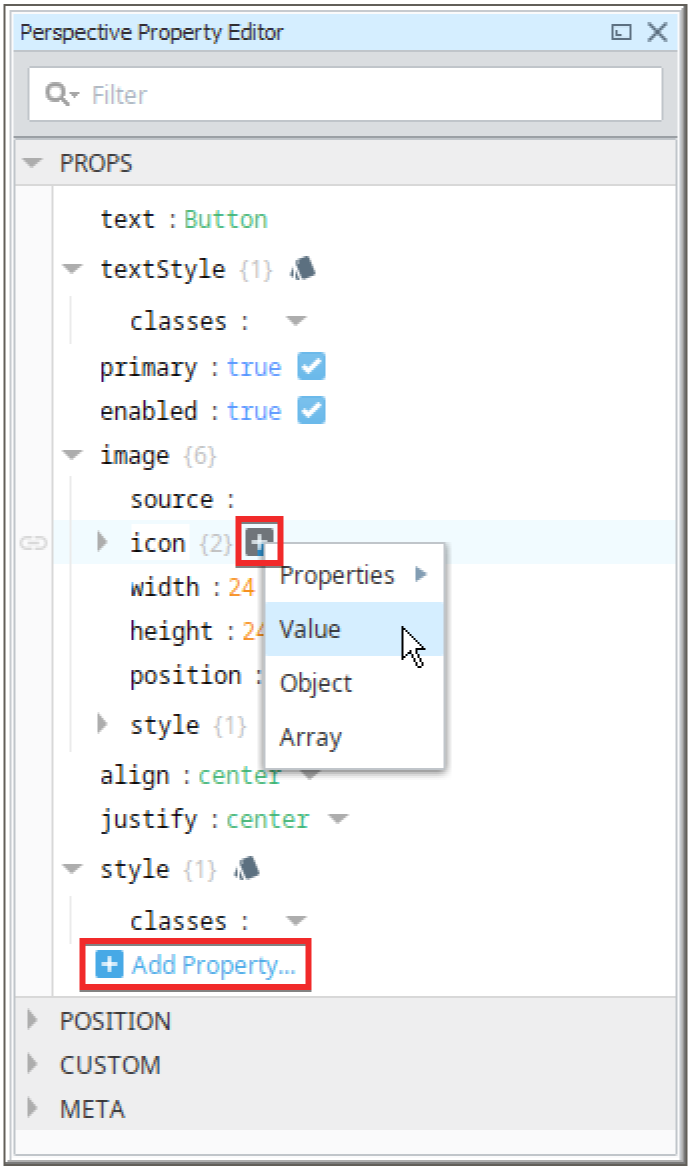
```
Perspective Property Editor
Filter
PROPS
  text : Button
  textStyle {1}
    classes :
  primary : true ☑
  enabled : true ☑
  image {6}
    source :
    icon {2}
    width : 24
    height : 24
    position : left
    style {1}
  align : center
  justify : center
  style {1}
    classes :
  + Add Property...
POSITION
CUSTOM
META
```

User Created Properties

User created properties can be added to any property category. New properties can be added by clicking the **Add Property** + icon.

It is strongly advised to add new properties to either the **Custom** or **Params** categories. C omponents can contain hidden properties in either the Props, Position, or Meta categories. These hidden properties are typically used on the backend to help the component function. By adding a user create property to Props, Position, or Meta, you could unintentionally create a property that matches the name of a hidden property, resulting in uninten ded behavior from the component. The Custom and Params categories don't have such properties, so they're a safe location for user created properties.

Alternatively, new elements can be added to objects and arrays by hovering over the object/array in the property editor, and clicking the **Add** + icon.

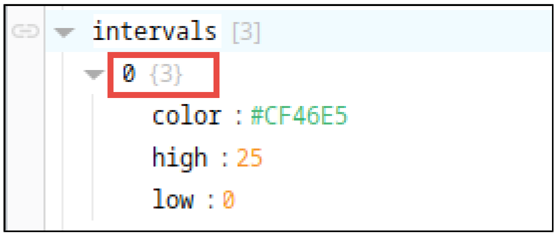
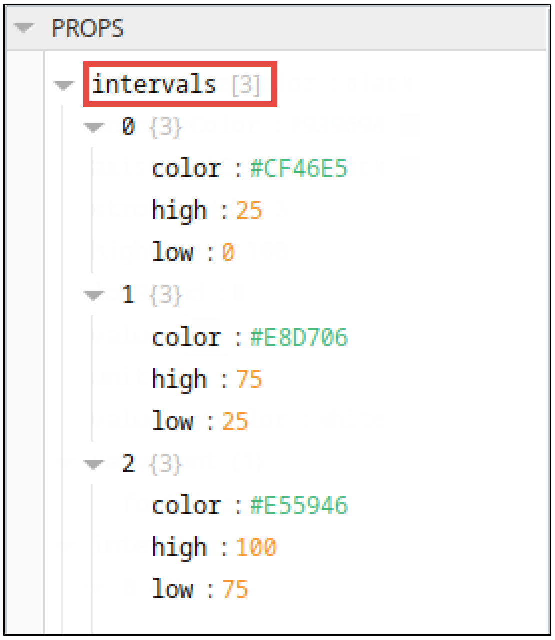
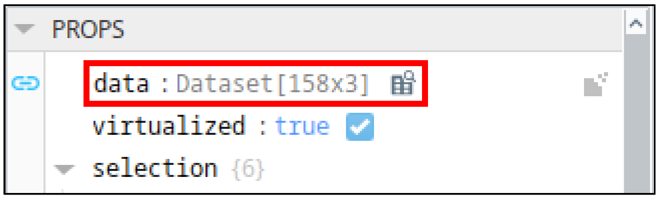


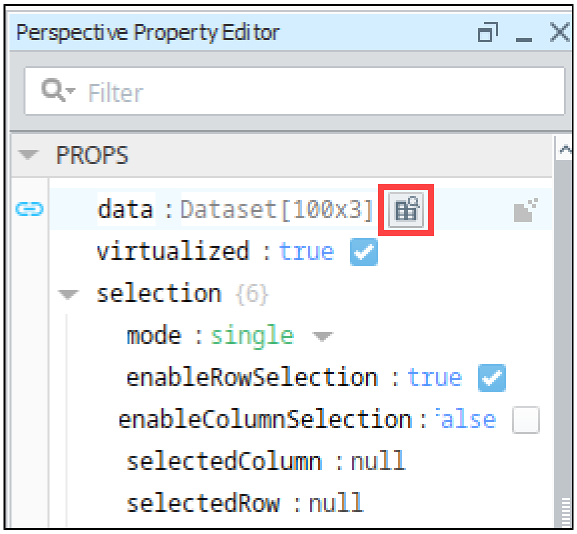
text : Button
textStyle {1}
classes :
primary : true ☑
enabled : true ☑
image {6}
source :
icon {2}
width : 24
height : 24
position :
style {1}
align : center
justify : center
style {1}
classes :

Property Data Types

Before you create a property, you must first understand the different property data types. When you create a new custom property, you must first select the appropriate property data type based on how you want the component to behave or appear. Technically, all values delivered to the frontend are encoded as JSON. Meaning they are either a Value (boolean, number, string, null), an array, or an object (string-keyed map). For convenience, Perspective offers some addition support for certain types, such as dates and datasets. The table below describes each property data type.

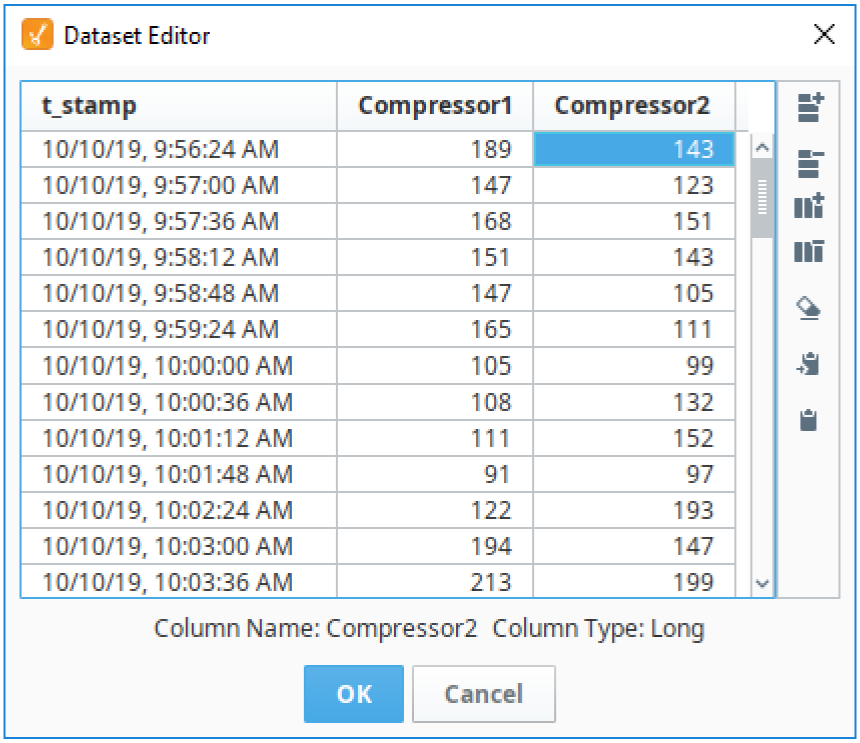
Property Types	
Value (primitive)	A value is an single variable for the property. It has a 'key" and a "value". key : value "Key" is the **name** given to the property, and "value" is the actual value of the property. Value types are as follows: • Boolean - A true/false value. • Numeric - An integer up to the maximum value for a long integer. • String - A string of characters can be numeric, alpha, or a combination
Object	

	An object is a one or more values stored under one variable name. Objects are indicated by curly braces { }. In this example, the Object has three sub- properties.  intervals [3] 0 {3} color : #CF46E5 high : 25 low : 0
Array	An array can hold many values under a single name, and you can access the values by referring to an index number. Array is indicated by square brackets [ ]. A good example of an array is the Thermometer component's default setup as shown in the example below. There is an array called "intervals" with three values, 0 through 2. Each of the array items is an object type that has three values: color (string), high (numeric), and low (numeric).  PROPS intervals [3] 0 {3} color : #CF46E5 high : 25 low : 0 1 {3} color : #E8D706 high : 75 low : 25 2 {3} color : #E55946 high : 100 low : 75
Dataset	The Dataset property type is a special variant of a Value. It can only be used when a binding returns data in a dataset format, or if a script writes a dataset to a property. A Dataset lists the number of rows and columns [**rows**x**columns**] and has an Edit Dataset icon that appears after a binding has been created. This icon brings up the Dataset Viewer panel and allows you to make changes to the raw data. Note that any changes will be overwritten the next time your binding polls. Datasets are generally only returned by SQL queries and Tag History bindings, though both have the ability to select from several different return formats. A good example of this is a table bound to a Historical Tag query.  PROPS data : Dataset [158x3] virtualized : true ☑ selection {6} Dataset properties have an dataset viewer and editor in Perspective's Property Editor. Once a property is bound to a dataset, you can access the viewer by clicking on the **Dataset Browser** ▦ icon.



```
data : Dataset[100x3]
virtualized : true
selection {6}
   mode : single
   enableRowSelection : true
  enableColumnSelection : 'alse
   selectedColumn : null
   selectedRow : null
```

With the Dataset Editor you can add and delete columns and rows, delete all rows, and copy information to or from the clipboard.

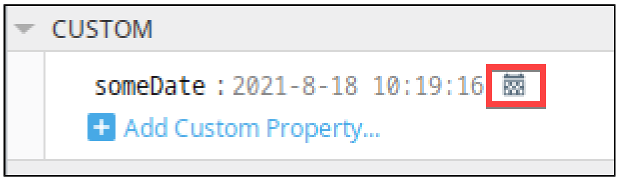
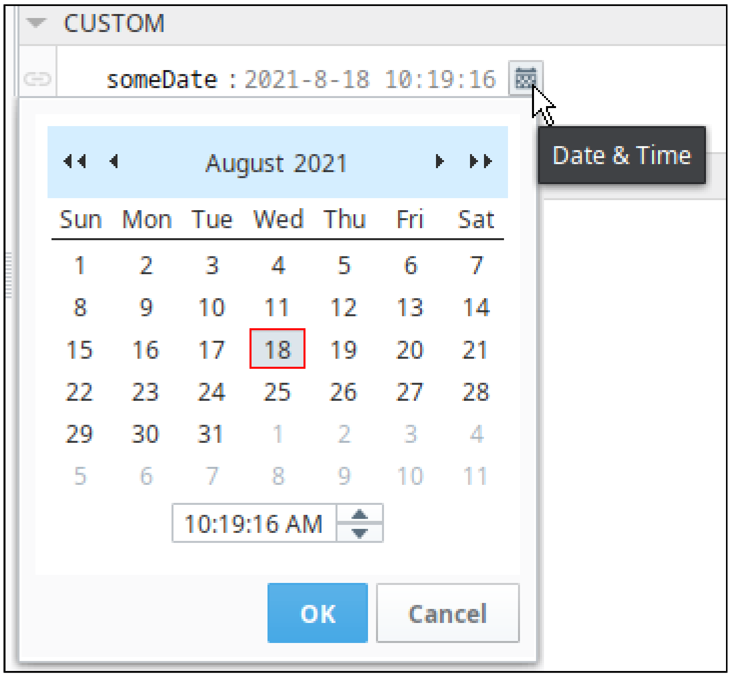
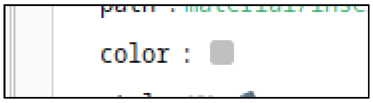


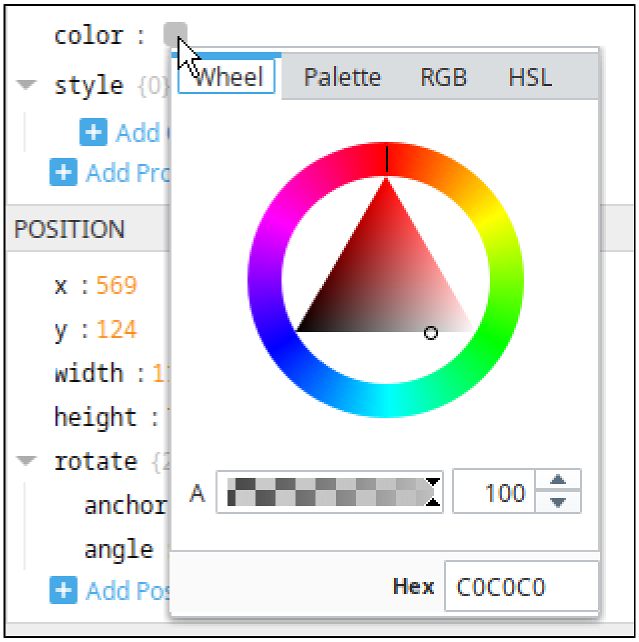
t_stamp	Compressor1	Compressor2
10/10/19, 9:56:24 AM	189	143
10/10/19, 9:57:00 AM	147	123
10/10/19, 9:57:36 AM	168	151
10/10/19, 9:58:12 AM	151	143
10/10/19, 9:58:48 AM	147	105
10/10/19, 9:59:24 AM	165	111
10/10/19, 10:00:00 AM	105	99
10/10/19, 10:00:36 AM	108	132
10/10/19, 10:01:12 AM	111	152
10/10/19, 10:01:48 AM	91	97
10/10/19, 10:02:24 AM	122	193
10/10/19, 10:03:00 AM	194	147
10/10/19, 10:03:36 AM	213	199

Column Name: Compressor2 Column Type: Long

The Dataset Editor icons and their corresponding actions are shown below.

Icon	Action
	Add row
	Delete selected rows
	Add a column
	Delete selected column
	Delete all rows
	Add to clipboard

	📋 Paste from clipboard
Date	Dates are a special variant of a Value. On the frontend they're represented in in a YYYY-MM-dd HH:mm:ss format, but on the backend they're treated as long integers. The date variant is used when a binding returns a date object (such as an expression binding that uses the now expression function). Date properties always feature a calendar icon.  CUSTOM someDate : 2021-8-18 10:19:16 Add Custom Property... Clicking the icon will cause a popup calendar to appear, making it easy to select a different date.  CUSTOM someDate : 2021-8-18 10:19:16 Date & Time August 2021 Sun Mon Tue Wed Thu Fri Sat 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 1 2 3 4 5 6 7 8 9 10 11 10:19:16 AM OK Cancel
Color	Color properties are another variant on the Value type. Many components provide color properties as a means to select the color of a certain object. On the backend, colors are simply strings, but the designer renders them as colored blocks.  color : ▨ Clicking the block will cause an color selector to appear, allowing you to change the color on the property.



Note:

JavaScript uses double-precision floating-point format numbers as specified in IEEE 754 and can only safely represent numbers between -(2^53 - 1) and 2^53 - 1. This means any value greater than 9,007,199,254,740,991 (or ~9 quadrillion) or less than -9,007,199,254,740,992 will be changed to a value calculated using floating point math up to a max value of 2^63.

This could potentially cause issues with very large numbers, especially when in a dataset property:

- Modifying a value in the Perspective Dataset Double or Long column could lead to an unexpected value.
- If bound to a Dataset tag, the value of the tag could be changed to an unexpected value.
- A Perspective Component rendering a dataset Double or Long could render an unexpected value.

Restricting Property Access

Perspective components have the ability to restrict access to properties from the app/browser. Property access settings do not restrict or inhibit built-in component interactions with bindings and python scripts. Instead they protect against malicious code execution in the browser. Normally a user can potentially execute arbitrary JavaScript code via developer tools (which generally are included with all web browsers) to interact with components and properties in the session. However setting property access level to **Private** or **Protected** will prevent such approaches, as browser-side script execution will be unable to access property values on the server side.

To understand what property access in Perspective is, you'll need to understand how the Document Object Model (DOM) works. In short, each active session is represented in a browser (which is the user interface side of the DOM) and on the Gateway (the back-end of the DOM). Interacting with components on the browser-side, such as writing to the text property on the text field, impacts the back-end and allows the gateway to react appropriately (i.e., trigger a property change script).

When a property is set to **Public**, then arbitrary JavaScript execution can freely write to the back-end, which is likely undesirable in most cases. However, a property set to **Protected** will disregard any such write requests from the browser, meaning only the back-end is allow to write to the property (i.e, Bindings, component Script Actions, etc). While the browser-side of the DOM is still interactable, the back-end will ignore such value changes.

In this same example, setting a property set to **Private** will also disregard write attempts, in addition to remaining hidden from any read attempts made by the arbitrary JavaScript.

One caveat to property access is that the names of style classes are always visible and interactable from the browser, so style class names should not included sensitive information.

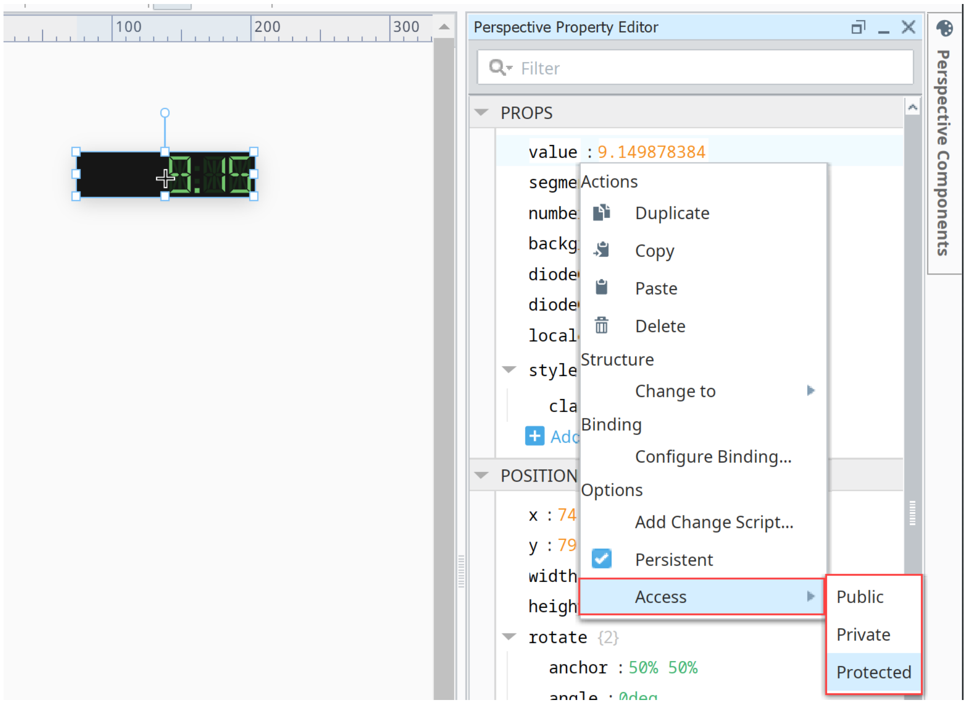
Property	Description
Public	Unrestricted Access. This is the default setting for all non-system properties.
Private	Hidden. The property is not readable from JavaScript, and write requests will be ignored (assuming the script correctly guesses the property path)

Protected	Read only. The browser's DOM may be interacted with via JavaScript, but the back-end will ignore any write requests.
System	This property's value is updated automatically. Not user-writable, read-only, and cannot be removed. System properties will not accept writes from the browser, and bindings will not be allowed to write to these properties either.

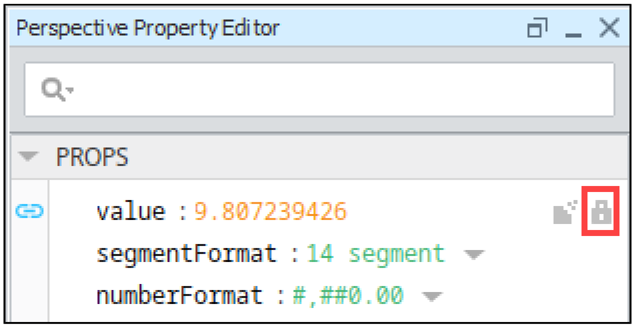
Restricting Access to Component Properties

In the example below, we used an LED Display component to restrict access on the components 'value' property.
Restricting Access to Component Properties

1. In the **Property Editor**, select the property you want to restrict access on.
2. Right-click on the property, select **Access** and choose the restriction level: Public, Private or Protected.



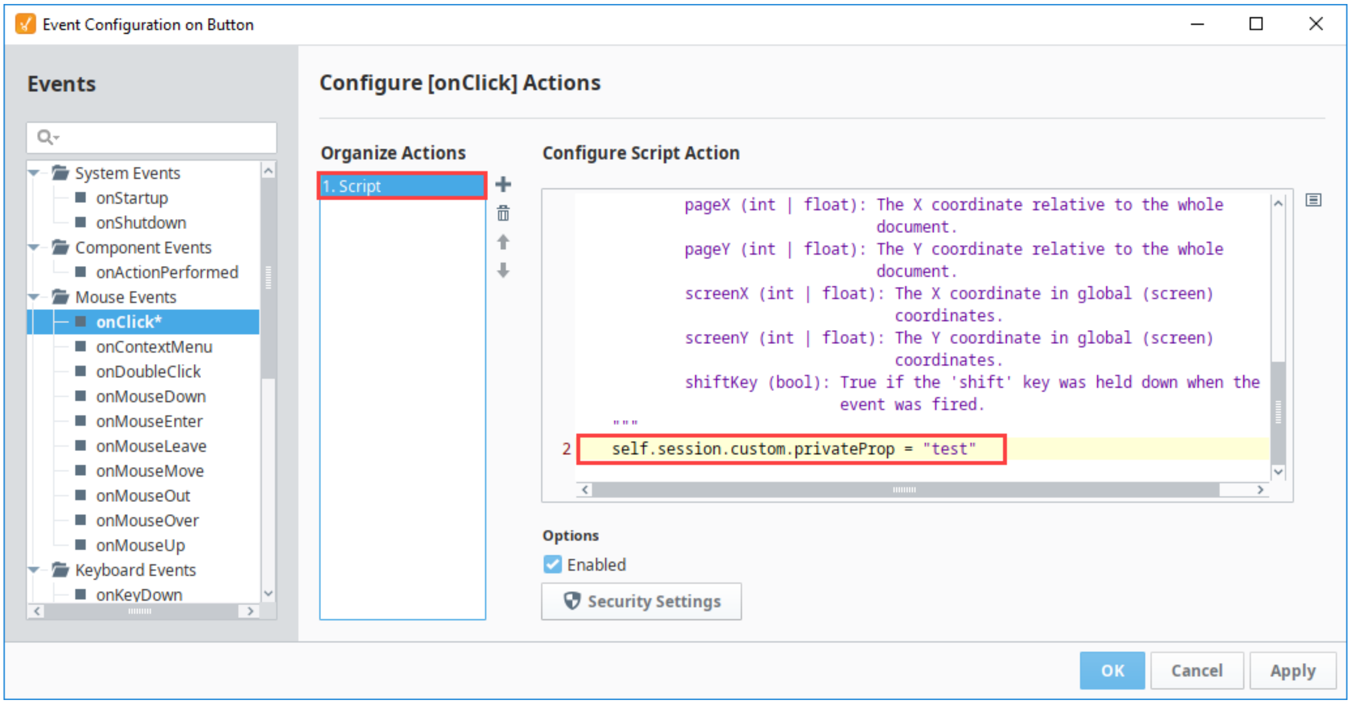
3. Once the restriction access is configured, the **Property Editor** will place one of three badges on any non-public property: Private , Protected , and System .



value : 9.807239426
segmentFormat : 14 segment
numberFormat : #,##0.00

Writing to Private or Protected Session Property

To write to a Private or Protected session prop, you can write to it through a scripting action.

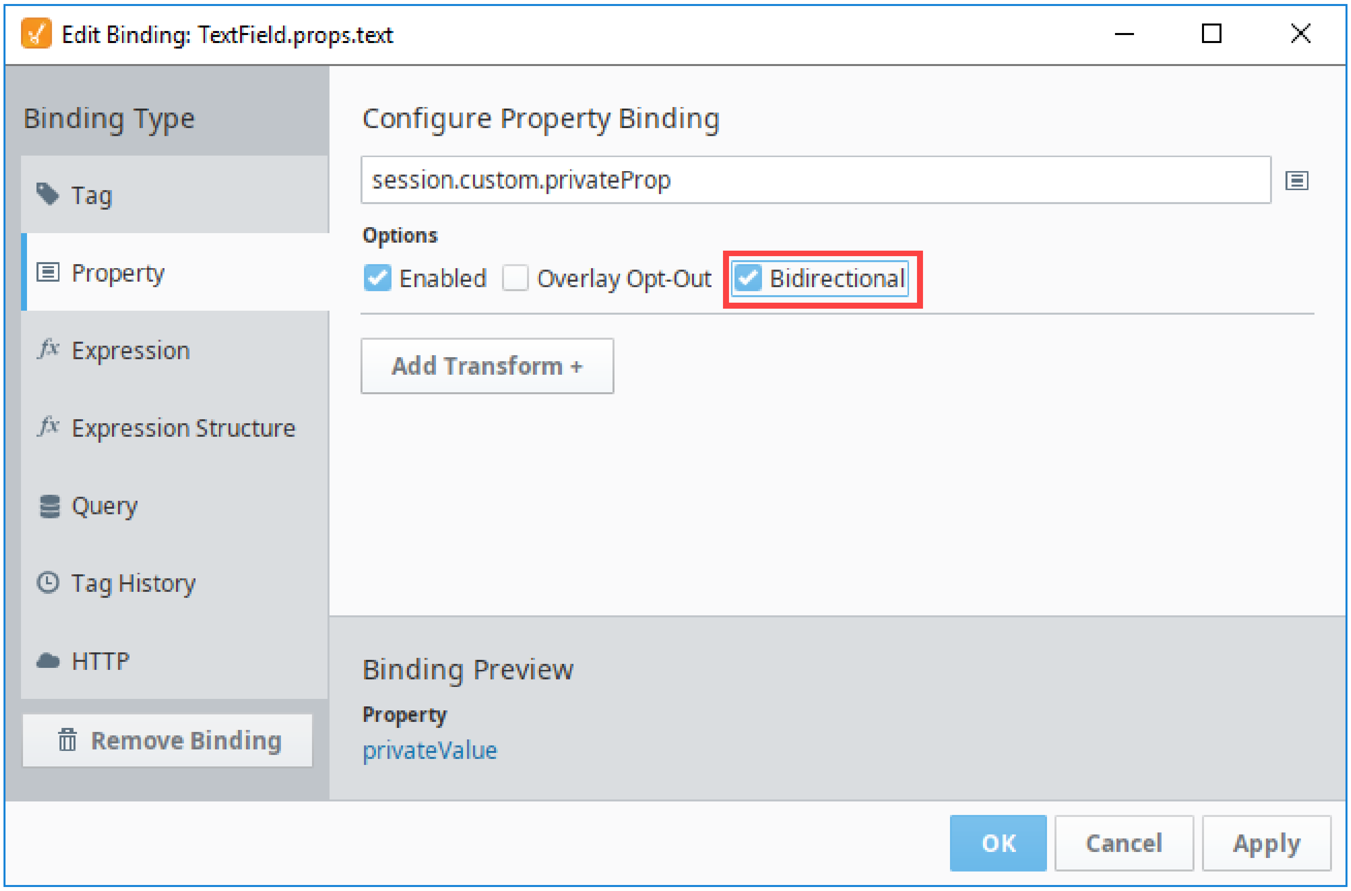


```
             pageX (int | float): The X coordinate relative to the whole
                                  document.
             pageY (int | float): The Y coordinate relative to the whole
                                  document.
             screenX (int | float): The X coordinate in global (screen)
                                    coordinates.
             screenY (int | float): The Y coordinate in global (screen)
                                    coordinates.
             shiftKey (bool): True if the 'shift' key was held down when the
                              event was fired.
     """
     self.session.custom.privateProp = "test"
```

Options

☑ Enabled

You can also use a property binding to write to a Private or Protected session prop. Remember to enable the **bidirectional** option.



session.custom.privateProp

Options

☑ Enabled ☐ Overlay Opt-Out ☑ Bidirectional

Binding Preview

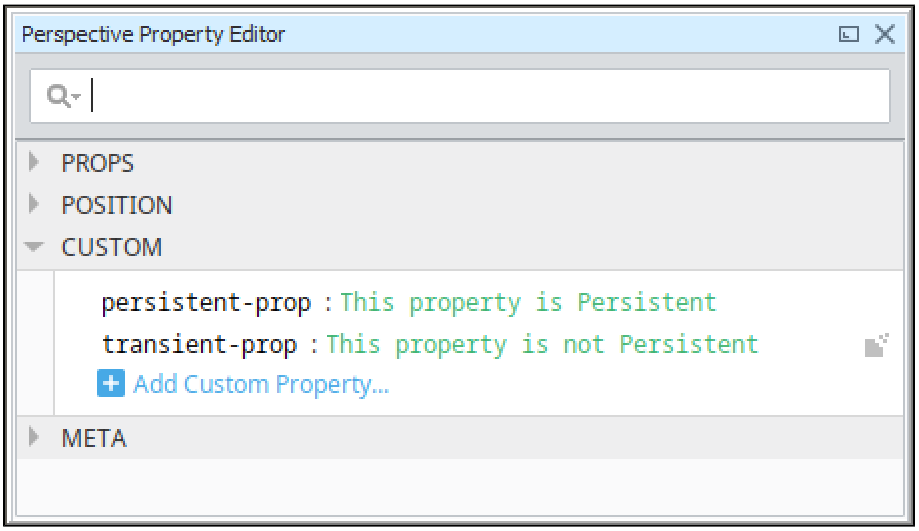
Property

privateValue

Persistent Properties

By default, component properties and their values are saved with the project. Meaning that a property in a Perspective Session will initialize with a value matching the last saved value. This is why you can create a label with a static text value, save, and then see the same text value in the session. These properties are considered **Persistent**.

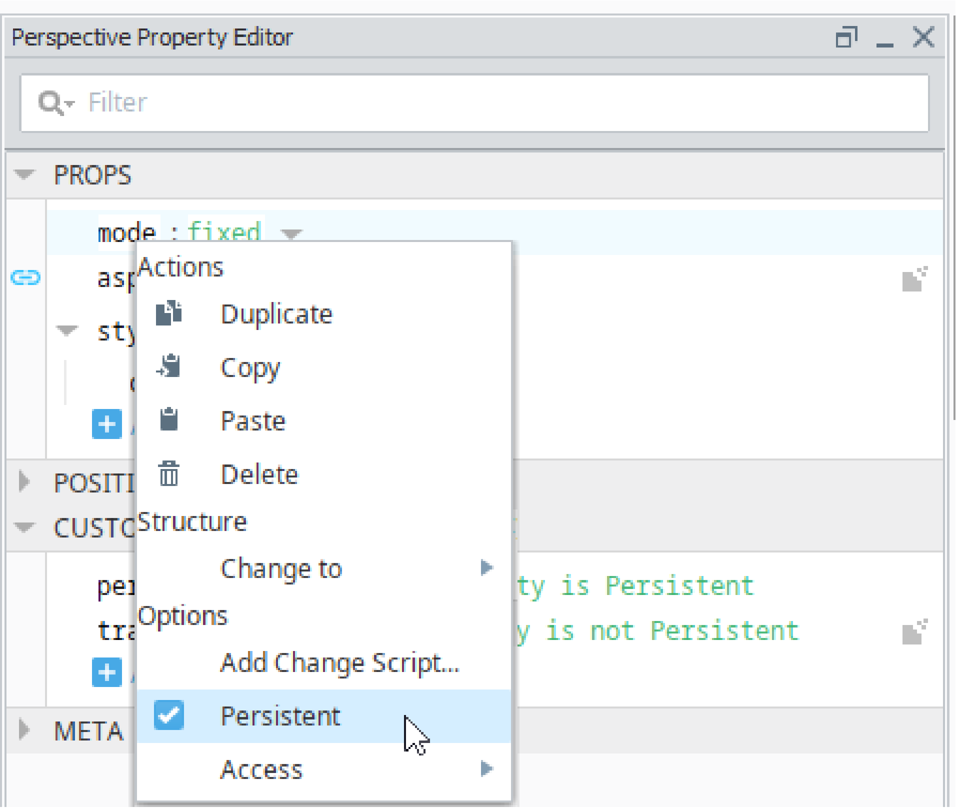
Conversely, properties can be configured to not retain their value in the session, meaning they'll initialize without a value. These properties are **not Persistent** . These properties are denoted by the **Transient** icon to the right of the property's value in the Property Editor.



persistent-prop : This property is Persistent

transient-prop : This property is not Persistent

Changing the Persistent state of a property can easily be accomplished by right-clicking on a property in the Property Editor, and toggling the **Persistent** option.



Persistence and User-Created Properties

User created properties missing the Persistent flag will not be saved in your project. Meaning, properties that are both user-created and not flagged as Persistent will be lost once the view containing the property is closed in the Designer (not just the value, but the property itself), regardless of whether or not the project was saved. This is also true for launching a session, as the user created property that is not Persistent will not be present in the session.

The one exception to this rule is if a binding was configured on the property. Binding configurations are always saved along with the component, and will execute in the session. When the binding executes and returns a valid value, the property will be re-created in the session.

Bindings and Persistence

When configuring a binding on a property, the property will automatically be configured to not persist. The idea being that properties with bindings generally don't need to save their value along with the view: when a view is opened, all bindings will need to evaluate on startup, which means the last saved value on the property is likely to differ from the result of the binding.

Imagine a table component, with a binding on the table's data property. In most cases, you'll likely want the data in the table to be generated in the session from the binding, retrieving the most update-to-date results. In this case, it doesn't make sense to persist the dataset in the table along with the table, as it's just extra data that will quickly be replaced by the binding at runtime.

Bound properties default to a non-persistent configuration, but sometimes this isn't desirable. For example, Embedded View components have an empty state when their "path" property is blank. When a binding is placed on the path property, opening the parent view will result in the Embedded View quickly transitioning between the empty state and the loaded view, which can cause an undesirable "flash" as the binding evaluates. This can be prevented by configuring the property as Persistent, and configuring an initial path for the Embedded View (which can lead to an empty placeholder view), allowing for a controlled transition.

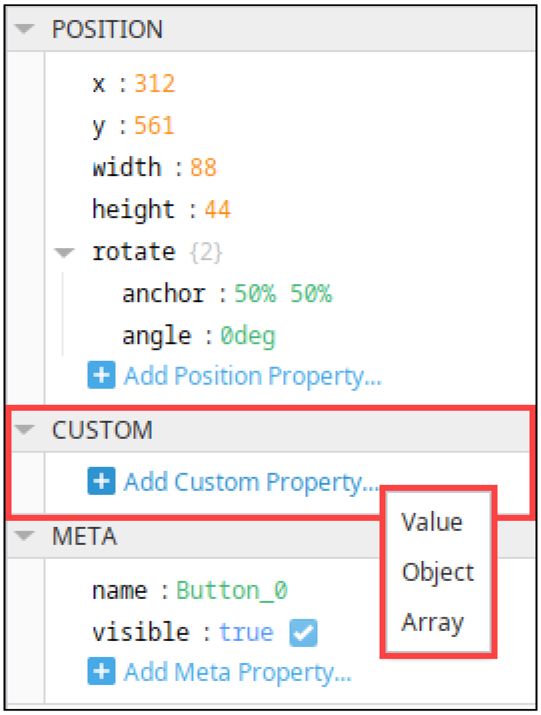
Custom Properties

User-created properties may be added to any property category such as the Props, Position, Custom and Meta Property Categories to enhance functionality. The Custom Category was designed as an ideal location in the Property Editor for users to create their own custom properties. These Custom properties allow components to store additional values which can be accessed by bindings and scripts. They are also important for passing parameters from one view to another.

Creating Custom Properties

In the following example, we used the Thermometer component to add some custom properties in the Custom category.

1. Custom Properties are created by clicking on the "Add Custom Property..." link, or by right clicking on a property in the Property Editor to bring up the Action Menu. There are three property types to choose from: Value, Object, and Array.

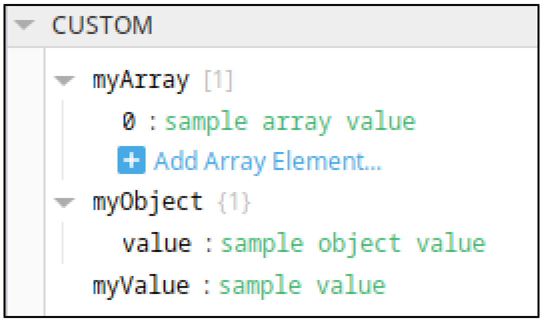


POSITION
x : 312
y : 561
width : 88
height : 44
rotate {2}
anchor : 50% 50%
angle : 0deg

CUSTOM

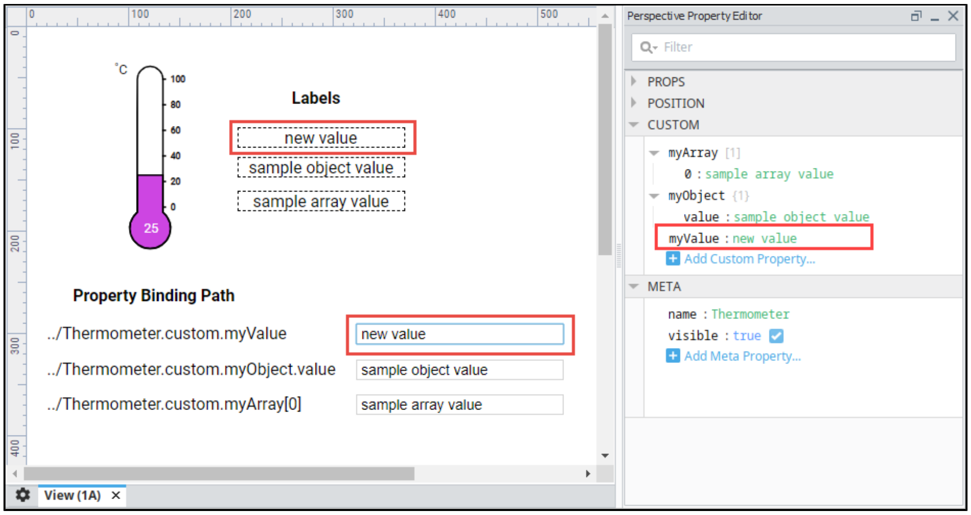
META
name : Button_0
visible : true ☑

2. Several custom properties were created under **Custom** in the Property Editor showing each of the different property data types (i.e., value, object, and array).



```
CUSTOM
  myArray [1]
    0 : sample array value
    Add Array Element...
  myObject {1}
    value : sample object value
  myValue : sample value
```

3. In the image below, next to the Thermometer, three Text Fields were added to a view to write to each of the different custom properties in the Thermometer. Each of their binding paths are listed next to the fields which are bound to the same custom property so you can see the difference when the values changes on the labels.



Labels

new value

sample object value

sample array value

Property Binding Path	
../Thermometer.custom.myValue	new value
../Thermometer.custom.myObject.value	sample object value
../Thermometer.custom.myArray[0]	sample array value

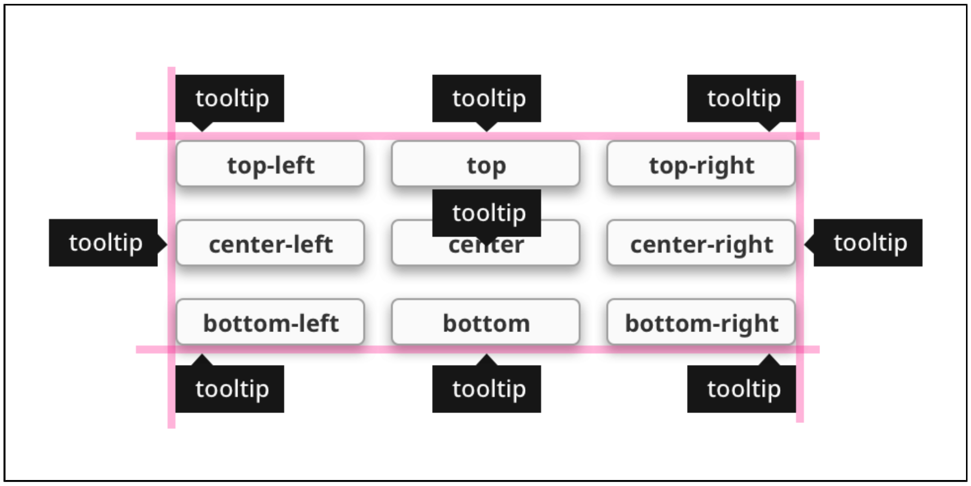
```
Perspective Property Editor
Filter
PROPS
POSITION
CUSTOM
  myArray [1]
    0 : sample array value
  myObject {1}
    value : sample object value
  myValue : new value
  Add Custom Property...
META
  name : Thermometer
  visible : true
  Add Meta Property...
```

Meta Properties

Meta properties are defined by the Perspective Module itself for common things like the component's name, and if the component is visible. Every Perspective component features Meta Properties. Each property under this category is listed below.

<table>
<tr><th>Name</th><th>Description</th><th>Dat
Typ</th></tr>
<tr><td>name</td><td>Name of the component used when navigating tree paths by name.</td><td>value
string</td></tr>
<tr><td>visible</td><td>Whether or not the component should be visible.</td><td>value
boole</td></tr>
<tr><td>tooltip</td><td>
The following feature is new in Ignition version <b>8.1.10</b>
Click here to check out the other new features

When enabled, the component will display a customizable tooltip. Tooltips feature the following properties:
<table>
<tr><th>Name</th><th>Description</th><th>Data Type</th></tr>
<tr><td>enabled</td><td>If true, a tooltip for the component will display on hover or when requested by a script.</td><td>value: boolean</td></tr>
<tr><td>width</td><td>Display width of the tooltip. Expects either numerical value that represents a number of pixels, or a string. Expected strings can be either "auto" (which allows the tooltip to decide its own width), or a number and unit. For example: "60px", or "100pt". Default value is auto.</td><td>value: string</td></tr>
<tr><td>text</td><td>The message to display in the component tooltip. See Multi-Line Tooltips for more information.</td><td>value:</td></tr>
</table>
</td><td>objec</td></tr>
</table>

		string
style	Standard style object for the tooltip. See Style Reference.	object
delay	Time, in milliseconds, to wait before the tooltip is displayed when requested or when the component is hovered over. A value of 0 results in immediate display.	value: numeric
sustain	Time, in milliseconds, to display tooltip before removing it. A value of 0 results in the tooltip displaying until the mouse exits the component or removeTooltip() is called on the component.	value: numeric
location	The tooltip display location relative to the component. Valid values: mouse, top/center/bottom, top/center /bottom- right, top/center/bottom - left Tooltip locations are constrained by the available space, and will display with the following priorities: 1. The tooltip must remain in the viewport at all times. 2. The tooltip will expand away from the component relative to the pink guidelines shown below.  tooltip tooltip tooltip top-left top top-right tooltip tooltip center-left center center-right tooltip bottom-left bottom bottom-right tooltip tooltip tooltip If the tooltip is too large to expand away from the component without leaving the viewport, the tooltip will still display at the configured location but may obscure the component.	value: string
tail	Enables a decorative triangle on the tooltip that points to the component. Ignored when location is mouse.	value: boolean

context Menu	The following feature is new in Ignition version **8.1.25** Click here to check out the other new features When enabled, the component displays a context menu when right-clicked. The contextMenu property features the following properties:	objec

Name	Description	Data Type
enabled	Determines if the context menu for this component is on or off. If true, a context menu will appear when items are added. The context menu is disabled by default.	value: boolean
items	Adds items to configure and display in the context menu. See Context Menu Item Types below for more description on the type options.	object

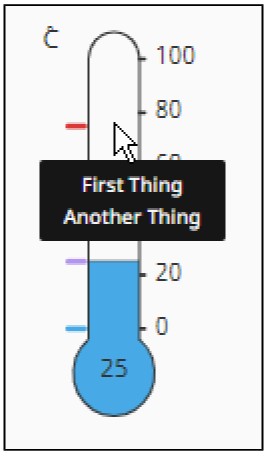
Name	Description	Data Type
text	Text to display on the context menu item.	value: string
icon	Configured icon to display before item text. Icons can be configured for all item types except separator.	object

Name	Description	Data Type
path	Shorthand path to icon source, using format: library /iconName.	value: boolean

			color	Selects the color of the icon to display. This is an optional property, color may also be configured using the style property.	value: boolean
			style	Standard style object for the icon.	Object
		style	Standard style object for the context menu item.		object
		type	Selects the type of context menu item to display. Type options include submenu, link, method, message, and separator.		value: string
	style	Standard style object for the context menu.			object
domId	Hidden by default. When added to the Meta category on a component, allows you to set the DOM "id" of the output element. This property is intended for testing purposes only, such as using a framework like Selenium to test a page.				value string

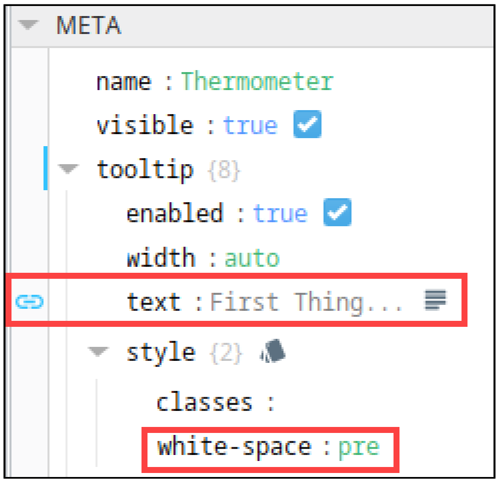
Multi-Line Tooltips

By default, the content in the `meta.tooltip.text` property is all rendered as a single line, only wrapping when text length exceeds the width. However it is possible to render text that spans multiple lines.



First, set add "`white-space : pre`" to meta.tooltip.style. Then, make sure the content of the meta.tooltip.text property contains multiple lines. This can be done by configuring an expression binding, and concatenating the items with "`\n`" characters:

```
"First Thing" + "\n" + "Another Thing"
```



META

name : Thermometer
visible : true
tooltip {8}
enabled : true
width : auto
text : First Thing...
style {2}
classes :
white-space : pre

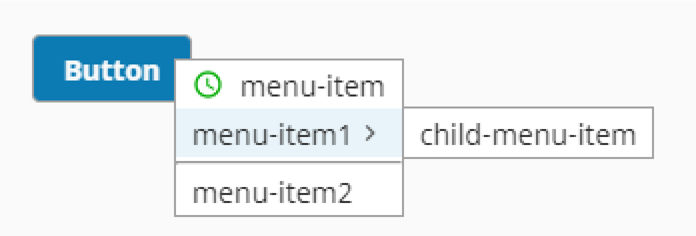
Context Menu Item Types

The following feature is new in Ignition version **8.1.25**
Click here to check out the other new features

Context menus allow users to have more component control when right-clicking on the configured component. Once opened, context menus will stay active until a new click is made elsewhere on the view. There are five item type options available to customize component context menus. Four of these options allow further navigation functionality, with one type allowing for category separation within the menu design.

Item Type	Description
submenu	This item type will display child items when clicked or hovered over and when child items exist, a simple arrow **>** icon will be included after the parent item text. . When selected, a children property will appear in the property editor that consists of the same item types and properties.
link	This item type will act like a link component, which allows users to specify a link to navigate to when clicked. When selected, a link property will appear containing url and target properties: ○ url: Contains the URL to be used for the link item. This can be an external URL or internal project page URL. To specify project pages, provide the exact configured URL for that page, complete with the leading slash (/someUrl/myPage). ○ target: Determines if the link will open in the current tab or a new tab.
method	When clicked, this item type will invoke a custom method defined on the component that the menu is enabled on. Any method defined here will become a custom method of the component the context menu belongs to. You may configure arguments to be passed into this custom method with the params object. When selected, a method property will appear containing name and params properties: ○ name: Displays the name of the method to be invoked. ○ params: Adds the object of params to pass to the method. Note that positional arguments are not supported.
message	When clicked, this item type will send a message with the given message type code, payload, and scope. This is allowed when specifying a configured message handler for a perspective session or perspective component, such as a toggle switch. When selected, a message property will appear containing type, payload, and scope properties: ○ type: Defines the message type that will trigger all component message handlers with the same type value. ○ payload: Object used to pass defined payload for message handlers. ○ scope: Defines the session, page, or view scope for sending the message.
separator	This item type displays a non-interactable horizontal hard-rule to keep context menu items separate.

The image below shows an example context menu configured on a Button component. The first item, **menu-item**, is a link type and includes a configured icon. **Menu-item1** is a submenu type with a single child item, **child-menu-item**. A separator item type divides **menu-item1** and **menu-item2**, which is a method type. This example demonstrates the context menu default style settings and you may notice different item type selections are not distinguishable in their default text display .   Context menus can be further stylized in their entirety or within the individual item properties as desired.

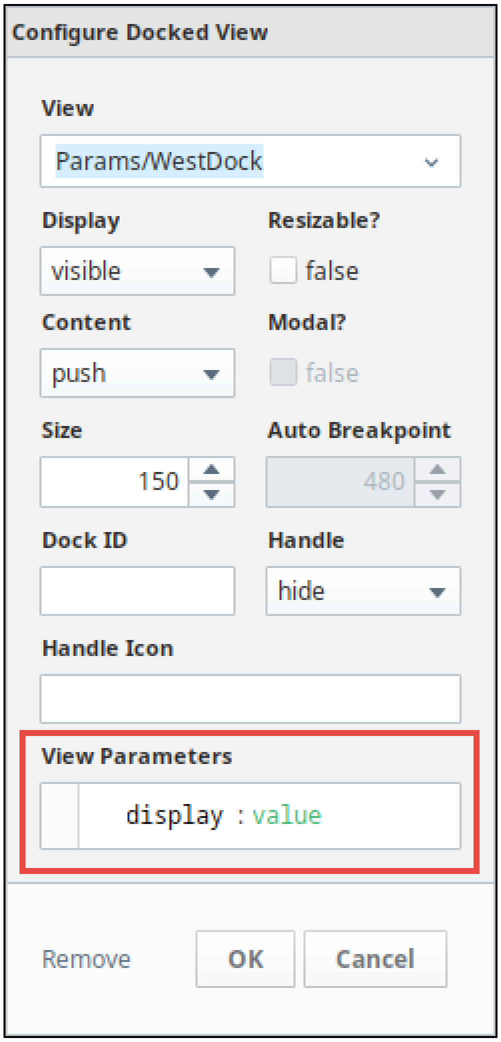


Params

Params are a category of properties that are used when passing parameters from one view to another view. The properties inside of the params category define parameters that can be passed in and out of a view. It's through the use of parameters that views interchange information with other entities such as a docked view, embedded view, or a page. To learn more about using parameters to pass properties across views, refer to Property Bindings in Perspective.

Docked View

When configuring a page, it's possible to pass a value to a docked view. When you click on a docked view, you can specifiy one or more param properties in the View Parameters field of the view that is docked. If you have any param category properties defined on that view, this interface allows you to pass a value when you navigate to the configured page. To learn more, go to Configure Docked View Parameters.



Configure Docked View

View: Params/WestDock

Display: visible

Resizable? □ false

Content: push

Modal? □ false

Size: 150

Auto Breakpoint: 480

Dock ID:

Handle: hide

Handle Icon:

View Parameters

```
display : value
```

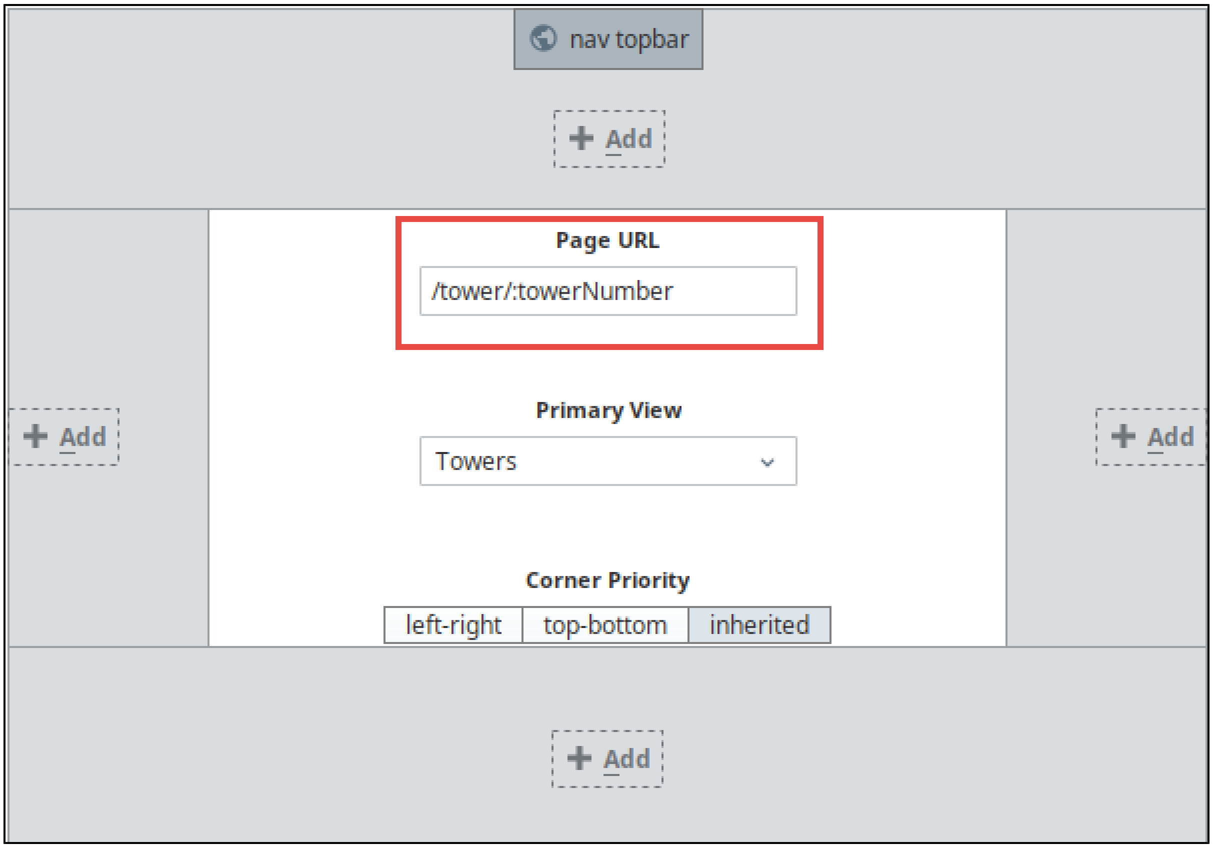
Embedded View

The embedded view component allows you to embed one view inside of another. The only way to pass a property across views is by passing a view parameter into an embedded view. Parameters can be defined as input, output, or both input/ouput. Once you have param properties configured on your views, the embedded view component provides you with access to these parameters. You have options on how to pass a property into an embedded view. How to set up passing a parameter depends on how you design your project. You can set up passing a property to an embedded view using a parameter with property bindings, Tag bindings, or even scripts. For more information on embedded views, refer to Pass a Property into an Embedded View Using a View Parameter and View Properties.

Page

Passing parameters to a Page essentially means passing a URL parameter to a page. Parameters are used to allow a page to be mounted at a dynamic URL, allowing information in the URL to be interpreted as input parameters to the page's primary view . A primary view can see whatever parameters are passed in, and components inside the view can bind or use the values in some useful way.

The way you add parameters to the Page URL is by mounting the page at a special URL using a parameter replacement syntax like so: <page> **/: towerNumber** . The dynamic URL mounting uses a colon to signify that a portion of the URL is meant to by dynamic and map to an input parameter on the page's primary view. To learn more, go to Passing parameters (URL Parameters).



Page URL

/tower/:towerNumber

Primary View

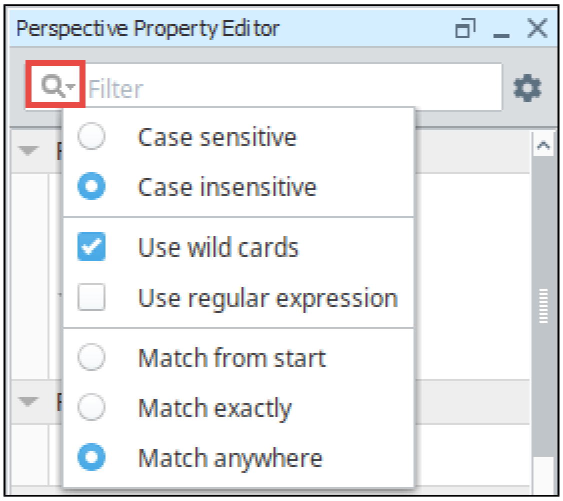
Towers

Corner Priority

left-right | top-bottom | inherited

Search Filter

In the Property Editor search bar, you can search for component properties. Start typing in the search bar and the property list updates based on the text you enter. You can also set filter options to make the search case sensitive, or use wildcards, regular expressions, match from start, match exactly, and match anywhere.



○ Case sensitive

● Case insensitive

☑ Use wild cards

☐ Use regular expression

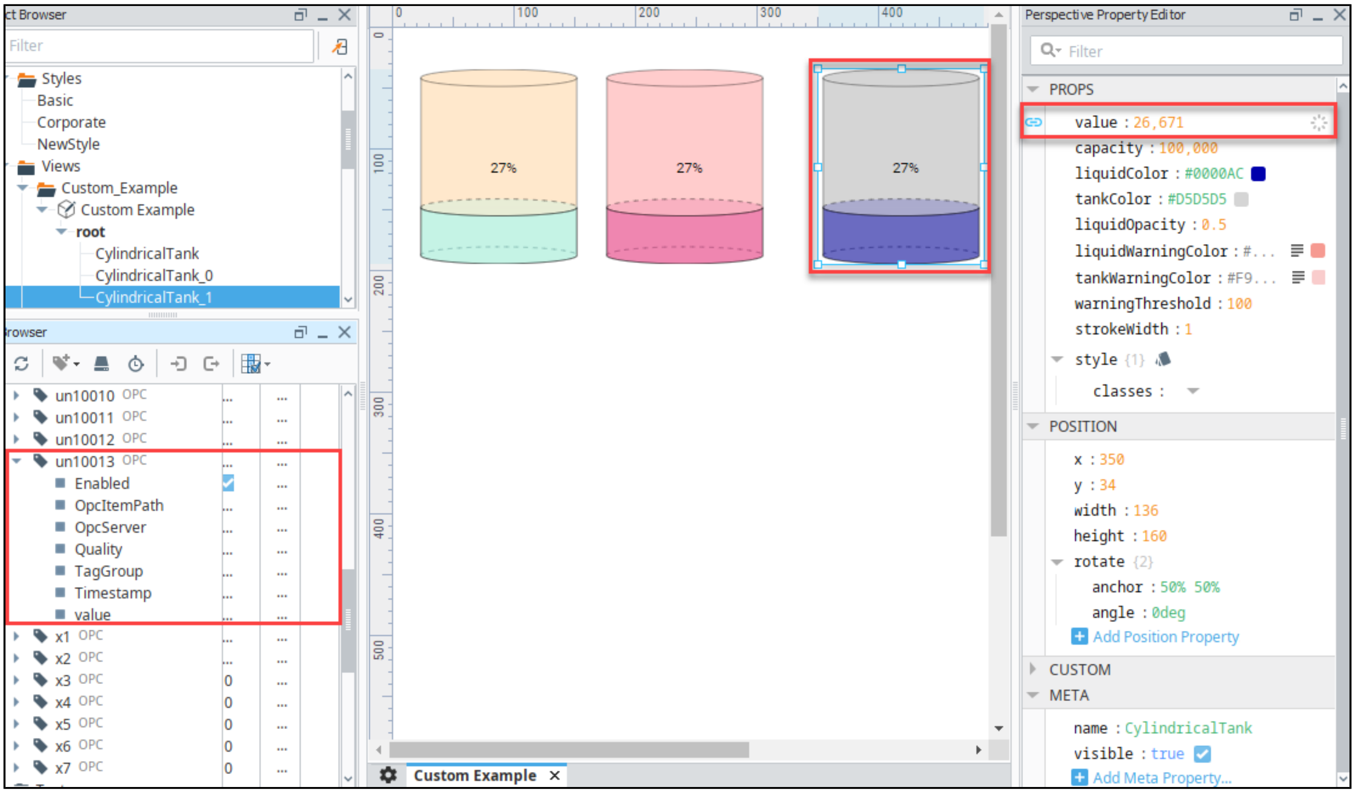
○ Match from start

○ Match exactly

● Match anywhere

Bindings

Most properties have binding options. For more information on bindings, see Types of Bindings in Perspective. The image below shows the properties in the Property Editor that are set on a Cylindrical Tank component. There are **Binding** 🔗 icons to the left of each of the properties that appear when you mouse over them. If you have a Cylindrical Tank on your view and click the Binding icon for the **value** property, it will open the Binding window and you can set what the Cylindrical Tank component is bound to. In this example, the **value** property of the Tank is bound to the un10013 OPC Tag.



value : 26,671
capacity : 100,000
liquidColor : #0000AC
tankColor : #D5D5D5
liquidOpacity : 0.5
liquidWarningColor : #...
tankWarningColor : #F9...
warningThreshold : 100
strokeWidth : 1
style {1}
classes :

x : 350
y : 34
width : 136
height : 160
rotate {2}
anchor : 50% 50%
angle : 0deg

name : CylindricalTank
visible : true

Styles

A full menu of style options is also available for text, background, margin and padding, border, shape and other miscellaneous settings to adjust the appearance of your components. Style classes enable you to set up a particular look and feel (fonts, colors, borders, etc.) once and then apply it to multiple components.

Right-Click Menu

Right-clicking in the Perspective Property Editor brings up a number of items. These items are described below by section.

INDUCTIVE UNIVERSITY

Persistent Properties

Watch the Video

Actions

The Actions items provides a means to modify properties on a component such as deleting a property, or inserting a new property. Common utilities (i.e., copy, paste, duplicate) are available.

Structure

- **Add Before** : Allows you to insert a new property into the hierarchy before an existing one. It is only visible when right-clicking on an element in either an Array or Object.
- **Add After** : Allows you to insert a new property into the hierarchy after an existing one. It is only visible when right-clicking on an element in either an Array or Object.
- **Insert** : Allows you to insert a new property into the hierarchy. The "Add Before" and "Add After" Structure options in the Perspective Property Editor context menu are two unique items that are only visible when right-clicking on an element in either an Array or Object.
- **Value** : Allows you to change the type of the selected property to a Value, Array, or Object

Binding

This feature was changed in Ignition version **8.1.26**:

The Binding section of items allows you to configure or manipulate existing bindings on a property. If any of the following are not current available options, they will not be visible selections. For example, if the clip board does not contain a binding configuration, the menu won't display a Paste Binding option.

- **Configure Binding...**: Opens the binding configuration window, allowing you to add a binding to the clicked property.
- **Copy Binding**: Copies the binding configuration from the clicked property.
- **Paste Binding**: If a binding configuration was previously copied via **Copy Binding**, this option will paste the binding configuration on the clicked property.
- **Disable Binding**: Disables the binding on the clicked property.
- **Remove Binding**: Removes the binding configuration from the clicked property.

Options

- **Add Change Script**: Enables you to add a script that will be called whenever the value of the property changes. For more information, see Property Change Scripts.
- **Persistent** : Determines if the property is Persistent or Transient .
- **Access:** Allows you to restrict property access.

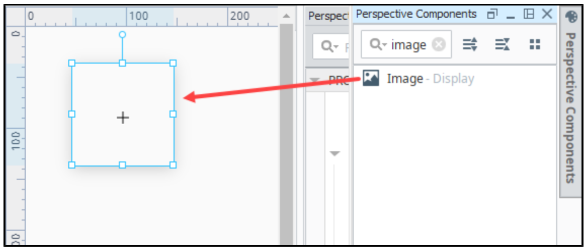
Images and Icons in Perspective

Perspective supports the import and display of images, including PNGs, JPGs, JPEGs, GIFs, and SVGs. Images can be loaded into Perspective via the Image Management utility, dragged and dropped directly into a View, or derived from an external web address. Regardless of approach, the Image component is used to display desired images, either as linked images with associated URLs, or as embedded images stored within the View. Each of these approaches is detailed below.

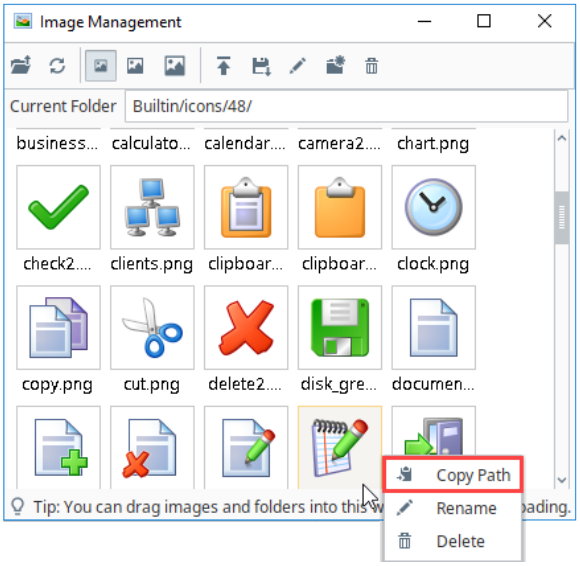
Using the Image Management Utility

The following example walks through bringing an image from the Image Management tool using an image component. For more information on the Image component and all of its properties, see Perspective - Image .

1. Drag an **Image** component from the component palette onto the container.



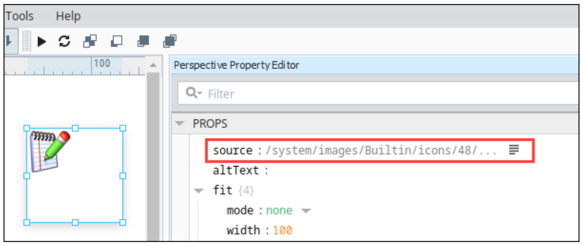
2. At the top of the Designer in the Menu Bar, select **Tools > Image Management.**
3. In the **Image Management tool**, look through the different folders for an image file you want.
4. Right-click on the image and select **Copy Path**.



Current Folder: Builtin/icons/48/

Tip: You can drag images and folders into this w ading.

5. In the Property Editor, paste the image path into the source property for the Image component. You need to add "/system/images" to the beginning of the path, to properly reference images stored by the Image Management tool.



```
source : /system/images/Builtin/icons/48/...
altText :
fit (4)
  mode : none
  width : 100
```

On this page ...

INDUCTIVE UNIVERSITY

Images in Perspective

Watch the Video

The image now appears inside the Image component on the container.

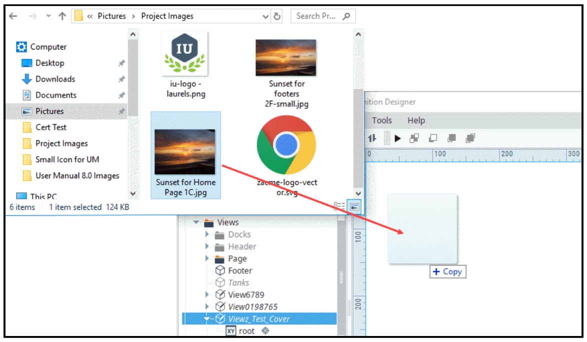
For more information on the Image component and all of its properties, see Perspective - Image.

Drag and Drop an Image onto a View

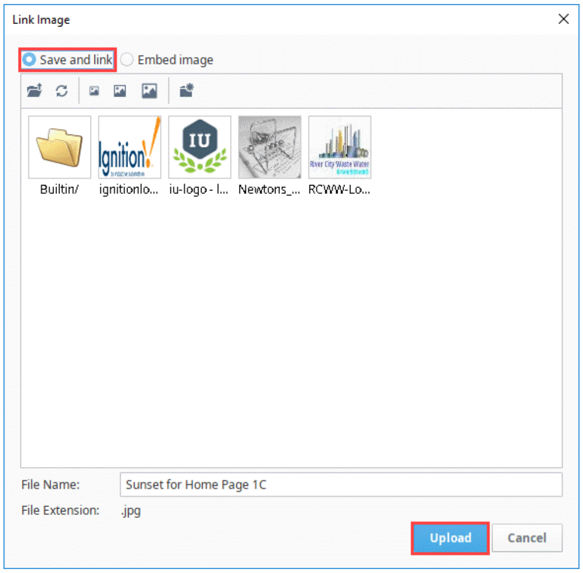
In Perspective, you can drag and drop images from your computer onto a view. When an image is dropped, it can either be uploaded into the Image Management tool and then referenced from an Image component, or it can be embedded directly into the view. In this example, we'll pull in a .jpg photo of a sunset to use as a background on a home page for our application.

1. In the Designer, open the view in which you want to put the image.
2. Locate the image on your computer and drag it onto the view.

> **Note:** We already resized this image in a graphics editing program so that it is the size we want.



3. You have the option of saving the image as a link or embedding the image in the view. To save the image as a link, make sure the **Save and Link** radio button is selected and then click **Upload** . This will save the image in the Image Management Tool where it can be used repeatedly.

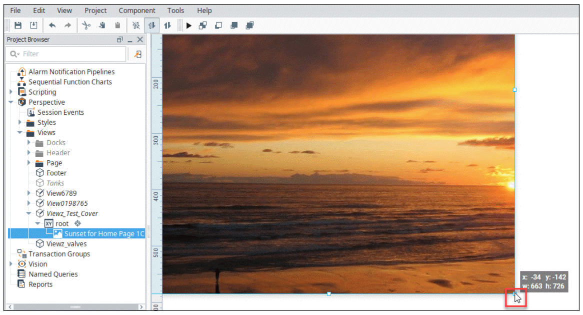


● Save and link ○ Embed image

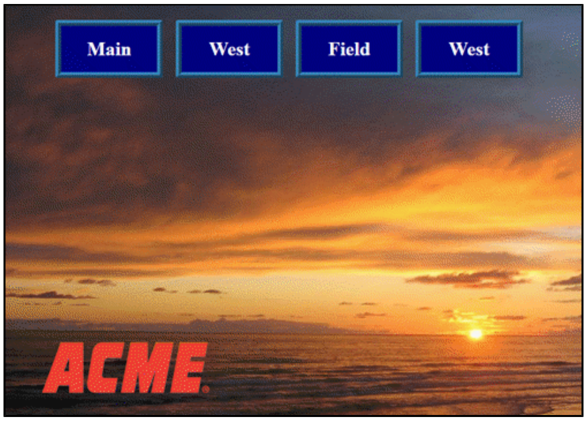
File Name: Sunset for Home Page 1C

File Extension: .jpg

4. The image now appears in the view. Drag one of the corner nodes to make the image component border the same size of the image.



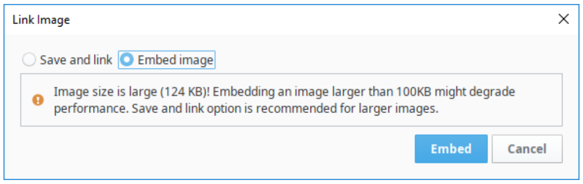
5. Now your image is on the view. Save your project. We added a company logo SVG file as well as some navigation buttons to this home page.



Embedding Images

If you don't want the image to be brought into the Image Management tool you can embed the image instead. Embedded images can degrade performance, but can also allow you to view and modify image elements directly in the Property Editor.

1. Locate the image on your computer and drag it onto the view.
2. After dragging the image onto the view, click the **Embed image** radio button. If the image is larger than 100KB, you will see a warning message in the confirmation popup, since large embedded images can degrade performance.



Link Image

○ Save and link ● Embed image

Image size is large (124 KB)! Embedding an image larger than 100KB might degrade performance. Save and link option is recommended for larger images.

3. Click **Embed** to embed the image.

Using Icons

Icons in Perspective are specially curated SVG (Scalable Vector Graphic) images chosen to be helpful in Perspective design. Perspective includes a starting collection of icons, and provides ways of including your own (see "Use a Custom Icon Repository" below). Icons can be used in Perspective directly using the Icon component, or integrated into the configurations of other components, as with the Horizontal Menu, Map, and Accordion components.

INDUCTIVE UNIVERSITY

Icons

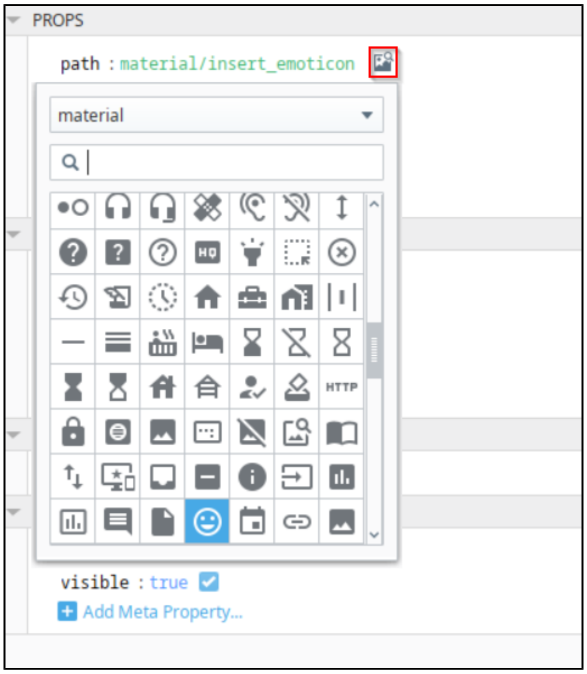
Watch the Video

Choosing Icons

The bulk of the icons included with Perspective are derived from the **materials** icon library, which is fully detailed, with included icon names, here (external link). Icons are referenced using icon paths within Perspective. For example, the path **material/location_on** will reference the **location_on** material design icon.

The following feature is new in Ignition version **8.1.6**
Click here to check out the other new features

All icon properties (via the Icon Component or other applicable components) include an interface for selecting icons from configured libraries, both built-in and custom. On the right-hand side of the icon path, the icon can be used to browse and search for available icons. The icons will be arranged into headings based on the originating icon files on the Gateway.



Use a Custom Icon Repository

The materials icon repository is a great source for icons, but if you have a custom library of icons, you can set it up to be accessible by Ignition. All icon repositories are stored as SVG files within the Gateway's Installation Directory. A typical path to an icon file will look like:

Windows

```
C:\Program Files\Inductive Automation\Ignition\data\modules\com.inductiveautomation.
perspective\icons\<repository name>.svg
```

MacOS

```
/usr/local/ignition/data/modules/com.inductiveautomation.perspective/icons/<repository name>.svg
```

Linux

```
/usr/local/bin/ignition/data/modules/com.inductiveautomation.perspective/icons/<repository name>.svg
```

You can create custom libraries by adding your own SVG file at this path. The properties inside the file are defined using XML, which should follow the pattern provided below. Note that each icon is placed in an enclosing SVG tag that defines values for the **id** attribute, which is used to specify the name of each icon, and the **viewbox** attribute, which must have large enough dimensions to fully enclose the graphic.

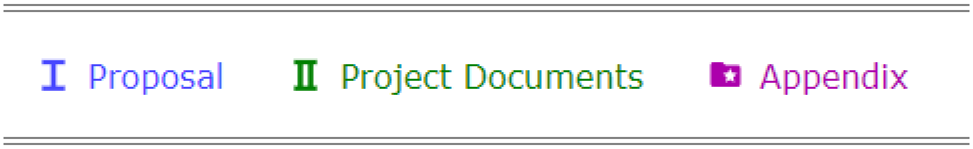
```
<?xml version="1.0" encoding="utf-8"?>

<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
    <!--Red Circle-->
        <svg viewBox="0 0 24 24" id="red-circle">
                <circle cx="12" cy="12" r="6" stroke="black" stroke-width="1" fill="red" />
        </svg>

    <!--Blue Circle-->
        <svg viewBox="0 0 24 24" id="blue-circle">
```

```
                <circle cx="12" cy="12" r="6" stroke="black" stroke-width="1" fill="blue" />
        </svg>
</svg>
```

Once an icon file has been added, a Designer restart is required to pull the new icons into the Designer for development.

Example with Custom Icon Repository

In this example we'll make a custom repository file with icons. The name of the file is `example.svg`. We'll then use the icons in this file and other repositories to create a Horizontal Menu component for Proposal, Project Documents, and Appendix headings.



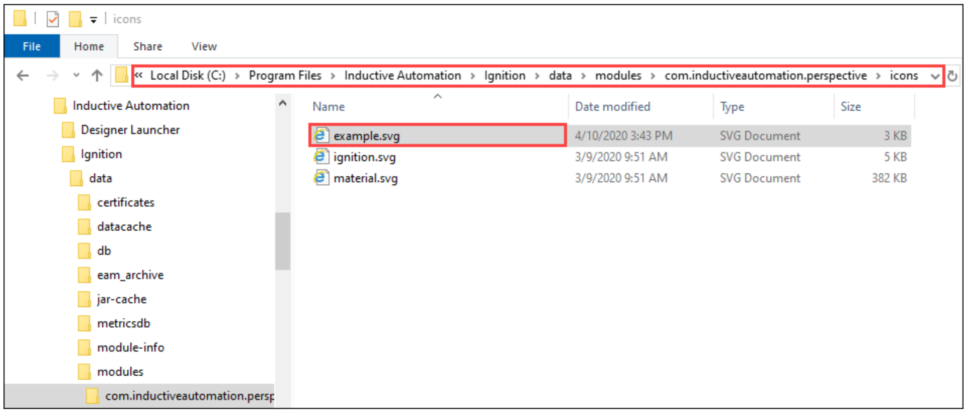
1. First we created the repository file in xml.

```
<?xml version="1.0" encoding="utf-8"?>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">

        <!--a red circle-->
        <svg viewBox="0 0 24 24" id="red-circle">
                <circle cx="12" cy="12" r="6" stroke="black" stroke-width="1" fill="red" />
        </svg>

        <!--a blue circle-->
        <svg viewBox="0 0 24 24" id="blue-circle">
                <circle cx="12" cy="12" r="6" stroke="black" stroke-width="1" fill="blue" />
        </svg>

        <!--a triangle-->
        <svg viewBox="0 0 24 24" id="triangle">
        <path d="M12 0 L3 22 L21 22 Z" />
    </svg>

        <!--roman numeral one icon-->
        <svg viewBox="0 0 24 24">
                <g id="one">
                        <rect
                           id="rect3723" width="15.09025" height="3.0526078" x="4.4548745" y="
2.4385488" />
                        <rect
                           id="rect3723-3" width="15.09025" height="3.0526078" x="4.4548745" y="
19.510113"/>
                        <rect
                           id="rect3755" width="3.4557824" height="16.990929" x="10.272108" y="
3.5045362" />
                </g>
        </svg>
        <!--roman numeral two icon-->
        <svg viewBox="0 0 24 24">
                <g id="two">
                        <rect
                           id="rect3723" width="15.09025" height="3.0526078" x="4.4548745" y="
2.4385488" />
                        <rect
                           id="rect3723-3" width="15.09025" height="3.0526078" x="4.4548745" y="
19.510113" />
                        <rect
                           id="rect3755" width="3.4557824" height="16.990929"  x="6.8435426" y="
3.5045362" />
                        <rect
                           id="rect3755-9" width="3.4557824" height="16.990929" x="13.700686"  y="
```

```
3.5045002"/>
                </g>
        </svg>
</svg>
```

2. Once created, save the file in the following directory (assuming a Windows install and the default choice of install folder):

   `C:Program Files\Inductive Automation\Ignition \data\modules\com.inductiveautomation.perspective\icons`



icons
File
Home
Share
View
« Local Disk (C:) › Program Files › Inductive Automation › Ignition › data › modules › com.inductiveautomation.perspective › icons
Inductive Automation
Designer Launcher
Ignition
data
certificates
datacache
db
eam_archive
jar-cache
metricsdb
module-info
modules
com.inductiveautomation.persp

Name	Date modified	Type	Size
example.svg	4/10/2020 3:43 PM	SVG Document	3 KB
ignition.svg	3/9/2020 9:51 AM	SVG Document	5 KB
material.svg	3/9/2020 9:51 AM	SVG Document	382 KB

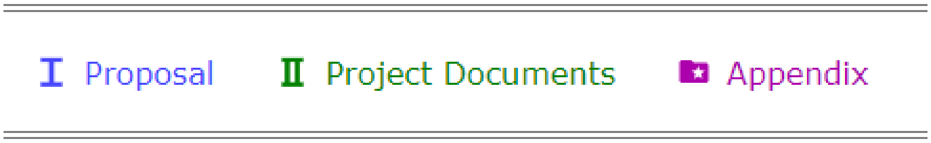
3. Next, we'll need to restart our Designer for the changes to be recognized.
4. Now we can use the icons in this icon repository in our components. For this example, we put a Horizontal Menu component in a Coordinate view.
5. We use two icons from the new example repository and one from the materials repository. We set the properties as follows for the Horizontal Menu component:

Top Level Properties

Property	Value
props.items.0.enabled	true
props.items.0.icon.path	example/one
props.items.0.icon.color	#4747FF
props.items.0.label	Proposal
props.items.0.style.color	#4747FF
props.items.0.style.fontFamily	Verdana
props.items.1.enabled	true
props.items.1.icon	example/two
props.items.1.color	#008000
props.items.1.label	Project Documents
props.items.1.style.color	#008000
props.items.1.style.fontFamily	Verdana
props.items.2.enabled	true
props.items.2.icon.path	material/folder_special
props.items.2.icon.color	#AC00AC
props.items.2.label	Appendix
props.items.2.style.color	#AC00AC
props.items.2.style.fontFamily	Verdana
props.style.borderBottomStyle	double
props.style.borderLeftStyle	none

props.style.borderRightStyle	none
props.style.borderTopStyle	double
props.style.borderTopWidth	4
props.style.borderBottomWidth	4

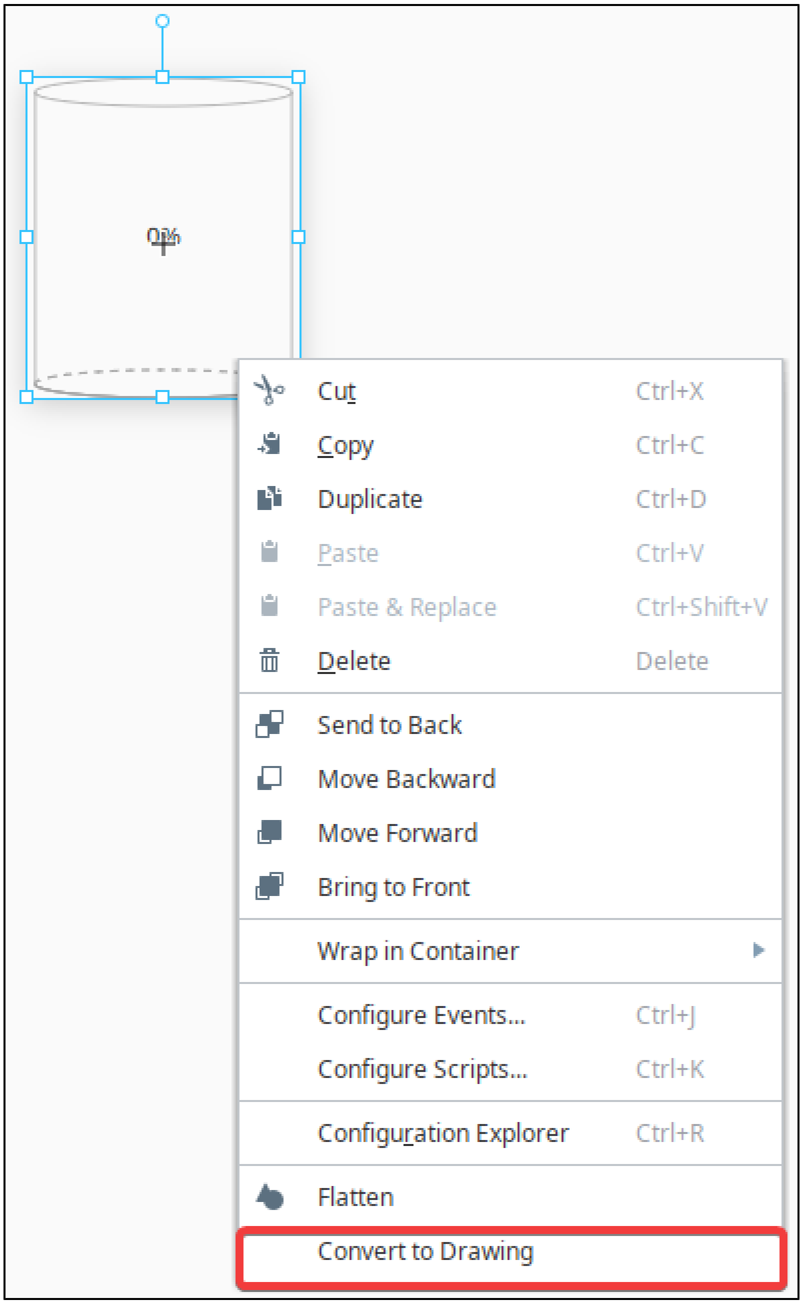
6. When we put the Designer into Preview ▶ mode, the Horizontal Menu appears.



Converting SVG-based Perspective Components to Drawings

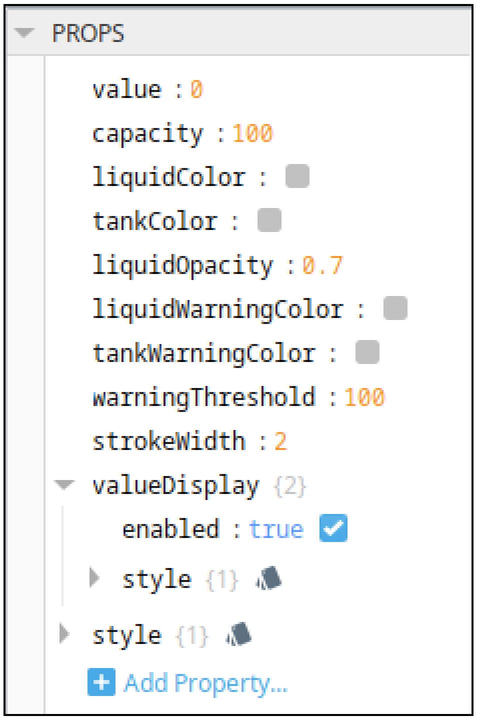
The following feature is new in Ignition version **8.1.22**
Click here to check out the other new features

Right-clicking on a built-in SVG-based Perspective component will give an option to convert the SVG into a drawing.



Converting the component to a drawing will result in the component having different properties. For example, the Cylindrical Tank component props list changes to include properties like viewBox, preserveAspectRatio, and elements instead when converted to a drawing.

Properties as a component:



PROPS

value : 0

capacity : 100

liquidColor :

tankColor :

liquidOpacity : 0.7

liquidWarningColor :

tankWarningColor :

warningThreshold : 100

strokeWidth : 2

valueDisplay {2}

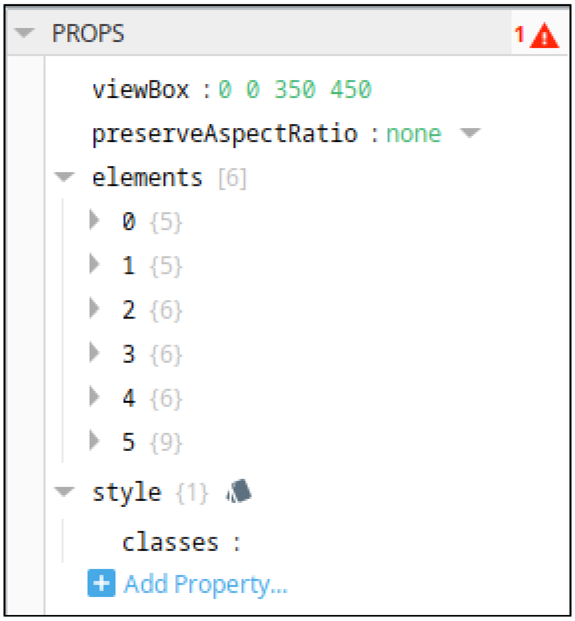
enabled : true ☑

style {1}

style {1}

Add Property...

Properties as a drawing:



PROPS 1 ⚠

viewBox : 0 0 350 450

preserveAspectRatio : none

elements [6]

0 {5}

1 {5}

2 {6}

3 {6}

4 {6}

5 {9}

style {1}

classes :

Add Property...

Related Topics ...

- Image Management Tool
- Symbol Factory

Localization in Perspective

Localization in the Perspective module utilizes terms from Ignition's Translation system. Once terms have been defined, translations can be enabled by the use of the "locale" session property. This page covers using localization in Perspective. For the Vision module, see Localization in Vision.

The following feature is new in Ignition version **8.1.2**
Click here to check out the other new features

Changing the locale of the session can also modify number formatting behavior (e.g., on the Numeric Entry Field component or Power Chart component).

On this page ...

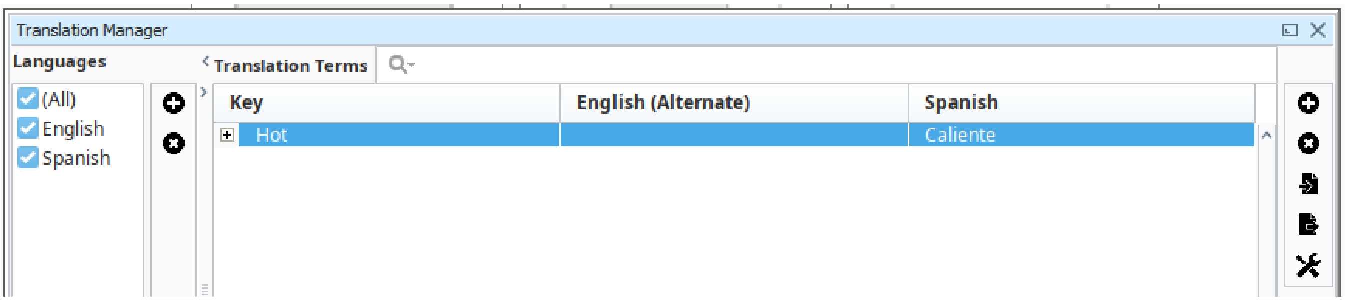
Selecting a Language

Selecting a language for your Perspective session is done through session properties, specifically the locale session property. By default, this property is set to the language tag "en-US" for English USA. Once you set up a additional languages in the Translation system, you can set your locale session property to different values depending on what language you want to see on your screen.

See this link for a full list of valid locale tags: https://www.oracle.com/java/technologies/javase/jdk8-jre8-suported-locales.html#util-text

Switching Languages within a Session

In this example, we'll set up a dropdown list from which the user can choose a language. For this example, we've already set up a Spanish Translation List. We've added a key "Hot" and its Spanish translation, "Caliente".

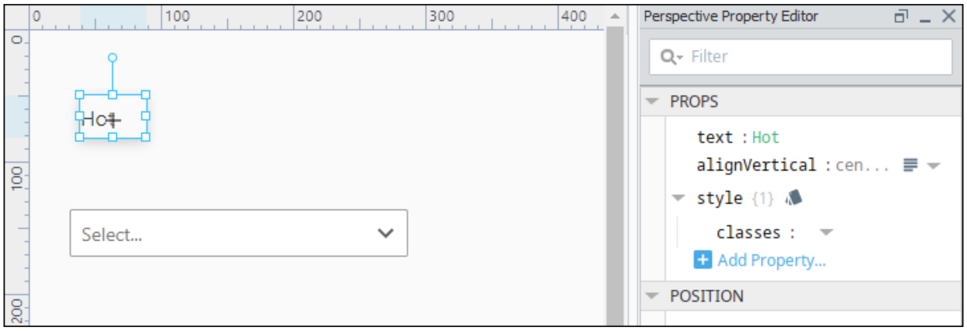


☑ (All)
☑ English
☑ Spanish

Key	English (Alternate)	Spanish
Hot		Caliente

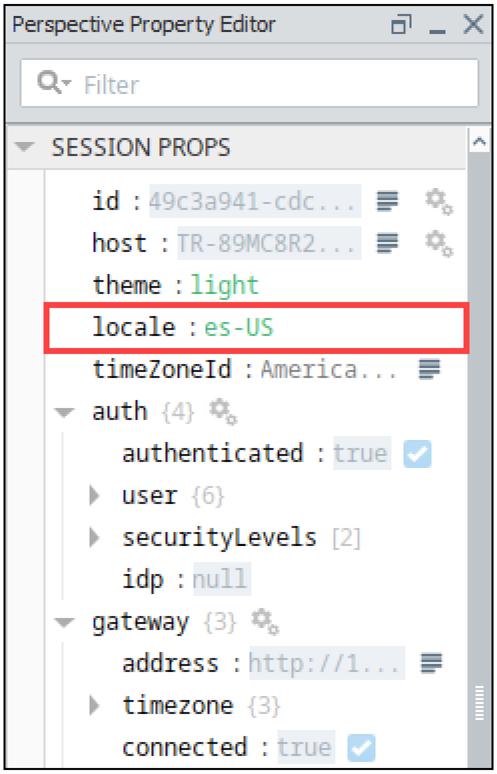
Now we'll set up a view where the user can choose between English and Spanish.

1. Create a new view and put a Label component and a Dropdown component on the view.
2. Set the text property of the label to "Hot."



text : Hot
alignVertical : cen...
style {1}
classes :

3. In the Project Browser, click on Perspective. The Session properties are displayed. Update the locale session property to "es-US" (the language tag for Spanish USA).



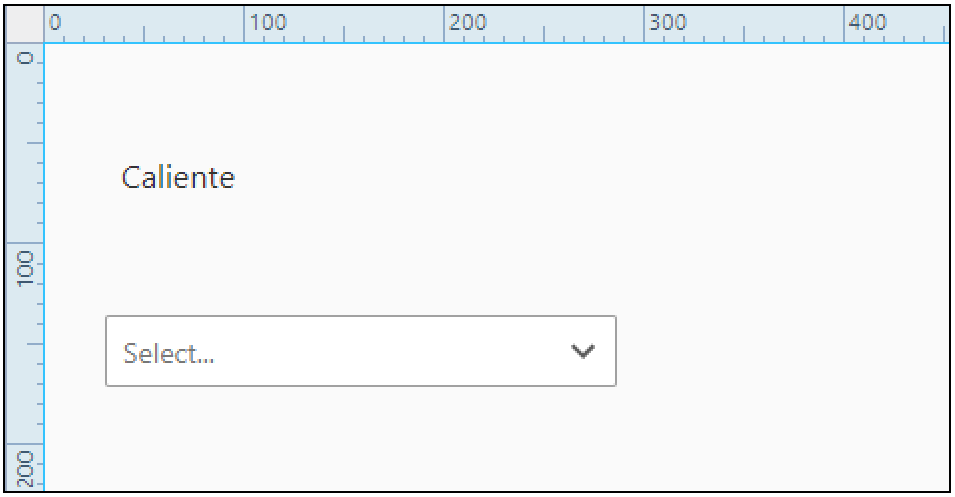
Perspective Property Editor

Filter

SESSION PROPS

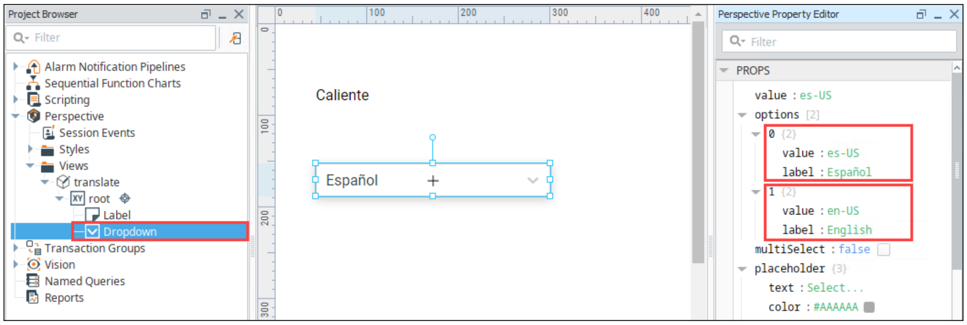
```
id : 49c3a941-cdc...
host : TR-89MC8R2...
theme : light
locale : es-US
timeZoneId : America...
auth {4}
  authenticated : true
  user {6}
  securityLevels [2]
  idp : null
gateway {3}
  address : http://1...
  timezone {3}
  connected : true
```

This results in anything in your session that contains the word "Hot" to be translated to its Spanish translation.



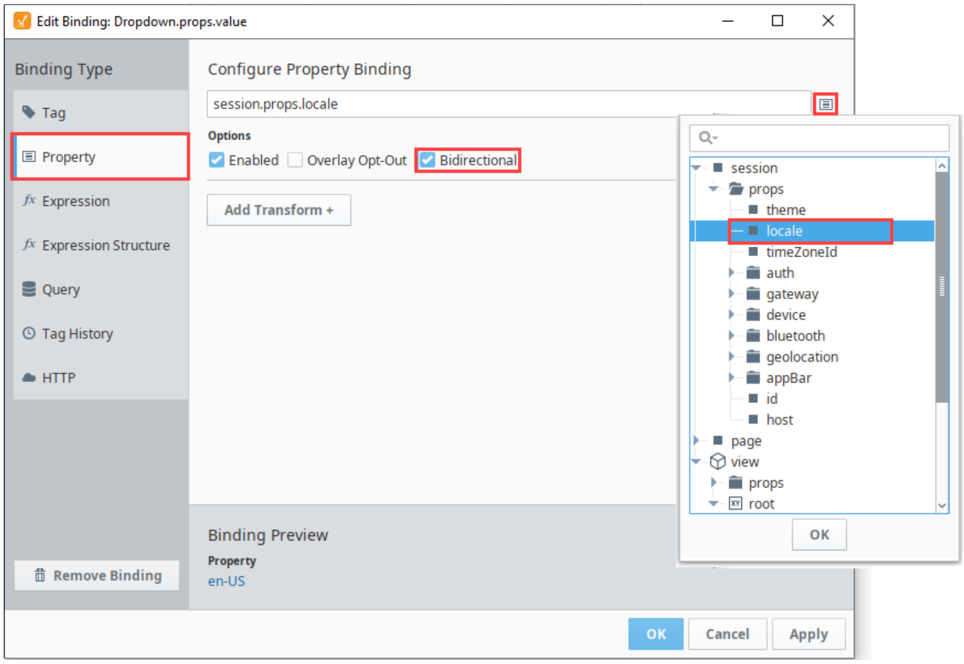
Similarly, if you set the locale session property back to "en-US", Ignition will translate your session text back to English.

4. Next we'll set up the Dropdown component with two options configured as shown below where the Spanish option is tied to the "es-US" language tag and the English option is tied to the "en-US" language tag.
   options.0.value: **es-US**
   options.0.label: **Espanol**
   options.1.value: **en-US**
   options.1.label: **English**



```
value : es-US
options [2]
  0 {2}
    value : es-US
    label : Español
  1 {2}
    value : en-US
    label : English
multiSelect : false
placeholder {3}
  text : Select...
  color : #AAAAAA
```

5. Next we'll bi-directionally bind the Dropdown's Value property to the Locale session property. In the Property editor, click the **Binding** 🔗 icon for the value property.
6. On the Edit Binding screen, select the **Property Binding** type.
7. Next click the **Property Details** ☰ icon. Scroll to the session.props.locale and click **OK**.
8. Select the Bidirectional option. Click **OK** to save the binding.



session.props.locale

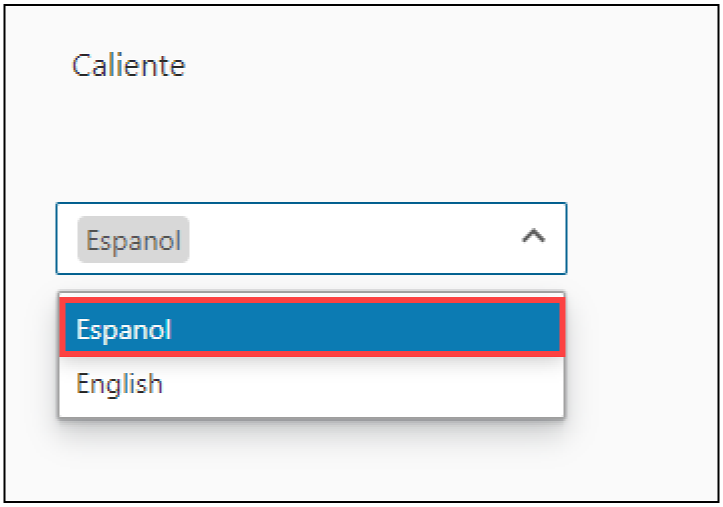
Options

☑ Enabled □ Overlay Opt-Out ☑ Bidirectional

Property

en-US

9. Put the Designer into **Preview** ▶ mode and test the Dropdown component. Since your Dropdown component now drives your Locale session property, you will be able to see your Label value go from its Spanish to English translation as you change the language selection.

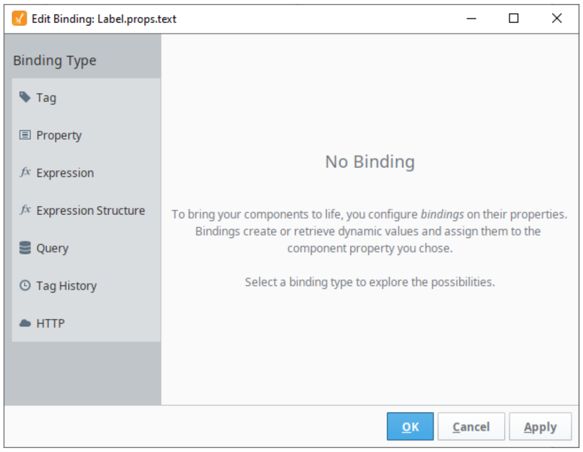


Related Topics ...

- Localization and Languages

Bindings in Perspective

Perspective allows for numerous types of bindings to allow for the dynamic updating of properties associated with Views or their child components.



No Binding

To bring your components to life, you configure *bindings* on their properties. Bindings create or retrieve dynamic values and assign them to the component property you chose.

Select a binding type to explore the possibilities.

On this page ...

When configuring a binding, it is initially unidirectional: the value on the property that contains the binding configuration will synchronize with whatever it is bound to. For example, if the text property on a Label component is bound to a Tag (via a Tag Binding), then the text on the Label will update to match the value of the Tag.

However, if the value of the Text property on the Label changed (say by a script, or someone opening the view in the Designer and manually changing its value), the binding would not cause the value on the Tag to change. However, it's possible to make a binding bidirectional.

Bidirectional Bindings

Tag and property bindings can be made bidirectional simply by checking the **Bidirectional** checkbox in the **Options** section of the **Property Binding** window. Typically this would be done on one of the PROPS properties of an Input component like a multi-state button or a numeric input.

Binding Interface

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.

In addition, bindings can make use of Transforms, which allow a binding to further manipulate the resulting value.

Tag Bindings in Perspective

A Tag binding is a straight-forward binding type. It simply binds a property directly to a Tag Property (typically the value). This sets up a subscription for that Tag. Every time the chosen Tag changes, the binding is evaluated and pushes 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.

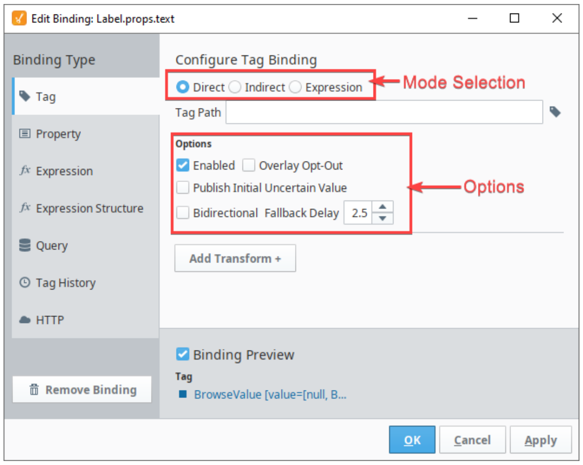
On this page ...

Tag Drop

Watch the Video

Tag Binding Interface

There are three different types of tag bindings in Perspective (direct, indirect, and expression), but they all share some common interface elements.



Configure Tag Binding

● Direct ○ Indirect ○ Expression

Tag Path

Options

☑ Enabled □ Overlay Opt-Out

□ Publish Initial Uncertain Value

□ Bidirectional Fallback Delay 2.5

☑ Binding Preview

Tag

BrowseValue [value=[null, B...

Mode Selection

Allows the binding to switch between the different tag binding modes, which are described further down this page.

- Direct Tag Binding Mode
- Indirect Tag Binding Mode
- Expression Binding Mode

Options

The following options are available, regardless of mode:

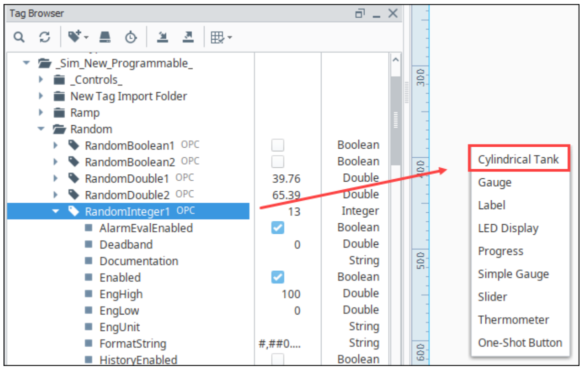
Option	Description
Enabled	Determines if the binding is enabled. Unchecking this option will prevent the binding from executing. Useful in cases where the binding needs to be temporarily disabled.
Overlay Opt-Out	If the target of the binding returns a non-good quality code, this setting indicates whether the component should reflect the quality code with a component overlay.
Publish Initial Uncertain Value	The following feature is new in Ignition version **8.1.8** Click here to check out the other new features Which switching to the view where this binding is located, the binding is briefly provided an initial value of "null" with a quality code of `Uncertain_InitialValue`. Because this quality code is not good, the component will show a quality overlay. Once the most recent value of the target tag has been received, the binding will replace the null value with the newly received value, and remove the quality overlay. If checked, this setting will cause the binding to ignore this initial null value. This prevents the quality overlay from appearing due to the initial null value. All other overlays will function normally.
Bidirectional	Determines if this binding should be allowed to write back to the target. Defaults to unchecked, meaning that the binding will only ever read the target's value. If checked, then changes made to the bound property will cause the binding to send a write request to the target of the binding.
Fallback	When a value is manually written to the bound property, a write request will be sent to the target

Delay	(assuming **Bidirectional** is checked). If the binding doesn't receive confirmation that the write request was accepted, the value on the bound property will instead fallback to the value on the target. This option determines the number of seconds until the binding value falls back.

Drag and Drop

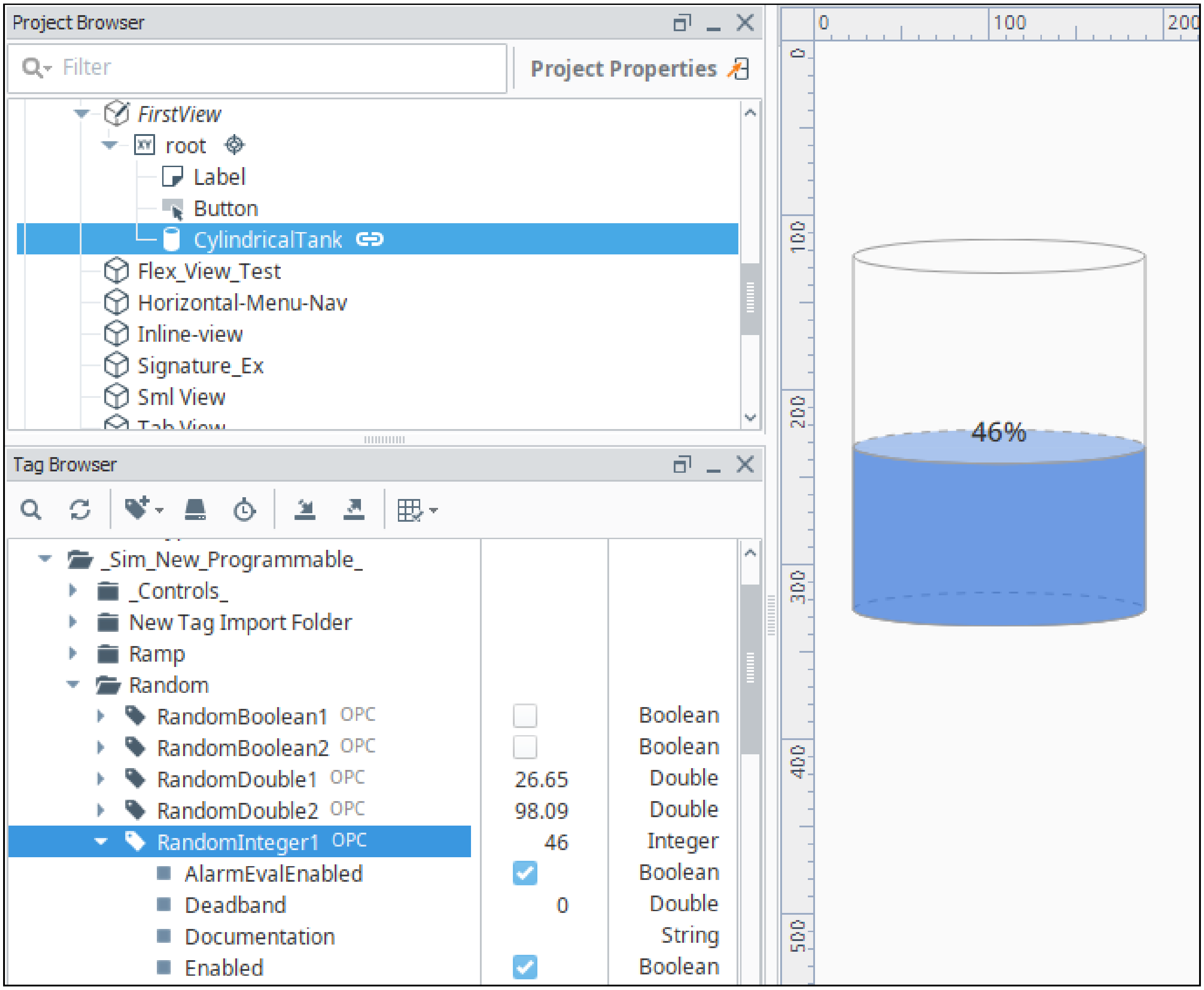
Ignition automatically creates the Tag bindings to several of the component properties when you choose to bind a Tag to a component by dragging and dropping. This is true for both creating a component by dragging and dropping a Tag onto empty space in a container, and by dragging and dropping a dropping a Tag directly onto a component property.

You can drag a Tag onto a container. Perspective will give you an option for the kind of component you'd like to use to represent the Tag. Configuration options for Tag drop are set in the Project properties. For more information, see Tag dropConfig. In this example we drag the Tag for a tank onto a view and choose the Cylindrical Tank option.



_Sim_New_Programmable_		
Controls		
New Tag Import Folder		
Ramp		
Random		
RandomBoolean1 OPC	□	Boolean
RandomBoolean2 OPC	□	Boolean
RandomDouble1 OPC	39.76	Double
RandomDouble2 OPC	65.39	Double
RandomInteger1 OPC	13	Integer
AlarmEvalEnabled	☑	Boolean
Deadband	0	Double
Documentation		String
Enabled	☑	Boolean
EngHigh	100	Double
EngLow	0	Double
EngUnit		String
FormatString	#,##0....	String
HistoryEnabled	□	Boolean

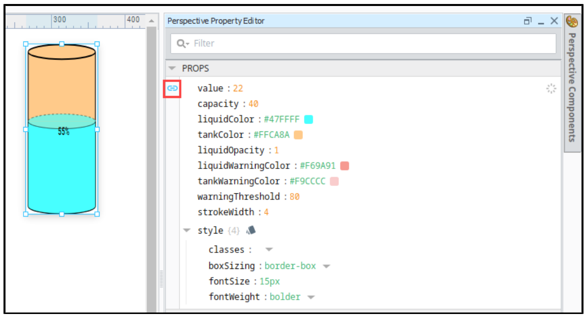
A Cylindrical Tank component is placed on the view. Notice that the value for the component and the value for the Tag match. The component's displayed value will update as the Tag updates.



RandomBoolean1 OPC	□	Boolean
RandomBoolean2 OPC	□	Boolean
RandomDouble1 OPC	26.65	Double
RandomDouble2 OPC	98.09	Double
RandomInteger1 OPC	46	Integer
AlarmEvalEnabled	☑	Boolean
Deadband	0	Double
Documentation		String
Enabled	☑	Boolean

Direct Tag Binding Mode

A direct binding binds a component property to a Tag path. Every time the Tag's value changes, the binding is evaluated and the new value is sent to the bound property. In the example below, the value of the Active_Tank Tag is displayed as a percentage of the capacity of the Cylindrical Tank component.

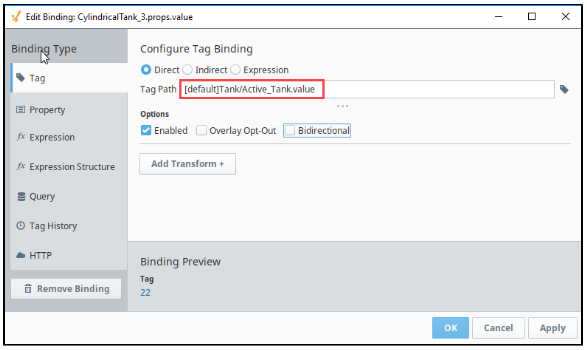


```
value : 22
capacity : 40
liquidColor : #47FFFF
tankColor : #FFCA8A
liquidOpacity : 1
liquidWarningColor : #F69A91
tankWarningColor : #F9CCCC
warningThreshold : 80
strokeWidth : 4
style {4}
  classes :
  boxSizing : border-box
  fontSize : 15px
  fontWeight : bolder
```

IU INDUCTIVE
UNIVERSITY

Tag Binding

Watch the Video



Edit Binding: CylindricalTank_3.props.value

Binding Type
- Tag
- Property
- Expression
- Expression Structure
- Query
- Tag History
- HTTP

Remove Binding

Configure Tag Binding

● Direct ○ Indirect ○ Expression

Tag Path: [default]Tank/Active_Tank.value

Options

☑ Enabled ☐ Overlay Opt-Out ☐ Bidirectional

Add Transform +

Binding Preview

Tag

22

OK Cancel Apply

Indirect Tag Binding Mode

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 session. These parameters are numbered starting at one, and denoted by braces, such as {intParameter}. 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.

For example, instead of binding directly to the FlowRate Tag inside the Valve4 folder, as shown below, you can build an indirect Tag path that will point to the FlowRate Tag for all valves between Valve0 to ValveN where N is any valve number.

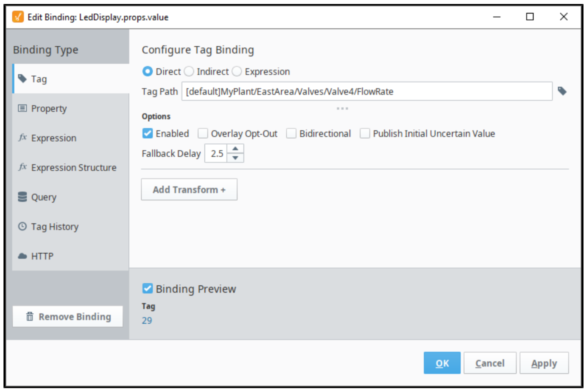
Tag Path

```
[default]MyPlant/EastArea/Valves/Valve4/FlowRate
```

INDUCTIVE UNIVERSITY

Indirect Tag Binding

Watch the Video



Edit Binding: LedDisplay.props.value

Binding Type
- Tag
- Property
- Expression
- Expression Structure
- Query
- Tag History
- HTTP

Remove Binding

Configure Tag Binding

● Direct ○ Indirect ○ Expression

Tag Path: [default]MyPlant/EastArea/Valves/Valve4/FlowRate

Options

☑ Enabled ☐ Overlay Opt-Out ☐ Bidirectional ☐ Publish Initial Uncertain Value

Fallback Delay: 2.5

Add Transform +

☑ Binding Preview

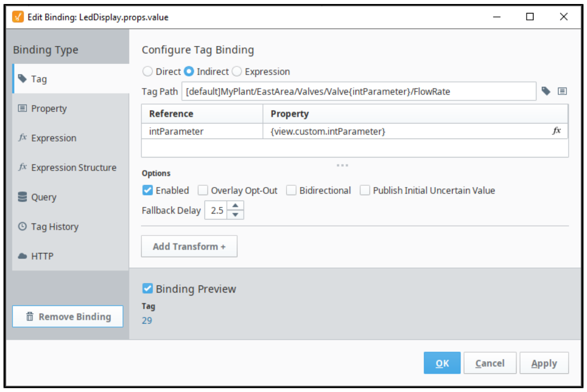
Tag

29

OK Cancel Apply

Below, the valve number, 4, is replaced by {intParameter} where {intParameter} is a placeholder for a dynamic reference that will be used to create a single Tag path capable of referencing every FlowRate Tag inside every existing Valve.

```
[default]MyPlant/EastArea/Valves/Valve{intParameter}/FlowRate
```



Edit Binding: LedDisplay.props.value

Configure Tag Binding

○ Direct ● Indirect ○ Expression

Tag Path [default]MyPlant/EastArea/Valves/Valve{intParameter}/FlowRate

Reference	Property
intParameter	{view.custom.intParameter}

Options

☑ Enabled □ Overlay Opt-Out □ Bidirectional □ Publish Initial Uncertain Value

Fallback Delay 2.5

☑ Binding Preview

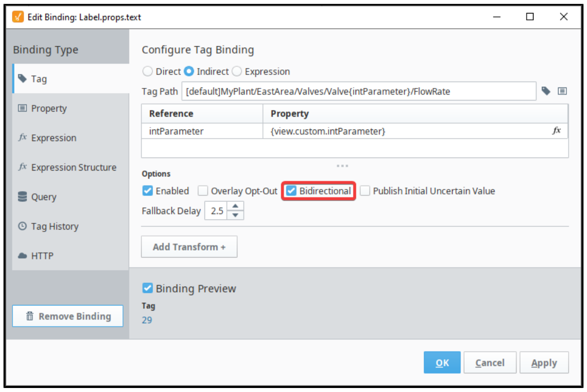
Tag

29

The {intParameter} dynamic **Reference** must be pointed to a **Property** or **View Parameter**. Your property can pointed to any component property value or Tag value to build an Indirect Tag Binding. In this example, the {intParameter} dynamic reference is pointing to a view custom property named valveNumber that is used to help users control which Valve the binding is reading a FlowRate from.

Indirect Tag Binding - Bidirectional

Indirect Tag Bindings can also be made Bidirectional by clicking the **Bidirectional** checkbox on the Edit Binding screen. This will allow any input from a user on that property to be written back to the Tag. To work properly, the Tag needs to have the proper security to accept writes.



Edit Binding: Label.props.text

Configure Tag Binding

○ Direct ● Indirect ○ Expression

Tag Path [default]MyPlant/EastArea/Valves/Valve{intParameter}/FlowRate

Reference	Property
intParameter	{view.custom.intParameter}

Options

☑ Enabled □ Overlay Opt-Out ☑ Bidirectional □ Publish Initial Uncertain Value

Fallback Delay 2.5

☑ Binding Preview

Tag

29

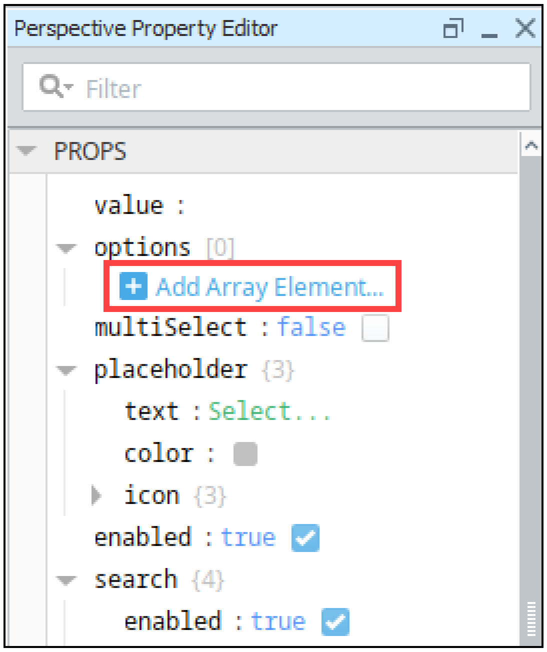
Indirect Tag Binding Example

In this example, we have some different motors, where each motor is a folder of Tags. Each motor has an amps Tag that is within the folder, so that our Tag paths look like the following:

```
Motors/Motor 1/amps

Motors/Motor 2/amps

Motors/Motor 3/amps

Motors/Motor 4/amps
```

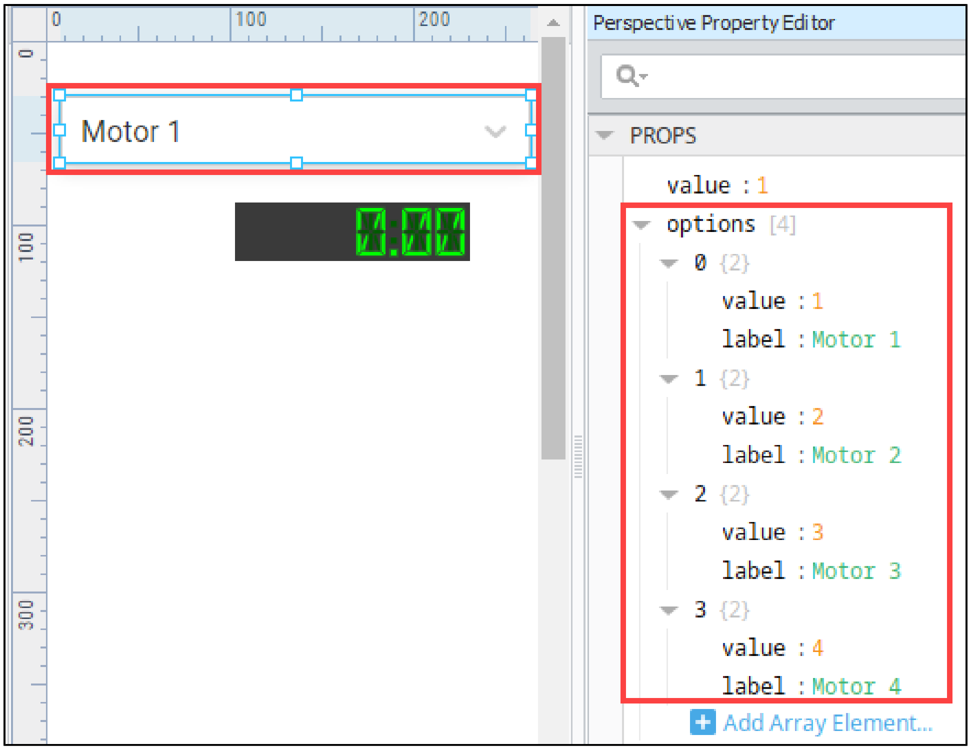
Instead of creating four different displays for these four different Tags, we can create a single display and make it indirect. We need two things for this example: A component to display the value in, and a component which allows the user to select which motor they are looking at.

1. Drag an LED Display component onto the view. This will be the display component.
2. Drag a Dropdown component onto the view. This will be used to allow the user to choose what motor the LED Display is showing amps for.
3. Select the Dropdown component. In the Property editor, under the options property, click **Add Array Element..**.



```
PROPS
  value :
  options [0]
  multiSelect : false
  placeholder {3}
    text : Select...
    color :
    icon {3}
  enabled : true
  search {4}
    enabled : true
```

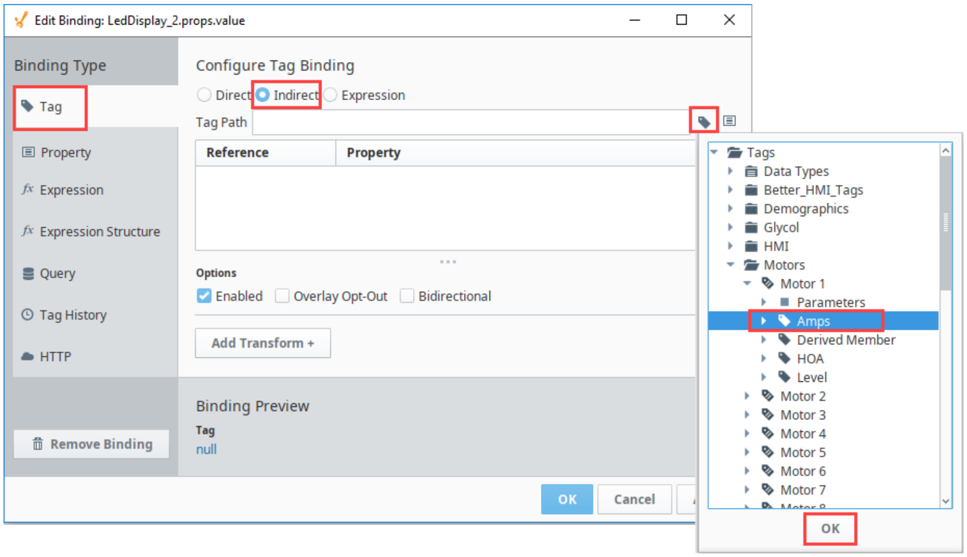
4. Set the value to 1 and the label to Motor 1.
5. Repeat steps 3 and 4 to add array elements for Motors 2, 3, and 4.



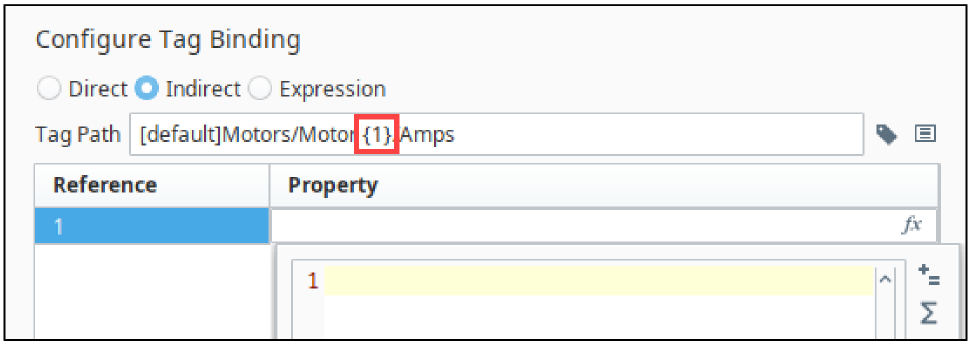
```
PROPS
  value : 1
  options [4]
    0 {2}
      value : 1
      label : Motor 1
    1 {2}
      value : 2
      label : Motor 2
    2 {2}
      value : 3
      label : Motor 3
    3 {2}
      value : 4
      label : Motor 4
```

6. Click on the LED Display component. Select the **Binding** 🔗 icon in the value property.
7. Click on the Tag binding type then click the **Indirect** button.

8. Click the **Tag** 🏷 icon and scroll down to the Motors/Motor 1/Amps Tag. Click **OK**.



9. In the Tag Path field, replace the 1 with {1}. We are replacing "[default]Motors/Motor 1/Amps" with "[default]Motors/Motor {1}/Amps" maintaining the space found between "Motor" and "1" in "Motor" and "{1}".

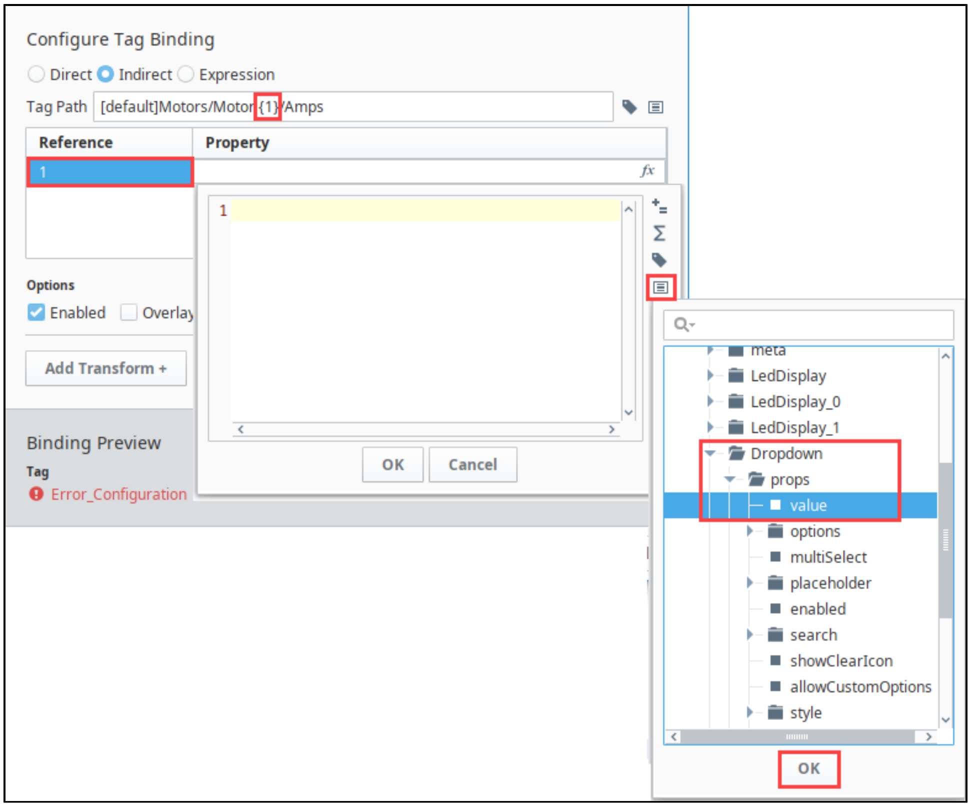

Configure Tag Binding

○ Direct ● Indirect ○ Expression

Tag Path [default]Motors/Motor {1} Amps

Reference	Property
1	

10. In the References list, select the row. Click the **Properties** ☰ icon.
11. Scroll to the Dropdown component and select the **value** property of the Dropdown. Click **OK**.



Configure Tag Binding

○ Direct ● Indirect ○ Expression

Tag Path [default]Motors/Motor{1}/Amps

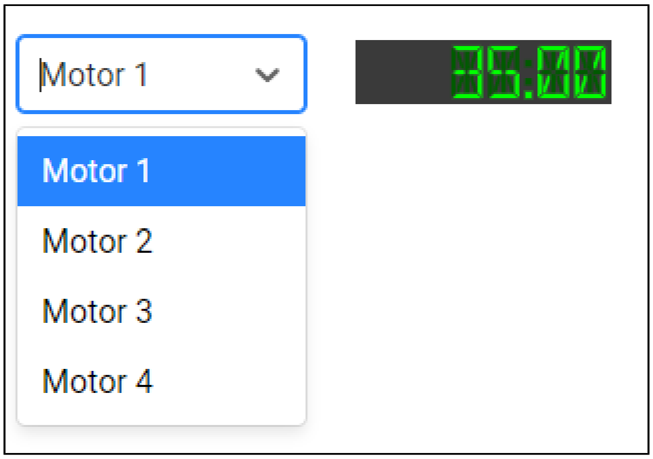
Reference	Property
1	

Options

☑ Enabled □ Overlay

What we have done is configured the Dropdown component's **value** property to be inserted into our indirect Tag path in place of "{1}". If we select "Motor 1" from the Dropdown component, its **value** property then becomes 1. The number 1 then takes the place of "{1}" in the indirect Tag path making it "[default]Motors/Motor 1/Amps". Similarly, selecting "Motor 2" from the Dropdown component makes its **value** property be 2. The number 2 then takes the place of "{1}" in the indirect Tag path making it "[default]Motors/Motor 2/Amps".

12. Click **OK** to save the binding.
13. Put the Designer into **Preview mode** ▶ to see the components and the indirect in action.
14. Select a motor in the Dropdown component. The value in the LED Display component will change depending on the Motor that is selected in the Dropdown list.



Tag Expression Binding

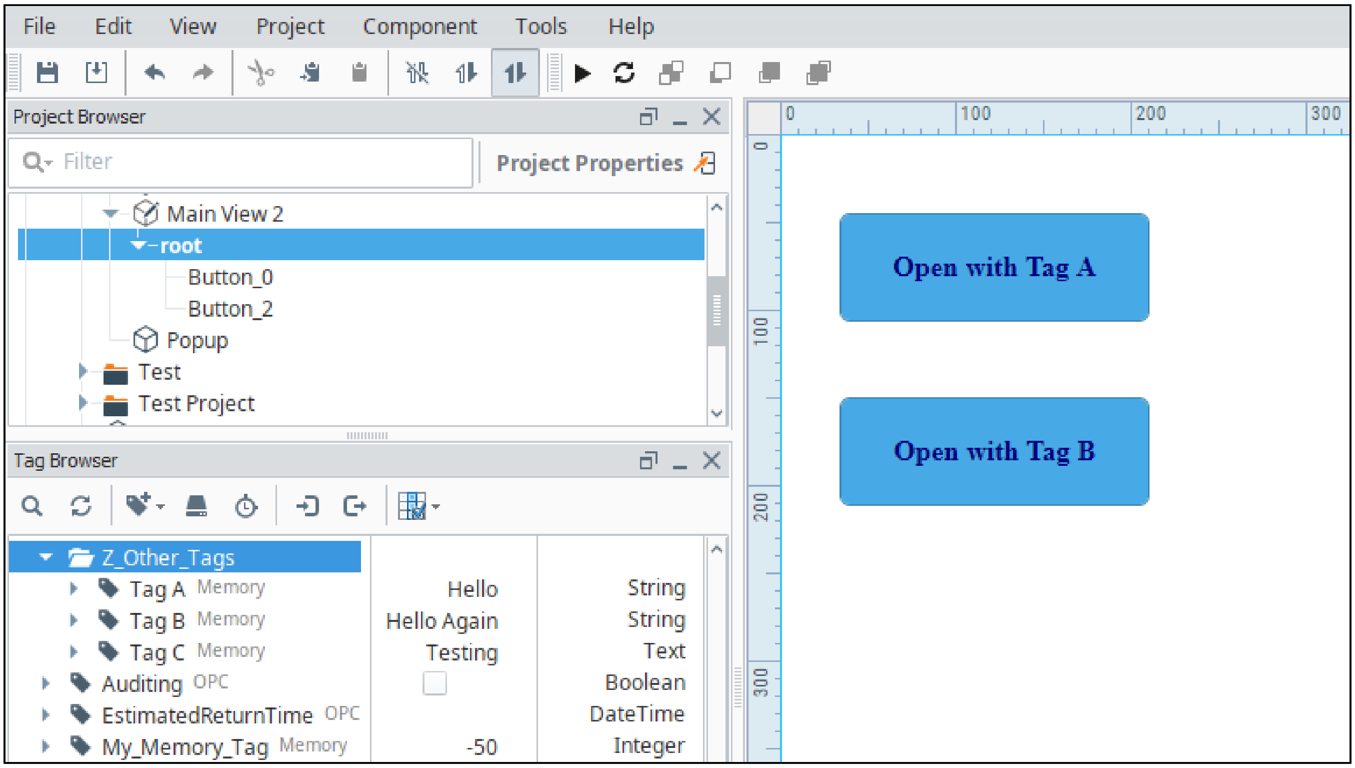
The Tag Expression binding uses the Expression language to specify an entire Tag path. This mode allows the bound property (Tag) to be bidirectional. The Tag path in the Expression is expected to be a string. Note that is different and not to be confused with an Expression Binding.

Tag Binding - Expression

Watch the Video

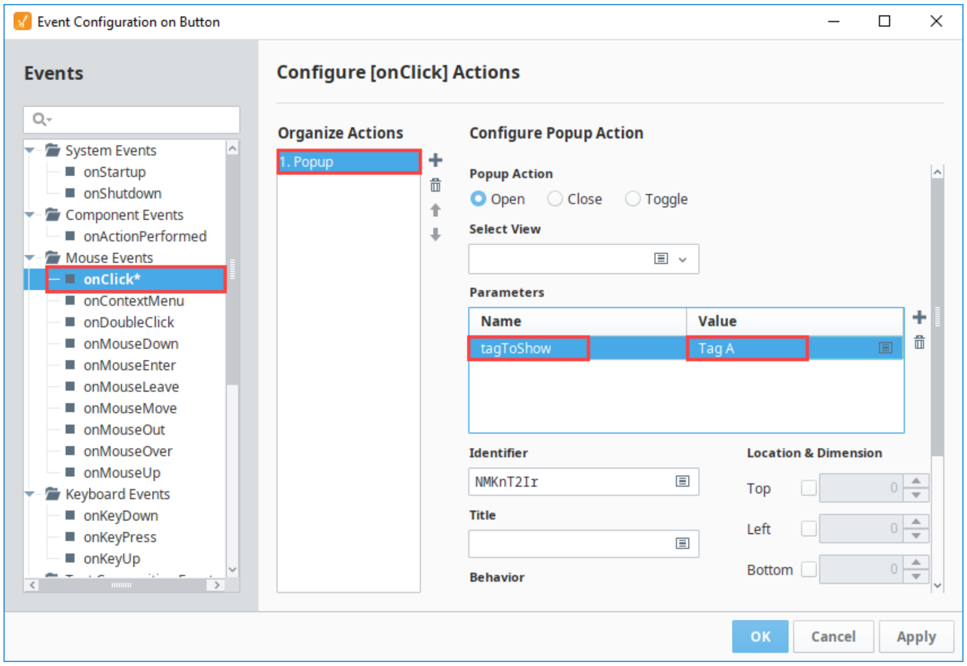
Tag Expression Binding - Example

For this example, we start with two Tags: Tag A and Tag B and two buttons on a view.



Z_Other_Tags			
Tag A	Memory	Hello	String
Tag B	Memory	Hello Again	String
Tag C	Memory	Testing	Text
Auditing	OPC	□	Boolean
EstimatedReturnTime	OPC		DateTime
My_Memory_Tag	Memory	-50	Integer

1. Right-click on the first button and select **Configure Events**.
2. Select **onClick**.
3. Under Organize Actions, click the Add ✚ icon and select **Popup**.
4. Under Parameters, click the **Add** ✚ icon to add a new parameter.
5. Enter "tagToShow" in the Name field and "Tag A" in the Value field. Click **OK**.



Event Configuration on Button

Configure [onClick] Actions

Organize Actions: 1. Popup

Configure Popup Action

Popup Action: ● Open ○ Close ○ Toggle

Select View:

Parameters

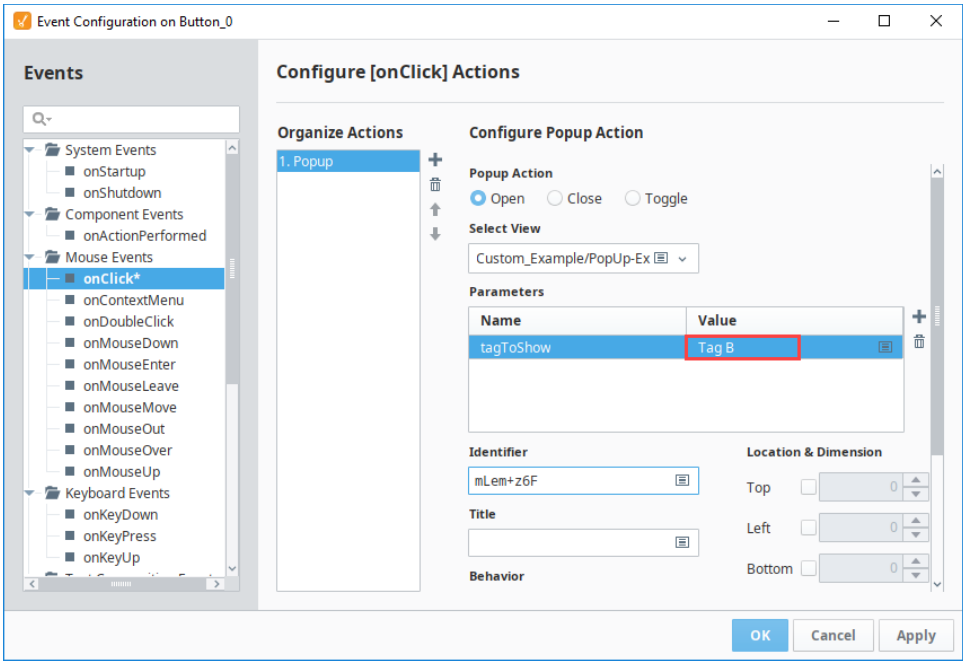
Name	Value
tagToShow	Tag A

Identifier: NMKnT2Ir

Title:

Location & Dimension: Top □ 0, Left □ 0, Bottom □ 0

6. Repeat steps 1 through 4 for the second button, but in the Value field, enter **Tag B**. Click **OK**.



Event Configuration on Button_0

Configure [onClick] Actions

Organize Actions: 1. Popup

Configure Popup Action

Popup Action: ● Open ○ Close ○ Toggle

Select View: Custom_Example/PopUp-Ex

Parameters

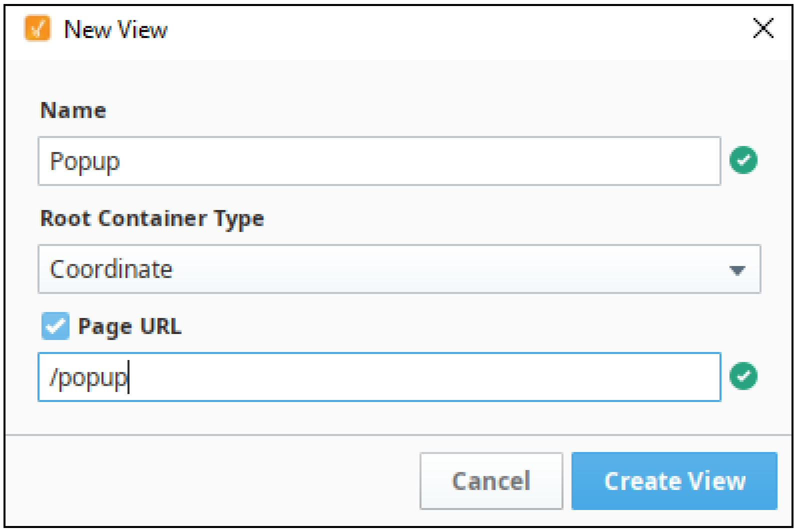
Name	Value
tagToShow	Tag B

Identifier: mLem+z6F

Title:

Location & Dimension: Top □ 0, Left □ 0, Bottom □ 0

7. Create a new view named **Popup**.



New View

Name

Popup

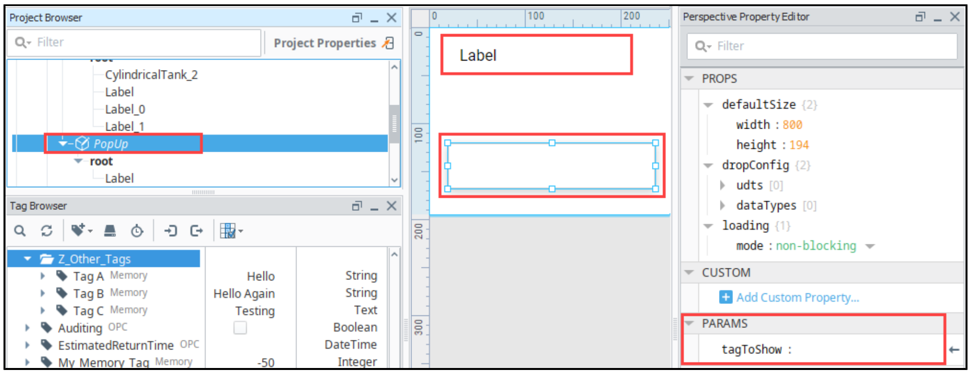
Root Container Type

Coordinate

☑ Page URL

/popup

8. Drag a Label component and a Text Field component onto the Popup view.
9. In the Popup view, click on **Add View ParameterA**.
10. Add a new parameter, **tagToShow**.

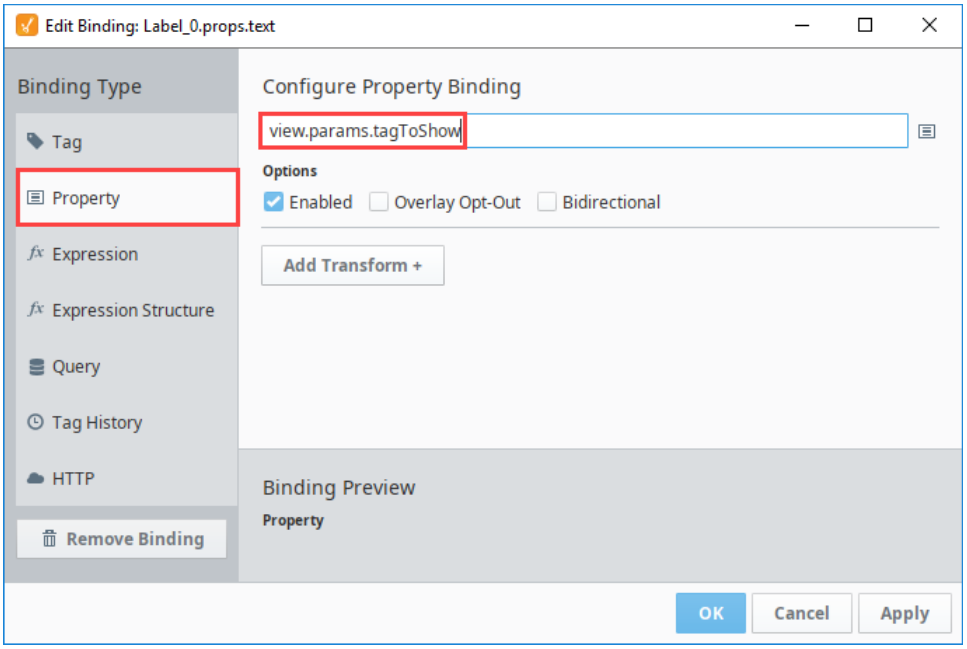


Tag A Memory	Hello	String
Tag B Memory	Hello Again	String
Tag C Memory	Testing	Text
Auditing OPC	□	Boolean
EstimatedReturnTime OPC		DateTime
My Memory Tag Memory	-50	Integer

```
defaultSize {2}
    width : 800
    height : 194
dropConfig {2}
    udts [0]
    dataTypes [0]
loading {1}
    mode : non-blocking
```

tagToShow :

11. Select the Label component. Click on the **Binding** 🔗 icon next to the text property.
12. On the Edit Binding screen, select the Property binding type. Enter **view.params.tagToShow** in the Configure Property Binding field.
13. Click **OK**.



Edit Binding: Label_0.props.text

Configure Property Binding

view.params.tagToShow

Options

☑ Enabled □ Overlay Opt-Out □ Bidirectional

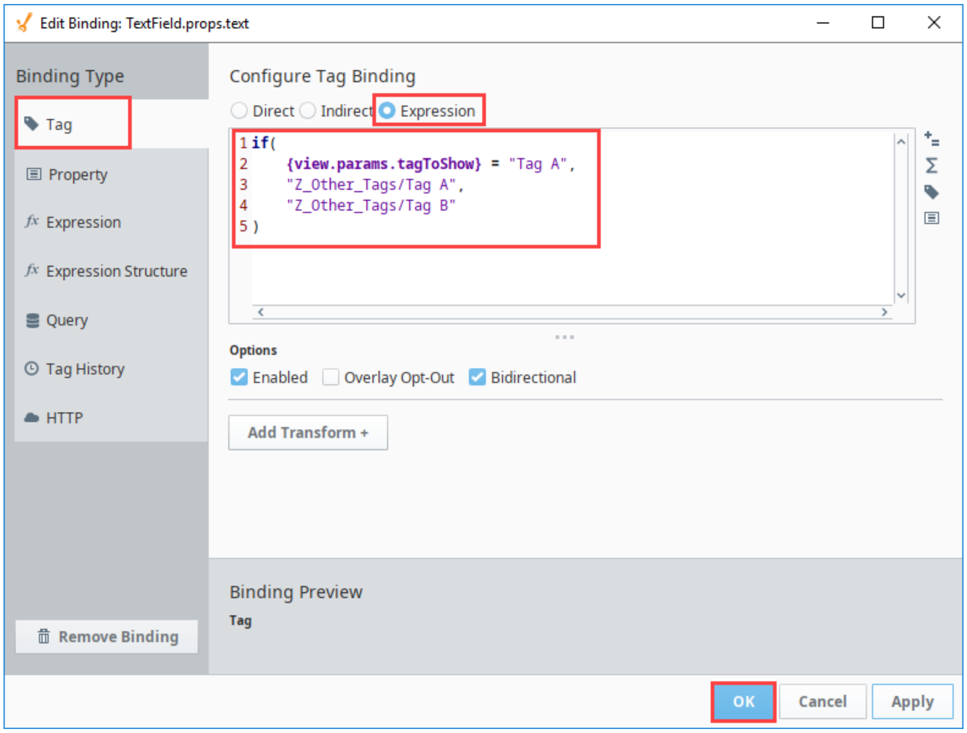
Binding Preview

Property

14. Next set a binding on the Text Field component. Select the component then click on the **Binding** 🔗 icon next to the text property.

15. On the Edit Binding screen, select the Tag binding type and select the **Expression** button.
16. Enter the following expression. This expression in the example tells the Text Field to display the value of Tag A if the view parameter is equal to Tag A. Otherwise display the value of Tag B.

```
if(
    {view.params.tagToShow} = "Tag A",
    "Z_Other_Tags/Tag A",
    "Z_Other_Tags/Tag B")
```



Edit Binding: TextField.props.text

Binding Type

- Tag
- Property
- Expression
- Expression Structure
- Query
- Tag History
- HTTP

Remove Binding

Configure Tag Binding

○ Direct ○ Indirect ● Expression

```
if(
    {view.params.tagToShow} = "Tag A",
    "Z_Other_Tags/Tag A",
    "Z_Other_Tags/Tag B"
)
```

Options

☑ Enabled ☐ Overlay Opt-Out ☑ Bidirectional

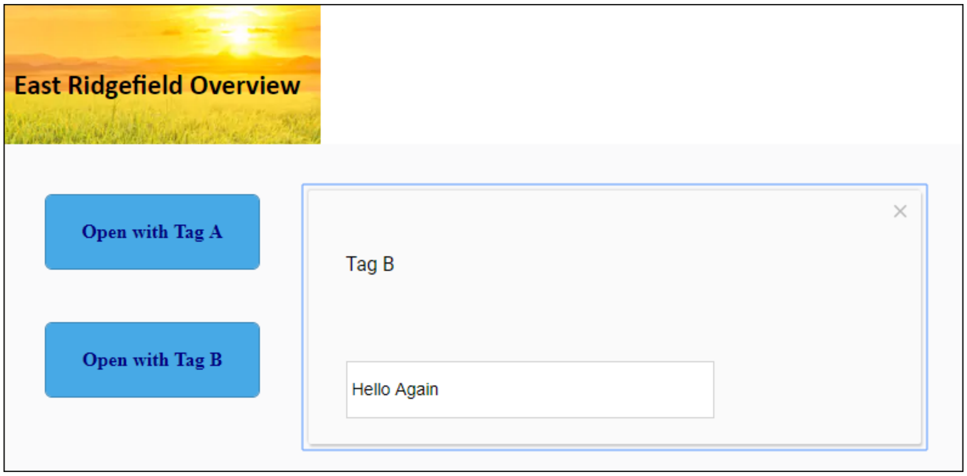
Add Transform +

Binding Preview

Tag

OK Cancel Apply

17. Click **OK** to save.
18. Put the Designer into **Preview mode** ▶. When you click on the Open with Tag B button, the popup appears to show the Tag B label.



East Ridgefield Overview

Open with Tag A

Open with Tag B

Tag B

Hello Again

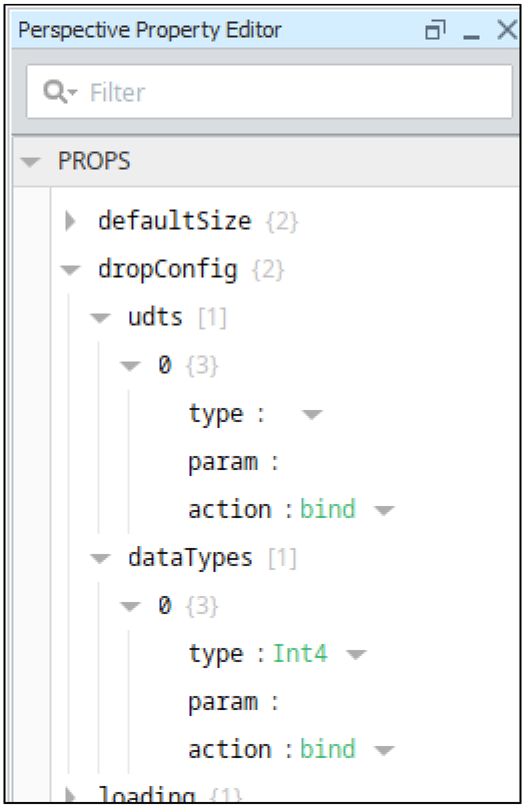
In This Section ...

Drop Configuration

On view objects in Perspective, the **dropConfig** property (also called **drop configuration**) enables you to associate a view with a Tag, allowing for drag-and-drop from the Tag browser to a view in the designer to automatically create a view. There are two general approaches, which are displayed under the **dropConfig** property:

- **udt**: Allows a view to be associated with a UDT. Useful in cases when a system is making use of UDTs, and you'd like to be able to multiple bindings simultaneously.
- **dataTypes:** Allows the associated to be based on the data type of a Tag. For example, you can automatically create and bind a specific view when an Integer Tag is dropped onto another view.

On this page ...



```
defaultSize {2}
dropConfig {2}
  udts [1]
    0 {3}
      type :
      param :
      action : bind
  dataTypes [1]
    0 {3}
      type : Int4
      param :
      action : bind
loading {1}
```

Both of these configurations are made directly on the view you wish to instantiate.

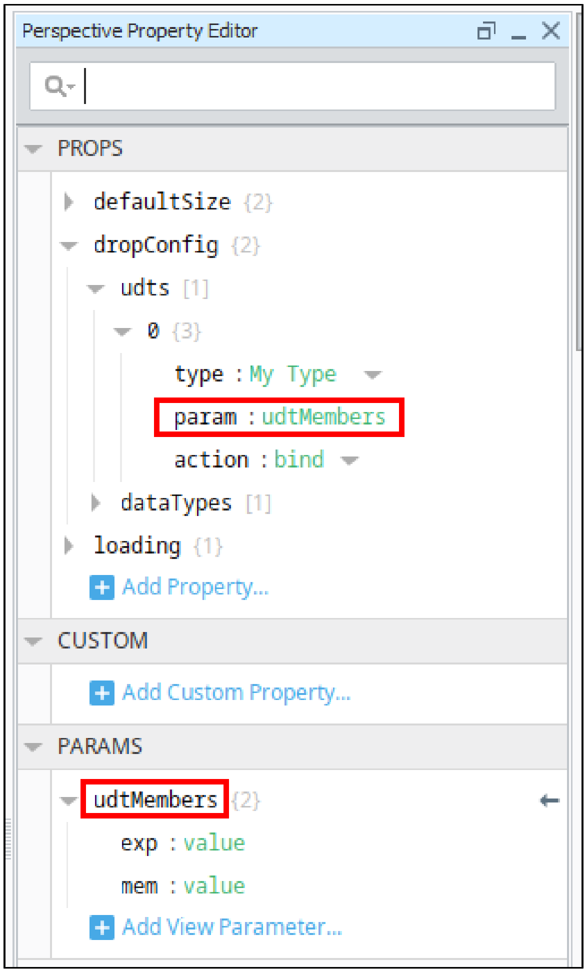
The param and action Properties

While udt and data types drop configurations are listed separately in the Property Editor, they share some properties: namely **param** and **action**.

param

When using a drop configuration, param expects a string value to a view parameter on the same view. The idea being that when a Tag drop creates an instance of this view, the view parameter on the instance will be automatically set with something (determined by the **action**).

In the image below, the view has an element under **dropConfig.udts** configured, with param set to "udtMembers". Further down, under the "PARAMS" heading, there is a parameter with the same name of "udtMembers". Thus the next time the configured UDT (type of "My Type", as shown in the image) is dropped onto a view, the "udtMembers" view param will automatically update its value.



Perspective Property Editor

```
PROPS
  defaultSize {2}
  dropConfig {2}
    udts [1]
      0 {3}
        type : My Type
        param : udtMembers
        action : bind
    dataTypes [1]
  loading {1}
  Add Property...
CUSTOM
  Add Custom Property...
PARAMS
  udtMembers {2}
    exp : value
    mem : value
  Add View Parameter...
```

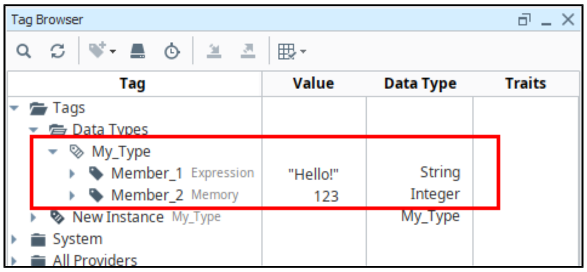
action

The **action** property determines what happens to the specified view parameters (determined by the **param** property). There are two options for **action**:

- **bind**: creates a Tag binding between the view parameter and the dropped Tag. This is useful if components in the view instance need to bind directly to the value of the dropped Tag.
- **path**: populates the view parameter with a Tag path leading to the dropped Tag. Useful if scripts and expression bindings in the view instance are more interested in the path to the Tag.

UDT Drop Config

The **udt** type drop configurations can associate an entire UDT instance, along with its members, to a view. Using a UDT drop configuration is exciting because if the associated view parameter is set to a JSON object, then the configurations will also expose values on the members of the UDT to elements in the JSON object. For example, say we have the following UDT.



Tag Browser

Tag	Value	Data Type	Traits
Tags			
Data Types			
My_Type			
Member_1 Expression	"Hello!"	String	
Member_2 Memory	123	Integer	
New Instance My_Type		My_Type	
System			
All Providers			

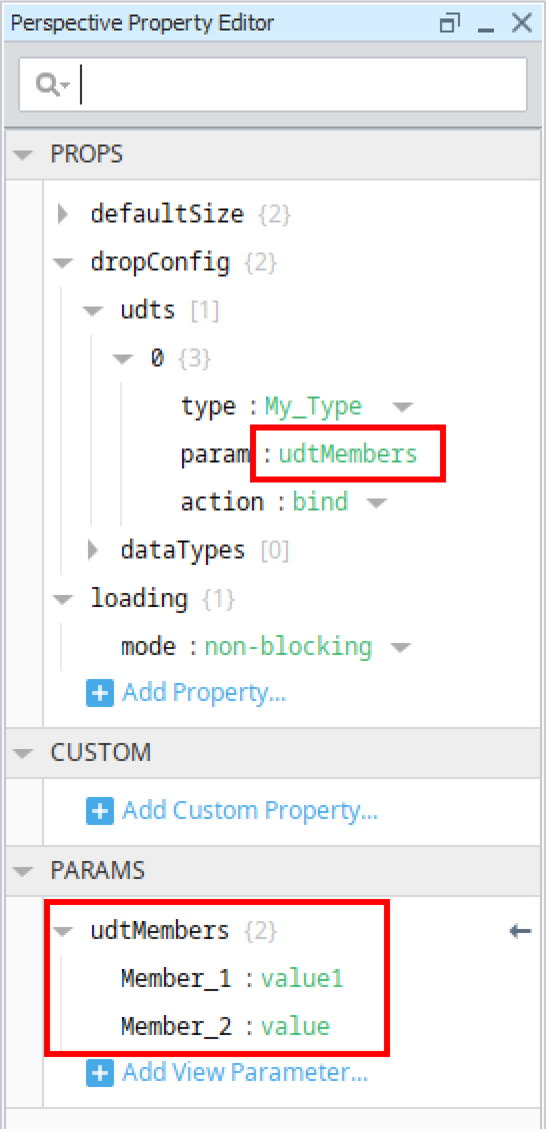
On a view we wish to embed elsewhere, we can add a view parameter as a JSON object, with a key ("value" type) for each member we want in the embedded view. As long as the name of the keys matches a UDT member, then the created view instance lookup values for the named members.

INDUCTIVE UNIVERSITY

Tag Drop

Watch the Video

In the image below, the dropConfig>udts parameter named **udtMembers** is a JSON object with two values. The keys for the those values are **Member_1** and **Member_2**, which matches the names of UDT members in our **My_Type** UDT. As a result, the drop configuration will result the new view instance having access to the value of the Member_1 and Member_2 Tags.



```
defaultSize {2}
dropConfig {2}
  udts [1]
    0 {3}
      type : My_Type
      param : udtMembers
      action : bind
  dataTypes [0]
loading {1}
  mode : non-blocking
```

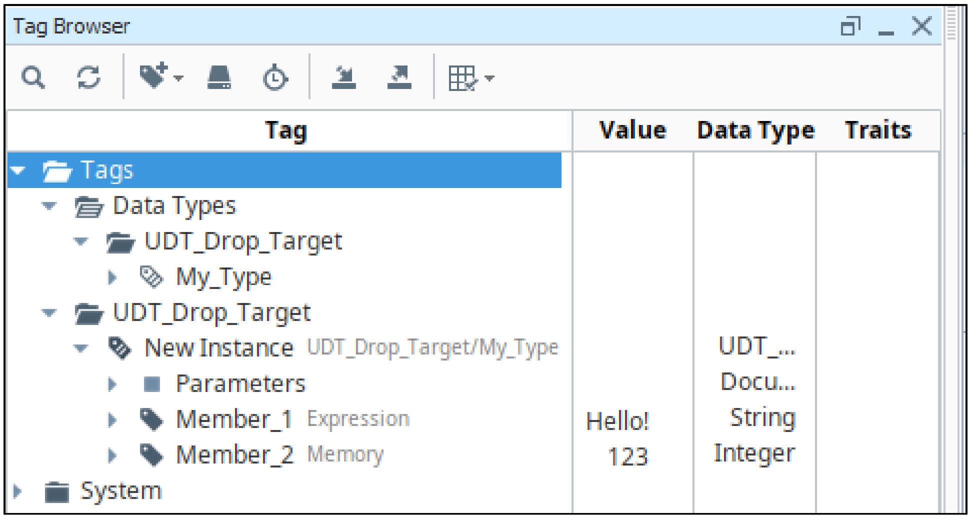
```
udtMembers {2}
  Member_1 : value1
  Member_2 : value
```

Example

1. First, create a UDT definition and instance. If you'd like to use the same Tags as this example, open the expand panel below, copy the JSON string, and paste it into the Tags folder in your Tag Provider.

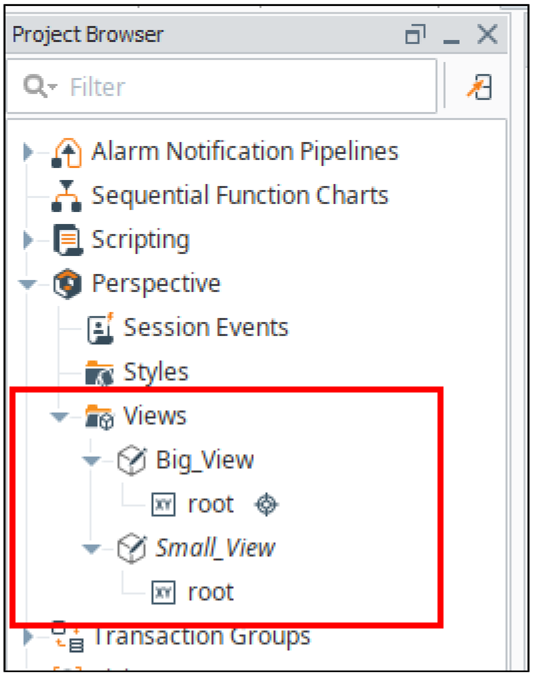
```
{
  "name": "",
  "tagType": "Provider",
  "tags": [
    {
      "name": "UDT_Drop_Target",
      "tagType": "Folder",
      "tags": [
        {
          "name": "New Instance",
          "typeId": "UDT_Drop_Target/My_Type",
          "parameters": {
            "thing1": {
              "dataType": "String",
              "value": "Hello!"
            }
          },
          "tagType": "UdtInstance",
          "tags": [
            {
              "name": "Member_1",
              "tagType": "AtomicTag"
            },
            {
              "name": "Member_2",
              "tagType": "AtomicTag"
            }
          ]
        }
      ]
    },
    {
      "name": "_types_",
      "tagType": "Folder",
      "tags": [
        {
          "name": "UDT_Drop_Target",
          "tagType": "Folder",
          "tags": [
            {
              "name": "My_Type",
              "tagType": "UdtType",
              "tags": [
                {
                  "valueSource": "memory",
                  "name": "Member_2",
                  "value": 123,
                  "tagType": "AtomicTag"
                },
                {
                  "valueSource": "expr",
                  "expression": "\"Hello!\"",
                  "dataType": "String",
                  "name": "Member_1",
                  "tagType": "AtomicTag"
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

It should look like the image below. Notice, both the instance and definition are placed into their own separate folders.

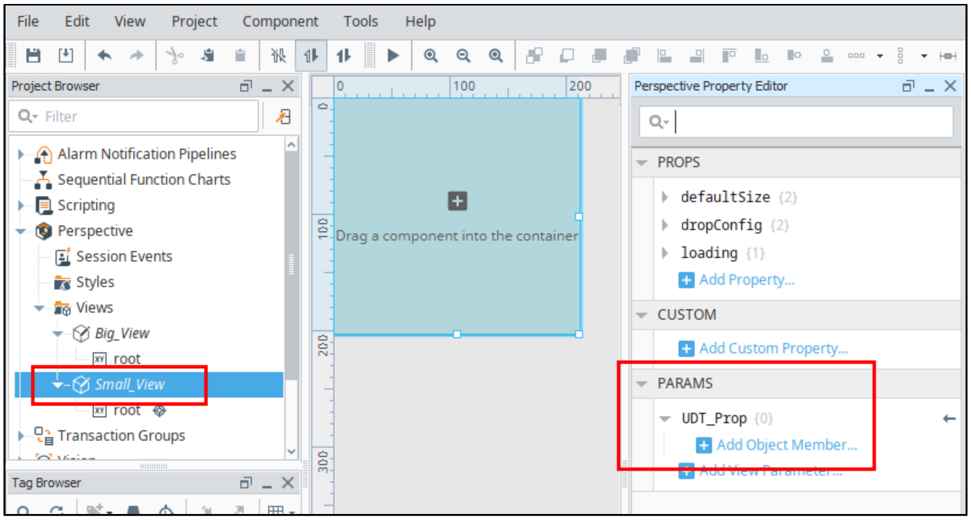


Tag	Value	Data Type	Traits
Tags			
Data Types			
UDT_Drop_Target			
My_Type			
UDT_Drop_Target			
New Instance UDT_Drop_Target/My_Type		UDT_...	
Parameters		Docu...	
Member_1 Expression	Hello!	String	
Member_2 Memory	123	Integer	
System			

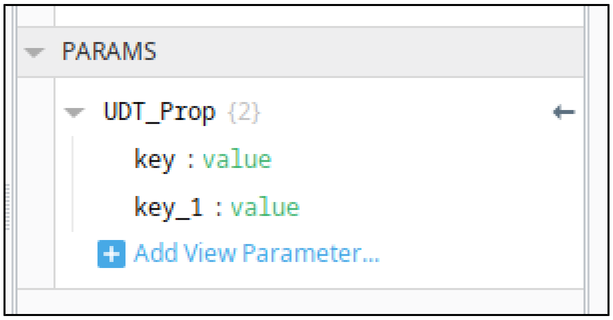
2. Create two views. The first will be Big_View that we'll be dragging Tags onto, and the second will be Small-View that we'll associated to the UDT above. For the purposes of this example, we'll use Coordinate Containers as the root container for both views.



3. In the **Small_View**, we'll make a minor cosmetic change so it is easier to see when we're embedding it later. We'll change the background color to something other than the default.
4. The, with the view selected, add a view parameter. When selecting the parameter type, choose **Object**. We'll name it **UDT_Prop**.



5. Add two object members to **UDT_Prop**. These members will represent the values on the two UDT members in our UDT. The values on our UDT members are integer and string, so both only have a single value. Thus, the object members we need to add here only need to show a single value each. Thus we'll make the object members Value typed.



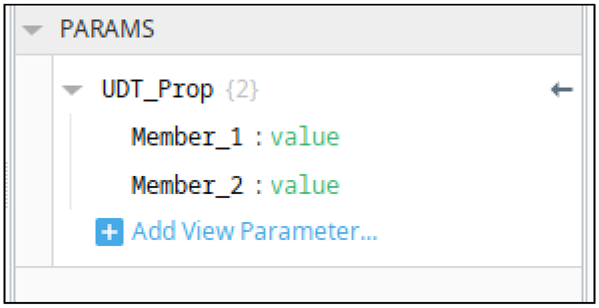
PARAMS

UDT_Prop {2}

key : value

key_1 : value

6. For the view parameters to know which members in the UDT to associate with, we need to rename them so they match the name of a UDT member. Change the name of the view parameters to **Member_1** and **Member_2**, to match our UDT members.



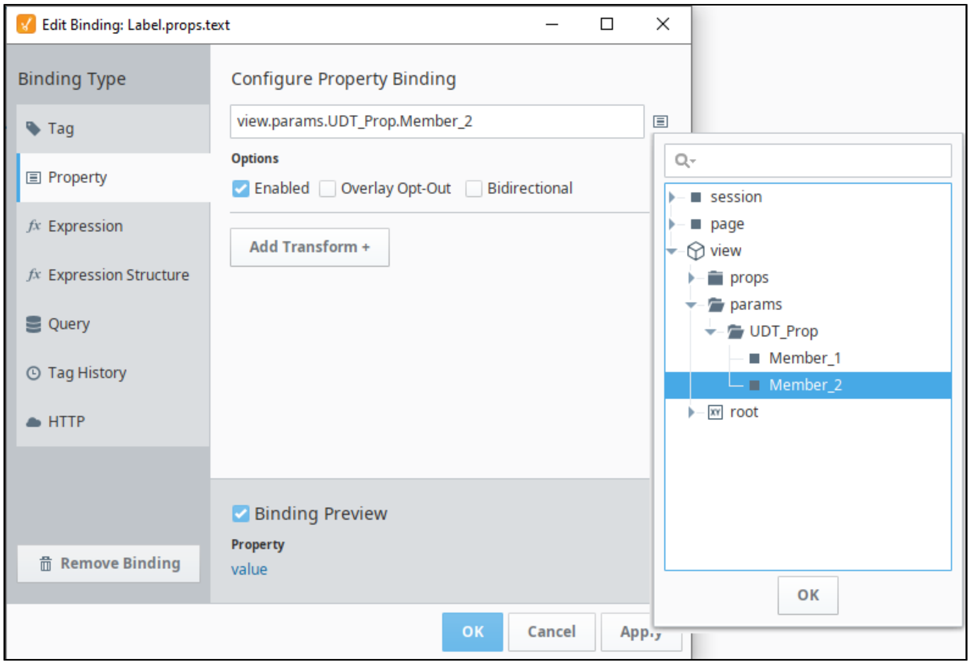
PARAMS

UDT_Prop {2}

Member_1 : value

Member_2 : value

7. Now that we have some parameters that will eventually house values from our UDT, we can create components that use these parameters. Add a **Label** component to **Small_View**.
8. On the Label, configure a **Property Binding** on the **Text** property, leading to the **Member_2**. We're using Member_2 here since it's a memory Tag, and we can easily change the value of it later on.



Edit Binding: Label.props.text

Configure Property Binding

view.params.UDT_Prop.Member_2

Options

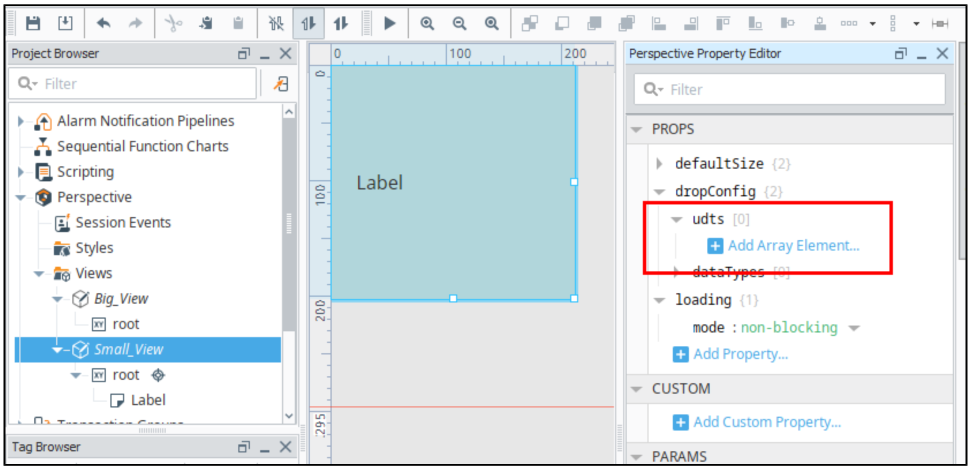
☑ Enabled □ Overlay Opt-Out □ Bidirectional

☑ Binding Preview

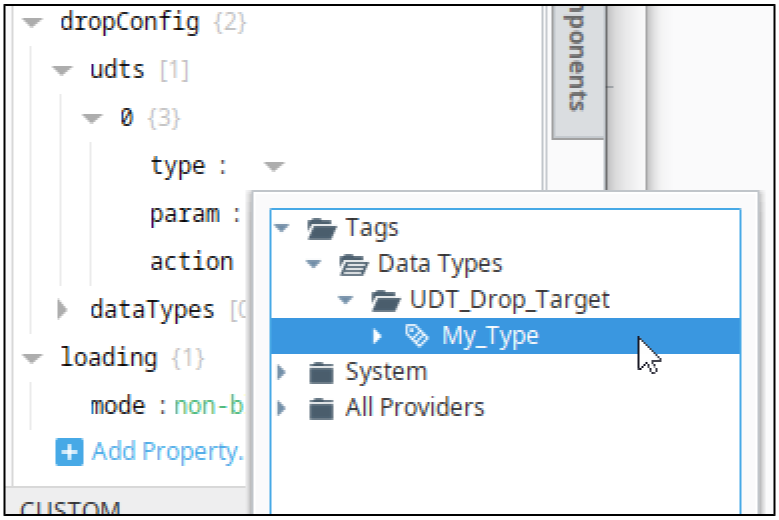
Property

value

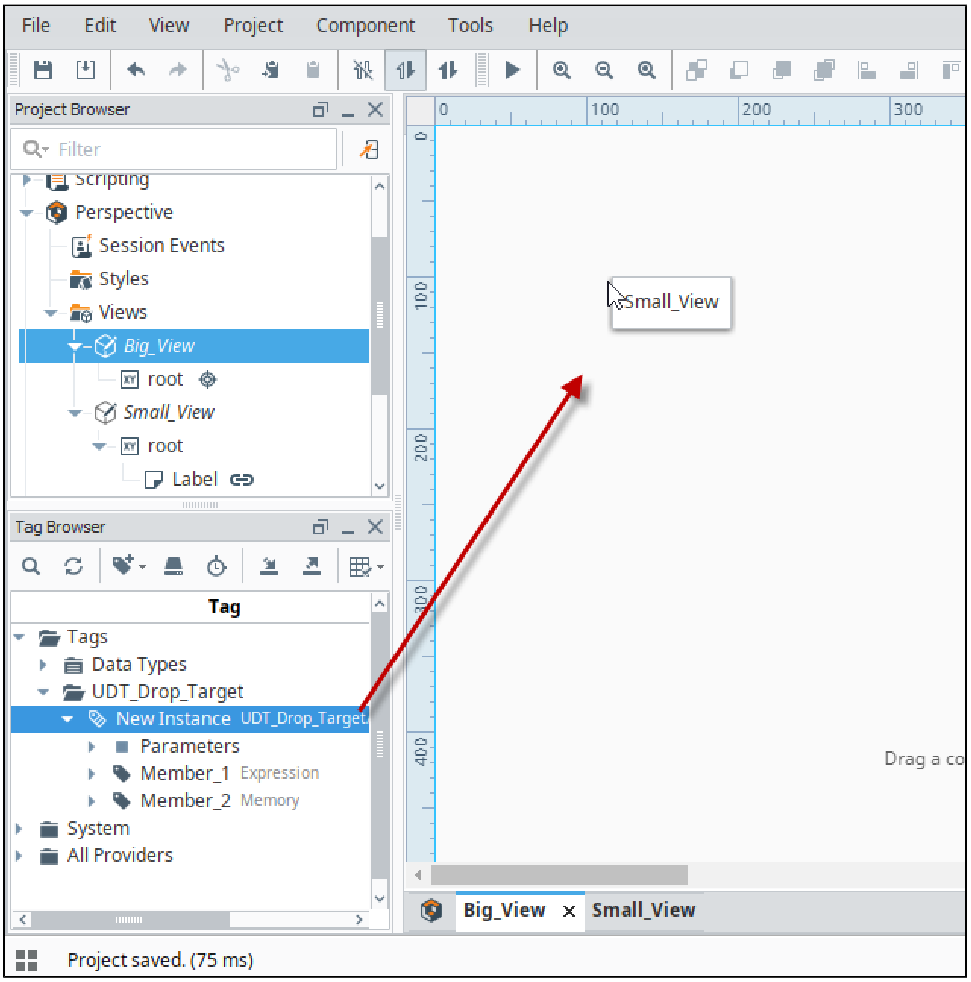
9. Next, we need to configure the drop configuration settings on **Small_View**. Select the view, and locate the **udts** property.



10. Add an array element to the **udts** array. For **type**, click the dropdown and select our UDT definition.

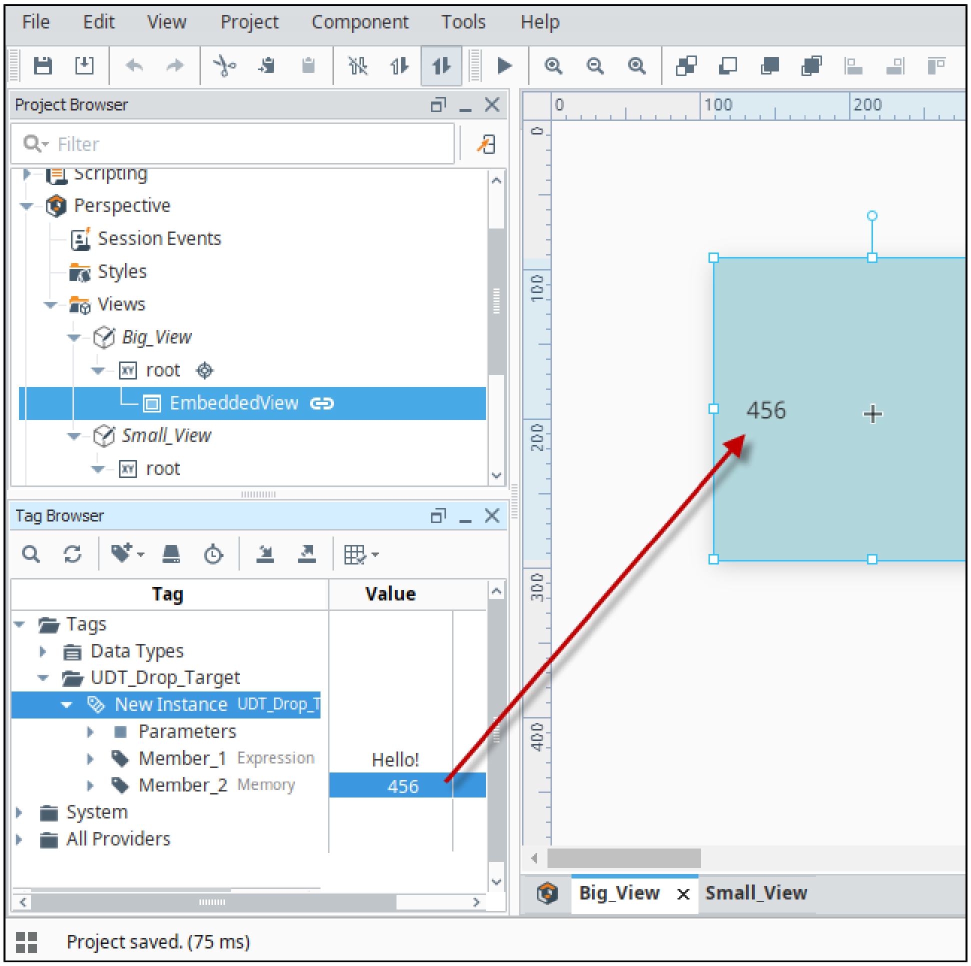


11. For **param**, type in the name of the root view parameter we created earlier. If you followed along with the example, the name would be **UDT_ Prop**.
12. For **action**, leave that with the default value of **bind**, which will create some bindings for us when we create an instance of this view.
13. Now we're ready to test this out. Save your project.
14. Switch over to **Big_View**.
15. Drag the UDT instance (or any UDT instance with a parent type of **My_Type**) onto **Big_View**. You'll be presented with a popup, which lists any and all views that are associated with that type.



16. Select **Small_View** from the popup, and you'll see an instance of **Small_View** created. Further more, the values in our UDT instance are made available to the view instance, so the value on the **Member_2** Tag will be shown on the label in the instance of **Small_View.**

17. Changing the value on the **Member_2** Tag will update the label on the instance of **Small_View**.

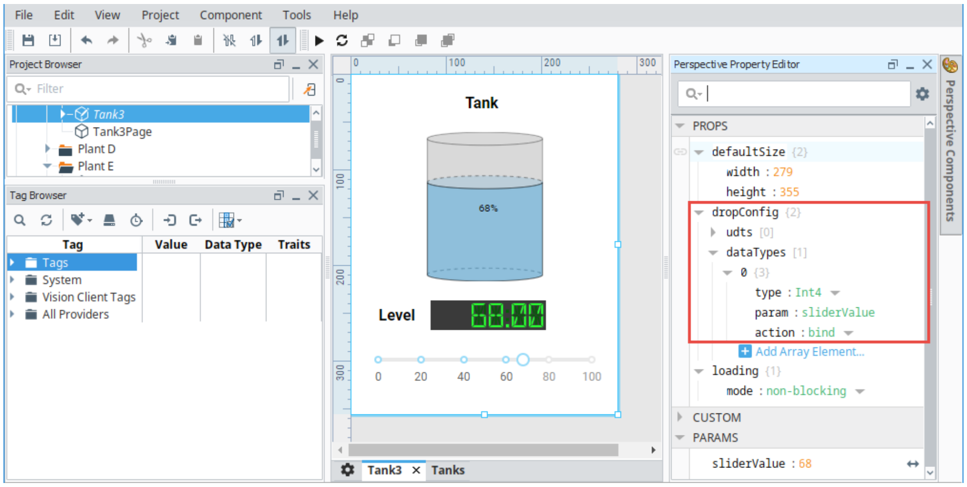


Data Type Drop Configurations

Data type drop configurations allow you to associate a view with a certain Tag data type, as well as specify what bindings occur when dropping the view.

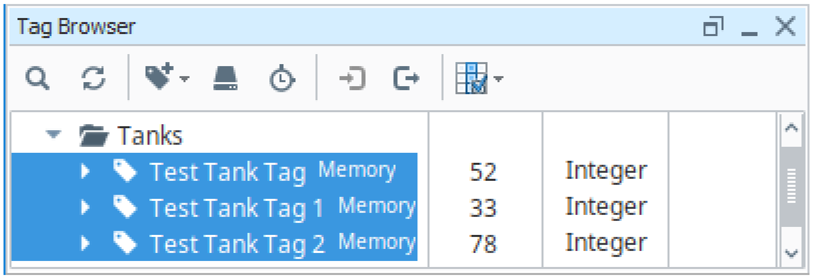
Let's use a Tag to create an embedded view with all the property bindings configured by simply dropping the Tag in a view.

1. In the **Project Browser**, select your original view that contains the Tank, LED Display, Slider, and Labels (i.e., Tanks3).
2. In the **Property Editor**, expand the **dropConfig** property, and then expand the **dataTypes** property. Configure the following properties:
    a. Select the data type - **Int4**
    b. Set the param - **sliderValue**
    c. Identify the action - **bind**



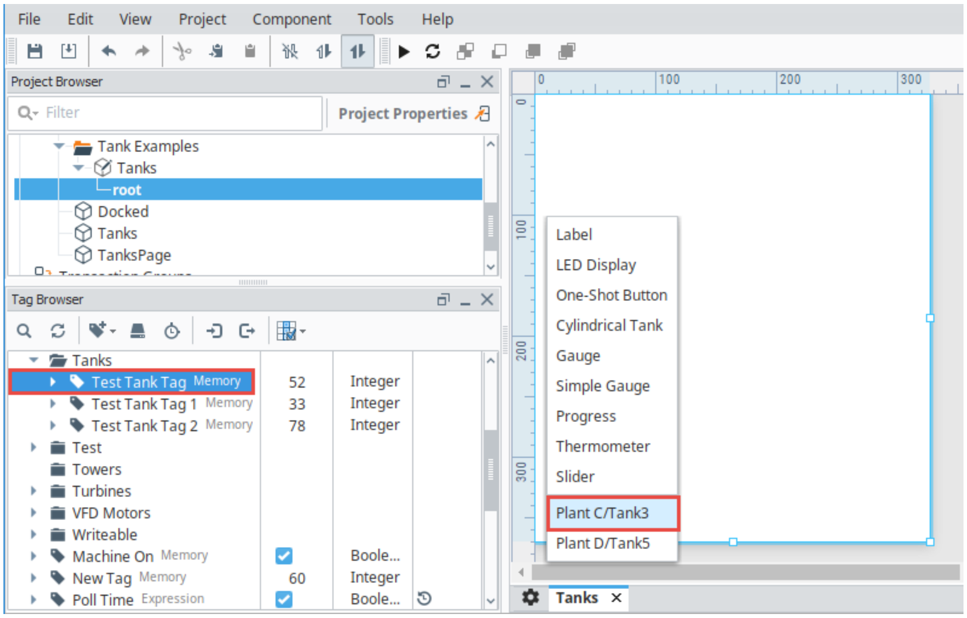
```
PROPS
  defaultSize {2}
    width : 279
    height : 355
  dropConfig {2}
    udts [0]
    dataTypes [1]
      0 {3}
        type : Int4
        param : sliderValue
        action : bind
      Add Array Element...
  loading {1}
    mode : non-blocking
CUSTOM
PARAMS
  sliderValue : 68
```

3. Let's create a couple of Tags so we can use the **'dropConfig'** feature to create multiple embedded views. Copy the Tag you created in the last example (i.e., Test Tank Tag) to make two more Tags, and change their values. This example uses 52, 33, and 78 for the values.



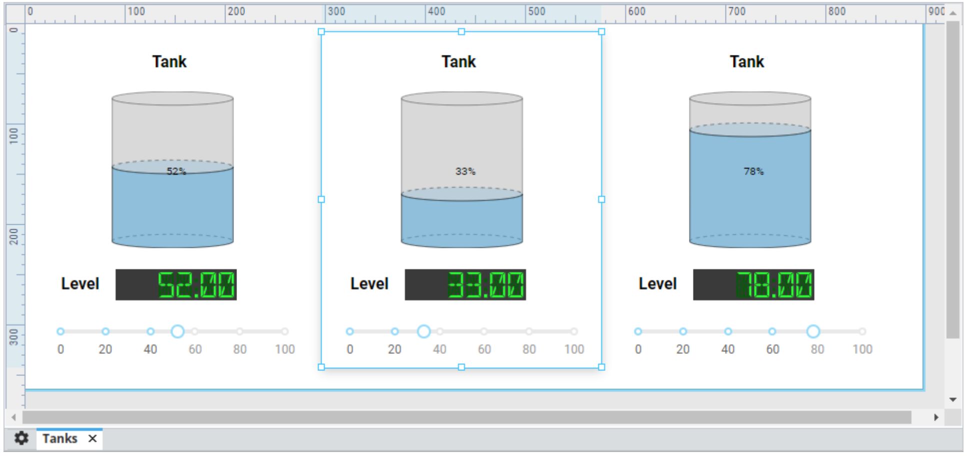
Tag	Value	Data Type
Tanks		
Test Tank Tag Memory	52	Integer
Test Tank Tag 1 Memory	33	Integer
Test Tank Tag 2 Memory	78	Integer

4. Create a new large view so we have plenty of room to drop multiple of these tank views on the screen.
5. Drag and drop each of your three Tags into your new large view. You will be prompted for what type of component or view you want to create. Choose your original view (i.e., Plant C / Tank3).

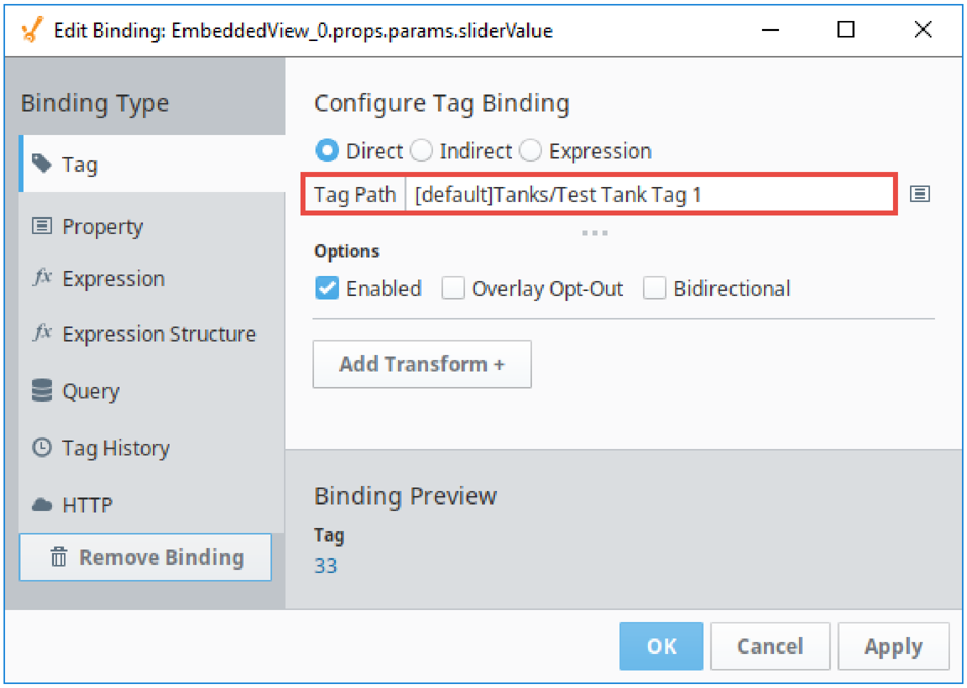


Tag	Value	Data Type
Tanks		
Test Tank Tag Memory	52	Integer
Test Tank Tag 1 Memory	33	Integer
Test Tank Tag 2 Memory	78	Integer
Test		
Towers		
Turbines		
VFD Motors		
Writeable		
Machine On Memory	☑	Boole...
New Tag Memory	60	Integer
Poll Time Expression	☑	Boole...

6. The three Tank components and their values are now displayed on your view.



7. If you check the individual tank views, you will see a binding on the **'sliderValue'** param. If you open the binding, you will see that each embedded view is pointing to one of the Tags.



Edit Binding: EmbeddedView_0.props.params.sliderValue

Binding Type

- Tag
- Property
- Expression
- Expression Structure
- Query
- Tag History
- HTTP

Remove Binding

Configure Tag Binding

● Direct ○ Indirect ○ Expression

Tag Path [default]Tanks/Test Tank Tag 1

Options

☑ Enabled □ Overlay Opt-Out □ Bidirectional

Add Transform +

Binding Preview

Tag

33

OK Cancel Apply

Property Bindings in Perspective

Property bindings are important when designing projects in Perspective. Each Perspective component has a number of properties that change the way a component looks and behaves, but it's through property bindings that bring your Perspective sessions to life to accomplish useful things.

A property binding is the simplest type of binding. It's a way of linking one component property in a view to another component's property in the same view. Not only can you link one property in a view to another property in the same view, but you can also link a component to a property within a UDT, and pass a property into an embedded view through a view parameter.

This page describes how to set up property bindings for properties in the same view, bind to a property within a UDT, and how to use a view parameter to pass a property into an embedded view.

> **Using Property Bindings Across Views**
>
> You cannot have a binding refer to a property in another view instance even view instances that may be embedded in a view. You can pass a property into an embedded view through a view parameter.

On this page ...

INDUCTIVE UNIVERSITY

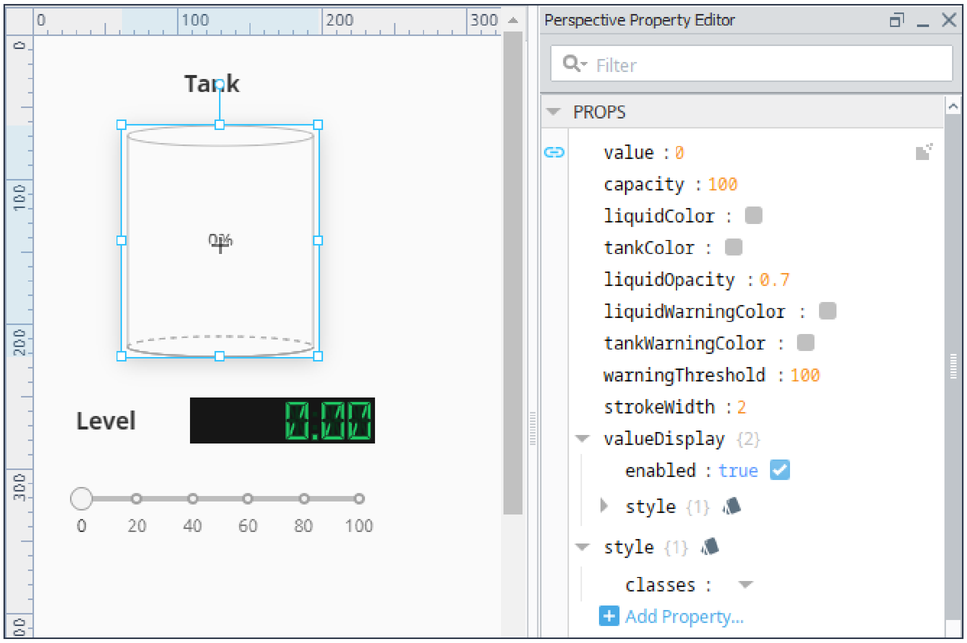
Property Binding

Watch the Video

Property to Property Binding

A property binding 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. In this example, we'll use a Tank, LED Display, and Slider components. We'll bind the 'value' properties of the Tank and LED Display to the 'value' property of the Slider in the same view so whenever the value of the Slider changes, the Tank and LED Display will reflect the same value.

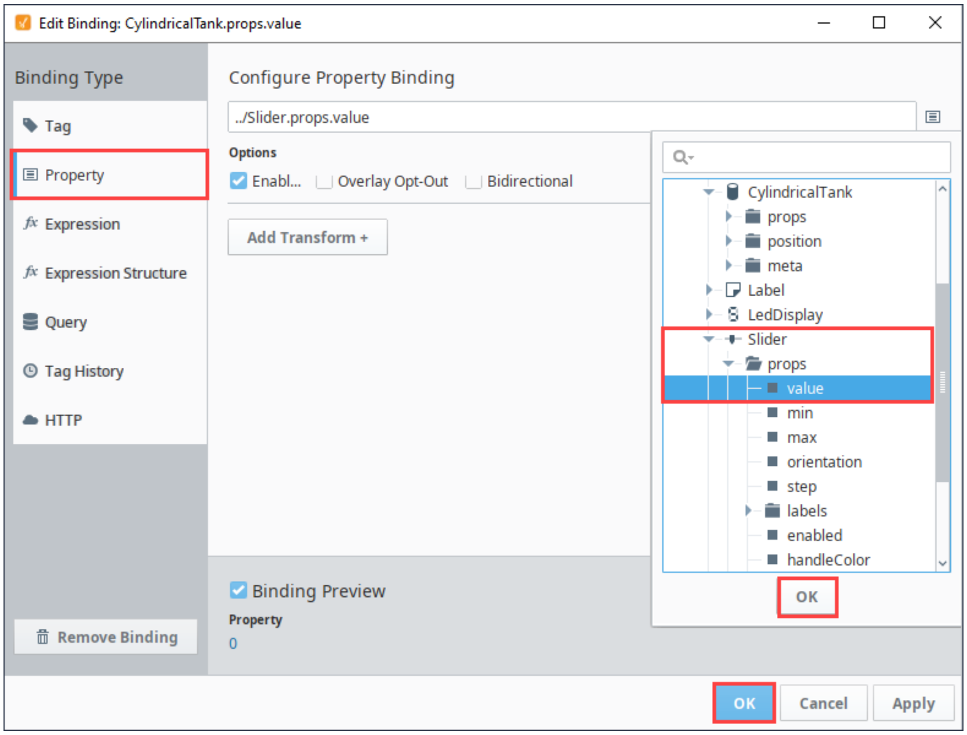
1. In the Designer, create a view.
2. From the Component Palette, drag **Cylindrical Tank**, **LED Display**, and **Slider** components into your Designer workspace.
3. Select the **Tank**. In the **Property Editor**, click on the **Binding** 🔗 icon for the 'value' property.



value : 0
capacity : 100
liquidColor :
tankColor :
liquidOpacity : 0.7
liquidWarningColor :
tankWarningColor :
warningThreshold : 100
strokeWidth : 2
valueDisplay {2}
enabled : true ☑
style {1}
style {1}
classes :

4. This opens the Edit Binding window. Configure the following settings:
   a. Select the **Property** binding type.

b. Under Configure Property Binding, click the **Insert Property** icon to open the **Property** popup box. Expand the folders and select the Slider **'value'** property. Click **OK**.
c. Click **OK** to save the binding settings.



Edit Binding: CylindricalTank.props.value

Configure Property Binding

../Slider.props.value

Options

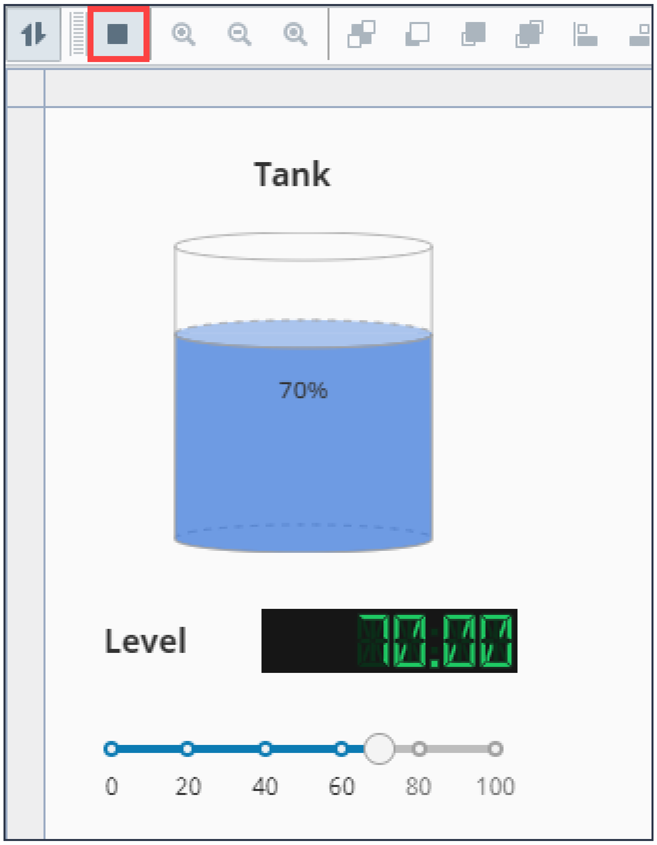
☑ Enabl... □ Overlay Opt-Out □ Bidirectional

☑ Binding Preview

Property

0

5. Select the **LED Display**. In the **Property Editor**, click on the **Binding** icon for the **'value'** property and repeat Step 4.
6. To see the labels on the Slider, select the Slider and set the **show** property to **'true'**.
7. Save your project. Put the Designer in **Preview Mode** ▶.
8. Move the slider and you'll see both the tank level and LED display change to the value of the Slider.



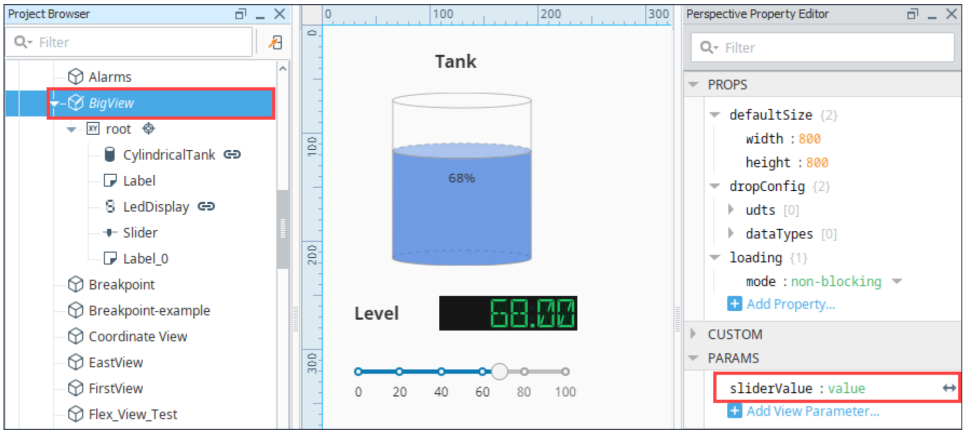
Pass a Property into an Embedded View Using a View Parameter

The only way to a pass a property across views is by passing a view parameter into an embedded view. You have options on how to pass a property into an embedded view, how you decide to set up passing a parameter depends on how you design your project. You can set up passing a property to a embedded view using a parameter with strictly property bindings, or with a Tag binding. Tag bindings allow you to store values in a database in the event you want to collect history but will force all sessions to see the same value.

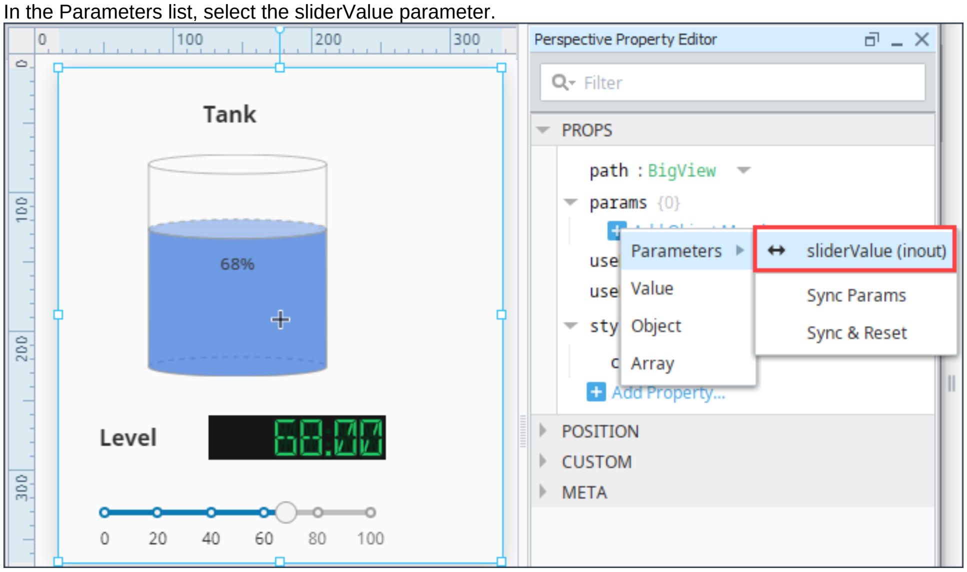
Use a Property Binding

In this example, we'll use a Tank view containing the Tank, LED Display, and Slider that shows passing the 'value' properties from the Tank and the LED Display to the 'value' property of the Slider. Now we will set the Slider to pass its value to a parameter on the embedded view. To demonstrate this, you can use the same Tank view from the Property to Property Binding section above, or you can create another view using the same components as in the example above, and assign the view a new name.

1. Using your original view from above, select the view in the Project Browser.
2. Create a view parameter under **Params** called **'**sliderValue**'** and make it bidirectional by toggling the **Arrow** ↔ icon until the arrowhead is at both ends.



3. In the Project Browser, create a new view (i.e., Tank3Page) that will contain the embedded view.
4. Drag a **Embedded View** from the Component Palette to the Designer workspace.
   a. With the Embedded View selected, set the 'path' property to your original view (i.e., BigView) from the dropdown list.
   b. In the **Property Editor**, click the **Add Object Member** ➕ icon under props.params.
   c. In the Parameters list, select the sliderValue parameter.

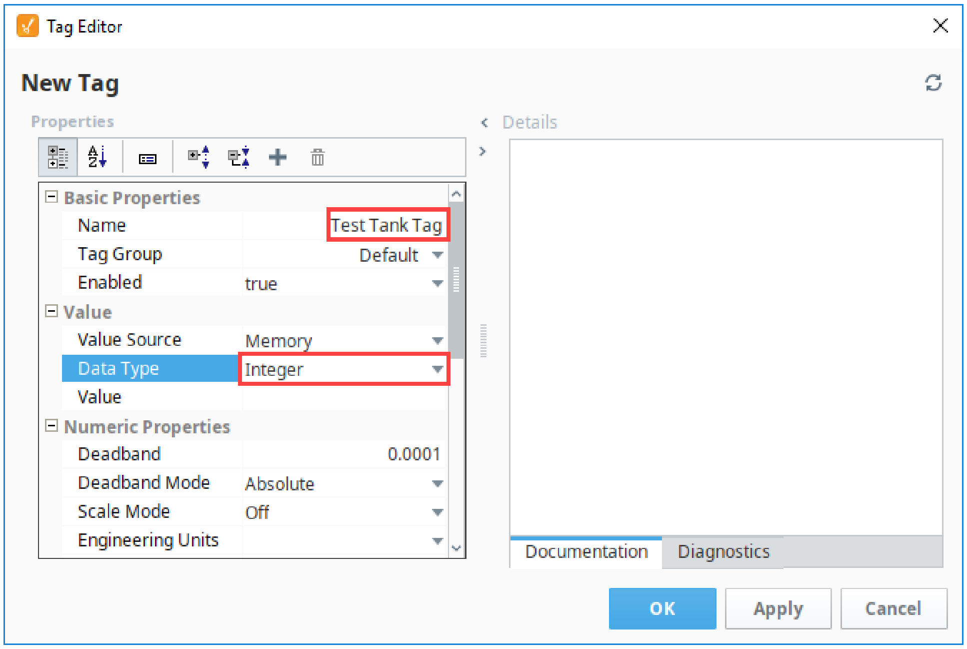


5. **Save** your project.
6. From this new view, put the Designer in **Preview Mode** ▶. Drag the slider to a value to change the value on the Tank and LED Display.
7. You can see from the Embedded View (i.e., Tank3Page), that your sliderValue reflects the same values as your slider.

Using a Tag Binding

Now, let's pass a property using a Tag binding. Using a Tag binding will allow you to maintain the value of the Slider when you relaunch a Client or Session and store values in a database in the event you want to collect history.

1. In the Tag Browser, create a Memory Tag with a data type of Integer, and assign a name (i.e.,Test Tank Tag).



Tag Editor

New Tag

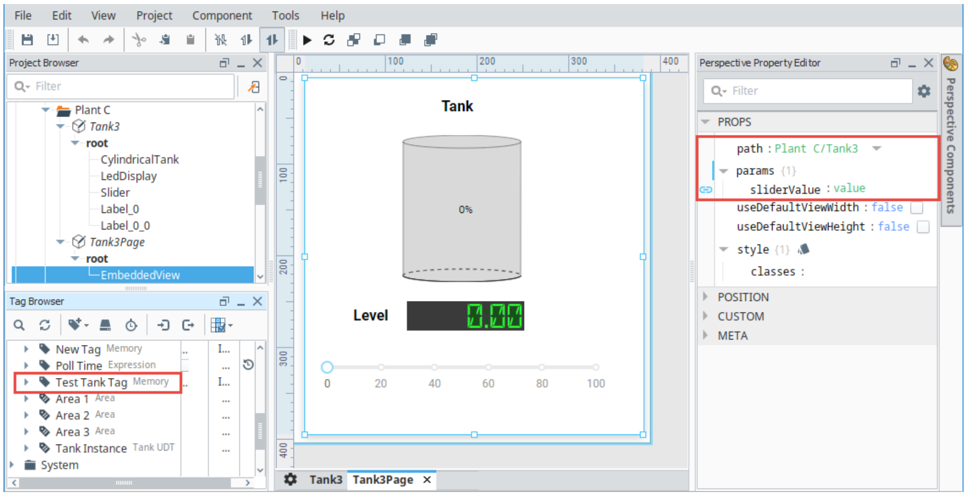
Properties

Basic Properties
Name: Test Tank Tag
Tag Group: Default
Enabled: true

Value
Value Source: Memory
Data Type: Integer
Value:

Numeric Properties
Deadband: 0.0001
Deadband Mode: Absolute
Scale Mode: Off
Engineering Units:

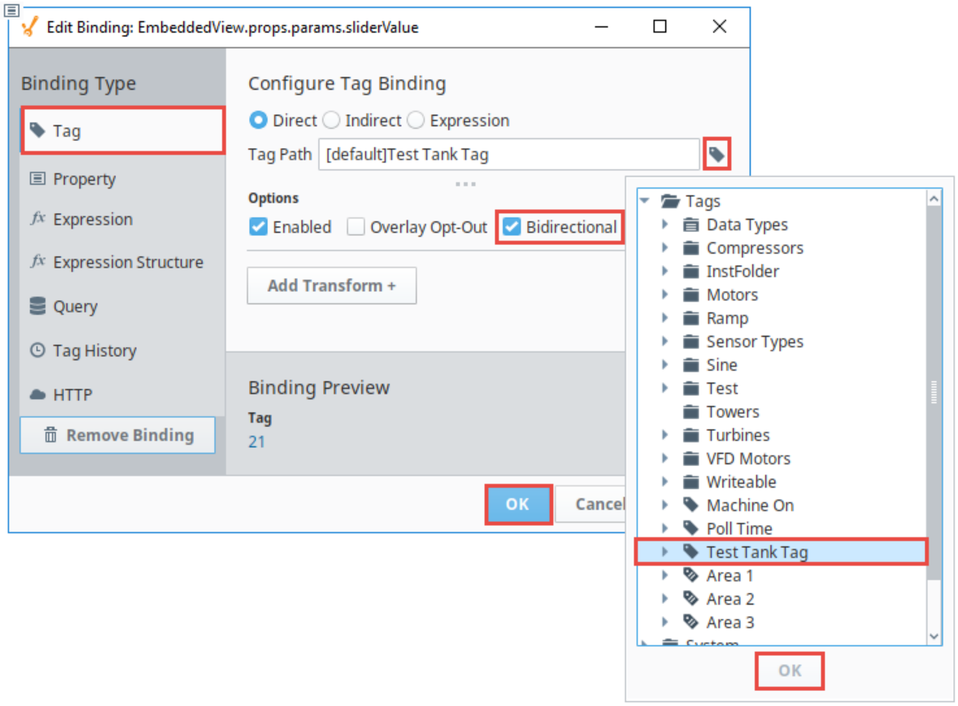
2. Select the **Embedded View.** Click on the **Binding** icon next to the **'sliderValue'** and bind it to the memory Tag (i.e., Test Tank Tag).



Perspective Property Editor

PROPS
path : Plant C/Tank3
params {1}
sliderValue : value
useDefaultViewWidth : false
useDefaultViewHeight : false
style {1}
classes :
POSITION
CUSTOM
META

3. This opens the Edit Binding window. Configure the following settings:
   a. Select the **Tag** binding type.
   b. Under Configure Tag Binding, click the **Tag** icon to open the Property popup box. Expand the folders and select the **'Test Tank Tag'**. Click **OK**.
   c. Click the **Bidirectional** checkbox. allowing the Tag to be updated by the embedded view.
   d. Click **OK** to save the binding settings.



Edit Binding: EmbeddedView.props.params.sliderValue

Binding Type

- Tag
- Property
- Expression
- Expression Structure
- Query
- Tag History
- HTTP

Remove Binding

Configure Tag Binding

● Direct ○ Indirect ○ Expression

Tag Path [default]Test Tank Tag

Options

☑ Enabled □ Overlay Opt-Out ☑ Bidirectional

Add Transform +

Binding Preview

Tag

21

OK Cancel

4. Now that all your bindings and Tag are configured, let's test out passing a parameter using a Tag. From the original view (Tanks), put the Designer in **Preview Mode** ▶ and move the Slider to different values. Then, go to the embedded view and see if the value was passed. Your embedded view should reflect the same values as in your original view.

Expression Bindings in Perspective

Binding Properties to the Outcome of an Expression

An expression binding is one of the most powerful kinds of property bindings. It uses a simple expression language to calculate a value. This expression can involve lots of dynamic data, such as other properties, Tag values, results of Python scripts, queries, and so on. Any time information needs to be massaged, manipulated, extracted, combined, split, and so on - think expressions!

Event Based and Polling

How an expression updates depends on what is being done in the expression. Expression bindings will always update immediately when the window they are in is opened. When they update again depends on if they are driven by events or polling. Typically, expressions are driven by events. If the expression was adding multiple values together, then when one of those values changed the expression would update, regardless of whether those values came from other properties or Tags. However, the expression function has some unique functions that can update at a set rate such as the now() function. When these functions are used within the expression, the expression binding will update based on the specified polling rate.

On this page ...

Expression Binding

Watch the Video

Using Expression Bindings

The expression language has lots of tools available that help calculate a specific value such as built in expression functions, multiple operators, and the ability to reference Tags. While all of these can be manually typed into the expression, the expression binding window makes it easy to reference these options.

To the right of the Expression Binding window, there are four buttons which can be used to reference specific objects or functions easily.

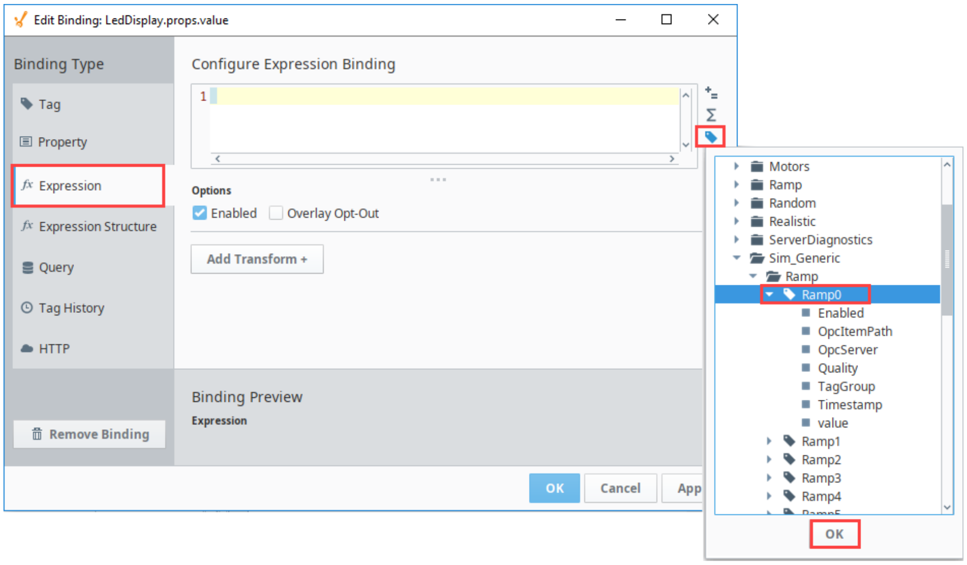
Icon	Function	Description
⁺₌	Operators	Places the operator into the expression at the cursor. Mostly used as a reference to what operators are available for use.
Σ	Functions	Places the function into the expression at the cursor. Can be used as a reference for what functions are available, as well as the parameters the function is expecting.
🏷	Tags	Places a Tag reference into the expression at the cursor, pulling in that Tag's value into the expression at the time of evaluation.
▤	Properties	Places a property reference into the expression at the cursor, pulling in that property's value into the expression at the time of evaluation.

Example 1

In this first example we'll use an Expression binding to combine and display the value of two Tags.

1. Create a new view and place two LED components on it.
2. Select the first LED component and click on the **Binding** 🔗 icon for the value property.
3. Select Expression the binding type.
4. In the Configure Expression Binding section, click on the **Tag** 🏷 icon.

5. Scroll down to the Tag you want to use (Ramp0 in the example) and click **OK**.



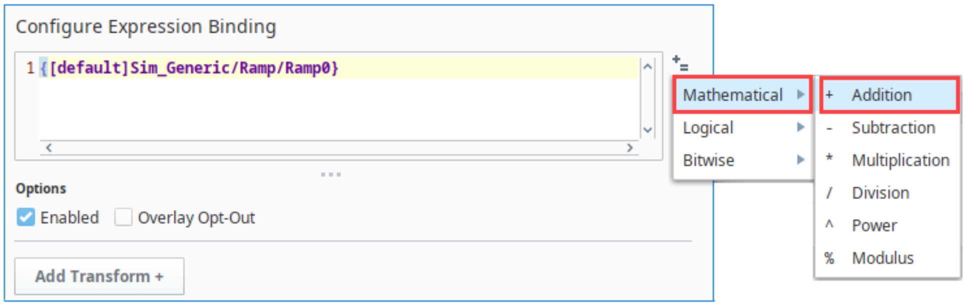
```
1
```

Options

☑ Enabled ☐ Overlay Opt-Out

Expression

6. Next click the **Operators** icon. Select **Mathematical**, and then choose the **Addition** option.



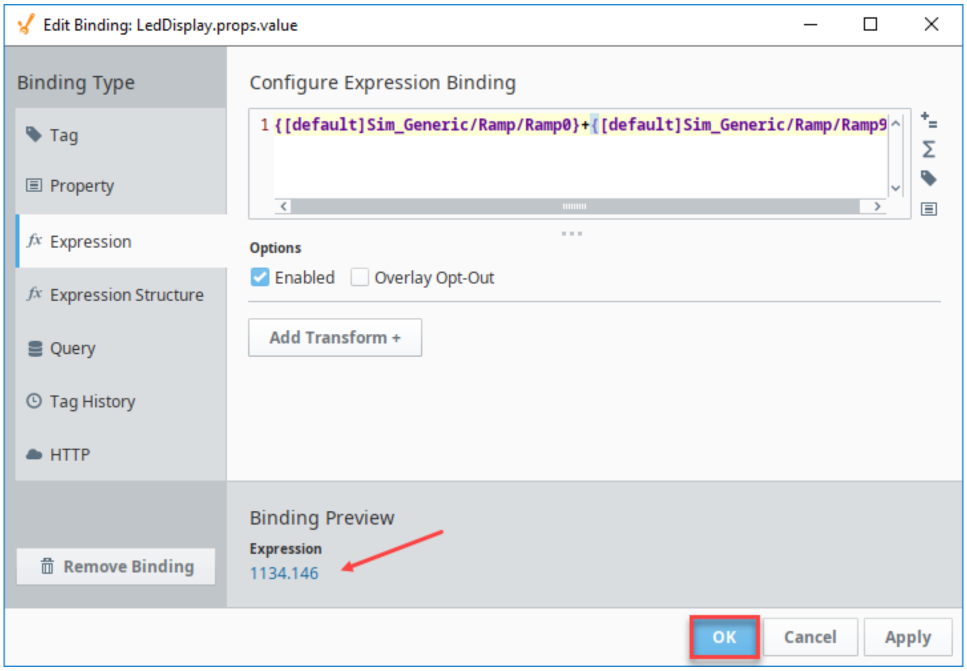
```
1 {[default]Sim_Generic/Ramp/Ramp0}
```

Options

☑ Enabled ☐ Overlay Opt-Out

7. Click the **Tag** icon again and select the second Tag. Click **OK**.

Now when this Expression is run, the value of this Tag will be added to the first Tag. Note that a preview of the Expression binding value is shown on the lower left.



```
1 {[default]Sim_Generic/Ramp/Ramp0}+{[default]Sim_Generic/Ramp/Ramp9
```

Options

☑ Enabled ☐ Overlay Opt-Out

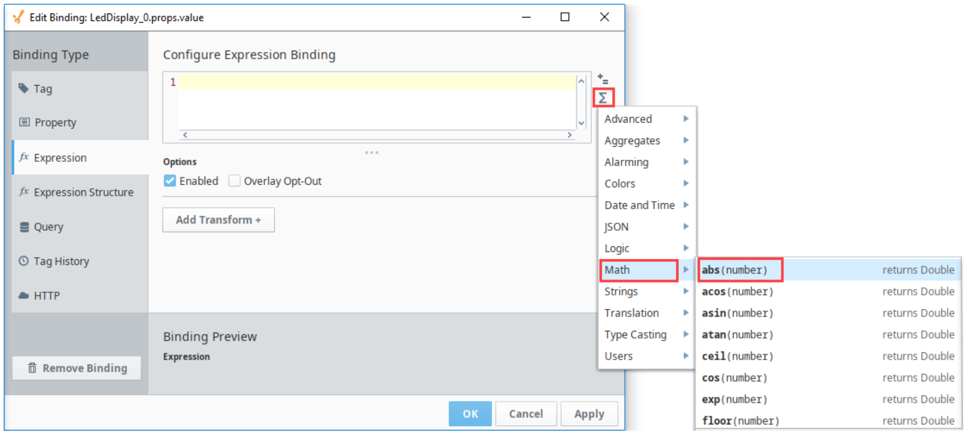
Expression

1134.146

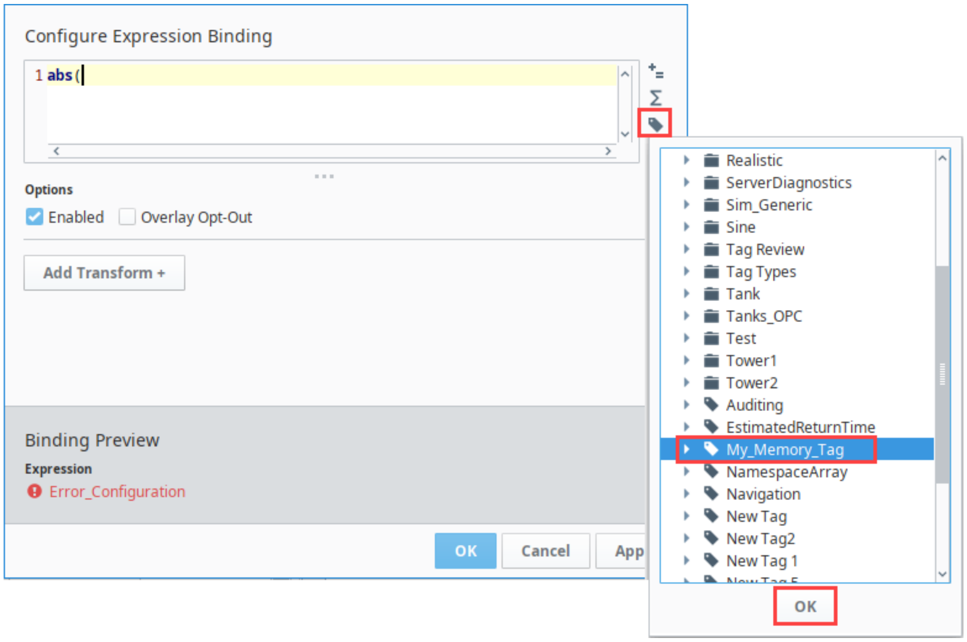
Example 2

Let's continue with the same view we set up in Example 1.

1. Select the second LED component and click on the **Binding** icon for the value property.
2. Select Expression the binding type.
3. In the Configure Expression Binding section, click on the **Functions** Σ icon. Scroll down to the **Math** functions and then select the **abs** (number) function.


Edit Binding: LedDisplay_0.props.value
Binding Type
Tag
Property
Expression
Expression Structure
Query
Tag History
HTTP
Remove Binding
Configure Expression Binding
1
Options
Enabled
Overlay Opt-Out
Add Transform +
Binding Preview
Expression
OK
Cancel
Apply
Advanced
Aggregates
Alarming
Colors
Date and Time
JSON
Logic
Math
Strings
Translation
Type Casting
Users
abs(number) returns Double
acos(number) returns Double
asin(number) returns Double
atan(number) returns Double
ceil(number) returns Double
cos(number) returns Double
exp(number) returns Double
floor(number) returns Double

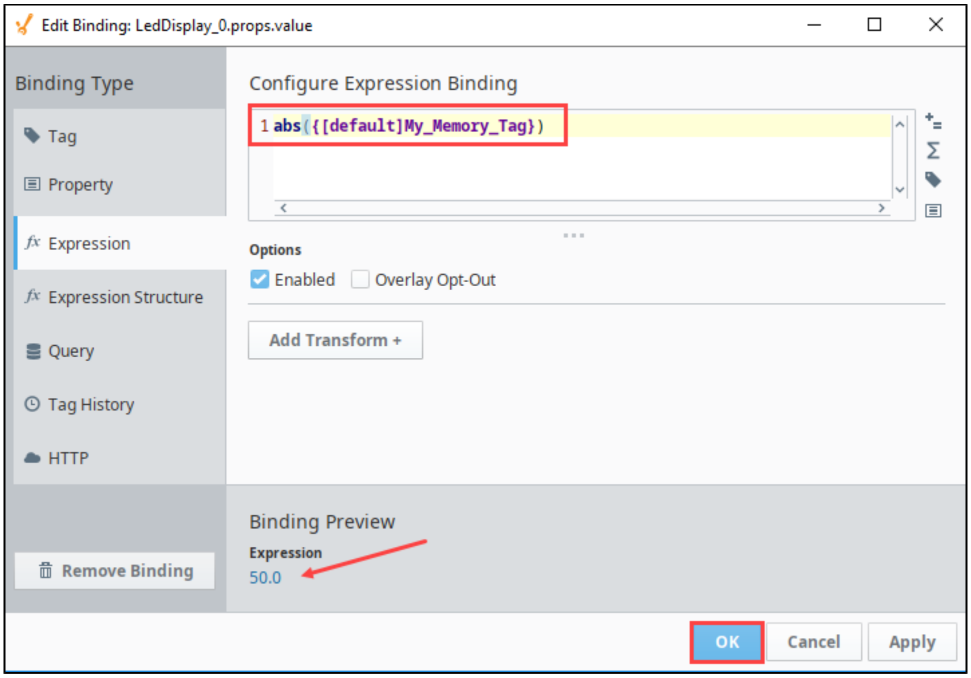
4. Next click the **Tag** icon and select the Memory Tag. Click **OK**.



```
1 abs (
```

Error_Configuration

5. Next close the function with a closed parenthesis). Note that a preview of the Expression binding value is shown on the lower left. Click **OK**.



Edit Binding: LedDisplay_0.props.value

Binding Type

- Tag
- Property
- Expression
- Expression Structure
- Query
- Tag History
- HTTP

Configure Expression Binding

```
abs({[default]My_Memory_Tag})
```

Options

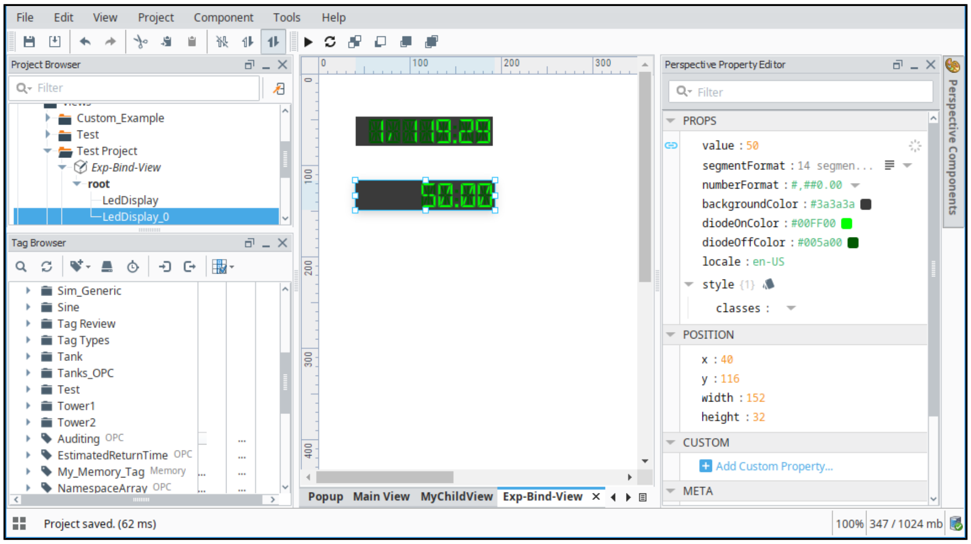
☑ Enabled □ Overlay Opt-Out

Binding Preview

Expression

50.0

6. The value is now displayed in the LED component.



PROPS

value : 50
segmentFormat : 14 segmen...
numberFormat : #,##0.00
backgroundColor : #3a3a3a
diodeOnColor : #00FF00
diodeOffColor : #005a00
locale : en-US
style (1)
classes :

POSITION

x : 40
y : 116
width : 152
height : 32

CUSTOM

Add Custom Property...

META

Project saved. (62 ms)

100% 347 / 1024 mb

Expression Structure Bindings in Perspective

What Is an Expression Structure?

An Expression structure is an **object** type of property where several of the sub-items get their values from expression bindings.

An Expression Structure bindings allow us to create a custom Expression binding where several expressions give you several values. That is, the output is an **object** rather than a **value**. This binding type is useful in configuring parameters for a script transform, or in cleanly creating a complex object from a single binding. It enables you to create a data structure using a separate expression to populate each entry in the structure.

Binding Interface

The Expression Structure binding is configured similarly to any object in Perspective, except that every **value** property in the object is evaluated via an individual expression.

Binding Properties

Property Name	Description
Enabled	Indicates whether the binding should fire.
Overlay Opt-Out	Indicates whether the component should reflect a bad quality binding via a tag overlay.
Wait On All	Indicates whether the binding should wait for every expression binding in the structure to finish before completing. If false, each expression in the structure will resolve individually and update their properties at that time. If true, all component properties will receive their new values at the same time.

Example

1. Create a new memory Tag called NewTag1. Set the following:
   Data Type: **String**
   Value: **It Works!**
2. Click **OK** to save the Tag.
3. Create View called MyParentView.
   a. Place a Carousel component and a Label component on the view.
   b. Set the label text as "Parent View."
4. Create another view called **MyChildView**.
5. Place a Label component and an Icon component on the view.
6. Now we need to add two view parameters MyChildView.

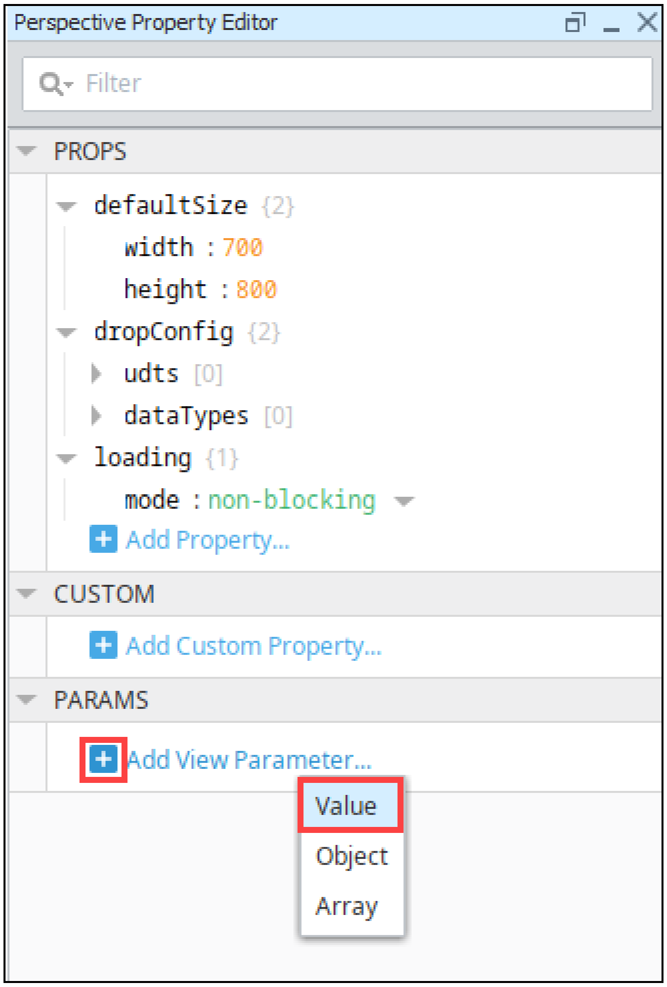
On this page ...

INDUCTIVE UNIVERSITY

Expression Structure Binding

Watch the Video

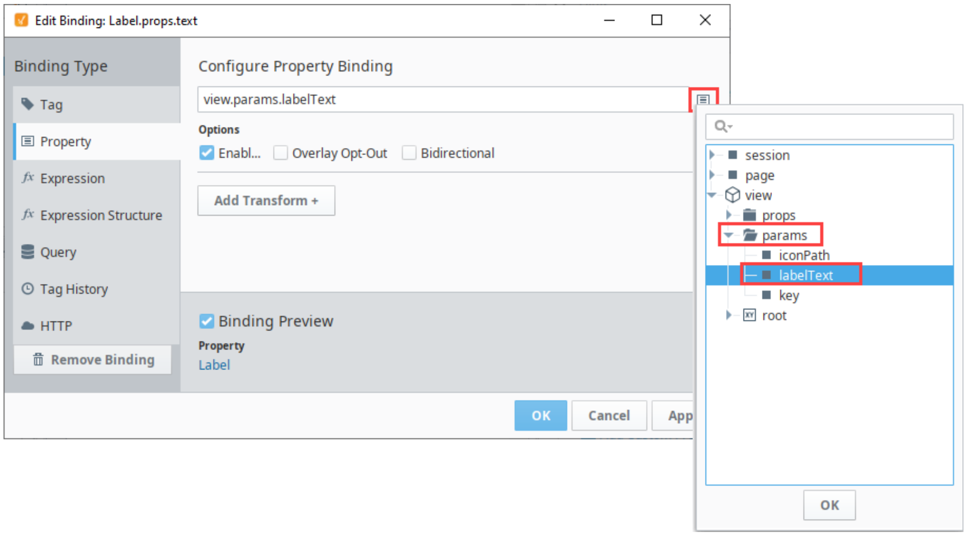
a. In the Property Editor under PARAMS, click **Add View** parameter and select the Value option.



defaultSize {2}
width : 700
height : 800
dropConfig {2}
udts [0]
dataTypes [0]
loading {1}
mode : non-blocking

b. Change the word "key" to the parameter name we want, which is "iconPath."
c. Click Add View Parameter again.
d. Name the second view parameter "labelText."

7. Next we'll bind the components to the view parameters. On MyChildView, select the Label component.
   a. Click the **Binding** 🔗 icon next to the text property.
   b. Select the Property binding type.
   c. Click the **Property Editor** ☰ icon.
   d. Scroll down to the labelText view parameter. Click **OK**. Click **OK** again to save the binding.



view.params.labelText

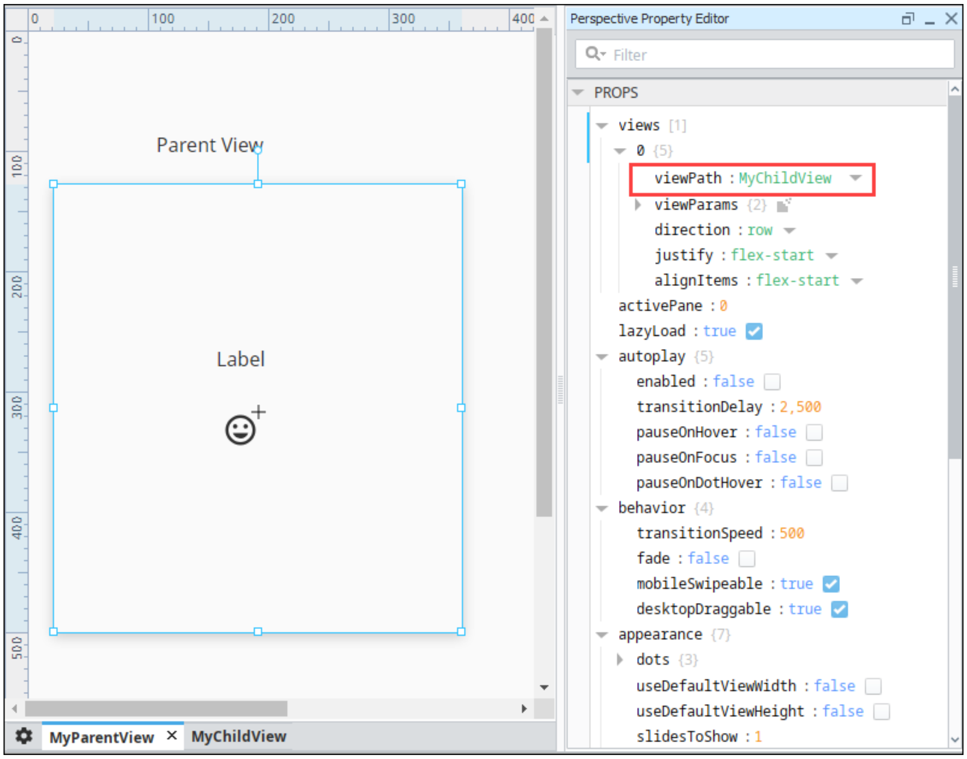
Options
☑ Enabl... ☐ Overlay Opt-Out ☐ Bidirectional

☑ Binding Preview
Property
Label

   e. Select the Icon component.
   f. Click the **Binding** 🔗 icon next to the path property.

g. Select the Property binding type.
h. Click the **Property Editor** ☰ icon.
i. Scroll down to the iconPath view parameter. Click **OK**. Click **OK** again to save the binding.

8. Back on MyParentView, select the Carousel component and set the viewPath property to **MyChildView**.



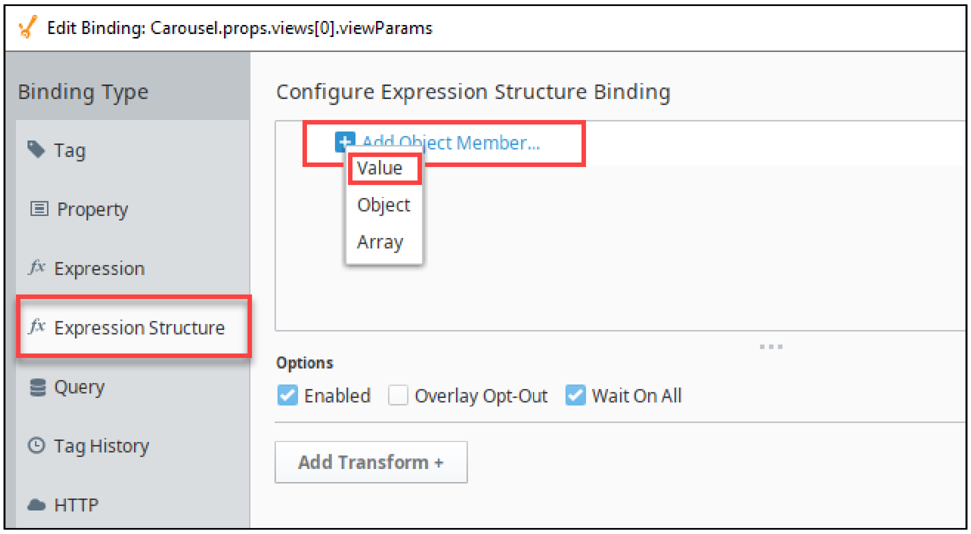
Perspective Property Editor

Filter

PROPS

```
views [1]
  0 {5}
    viewPath : MyChildView
    viewParams {2}
    direction : row
    justify : flex-start
    alignItems : flex-start
activePane : 0
lazyLoad : true ☑
autoplay {5}
  enabled : false ☐
  transitionDelay : 2,500
  pauseOnHover : false ☐
  pauseOnFocus : false ☐
  pauseOnDotHover : false ☐
behavior {4}
  transitionSpeed : 500
  fade : false ☐
  mobileSwipeable : true ☑
  desktopDraggable : true ☑
appearance {7}
  dots {3}
  useDefaultViewWidth : false ☐
  useDefaultViewHeight : false ☐
  slidesToShow : 1
```

9. Next we need to add two view parameters. Still on the MyParentView, click the **Binding** 🔗 icon next to **viewParams**. The Edit Binding screen is displayed.
10. Choose Expression Structure as the binding type. Click **Add Object Member...** and select **Value**.



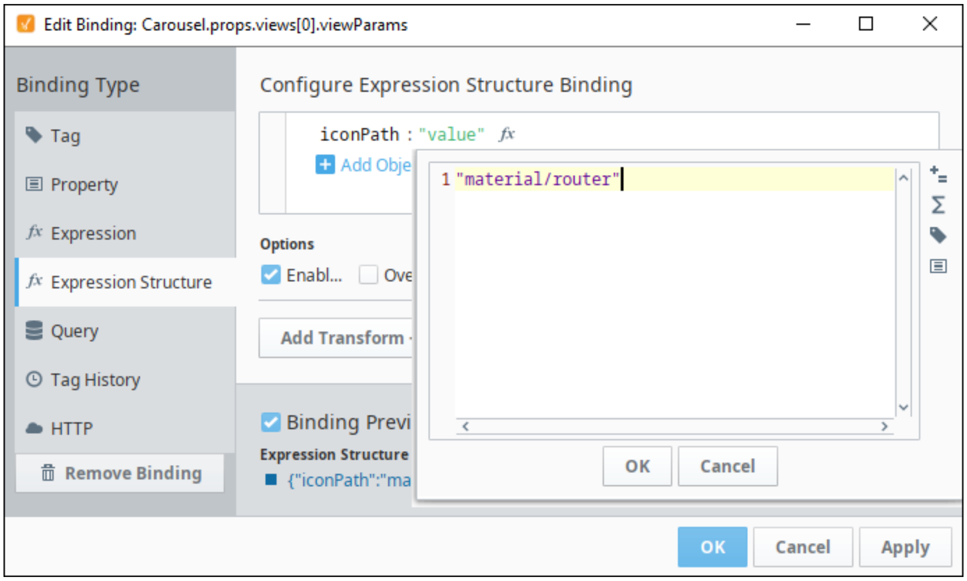
Edit Binding: Carousel.props.views[0].viewParams

Configure Expression Structure Binding

Options

☑ Enabled ☐ Overlay Opt-Out ☑ Wait On All

11. Name this parameter **iconPath**.
12. Click the **Expression** *fx* icon, then enter "material/insert_emoticon" as the expression. Click **OK**.



Edit Binding: Carousel.props.views[0].viewParams

Binding Type

- Tag
- Property
- Expression
- Expression Structure
- Query
- Tag History
- HTTP

Remove Binding

Configure Expression Structure Binding

```
iconPath : "value"
```

Add Obje

```
1 "material/router"
```

Options

☑ Enabl... ☐ Ove

Add Transform

☑ Binding Previ

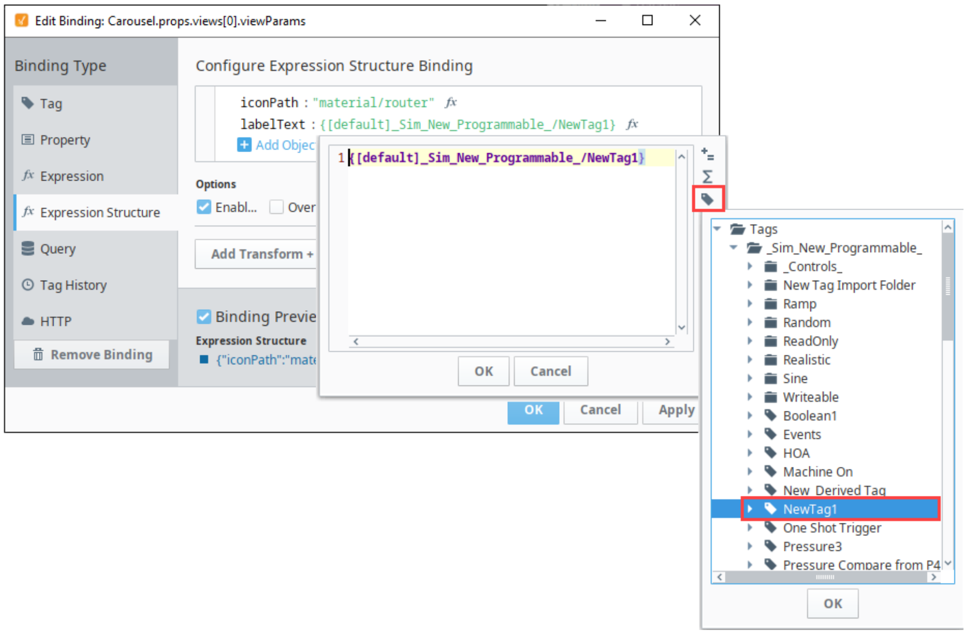
Expression Structure

{"iconPath":"ma

OK Cancel

OK Cancel Apply

13. Click the **Expand** icon and select Value to add another parameter.
14. Name this next parameter labelText then click the **Expression** *fx* icon.
15. For the value, click on the **Tag** icon then choose the tag. Click **OK**.



Edit Binding: Carousel.props.views[0].viewParams

Binding Type

- Tag
- Property
- Expression
- Expression Structure
- Query
- Tag History
- HTTP

Remove Binding

Configure Expression Structure Binding

```
iconPath : "material/router"
labelText : {[default]_Sim_New_Programmable_/NewTag1}
```

Add Objec

```
1 {[default]_Sim_New_Programmable_/NewTag1}
```

Options

☑ Enabl... ☐ Over

Add Transform +

☑ Binding Previe

Expression Structure

{"iconPath":"mate

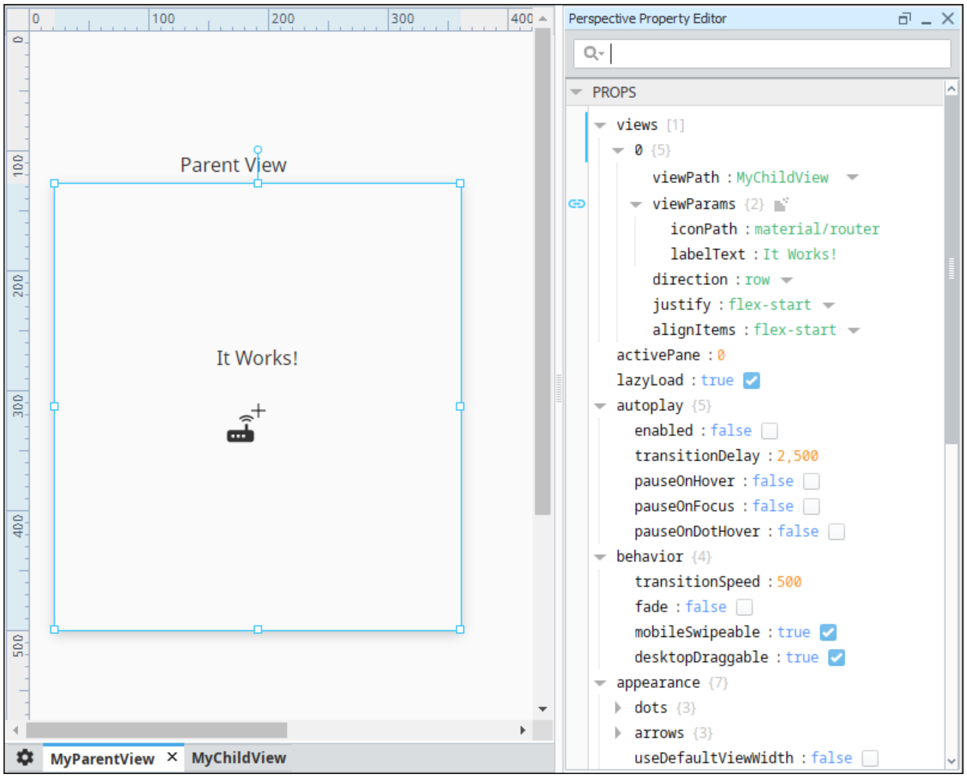
OK Cancel

OK Cancel Apply

- Tags
  - _Sim_New_Programmable_
    - _Controls_
    - New Tag Import Folder
    - Ramp
    - Random
    - ReadOnly
    - Realistic
    - Sine
    - Writeable
    - Boolean1
    - Events
    - HOA
    - Machine On
    - New Derived Tag
    - NewTag1
    - One Shot Trigger
    - Pressure3
    - Pressure Compare from P4

OK

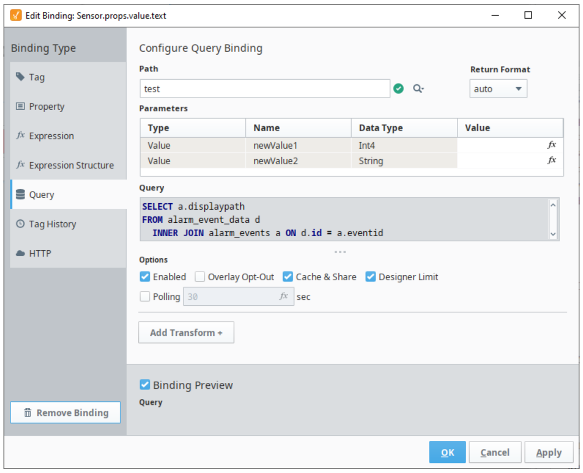
16. Select the **Wait On All** option. This will ensure that all expressions provide a value before this binding will publish its initial value.
17. Click **OK** to save the binding. Now the label text we're using in MyChildView will be populated by this tag (NewTag1).



```
views [1]
  0 {5}
    viewPath : MyChildView
    viewParams {2}
      iconPath : material/router
      labelText : It Works!
    direction : row
    justify : flex-start
    alignItems : flex-start
activePane : 0
lazyLoad : true ☑
autoplay {5}
  enabled : false □
  transitionDelay : 2,500
  pauseOnHover : false □
  pauseOnFocus : false □
  pauseOnDotHover : false □
behavior {4}
  transitionSpeed : 500
  fade : false □
  mobileSwipeable : true ☑
  desktopDraggable : true ☑
appearance {7}
  dots {3}
  arrows {3}
  useDefaultViewWidth : false □
```

Query Bindings in Perspective

The Query Binding allows you to pull data from the database using a named query. In Perspective, the Query Binding requires the use of Named Queries. You can't type a query here from this interface, a Named Query must already exist. You can also add transforms. For more information, see Transforms.



Edit Binding: Sensor.props.value.text

Binding Type: Tag, Property, Expression, Expression Structure, Query, Tag History, HTTP

Configure Query Binding

Path: test

Return Format: auto

Parameters

Type	Name	Data Type	Value
Value	newValue1	Int4	
Value	newValue2	String	

Query

```
SELECT a.displaypath
FROM alarm_event_data d
  INNER JOIN alarm_events a ON d.id = a.eventid
```

Options

☑ Enabled □ Overlay Opt-Out ☑ Cache & Share ☑ Designer Limit

□ Polling 30 sec

☑ Binding Preview

Query

Query Binding

Watch the Video

Binding Properties

Property Name	Description
Path	Here you can enter in the path to the Named Query. Click on the **Search** 🔍 icon to get a list of all available Named Queries.
Return Format	The Return format specifies how the query results are returned. Options are auto, json, dataset, or scalar. • **auto**: Query results are returned in the format native to the database (typically dataset). • **json**: Query results are returned in json format. • **dataset**: Query results are returned in dataset format. • **scalar**: Returns the first element from the query result. This format is best when a single value is expected.
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 icon or the Tag icon. **Note:** The fields under the Value column are evaluated as expressions, so strings will require quotations marks.
Query	In the query section, there is nothing to configure, but you can see what the Named Query you have selected looks like.
Options	
Enabled	Allows the component to be active/in use /interactive on the screen.
Overlay Opt-Out	If the target of the binding returns a non-good quality code, this setting indicates whether the component should reflect the quality code with a component overlay.
Bypass cache	This will cause the query to bypass/ignore any cached values from the Named Query and run every time it is called. This feature was changed in Ignition version **8.1.12**: This option was removed in 8.1.12 and replaced with Cache & Share
Cache & Share	

	The following feature is new in Ignition version **8.1.12** Click here to check out the other new features When this setting is enabled, the query will utilize a polling engine that is shared across all running Perspective sessions. The polling engine will run the query once, cache the value, and deliver the results to all registered consumers. The cache persists for a period of time that matches the configured poll rate. If polling is off, the default cache duration is 250 milliseconds. This optimization is helpful for consolidating multiple identical polling tasks, particularly when you have many Perspective sessions running at once. Disabling this option will bypass the cache.
Designer Limit	This setting will force the results of the query to be limited to a few rows when run in the Designer.
Polling	Here you can set the Polling Mode of the Named Query binding based on the Polling rate.

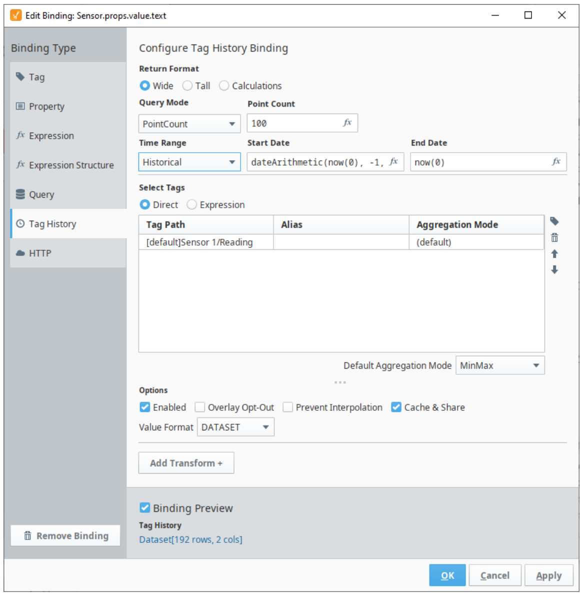
Related Topics ...

- Named Queries
- Transforms

Tag History Bindings in Perspective

Tag History Binding

Tag History Bindings allow you to pull Tag History data that is stored in the database into a component through a binding.



Configure Tag History Binding

Return Format: ● Wide ○ Tall ○ Calculations

Query Mode: PointCount

Point Count: 100

Time Range: Historical

Start Date: dateArithmetic(now(0), -1,

End Date: now(0)

Select Tags: ● Direct ○ Expression

Tag Path	Alias	Aggregation Mode
[default]Sensor 1/Reading		(default)

Default Aggregation Mode: MinMax

Options: ☑ Enabled □ Overlay Opt-Out □ Prevent Interpolation ☑ Cache & Share

Value Format: DATASET

☑ Binding Preview

Tag History

Dataset[192 rows, 2 cols]

On this page ...

INDUCTIVE UNIVERSITY

Tag History Binding

Watch the Video

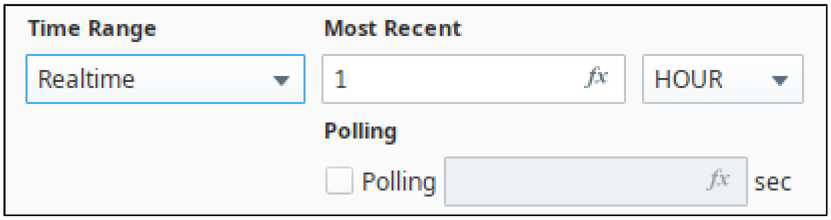
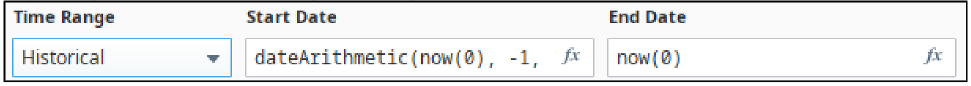
INDUCTIVE UNIVERSITY

Tag History Binding - Expression

Watch the Video

<table>
<tr><th>Configuration Property</th><th>Description</th></tr>
<tr><td>Return Format</td><td>Allows you to select the return format of the data. Possible options are:
<table>
<tr><th>Property</th><th>Description</th></tr>
<tr><td>Wide</td><td>Every column is a different tag, and each row is their values at different times.</td></tr>
<tr><td>Tall</td><td>There are columns for Value, Quality, Timestamp, and Path, and each row is a new tag value at a specific time.</td></tr>
<tr><td>Calculations</td><td>Will perform a calculation on the returned data, and return the calculated values instead. For example, using the Ave average of each tags value over the time range selected.</td></tr>
</table></td></tr>
<tr><td>Query Mode</td><td>How you want to query out the data. Possible options are:
<table>
<tr><th>Property</th><th>Description</th></tr>
<tr><td>PointCount</td><td>Will return the number of records defined in the Point Count property.</td></tr>
<tr><td>Periodic</td><td>Will return records separated by an amount of time specified in the Period property.</td></tr>
<tr><td>AsStored</td><td>Will return the records as stored in the database. While querying data with this mode, multiple value changes at the s rows, one row for each unique value.</td></tr>
</table></td></tr>
<tr><td></td><td></td></tr>
</table>

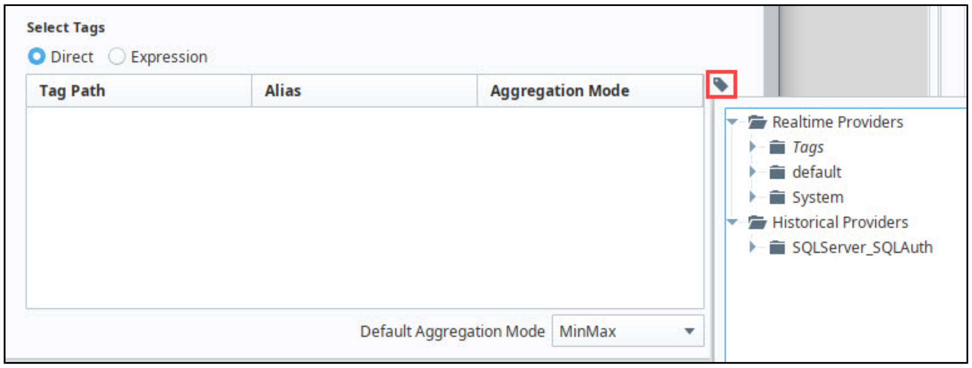
<table>
<tr><td>Time Range</td><td>The time range to pull data values from. Possible options are:
<table>
<tr><th>Property</th><th>Description</th></tr>
<tr><td>Realtime</td><td>The start date will go back as far from the current time as the Most Recent property specified and the end date will be
evaluates. Options are MS, SEC, MIN, HOUR, DAY, WEEK, MONTH, or YEAR.

Polling: You can specify a Polling rate to determine how often to update the times. Click the Functions 𝑓x icon to use
properties.

Time Range: Realtime
Most Recent: 1 HOUR
Polling
☐ Polling sec</td></tr>
<tr><td>Historical</td><td>You can specify the Start and End Date in an expression. Click the Functions 𝑓x icon to use operators, expressions,
only changed if bound to something that changes.

Time Range: Historical
Start Date: dateArithmetic(now(0), -1,
End Date: now(0)

⚠ Please note that intervals returned by Historical queries are inclusive of the End Date, including when the En
you may see one additional interval than expected that only contains future dates, which get interpolated to (

For example, if you want data from 10am - 11am in 1 minute windows, you'll need to set your query from 10a
would create an interval to contain it and that window will often return 0 since there is typically no future value
the results of two queries of adjoining times, such as 10am - 11am and 11am - 12pm, the first window of the
data to the last window of the first period.</td></tr>
</table></td></tr>
<tr><td>Select Tags</td><td>An area to select the Tags to trend. Tag Paths can be defined directly or using an expression. See <b>Using Dynamic Tag Paths</b> belo

The following feature is new in Ignition version <b>8.1.13</b>
Click here to check out the other new features

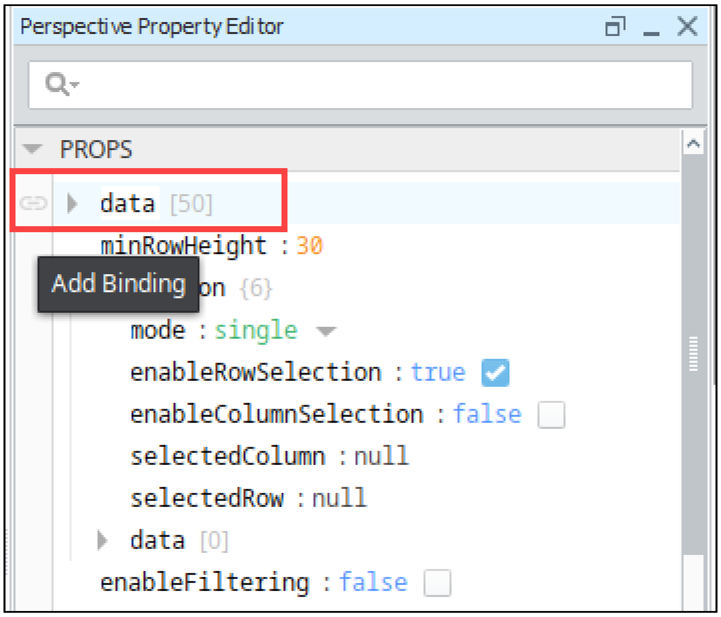
Users may now browse for and select tags from both Realtime and Historical Tag Providers.

Select Tags
● Direct ○ Expression
Tag Path | Alias | Aggregation Mode
Default Aggregation Mode: MinMax</td></tr>
<tr><td>Aggregation Mode</td><td>The aggregation mode that will be applied to each time slice, unless a more specific aggregation mode is defined on a Tag Path. Fo
populated, see the How the Tag Historian System Works page.
<table>
<tr><th>Aggregation Mode</th><th>Description</th></tr>
<tr><td>(default)</td><td>Use the mode set in the Default Aggregation Mode field.</td></tr>
<tr><td>Average</td><td>The values are averaged together, weighted for the amount of time they cover within a time slice.</td></tr>
<tr><td>MinMax</td><td>The minimum and maximum values will be returned for the window. In other words, two rows will be returned. If</td></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td></td><td>only one row will be returned.</td></tr>
<tr><td>LastValue</td><td>The value closest to the ending time of the time slice will be returned.</td></tr>
<tr><td>SimpleAverage</td><td>The values are summed together and divided by the number of values.</td></tr>
<tr><td>Sum</td><td>The values in the time slice are summed together.</td></tr>
<tr><td>Minimum</td><td>The minimum value in the time slice.</td></tr>
<tr><td>Maximum</td><td>The maximum value in the time slice.</td></tr>
<tr><td>DurationOn</td><td>Returns the number of seconds that the value was recorded as non-zero.</td></tr>
<tr><td>DurationOff</td><td>Returns the number of seconds that the value recorded as zero.</td></tr>
<tr><td>CountOn</td><td>Returns the number of times the tag's value went from a zero value to non-zero.</td></tr>
<tr><td>CountOff</td><td>Returns the number of times the tag's value changed from a non-zero value to zero.</td></tr>
<tr><td>Count</td><td>Returns the number of times a value was recorded</td></tr>
<tr><td>Range</td><td>Returns the range between the highest and lowest value for the period.
This feature was changed in Ignition version <b>8.1.17</b>:
Range mode will return "0" if the historical tag value remains static over the given Time Range.</td></tr>
<tr><td>Variance</td><td>Returns the variance of values. Similar in concept to standard deviation. Only good quality values are used wher</td></tr>
<tr><td>StdDev</td><td>Returns the standard deviation of values, or how much spread is present in the data; low standard deviation sho and high standard deviation shows that the data points are spread out over a large range of values. Only good q</td></tr>
<tr><td>PctGood</td><td>Time-weighted percentage of good values over the date range.</td></tr>
<tr><td>PctBad</td><td>Time-weighted percentage of bad values over the date range.</td></tr>
</table>
</td></tr>
<tr><td>Default Aggregation Mode</td><td>Aggregation mode to use as a default if the Select Tags are set to Default Aggregation mode.</td></tr>
<tr><td>Options</td><td>Allows you to specify various options that will apply to the binding.
<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>Enabled</td><td>Enable these options.</td></tr>
<tr><td>Overlay Opt-Out</td><td>Opt out of displaying the Tag quality overlay.</td></tr>
<tr><td>Ignore Bad Quality</td><td>Only data with "good" quality will be loaded from the data source.</td></tr>
<tr><td>Prevent Interpolation</td><td>Requests that values not be interpolated, if the row would normally require it. Also instructs the system to not write resu interpolated values. In other words, if the raw data does not provide any new values for a certain window, that window</td></tr>
<tr><td>Cache &amp; Share</td><td>The following feature is new in Ignition version <b>8.1.12</b>
Click here to check out the other new features
When this setting is enabled, the binding will utilize a polling engine that is shared across all running Perspective sessic cache the value, and deliver the results to all registered consumers. The cache persists for a period of time that matche off, the default cache duration is 250 milliseconds. This optimization is helpful for consolidating multiple identical polling many Perspective sessions running at once. Disabling this option will bypass the cache.</td></tr>
<tr><td>Value Format</td><td>Can be Dataset or Document.</td></tr>
</table>
</td></tr>
</table>

Tag History Binding Example

In this example, we'll use a Table component to show the records from some Tags that have Tag History enabled.

1. Drag a Table component onto a view. Select the Table component.
2. In the Property Editor, click on the **Binding** icon next to the **data** property. The Edit Binding screen is displayed.

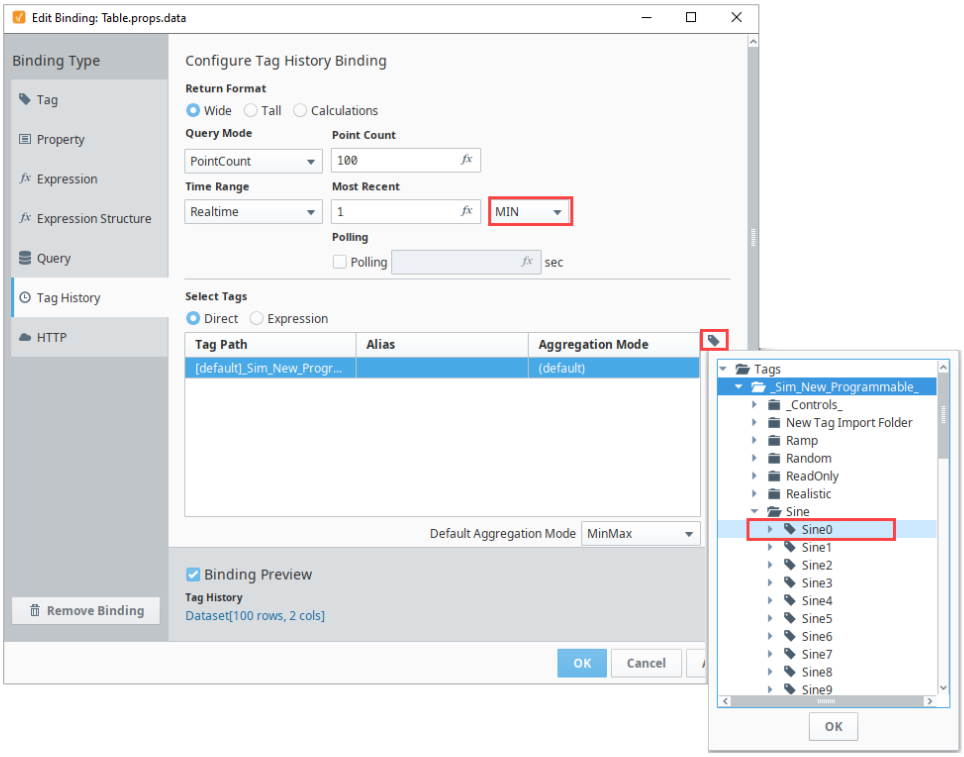


Perspective Property Editor

PROPS

- data [50]
- minRowHeight : 30
- Add Binding
- on {6}
  - mode : single
  - enableRowSelection : true ☑
  - enableColumnSelection : false □
  - selectedColumn : null
  - selectedRow : null
  - data [0]
- enableFiltering : false □

3. Select **Tag History** as the binding type.
4. In the **Time Range** section, select and the last 1 minute of data.
5. Under the **Select Tags** section, click on the **Tag** icon. Navigate to the first Tag and click **OK**. Repeat for additional Tags. In our example, we chose the Sine0 and Sine2 Tags.



Edit Binding: Table.props.data

Configure Tag History Binding

Return Format: ● Wide ○ Tall ○ Calculations

Query Mode: PointCount

Point Count: 100

Time Range: Realtime

Most Recent: 1 MIN

Polling: □ Polling sec

Select Tags: ● Direct ○ Expression

Tag Path	Alias	Aggregation Mode
[default]_Sim_New_Progr...		(default)

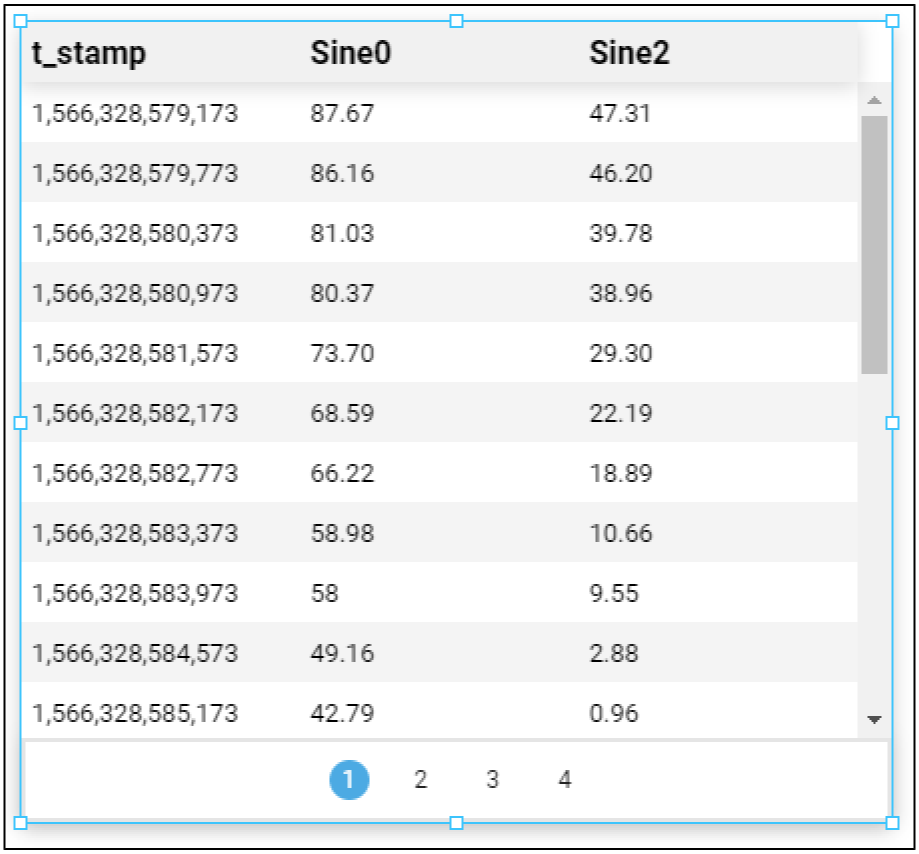
Default Aggregation Mode: MinMax

☑ Binding Preview

Tag History

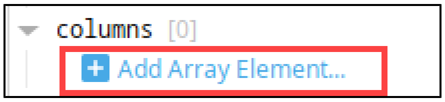
Dataset[100 rows, 2 cols]

6. Notice the Binding Preview at the bottom of the screen. Click **OK** to save the Tag History binding.
7. Back in the view, the Table component now contains a column for the timestamp and one for each Tag. Notice how the timestamp does not have a date format.

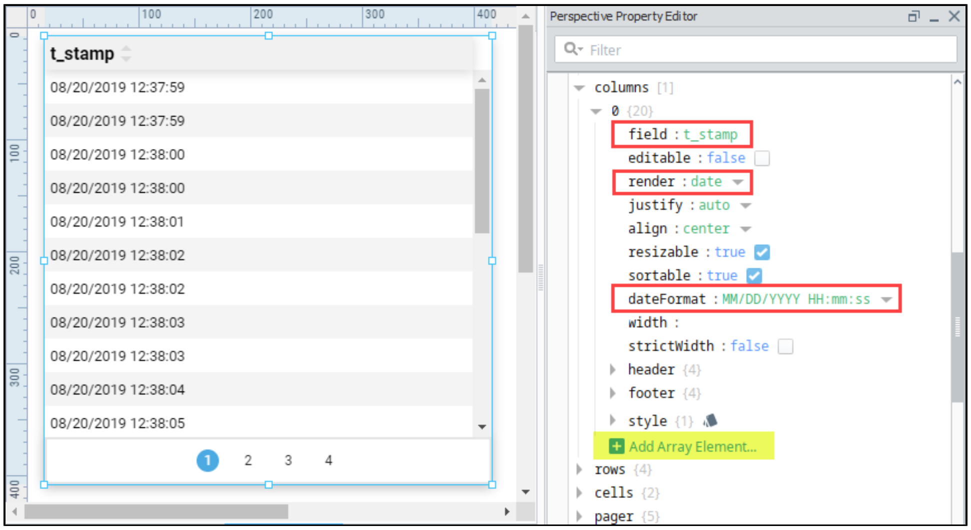


t_stamp	Sine0	Sine2
1,566,328,579,173	87.67	47.31
1,566,328,579,773	86.16	46.20
1,566,328,580,373	81.03	39.78
1,566,328,580,973	80.37	38.96
1,566,328,581,573	73.70	29.30
1,566,328,582,173	68.59	22.19
1,566,328,582,773	66.22	18.89
1,566,328,583,373	58.98	10.66
1,566,328,583,973	58	9.55
1,566,328,584,573	49.16	2.88
1,566,328,585,173	42.79	0.96

8. In order to change the t-stamp values into a date format, in the Property Editor under columns, click **Add Array Element...**



9. You'll notice that once the Array Element is added, the other two columns (Sine0 and Sine2) disappeared. We'll add them back in Step 10.
   a. In the **field** property, enter **t_stamp** (which is the column name).
   b. For the **render** property, select **date** from the dropdown.
   c. Select a **dateFormat** from the dropdown. In this example, we used the date time format.

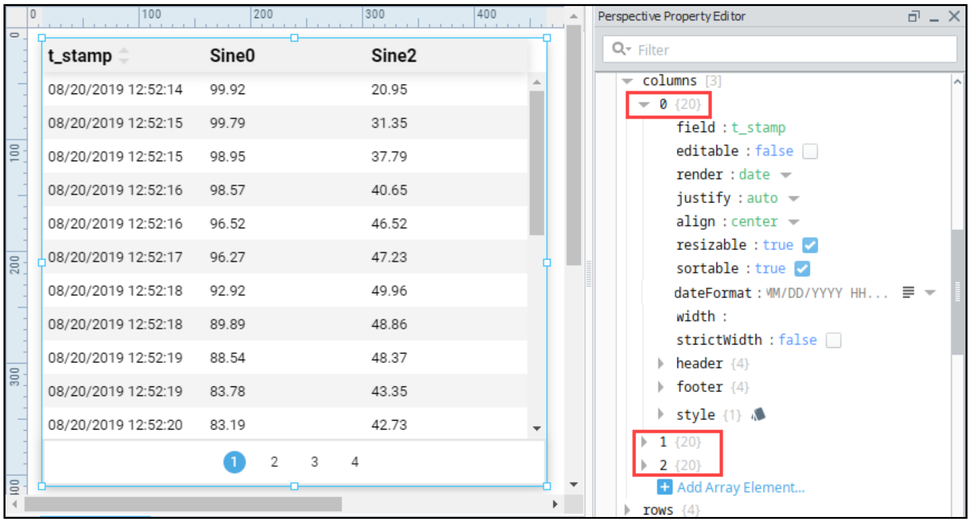


t_stamp
08/20/2019 12:37:59
08/20/2019 12:37:59
08/20/2019 12:38:00
08/20/2019 12:38:00
08/20/2019 12:38:01
08/20/2019 12:38:02
08/20/2019 12:38:02
08/20/2019 12:38:03
08/20/2019 12:38:03
08/20/2019 12:38:04
08/20/2019 12:38:05

```
columns [1]
  0 {20}
    field : t_stamp
    editable : false ☐
    render : date
    justify : auto
    align : center
    resizable : true ☑
    sortable : true ☑
    dateFormat : MM/DD/YYYY HH:mm:ss
    width :
    strictWidth : false ☐
    header {4}
    footer {4}
    style {1}
  Add Array Element...
rows {4}
cells {2}
pager {5}
```

10. Now, let's add the Sine0 and Sine2 columns back into our table by clicking on **Add Array Elements...** 2 times (refer to the yellow highlights in the image above). There is no need to change any of the Sine0 or Sine2 column properties unless you want to change the default settings or rearrange columns.

Notice that columns 1 and 2 (Sine0 and Sine2, respectively) are collapsed in this image. To see all the columns properties, expand the columns.



t_stamp	Sine0	Sine2
08/20/2019 12:52:14	99.92	20.95
08/20/2019 12:52:15	99.79	31.35
08/20/2019 12:52:15	98.95	37.79
08/20/2019 12:52:16	98.57	40.65
08/20/2019 12:52:16	96.52	46.52
08/20/2019 12:52:17	96.27	47.23
08/20/2019 12:52:18	92.92	49.96
08/20/2019 12:52:18	89.89	48.86
08/20/2019 12:52:19	88.54	48.37
08/20/2019 12:52:19	83.78	43.35
08/20/2019 12:52:20	83.19	42.73

```
columns [3]
  0 {20}
    field : t_stamp
    editable : false ☐
    render : date
    justify : auto
    align : center
    resizable : true ☑
    sortable : true ☑
    dateFormat : MM/DD/YYYY HH...
    width :
    strictWidth : false ☐
    header {4}
    footer {4}
    style {1}
  1 {20}
  2 {20}
rows {4}
```

Editor notes are only visible to logged in users
Tag History Binding - Expression

This is supplemental video. Not sure if we need to put a full example in here (complicated). It can possibly be lower priority for now.

Using Dynamic Tag Paths

Tag History bindings have the option to list out Tag paths, or to use an expression to build a Tag path. It is common to create a dynamic path or set of paths as a component property, that you then reference in other places. You can use as many Tag paths as you want, but they must all follow this format:

- **key** [array]
    - **[0]** {object}
        - **aggregate** value
        - **alias** value
        - **path** value
    - **[1]**{object}
        - **aggregate** value
        - **alias** value
        - **path** value
    - ...

For an example, you can create a custom property on a chart and use it to fuel the historical data.

1. Create a custom property named **Key** on a chart component that is an **Array** type.
2. Copy and paste the JSON below into this Key property.

JSON for Key Array

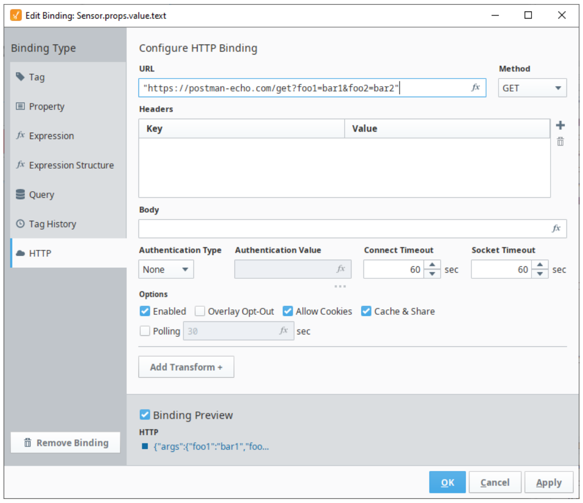
```
[
  {
    "aggregate": "Average",
    "alias": "tank_temp",
    "path": "[default]Tank/03/Temperature"
  },
  {
    "aggregate": "Average",
    "alias": "setpoint",
    "path": "[default]Tank/03/Setpoint"
  }
]
```

3. Open the property binding on your chart and select the **Tag History** binding type.

4. Select the **Expression** radio button to create your own tag structure.
5. Click on the property selector button on the right and find your new custom property.
6. Click **OK** in the lower right.

HTTP Bindings in Perspective

The HTTP Binding type allows you to use HTTP get/post protocols to interface with other API's.



Configure HTTP Binding

URL: "https://postman-echo.com/get?foo1=bar1&foo2=bar2"

Method: GET

Headers

Key	Value

Body:

Authentication Type: None

Authentication Value:

Connect Timeout: 60 sec

Socket Timeout: 60 sec

Options

☑ Enabled ☐ Overlay Opt-Out ☑ Allow Cookies ☑ Cache & Share

☐ Polling 30 sec

☑ Binding Preview

HTTP

■ {"args":{"foo1":"bar1","foo...

On this page ...

INDUCTIVE UNIVERSITY

HTTP Binding

Watch the Video

JSON Support

One perk of the HTTP binding is the ability to fetch JSON documents from a website or an API. Since the Perspective property tree is also JSON-formatted, this allows you to dynamically create a property structure from a JSON document directly.

HTTP Binding Configuration

<table>
<tr><th>Binding Property</th><th>Description</th></tr>
<tr><td>URL</td><td>An expression indicating what web address to reference on the binding. If entering a static URL, quotation marks must be used.</td></tr>
<tr><td>Method</td><td>Any HTTP method. Used to send HTTP requests to the specified URL. Options as follows:
<table>
<tr><th>Method</th><th>Definition</th></tr>
<tr><td>GET</td><td>The GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URI.</td></tr>
<tr><td>HEAD</td><td>The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response.</td></tr>
<tr><td>POST</td><td>The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line.</td></tr>
<tr><td>PUT</td><td>The PUT method requests that the enclosed entity be stored under the supplied Request-URI.</td></tr>
<tr><td>DELETE</td><td>The DELETE method requests that the origin server delete the resource identified by the Request-URI.</td></tr>
<tr><td>TRACE</td><td>The TRACE method is used to invoke a remote, application-layer loop- back of the request message.</td></tr>
<tr><td>CONNECT</td><td>The CONNECT method starts two-way communications with the requested resource. It can be used to open a tunnel.</td></tr>
</table></td></tr>
<tr><td>Headers</td><td>Used to pass key/value pairs in the header of our HTTP requests.
<table>
<tr><th>Field</th><th>Definition</th></tr>
<tr><td><b>Key</b></td><td>Allows dropdown selection from common header keys, or the ability to specify a custom one.</td></tr>
<tr><td><b>Value</b></td><td>The Value field is an expression.</td></tr>
</table></td></tr>
<tr><td></td><td></td></tr>
</table>

Body	An expression indicating what to send in the body of our HTTP requests.
Authentication Type	Indicates what HTTP authentication type to use. Equivalent to specifying the **Authorization** key in the header. Options are None, Basic, Bearer, or Digest.
Authentication Value	The **Authentication Value** field takes an expression that should indicate what authorization string or token should be used in combination with the associated authentication type. For example, if the header should contain the string: `Authorization: Basic aWduaXRpb246cGFzc3dvcmQ=` Then the Authentication Type should be **Basic** and the Authentication Value should be **aWduaXRpb246cGFzc3dvcmQ=**.
Connect Timeout	Indicates how long the Ignition Gateway should wait for a response to our connect request.
Socket Timeout	Indicates how long the Ignition Gateway should wait for a response to a given HTTP request.
Enabled	Indicates whether the binding should be active.
Overlay Opt-Out	Indicates whether the component should reflect a bad quality binding via a Tag overlay.
Allow Cookies	Indicates whether to allow the remote web server to store cookies.
Cache & Share	The following feature is new in Ignition version **8.1.12** Click here to check out the other new features When this setting is enabled, the binding will utilize a polling engine that is shared across all running Perspective sessions. The polling engine will poll once, cache the value, and deliver the results to all registered consumers. The cache persists for a period of time that matches the configured poll rate. If polling is off, the default cache duration is 250 milliseconds. This optimization is helpful for consolidating multiple identical polling tasks, particularly when you have many Perspective sessions running at once. Disabling this option will bypass the cache.
Polling	Controls how frequently HTTP requests should be issued, and therefore how often the binding should be updated.

MongoDB Bindings in Perspective

The MongoDB Cloud Connector module introduces a Perspective Binding type called MongoDB. The MongoDB binding follows the same format as the Perspective Expression Structure binding by allowing users to to bind JSON data to individual aspects of a Perspective project such as properties. In other words, building a MongoDB binding query is almost identical to building an Expression Structure binding.

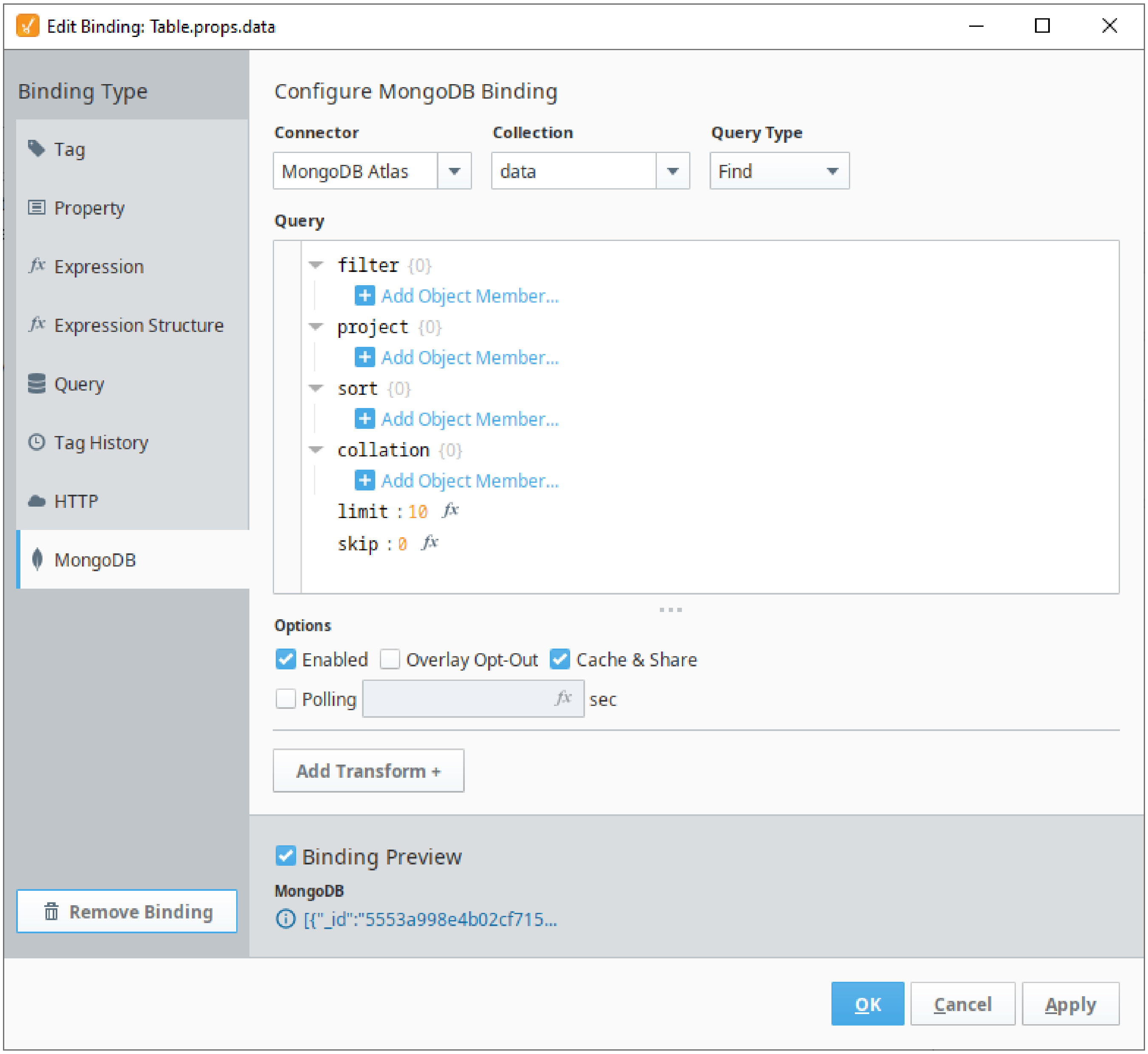
Users can set polling on a MongoDB binding, updating the displayed result at set intervals.

MongoDB bindings are read-only, and will not allow any writes to your bound MongoDB collection. If you need to write to a collection or document, use the MongoDB Cloud Connector's included system functions instead.

On this page ...

MongoDB Binding Interface

The image and table below detail various elements of the MongoDB binding user interface.



Configure MongoDB Binding

Connector: MongoDB Atlas
Collection: data
Query Type: Find

Query

```
filter {0}
project {0}
sort {0}
collation {0}
limit : 10
skip : 0
```

Options

☑ Enabled ☐ Overlay Opt-Out ☑ Cache & Share

☐ Polling sec

☑ Binding Preview

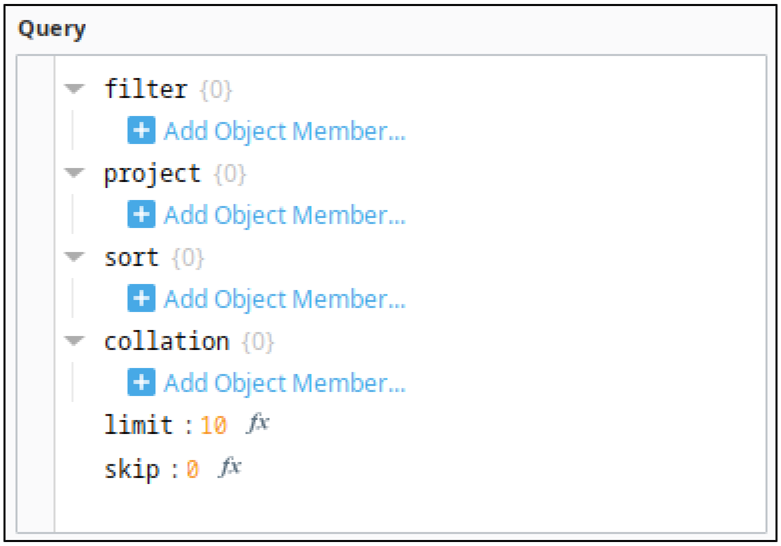
MongoDB

[{"_id":"5553a998e4b02cf715...

Source and Query Type Selection

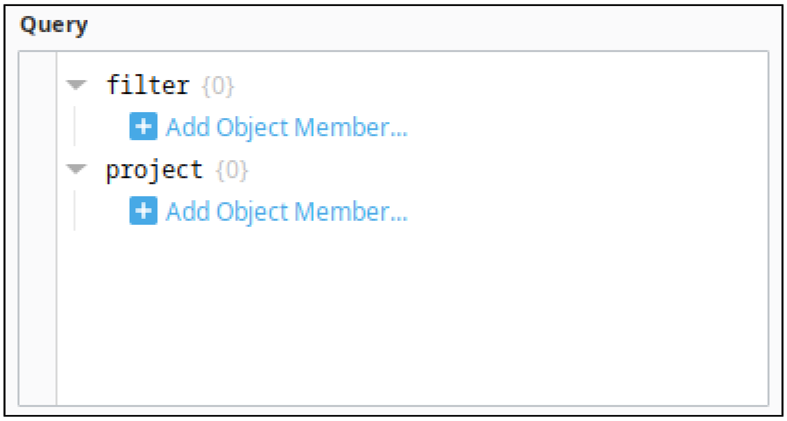
The three dropdown menus at the top of the MongoDB binding window allow you to specify where the binding's data is coming from and what you want to do with the information.

- Connector
  - The Connector dropdown will show available configured connections to your MongoDB databases.
- Collection
  - The Collection dropdown will display a list of available collections from the specified connection's database.
- Query Type
  - The Query Type dropdown will offer different actions, including Find, FindOne, or Aggregate. The Query Builder will also change depending on what Query Type you select:
    - Find contains filter, project, sort, collation, limit, and skip.



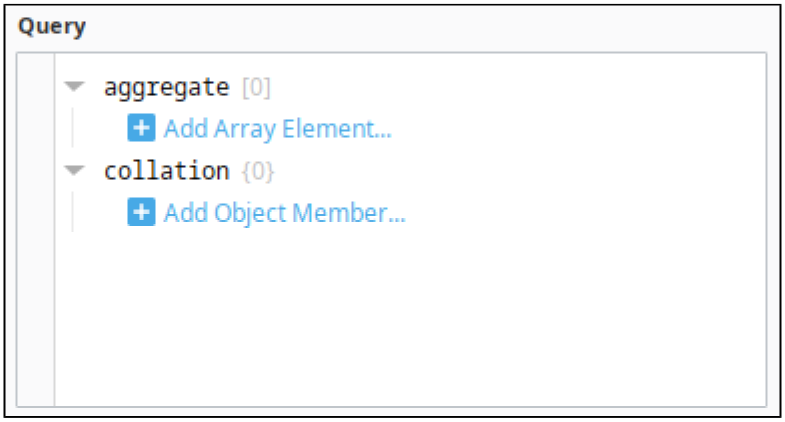
```
Query
filter {0}
    Add Object Member...
project {0}
    Add Object Member...
sort {0}
    Add Object Member...
collation {0}
    Add Object Member...
limit : 10
skip : 0
```

- FindOne contains filter and project.



```
Query
filter {0}
    Add Object Member...
project {0}
    Add Object Member...
```

- Aggregate contains aggregate and collation.



```
Query
aggregate [0]
    Add Array Element...
collation {0}
    Add Object Member...
```

Binding Options

Option	Description
Enabled	Indicates whether the binding should fire.
Overlay Opt-Out	Indicates whether the component should reflect a bad quality binding via a Tag overlay.
Cache & Share	When this setting is enabled, the query will utilize a polling engine that is shared across all running Perspective sessions. The polling engine will run the query once, cache the value, and deliver the results to all registered consumers. The cache persists for a period of time that matches the configured poll rate. If polling is off, the default cache duration is 250 milliseconds. This optimization is helpful for

	consolidating multiple identical polling tasks, particularly when you have many Perspective sessions running at once. Disabling this option will bypass the cache.
Polling	Controls how often the binding should update.

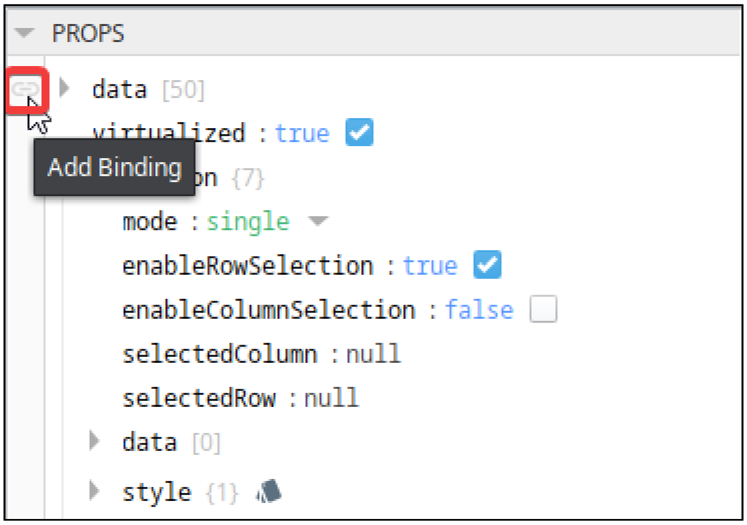
MongoDB Expression Functions

The MongoDB Cloud Connector module also adds three new expression functions to assist in building Perspective bindings expression structure. See the MongoDB expression functions for more details.

Example

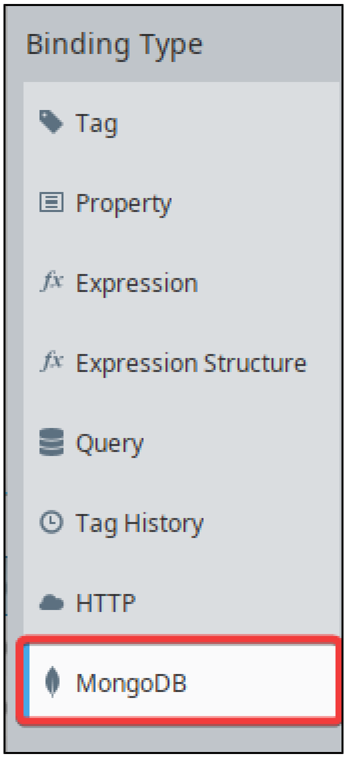
The following example pulls data from a MongoDB Atlas connection to display in a Perspective Table.

1. Set up a MongoDB connection, if you haven't already. The steps outlined on the MongoDB User Manual page can help guide you through setting up a connection.
2. Click on the **Binding** 🔗 icon next to the data property.

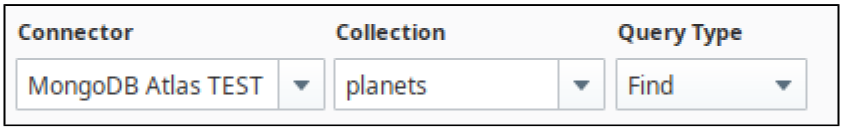


```
PROPS
  data [50]
  virtualized : true ☑
  ...on {7}
    mode : single
    enableRowSelection : true ☑
    enableColumnSelection : false ☐
    selectedColumn : null
    selectedRow : null
    data [0]
    style {1}
```

3. Select the **MongoDB** binding type.

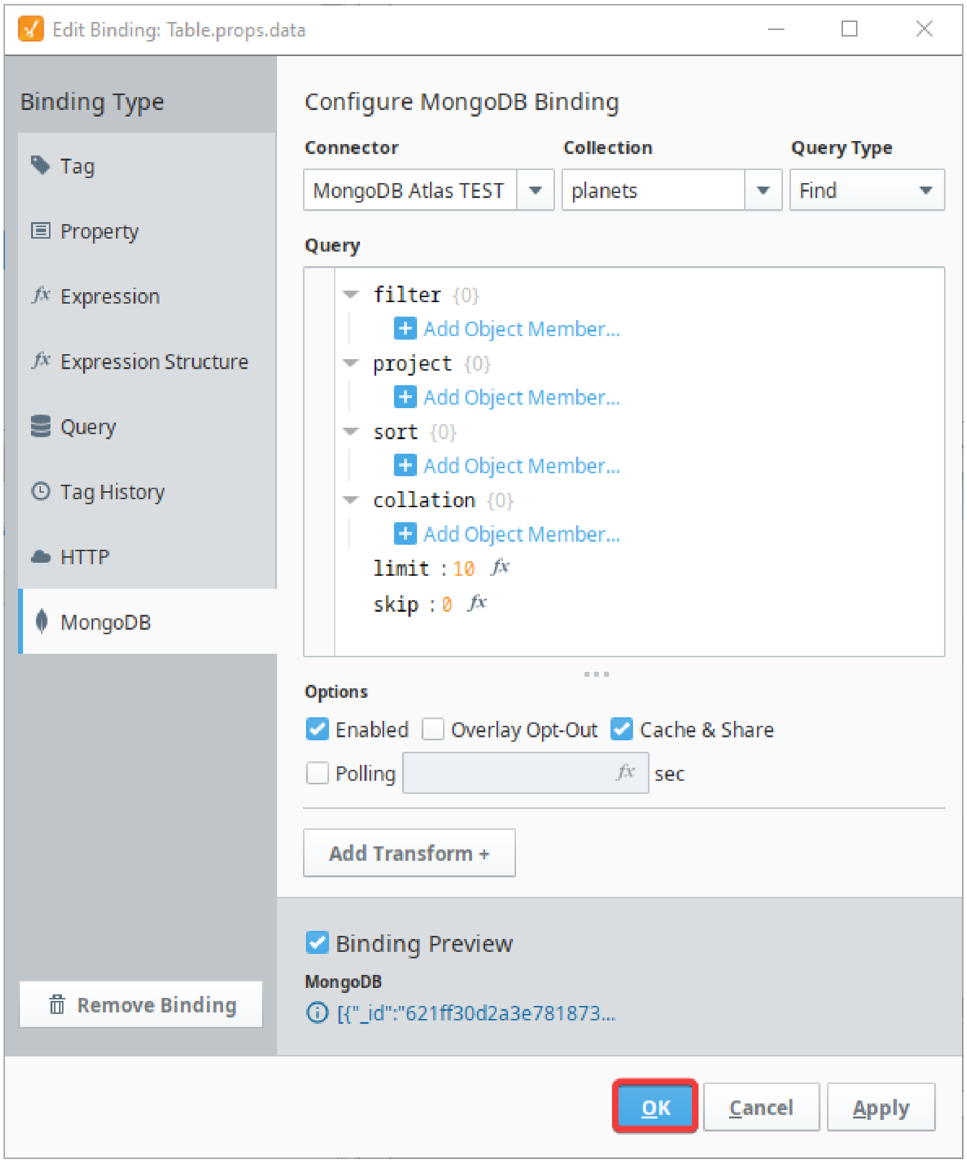


4. Select the **Connector**, **Collection**, and **Query Type** you want to use. In this example, the Connector we are using is called `MongoDB Atlas TEST`, the Collection we are pulling data from is called `planets`, and we are using a `Find` Query Type.



Connector	Collection	Query Type
MongoDB Atlas TEST	planets	Find

5. You can use the Query Builder to customize your data how you want. In this example, we are pulling in all the data in the `planets` collection.
6. Toggle or set any options you want, such as Polling or Cache & Share.
7. Once you have finished building your MongoDB binding, click **OK**.



Configure MongoDB Binding

Connector: MongoDB Atlas TEST
Collection: planets
Query Type: Find

Query

```
filter {0}
    Add Object Member...
project {0}
    Add Object Member...
sort {0}
    Add Object Member...
collation {0}
    Add Object Member...
limit : 10
skip : 0
```

Options

☑ Enabled □ Overlay Opt-Out ☑ Cache & Share

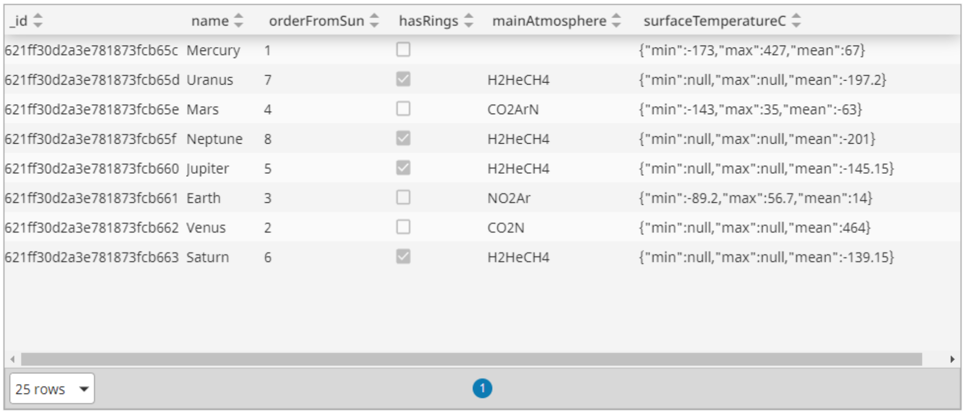
□ Polling sec

☑ Binding Preview

MongoDB

[{"_id":"621ff30d2a3e781873...

8. Our Perspective Table is now populated with data from our `planets` collection:



_id	name	orderFromSun	hasRings	mainAtmosphere	surfaceTemperatureC
621ff30d2a3e781873fcb65c	Mercury	1	□		{"min":-173,"max":427,"mean":67}
621ff30d2a3e781873fcb65d	Uranus	7	☑	H2HeCH4	{"min":null,"max":null,"mean":-197.2}
621ff30d2a3e781873fcb65e	Mars	4	□	CO2ArN	{"min":-143,"max":35,"mean":-63}
621ff30d2a3e781873fcb65f	Neptune	8	☑	H2HeCH4	{"min":null,"max":null,"mean":-201}
621ff30d2a3e781873fcb660	Jupiter	5	☑	H2HeCH4	{"min":null,"max":null,"mean":-145.15}
621ff30d2a3e781873fcb661	Earth	3	□	NO2Ar	{"min":-89.2,"max":56.7,"mean":14}
621ff30d2a3e781873fcb662	Venus	2	□	CO2N	{"min":null,"max":null,"mean":464}
621ff30d2a3e781873fcb663	Saturn	6	☑	H2HeCH4	{"min":null,"max":null,"mean":-139.15}

Note: You may need to add the column names from your target collection in order for users to be able to perform actions such as column resizing or reordering. See the column row from the Properties section of the Perspective Table User Manual page for more details.

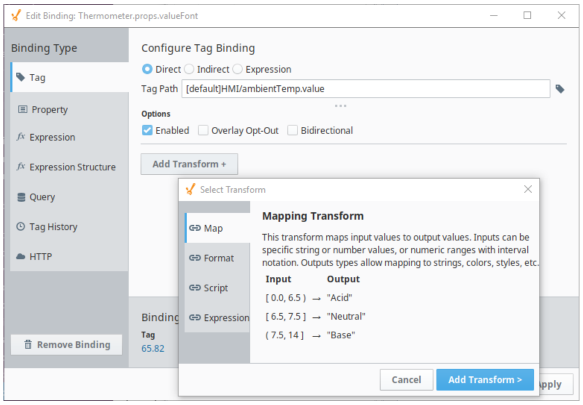
Transforms

On this page ...

Transforms offer a chance to alter the value returned from a binding. For example, you can bind a property to an integer value and use a transform to map the numerical value to a particular color, all from the same interface.

Transforms can be used as a way of splitting up complex expressions. You can make several simple expressions and chain them together as several transforms to manipulate your original value.

When multiple transforms are applied to a single binding, they are executed in order from top to bottom.



Configure Tag Binding

● Direct ○ Indirect ○ Expression

Tag Path [default]HMI/ambientTemp.value

Options

☑ Enabled □ Overlay Opt-Out □ Bidirectional

Binding

Tag

65.82

Mapping Transform

This transform maps input values to output values. Inputs can be specific string or number values, or numeric ranges with interval notation. Outputs types allow mapping to strings, colors, styles, etc.

Input		Output
[0.0, 6.5)	→	"Acid"
[6.5, 7.5]	→	"Neutral"
(7.5, 14]	→	"Base"

Map

The Map Transform allows you to set up a map of input values to output values. Inputs can be anything from specific values or strings, to numeric ranges of values. Each input value can then be mapped to an output value, such as a string, integer, color, style, etc. A great example of this is mapping values to colors, so that you can easily show state changes.

Format

The Format Transform applies a format string to the value returned from the binding, allowing you to format the output in any way. The format can be something like a date format or a number format.

Script

The Script Transform will run a script that allows you to manipulate the value returned from the binding using any Python script you want.

Expression

The Expression Transform runs an expression on the original value so you can manipulate it without creating more complex logic. This is most useful when you are indirectly binding to a Tag but still want to apply an expression to it, it can now be done in two distinct steps.

In This Section ...

Map Transform

The Map Transform allows you to set up a map of input values to output values. Inputs can be anything from specific values or strings, to numeric ranges of values. Each input value can then be mapped to an output value, which can be a string, integer, color, style, etc. A great example of this is mapping specific values to specific colors, so that you can easily bind visual properties to tag values.

On this page ...

INDUCTIVE UNIVERSITY

Map Transform

Watch the Video

The mapping table defaults to a value for both the input and output, but they can both be changed to one of several types. Select the **Input Type** and **Output type** for the mapping using the pull down arrows next to the Input/Output Type headers to make your selections.

Input Type

Input Type is The value coming into the transform (from the binding or previous transform)

- **Value:** Individual numeric values designed to exactly match the input value.
- **Numeric Range:** A range of values that the incoming value will fall between. Use brackets [x,y] to indicate values that are to be inclusive, or parenthesis for values that are exclusive (x,y), and you can mix them as needed. You can also omit start or end values to indicate no end to the range. See the examples below for clarification.
- **Expression:** Used in a similar manner to a Value Input Type. Use the Expression language to create a Value.

Output Type

Output Type is the outgoing value from the transform.

- **Value:** An alphanumeric value.
- **Color:** A color.
- **Expression:** A value result calculated by an expression.
- **Document:** A manually created JSON document (You can also copy an existing or custom made property and paste it in).
- **Style:** A formatting style (best for simple edits of formatting style). To map to a named Style, use the "Value" output type to call the name.
- **Style Class:** A style class to use. Once selected, a dropdown list shows the available style classes.

Use the **Add** ✚ icon to add rows to the Map table. The **Delete** 🗑 icon will delete the selected row. The up/down arrows will sort the order execution for the mapping. The first mapping (from top to bottom) that matches will be the mapping that is assigned.

You can then double-click to set values for each cell except the Fallback cell. The Fallback allows you to create a value to use in case non of your values, expressions, or ranges match to the input.

Examples

Numeric Range to Integer Value

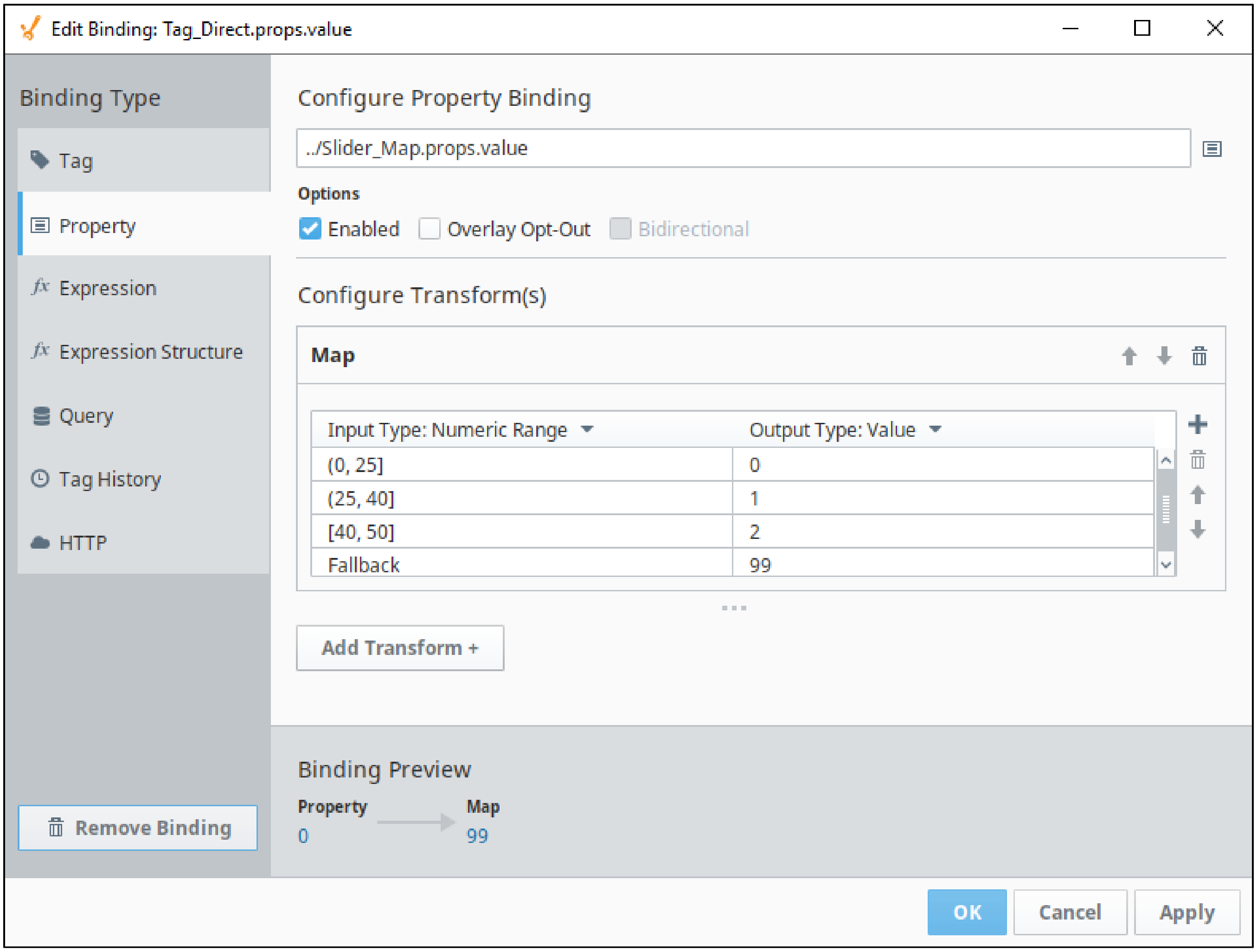
Input Type: **Numeric Range**

Output Type: **Value**

You can see in the Binding Preview at the bottom that the original value of the property binding is zero. Since the Map Transform input value is 0, we get an output value of 99, the Fallback value. This is because the first mapping uses an exclusive 0 (with a parenthesis) so it is testing for strictly greater than 0, not "greater than or equal to."

Numeric Range to Number Mapping

```
(0,25]                        0
(25,40]                        1
[40,50]                        2
Fallback                99
```



Configure Property Binding

../Slider_Map.props.value

Options

☑ Enabled ☐ Overlay Opt-Out ☐ Bidirectional

Configure Transform(s)

Map

Input Type: Numeric Range	Output Type: Value
(0, 25]	0
(25, 40]	1
[40, 50]	2
Fallback	99

Binding Preview

Property → Map

0 → 99

Numeric Range to String Mapping - Range Values Omitted

```
(,25]                        "Below 25"
(25,)                        "Above 25"
Fallback                "Invalid Value"
```

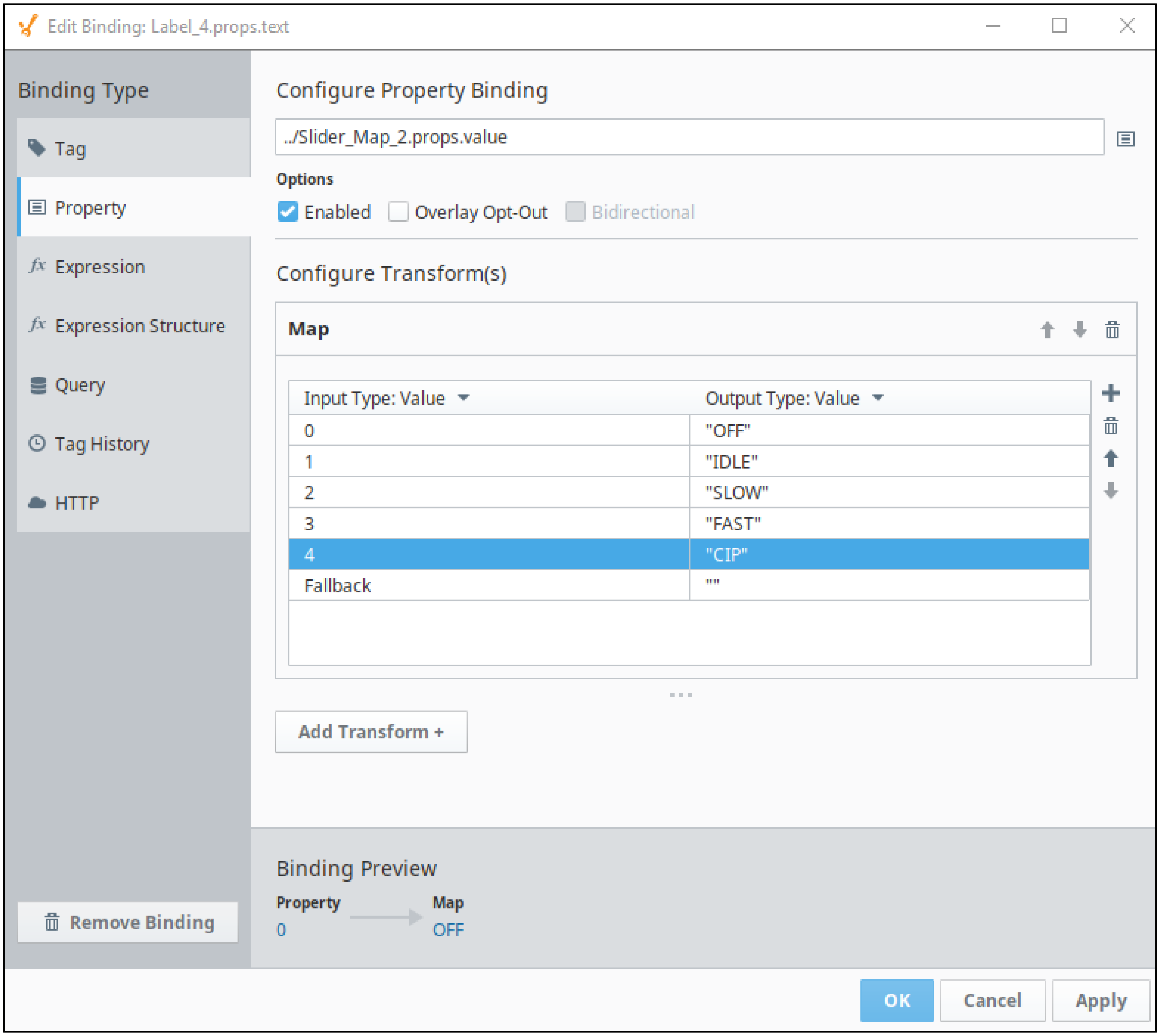
Integer to String

This is linked to a status text display.

Input Type: **Value**

Output Type: **Value**

You can see in the Binding Preview at the bottom that the original value of the property binding is zero. Since the Map Transform input value is 0, we get an output value of "OFF".



Edit Binding: Label_4.props.text

Binding Type

- Tag
- Property
- Expression
- Expression Structure
- Query
- Tag History
- HTTP

Configure Property Binding

../Slider_Map_2.props.value

Options

☑ Enabled □ Overlay Opt-Out □ Bidirectional

Configure Transform(s)

Map

Input Type: Value	Output Type: Value
0	"OFF"
1	"IDLE"
2	"SLOW"
3	"FAST"
4	"CIP"
Fallback	""

Binding Preview

Property 0 → Map OFF

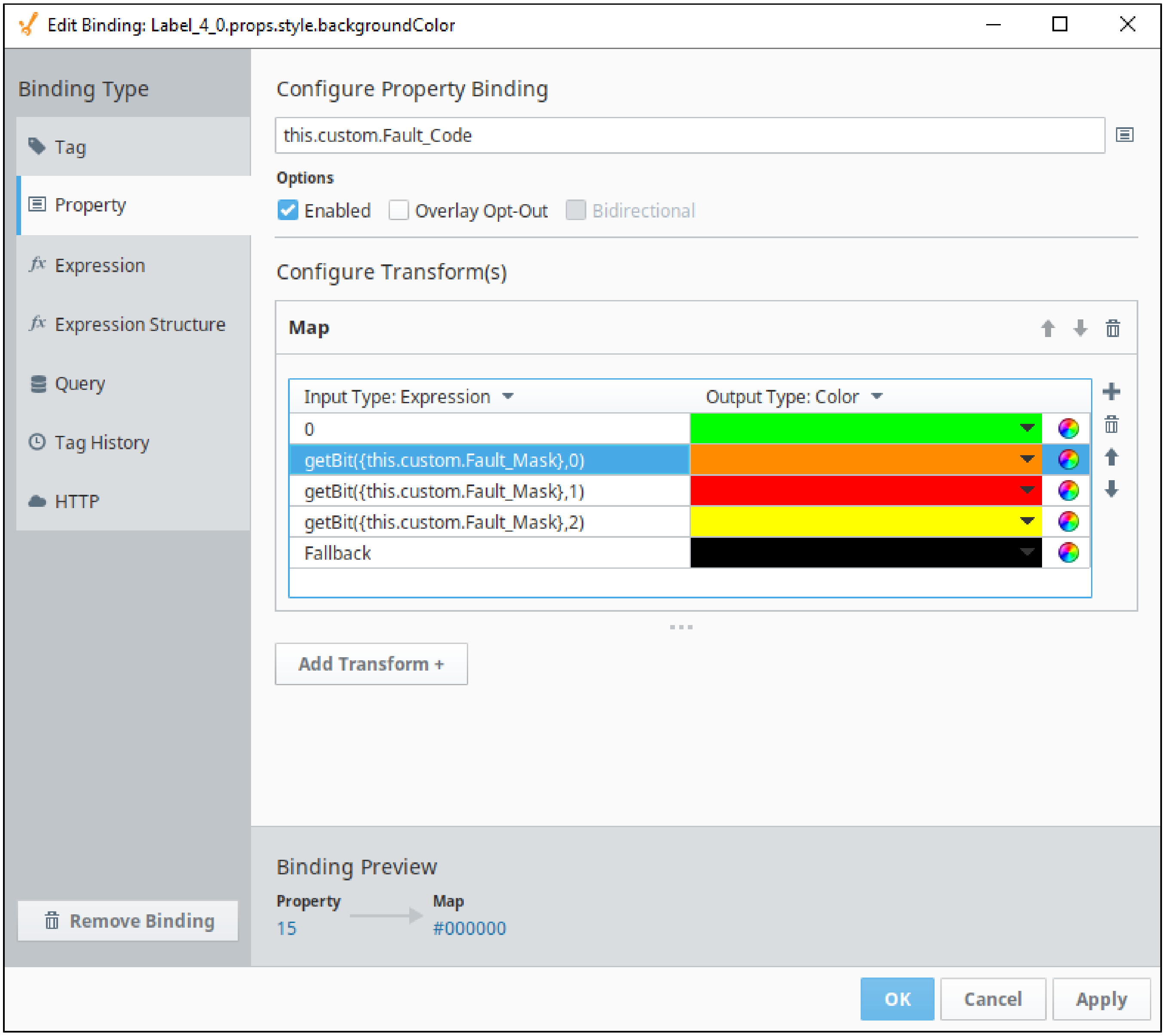
Expression to Color

This is linked to the background color property of a text field.

Input Type: **Expression**

Output Type: **Color**

This input is passed through several expressions before selecting the Fallback value of black (#000000). You can see in the Binding Preview at the bottom that the original value of the property binding is 15. For each expression past the first, a single bit of that integer is being tested. Since none of bits 0, 1, or 2 are true, we get the fallback. For more information on the getBit() function, see the appendix.



Configure Property Binding

this.custom.Fault_Code

Options

☑ Enabled ☐ Overlay Opt-Out ☐ Bidirectional

Configure Transform(s)

Map

Input Type: Expression	Output Type: Color
0	🟩
getBit({this.custom.Fault_Mask},0)	🟧
getBit({this.custom.Fault_Mask},1)	🟥
getBit({this.custom.Fault_Mask},2)	🟨
Fallback	⬛

Binding Preview

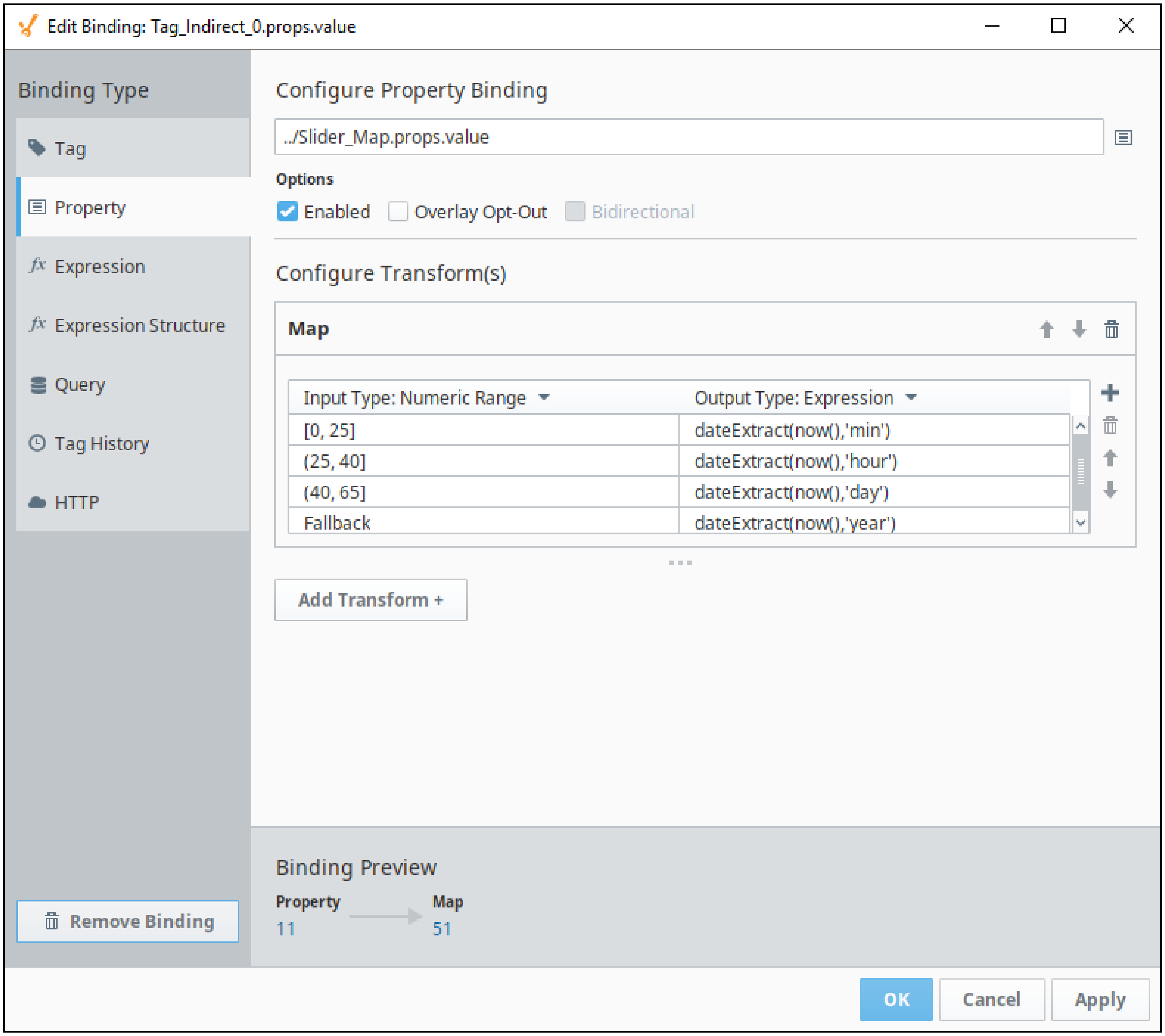
Property	Map
15	#000000

Numeric Range to Expression

Input Type: **Range**

Output Type: **Expression**

This mapping maps different ranges of values to different parts of the current time. You can see in the Binding Preview at the bottom that the original value of the property binding is 11 so we get an output value that pulls the minute out of the current time. That was 51 at the time this screenshot was taken.



Edit Binding: Tag_Indirect_0.props.value

Configure Property Binding

../Slider_Map.props.value

Options

☑ Enabled ☐ Overlay Opt-Out ☐ Bidirectional

Configure Transform(s)

Map

Input Type: Numeric Range	Output Type: Expression
[0, 25]	dateExtract(now(),'min')
(25, 40]	dateExtract(now(),'hour')
(40, 65]	dateExtract(now(),'day')
Fallback	dateExtract(now(),'year')

Binding Preview

Property 11 → Map 51

Format Transform

On this page ...

Format Transform

Watch the Video

The Format Transform applies a format string to the value returned from the binding, allowing you to format the way the output is displayed. Typically this applies to a date or number format.

Select Format Type:

- **Datetime:** A date will be formatted to appear a certain way.
- **Numeric:** A number will be formatted to appear a certain way.

Format (Numeric only)

- **Pattern:** Applies a number pattern (using 0 and #) and converts the results to string (good for putting in leading 0's on a display).
- **Integer:** Rounds input (float) to integer.
- **Number:** Formats number based on Language setting (puts in comma or other number separator).
- **Percent:** Converts the input number into percent (0.123 would format to 12.3%).
- **Currency:** Uses the currency specified by the Language setting.

Format (Datetime only)

- **Pattern:** Custom date format string (for example - M/D/YYYY h:m:s a).
- **Date:** Date only will be shown.
- **Time:** Time only will be shown.
- **Datetime:** Date and Time will be shown.

Date (Datetime only)

- **Full:** Example format - Monday, March 18, 2019.
- **Long:** Example format - March 18, 2019.
- **Medium:** Example format - Mar 18, 2019.
- **Short:** Example format - 3/18/19.

Time (Datetime only)

- **Full:** Example Format - 2:02:46 PM Pacific Standard Time.
- **Long:** Example format - 2:02:46 PM PST.
- **Medium:** Example format - 2:02:46 PM.
- **Short:** Example format - 2:02 PM.

Locale

- **auto:** Default time zone of Client.
- ...(Language Selection):

Time Zone (Datetime only)

- **auto:** Default time zone of Client.
- ...(Language Selection):

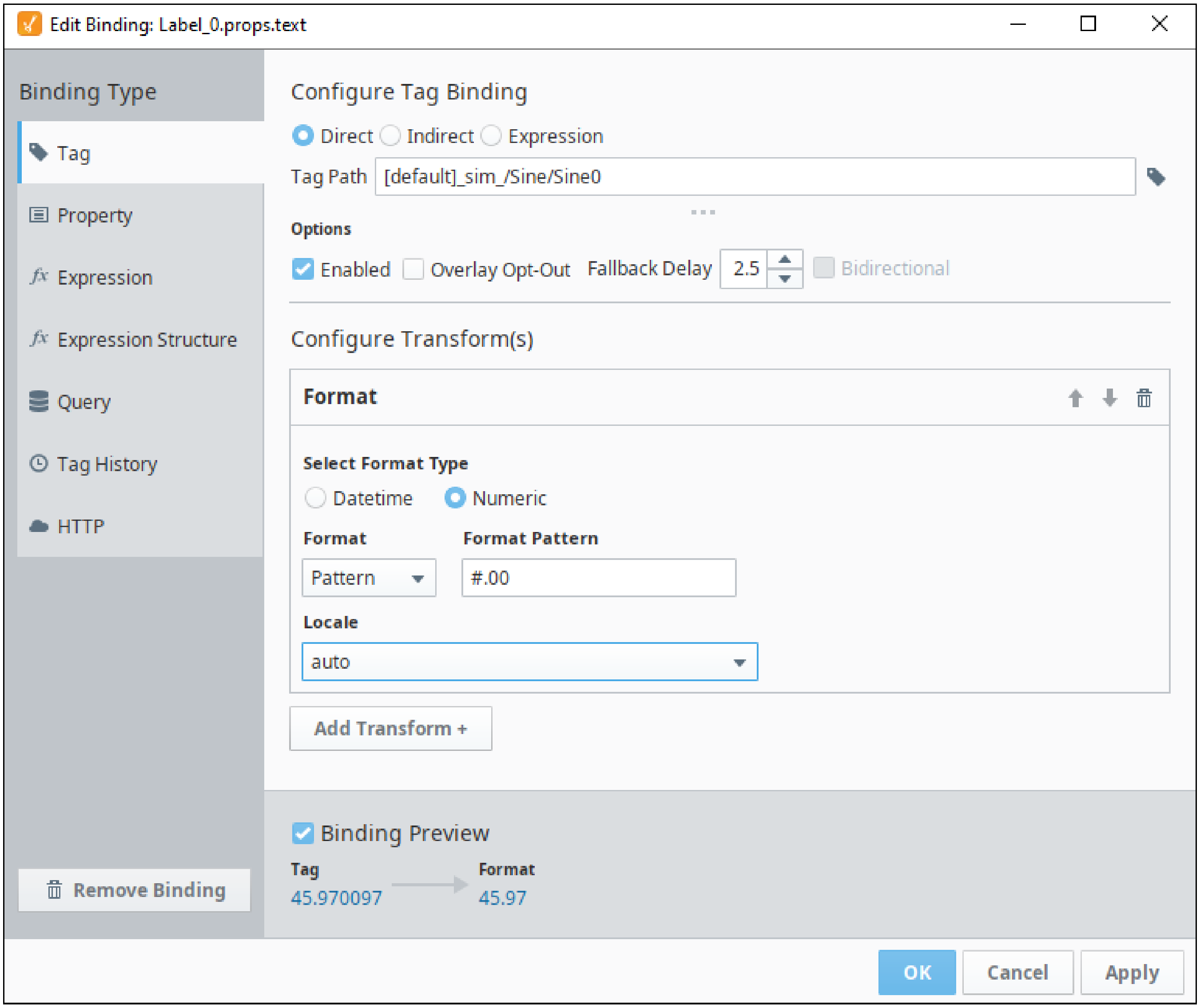
See the Data Type Formatting Reference page for more information.

Examples

Numeric Pattern

In the image below, an expression on a Tag binding is retrieving the value from a Tag. As shown in the preview, the value on the Tag shows **45.970097** .

The Format transform is taking that Tag value, then applying a numeric pattern of **#.00**, which denotes that two digits must always be shown after the decimal point. As a result, the preview shows a Format value of **45.97**, since any digits beyond the first two decimal places are ignored by the transform.



Edit Binding: Label_0.props.text

Configure Tag Binding

● Direct ○ Indirect ○ Expression

Tag Path [default]_sim_/Sine/Sine0

Options

☑ Enabled □ Overlay Opt-Out Fallback Delay 2.5 □ Bidirectional

Configure Transform(s)

Format

Select Format Type

○ Datetime ● Numeric

Format: Pattern

Format Pattern: #.00

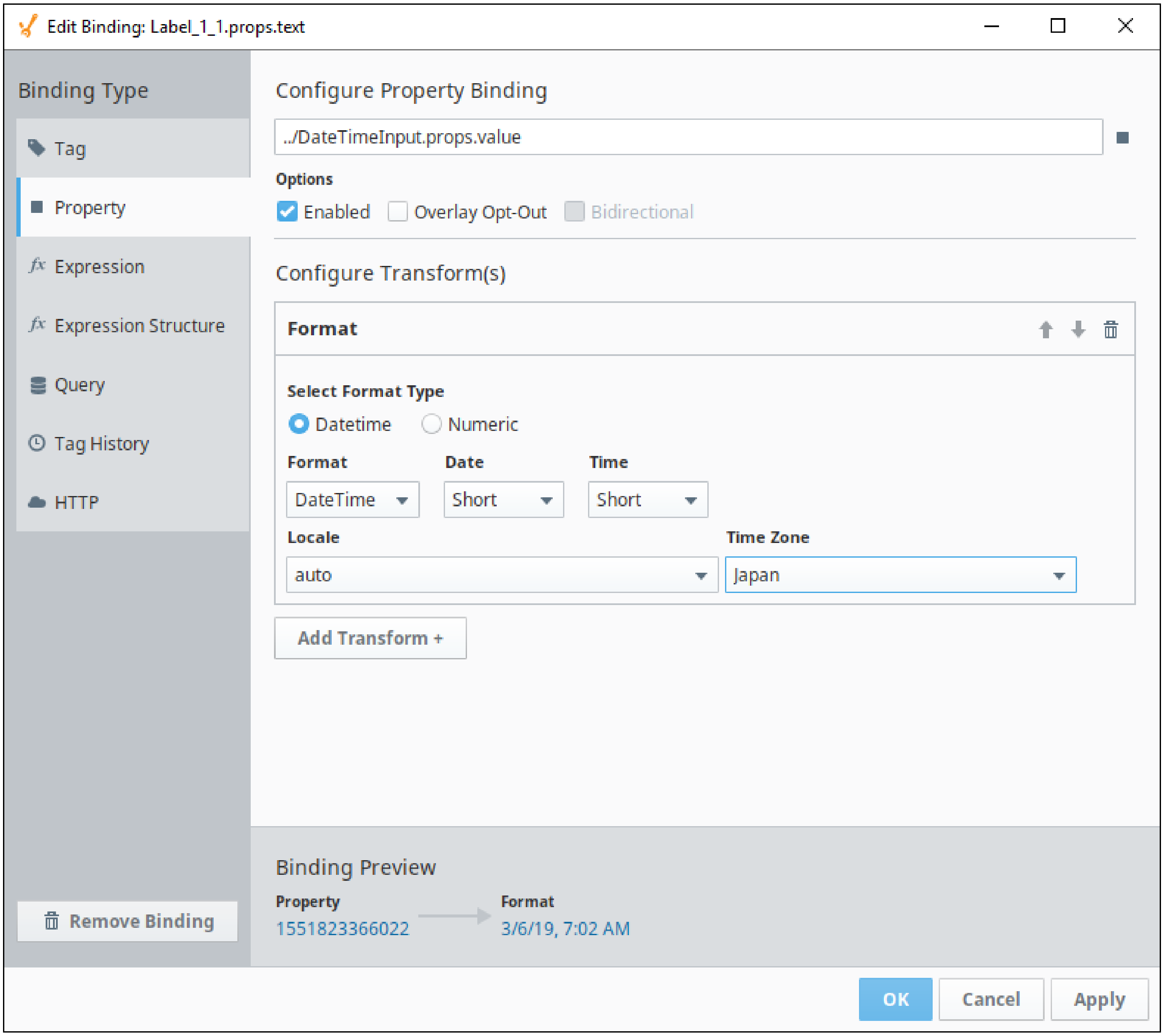
Locale: auto

☑ Binding Preview

Tag: 45.970097 → Format: 45.97

Datetime Short, Short, Time Zone Adjusted

The image below is taking a Unix timestamp value (including milliseconds) of **1551823366022**, and turning it into a human readable date (set to Japan's time zone) of **3/6/19, 7:02 AM**.



Edit Binding: Label_1_1.props.text

Binding Type

- Tag
- Property
- Expression
- Expression Structure
- Query
- Tag History
- HTTP

Remove Binding

Configure Property Binding

../DateTimeInput.props.value

Options

☑ Enabled □ Overlay Opt-Out □ Bidirectional

Configure Transform(s)

Format

Select Format Type

● Datetime ○ Numeric

Format	Date	Time
DateTime	Short	Short

Locale	Time Zone
auto	Japan

Add Transform +

Binding Preview

Property	Format
1551823366022	3/6/19, 7:02 AM

Script Transform

Script Transforms are special functions that are applied only on an existing binding. They take the result of a binding (or transform) as an input and produce a single output. This will allow you to manipulate a binding result using whatever python code you want.

When a script transform is first created it generates a template script with a few assumed input Arguments (see below). From this template you can write your own script and reference an incoming value as well as a few other parameters. It is assumed the custom script will end with a Return comment that is a single output value, dataset, document, or other type of data.

Arguments:

- **self:** A reference to the component this binding is configured on.
- **value:** The incoming value from the binding or the previous transform.
- **quality:** The quality code of the incoming value.
- **timestamp:** The timestamp of the incoming value as a java.util.Date

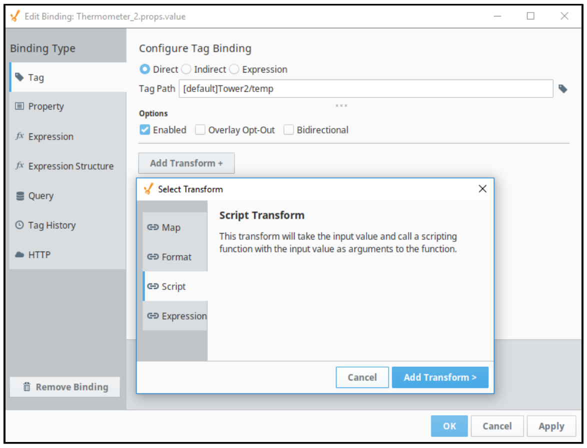
Return:

- Any single value of any data type. As with all other transforms, the type of data that is returned will overwrite the data type of the property that is being bound.

On this page ...

Script Transform

Watch the Video



Edit Binding: Thermometer_2.props.value

Configure Tag Binding

● Direct ○ Indirect ○ Expression

Tag Path [default]Tower2/temp

Options

☑ Enabled □ Overlay Opt-Out □ Bidirectional

Select Transform

Script Transform

This transform will take the input value and call a scripting function with the input value as arguments to the function.

Using Complex Properties in Your Transform

Any time you reference a complex component property (like an array or object), it will contain a Qualified Value(s). This means they have a quality code and timestamp attached to each piece of data. Using your script to access it will result in a Qualified Value but you can access the actual value manually using the **.value()** function. This means that fetching a complex property like an array must be looped through and converted manually if you want a basic array. For ease, the output of any binding or transform will automatically be stripped of quality and timestamp so you can use the value argument directly.

Example 1) Using the results of an array Property Binding in your script transform: You can reference the property value directly by calling the value parameter that is passed in.

Example 2) Fetching an additional array Property inside your Script Transform: You must manually convert the array property to a simple array.

Example Scripts

A lot of the components in Perspective expect arrays or JSON structured data in order to display data, but several binding types return single values or datasets. In order to transform one into the other, you can add a Transform to the binding and fill in a script to change the data format. Here are a few examples of code that can be added to a **Script Transform**.

Note: Don't forget to tab in correctly if you are copying scripts from this page. The Script Transform is a function (starts with "def") and every line of the code below should be tabbed in one from the edge.

These examples are not necessary with Tag History or Query binding types. For both of these, there is a dropdown setting at the bottom of the binding page that allows you to select the return type of DOCUMENT.

Dataset to Array of Objects

This is a script to take a dataset and transform it into a json array. All header names will be included in the resulting structure. This type of array is expected on many Perspective components like the Table component.

Dataset to Array of Objects

```
# convert the incoming value data
pyData = system.dataset.toPyDataSet(value)
# get the header names
header = pyData.getColumnNames()
# create a blank list so we can append later
newList = []

# step through the rows
for row in pyData:
        # create a new blank dictionary for each row of the data
        newDict = {}
        # use an index to step through each column of the data
        for i in range(len(row)):
                # set name/value pairs
                newDict[ header[i] ] = row[i]

        # append the dictionary to list
        newList.append(newDict)

# return the results
return newList
```

Dataset to Array with Renamed Columns

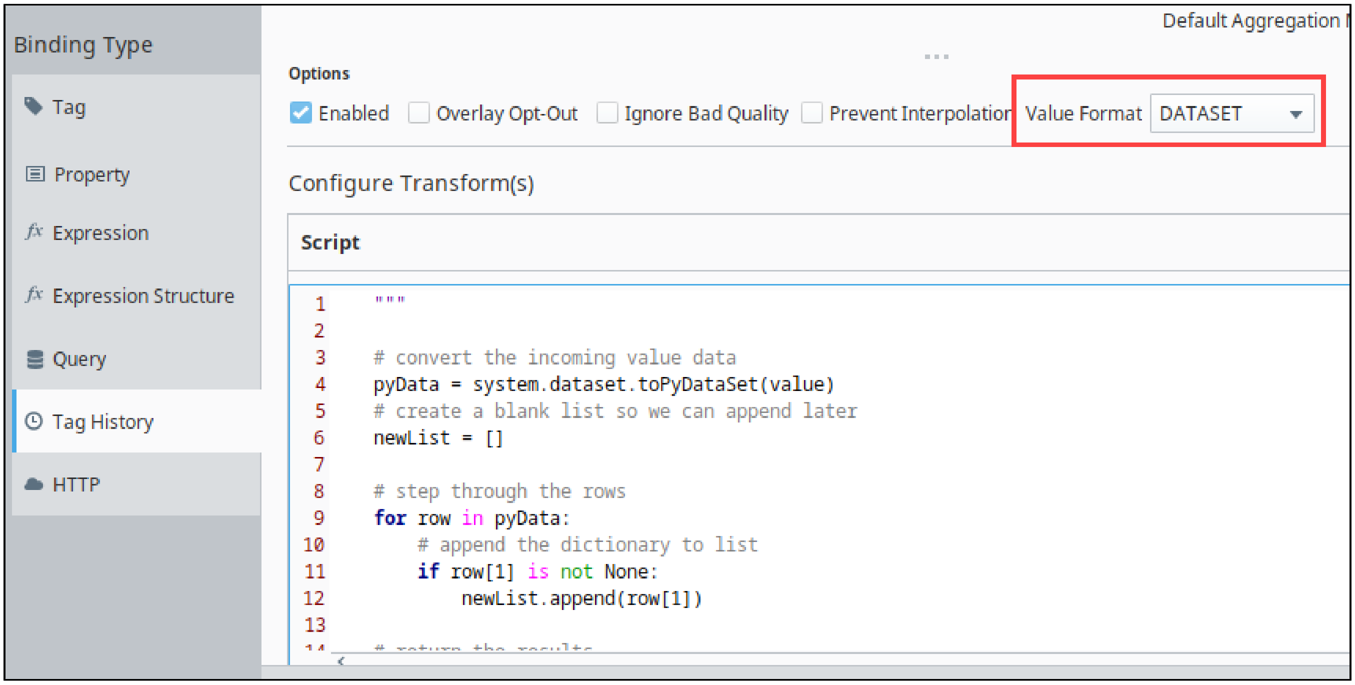
This is another script to take a dataset and transform it into a json array. In this example, you create the header names to be included in the resulting structure. This type of array is expected on many perspective components like the XY chart, Dropdown, etc.

Dataset to Array of Named Objects

```
# set the header names. For the XY chart, these must match the values in the series property.
header = ["Column 1", "Column 2"]

# convert the incoming value data
pyData = system.dataset.toPyDataSet(value)
# create a blank list so we can append later
newList = []

# step through the rows
for row in pyData:
        # create a new blank dictionary for each row of the data
        newDict = {}
        # use an index to step through each column of the data
        for i in range(len(row)):
                # set name/value pairs
                newDict[ header[i] ] = row[i]

        # append the dictionary to list
        newList.append(newDict)

# return the results
return newList
```

Sparkline Chart

The data expected in a Sparkline chart is a bit different than other charts. Instead of having values paired with a timestamp, it just takes an array of values to draw in order. This is a script to take a Tag History binding, apply the Script Transform, and output only the values in an array. Make sure the Value Format dropdown is set to Dataset.



Options

☑ Enabled □ Overlay Opt-Out □ Ignore Bad Quality □ Prevent Interpolation Value Format DATASET

```
"""

# convert the incoming value data
pyData = system.dataset.toPyDataSet(value)
# create a blank list so we can append later
newList = []

# step through the rows
for row in pyData:
    # append the dictionary to list
    if row[1] is not None:
        newList.append(row[1])

```

Tag History to list of Data Points

```
# convert the incoming value data
pyData = system.dataset.toPyDataSet(value)
# create a blank list so we can append later
newList = []

# step through the rows
for row in pyData:
        # append the dictionary to list
    if row[1] is not None:
                newList.append(row[1])

# return the results
return newList
```

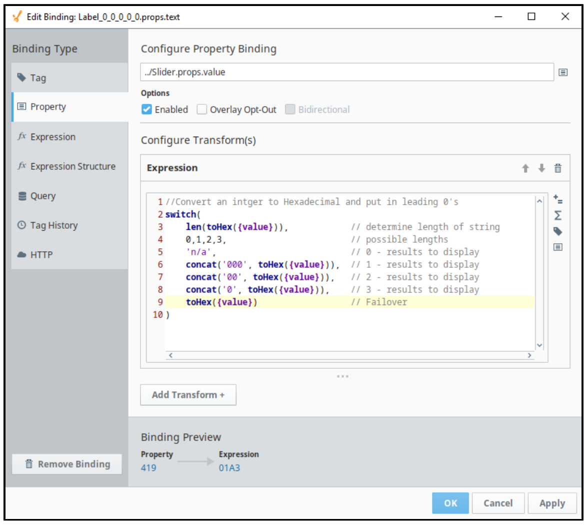
Expression Transform

The Expression Transform runs an expression that allows you to manipulate the value of the binding using an expression. An Expression transform uses the Ignition Expression binding language and has built in links to several toolsets.

Those expression tools are as follows:

- **Operators:** Mathematical, Logical, Bitwise operators to adjust the incoming value.
- **Functions:** A library of expression functions to adjust the incoming value
- **Browse Tags:** A link to the Tag browser
- **Browse Properties:** A link to Session Properties or other properties in the same View.

Example - Function



Configure Property Binding

../Slider.props.value

Options

☑ Enabled □ Overlay Opt-Out □ Bidirectional

Configure Transform(s)

Expression

```
//Convert an intger to Hexadecimal and put in leading 0's
switch(
    len(toHex({value})),            // determine length of string
    0,1,2,3,                        // possible lengths
    'n/a',                          // 0 - results to display
    concat('000', toHex({value})),  // 1 - results to display
    concat('00', toHex({value})),   // 2 - results to display
    concat('0', toHex({value})),    // 3 - results to display
    toHex({value})                  // Failover
)
```

Binding Preview

Property	Expression
419	01A3

Convert Into Hex

```
//Convert an integer to a Hexadecimal and put in leading 0's
switch(
        len(toHex({value})),                        // determine length of
string
        0,1,2,3,                                                //
possible lengths
        'n/a',                                                        // 0
- results to display
        concat('000', toHex({value})),        // 1 - results to display
        concat('00', toHex({value})),        // 2 - results to display
        concat('0', toHex({value}))        ,        // 3 - results to
display
        toHex({value})                                        // Failover
)
```

On this page ...

Expression Transform

Watch the Video

Binding Property Path Reference

Perspective component bindings can reference the value of other properties. These references take the form of a relative path that leads to the referenced property.

Because of this, it can be helpful to understand how the paths work. This section details the various keywords and operators associated with these paths. Only properties on components in the same view are eligible to be used in this way.

The format of the property path is like a file system path to get to the component combined with a dot-referenced object path to get to the property. The section referencing the property must begin with the property scope (e.g., "props" or "position" or "meta"). For the following examples, suppose we are designing a View with the following component hierarchy, and that each of these components has an "x", "y" property in "position", as well as a property called "complex" in "props" which is a map containing "foo", which is a number, and "bar" which is an array of numbers.

- View
    - root
        - LabelA
        - LabelB
        - Sub_Container1
            - ButtonA
            - ButtonB
        - Sub_Container2
            - ButtonA
            - ButtonB

Operator /Keyword	Description	Example

`/`	Slash Operator - When a path starts with this operator, then it defines an absolute path. That is, a path that starts at the top of the view hierarchy and is not relative to where the binding is being configured.
	When not at the start of a path, the / operator moves further into a container, drilling further down into the hierarchy.
		`// Absolute path. Sequential slashes allow for movement into a container`
		`/root/LabelA.position.x`
		`/root/Sub_Container1/ButtonA.position.y`

`.`	Dot Operator - You may access properties deep within a component's property document structure using the Dot Operator. Assuming the component LabelA had a META property named "foo", then we could use the example on the right to retrieve the value of foo. The Dot Operator can also be used to move further into a complex component. Assuming LabelA has object under META named "rotate", we can move into rotate with further use of the Dot Operator.	`/root/LabelA.meta.foo` `/root/LabelA.meta.rotate.angle`
`[]`	Brackets - When referencing an array property, brackets allow you to specify an individual index within the array.	`/root/LabelA.props.complex.bar[5]`

`../`	Parent Container Operator - This operator acts as a shorthand reference to the parent container. Because the operator always returns the immediate parent container, the operator is relative to the component trying to utilize the operator.
	When moving up in the hierarchy, multiple uses of this operator may be used in sequence to climb up multiple containers.
	Alternatively, you may simply add additional dots to move up levels. Each additional dot moves up another level.
		`// From ButtonA, we can use this operator quickly move to a sibling component`
		`../ButtonB.position.x`
		Move Up Multiple Parent Containers
		`// Moving up multiple parent containers`
		`../../LabelA.position.x`
		`// Also moves up: each additional dot is another`

		parent container .../LabelA.position.x
./	Container Self Operator - When configuring a binding from a container, this operator acts as a shorthand reference to the container. This is similar in concept to the this keyword, but still allows for the user of the other operators. **Note:** This operator only works when the path is on a binding configured on a container.	./LabelA.position.x
this	The this keyword allows you to easily reference the same component the binding has been placed on. This works on any object, including containers, views, and even the session.	this.meta.name
parent	Parent shortcut - References your immediate parent. This keyword is only valid when being evaluated from the scope of a component. For example, LabelA could reference the root container variables. All of these shortcuts cannot be used with any other path separators, so a path like this /MyChild.position.x is invalid, for that, you'd use "./"	parent.props.complex.foo or parent.position.x
view	View keyword - Refers to the view that a component is contained in. This is only valid when being evaluated from the scope of a component. Lastly, the view shortcut references the view itself. Views may have input and output parameters, and to reference these parameters simply specify the category and name of the parameter, as shown in the example.	view.params.paramName
page	Page keyword - refers to the page that the object is contained in. This is only valid when being evaluated from the scope of a view.	
session	Session keyword - Refers to the session object. This keyword is valid from any object type. Useful in cases where a binding needs to reference session properties. **Note:** Normally, bindings can not reference properties on components in other views. However, views can use custom session properties as a means of sharing or synchronizing a value.	

Component Events and Actions

In Perspective, events and actions are some of the fundamental building blocks of project functionality. Actions give you the ability to respond to specific user inputs (such as mouse, keyboard, and touch inputs), as well as broad session events (like the beginning and end of the session) in many different ways. Thus, *actions* are a response to *events*.

Example uses for events and actions include:

- Navigating to a new page when the user presses a button.
- Opening a popup containing details on a specific PLC when a user double-clicks on it in a diagram.
- Logging the user out of the session when they press Ctrl + L on the keyboard.
- Scanning a barcode from a mobile device, and sending it to the Gateway.

In this section, we'll cover the basics on how to configure actions and events to suit your project's needs.

On this page ...

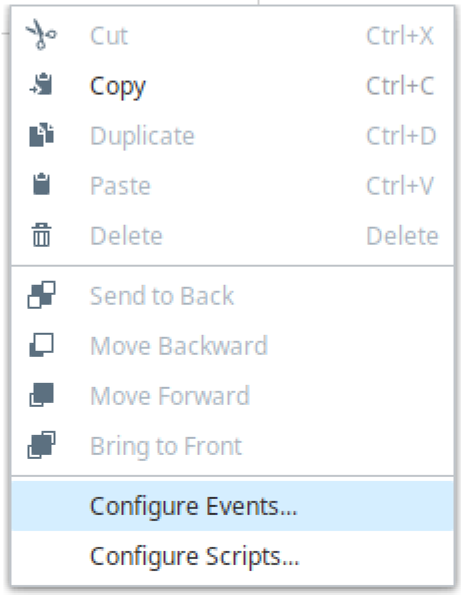
Events and Actions

Watch the Video

Configure Events and Actions

To configure an event and action on a component:

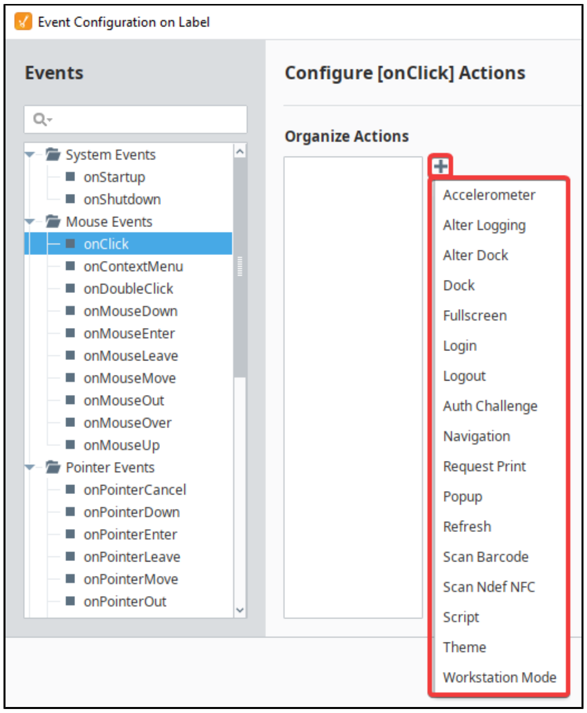
1. Select the component.
2. **Right-click** on the component, and select **Configure Events**.



3. Choose an event from the left-hand side.

> **Note:** You can configure actions for as many events as you'd like, but you'll need to configure the actions separately for each one.

4. Next choose one or more actions to associate with the event, by clicking the **Add** + icon. Descriptions of the options for each component action are described in the sections on this page.



5. Configure the actions as you want. Click **OK**.

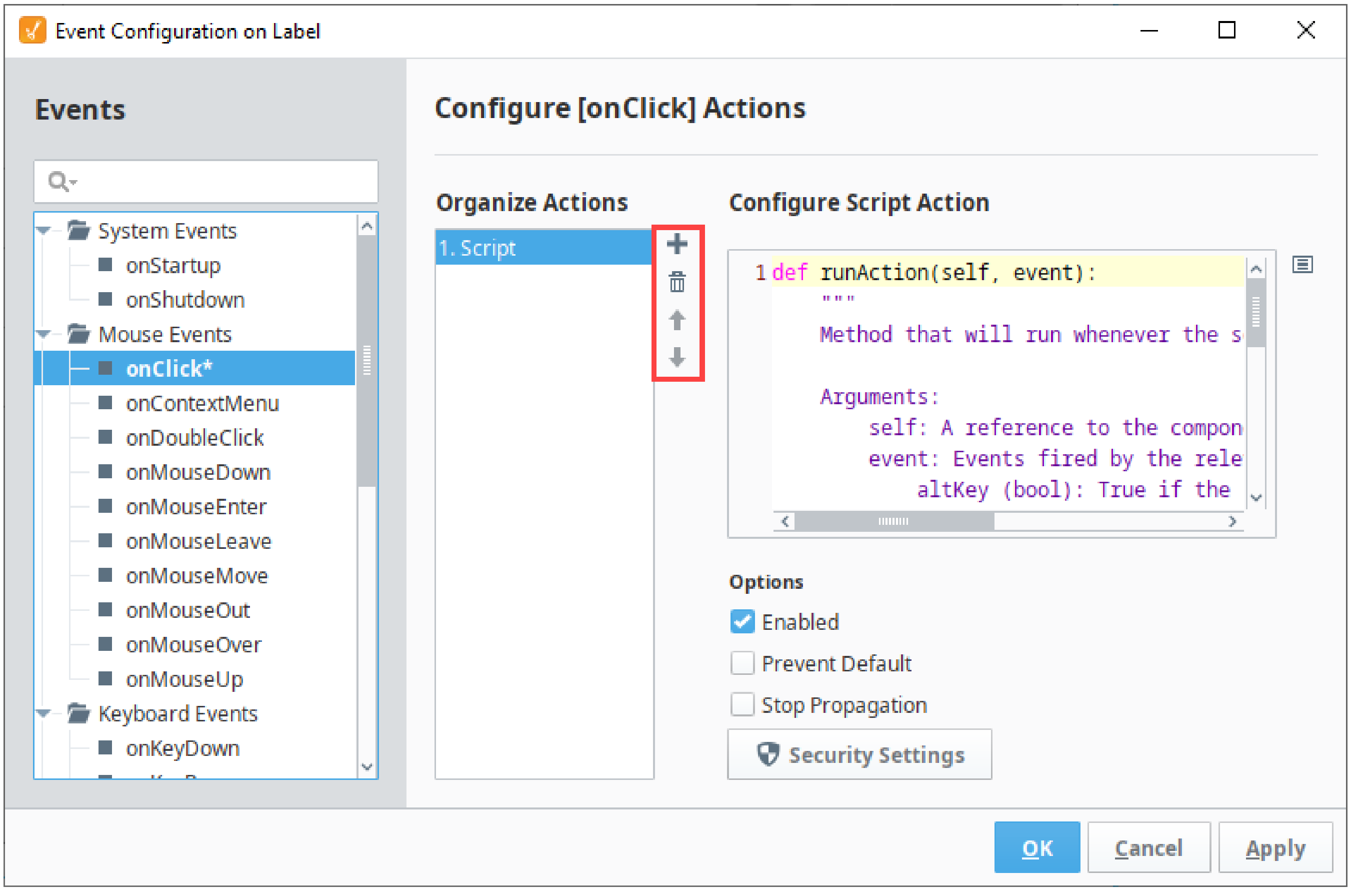
Perspective Events

Perspective offers a wide range of possible events, but in this section we'll highlight a few common ones. You can find details on all configurable events at Perspective Event Types Reference.

Event Type	Example Event
Component Event	Many components have their own events that are related directly to the functionality of the component. For a full list of components with configurable component events, see Perspective Event Types Reference.
System Event	• **onStartup** events occur when the View or component is loaded into the session. For example, if you configure an onStartup event on a component, the event will occur when the view it is on is opened. • **onShutdown** events occur when the View or component is removed from the session. Typically this will occur when you navigate away from the View containing the component, or if the session times out.
Mouse Event	• **onClick** events occur when the user clicks on any of their mouse's buttons, while the cursor is hovered over the component. • **onContextMenu** events occur when the user clicks the mouse button associated with a context menu (typically the *right* mouse button, on a two-button mouse). • **onMouseOver** events occur when the mouse pointer enters the component's borders.
Keyboard Event	• **onKeyUp** events occur when a key on the keyboard is released, while the component is *focused.* • **onKeyPress** events will be run repeatedly, while a key is held down and the component is *focused.*
Touch Event	• **onTouchStart** Fired when the user has touched the surface of a touch capable device.
Wheel Event	• **onWheel** events occur when a user moves the scroll wheel while hovered over the component.

Action Types

Each Event can have actions assigned to them, and each Action has some specific purpose. Each event can have any number of actions, and different types of actions can do different things.



```
def runAction(self, event):
    """
    Method that will run whenever the s

    Arguments:
        self: A reference to the compon
        event: Events fired by the rele
            altKey (bool): True if the
```

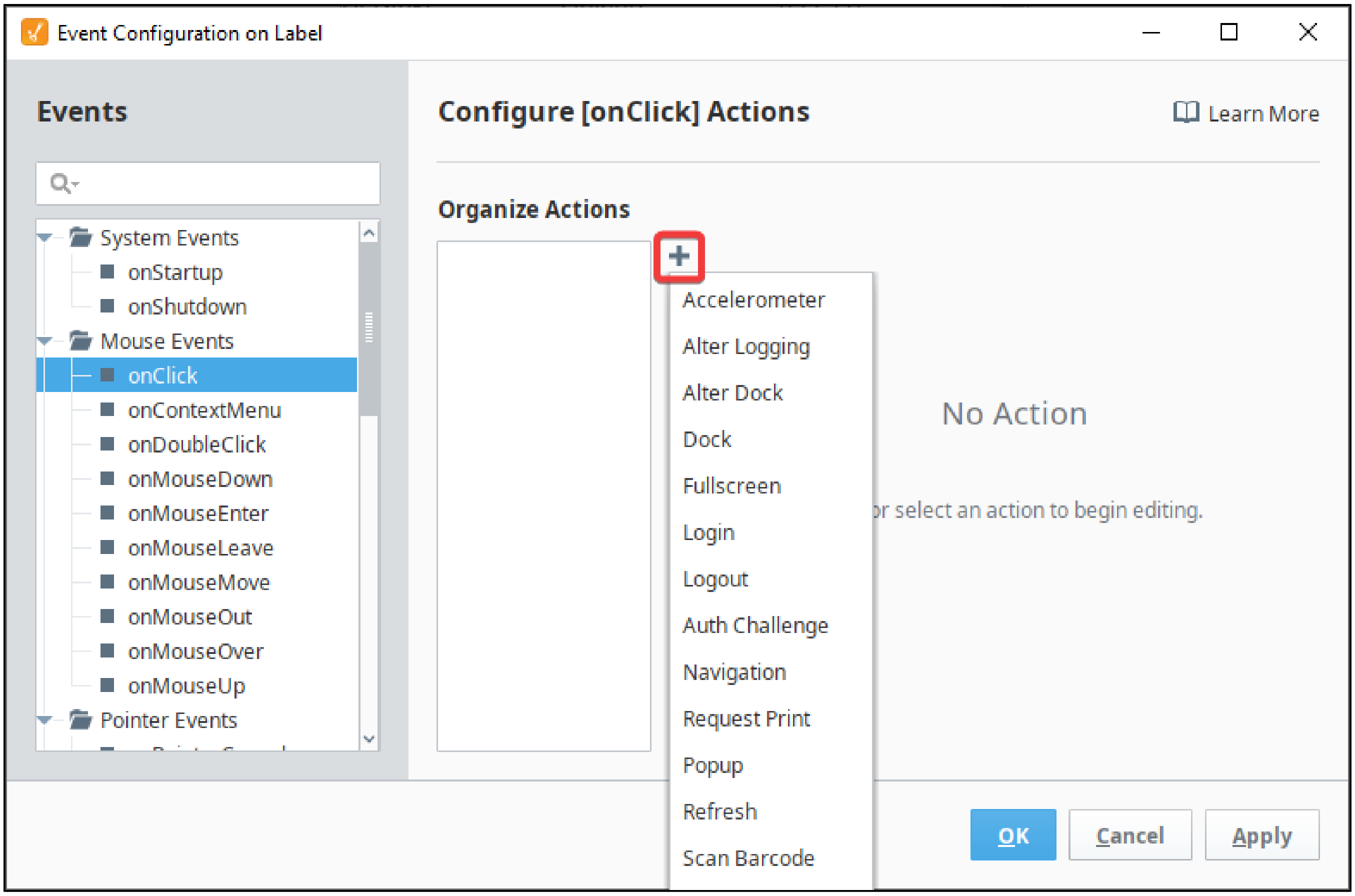
☑ Enabled

☐ Prevent Default

☐ Stop Propagation

If you want to delete an action, use the **Delete** 🗑 icon.

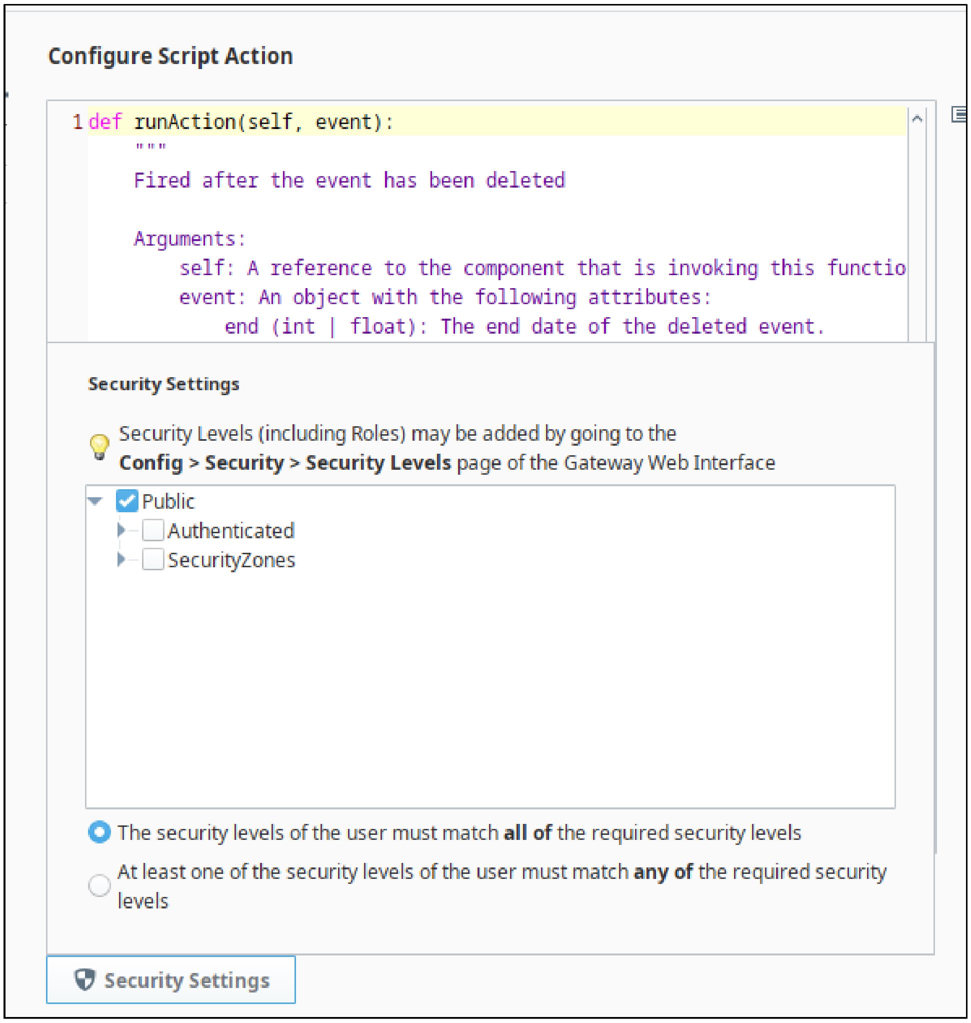
Each Action is called in order from top to bottom. To control this execution order, you can reorder the list of Actions using the **Up Arrow** ⬆ and **Down Arrow** ⬇ icons next to the list. However, Actions are not executed synchronously: sequential actions do not wait for any prior Actions to finish executing before running. Thus, if Action 1 is a long-running script, while Action 2 is quick to finish, it is possible that Action 2 will finish before Action 1.



Shared Action Options

The bottom of each action lists a set of options. The options listed in the following table are shared across different action types.

Enabled	Specifies whether the action should be used or ignored.
Prevent Default	Prevents the browser's default behavior from occurring. Useful when you want to prevent the browser's built-in right-click menu from showing.
Stop Propagation	Prevents events from higher up in the component hierarchy from triggering when the selected event triggers.
Security Settings	Opens a panel where you can specify required security levels that must be present for the action to trigger.



Configure Script Action

```
def runAction(self, event):
    """
    Fired after the event has been deleted

    Arguments:
        self: A reference to the component that is invoking this functio
        event: An object with the following attributes:
            end (int | float): The end date of the deleted event.
```

Security Settings

Security Levels (including Roles) may be added by going to the **Config > Security > Security Levels** page of the Gateway Web Interface

- ☑ Public
  - □ Authenticated
  - □ SecurityZones

● The security levels of the user must match **all of** the required security levels

○ At least one of the security levels of the user must match **any of** the required security levels

Action Setting Reference

The following headings detail unique settings for each action type. Shared options for each action are listed under Shared Action Options.

Accelerometer Action

Retrieves accelerometer data from the device's accelerometer (a common feature on smartphones). This is a Native App Action, designed to help in gathering data from a mobile device. Each action needs to be configured in two parts:

1. The action is run on a mobile device, indicating that a type of data should *begin* to be gathered.
2. As the data is gathered, or once it's finished being gathered, it is sent to the Gateway to be handled by the corresponding *Session Event*.

Note: This action is designed specifically for the Ignition Perspective App for Android and iOS devices. If it is run in a browser session, it will be ignored.

Action Setting	Description
Continuous	Begins recording accelerometer data, which repeatedly updates the **accelerometer** object in the current session's Session Properties. • **Sample Rate** indicates how often the accelerometer object should be updated. • **Context** provides the opportunity to pass a custom object through to the Accelerometer Data Received Session Event. A **session** object is already provided to that event script.
Batch	Records accelerometer data at a specified interval for a specified rate, then sends the accumulated data to the Gateway, to be handled by the Accelerometer Data Received Session Event.

	- **Sample Rate** indicates how often a reading should be made. - **Duration** specifies for how long data should be logged. - **Context** provides the opportunity to pass a custom object through to the Accelerometer Data Received Session Event. **Note:** A **session** object is already provided to that event script.
Off	Turns off accelerometer data recording.

Alter Logging Action

Perspective sessions log their session activity and errors to the *logs of the browser they run in*. Alter Logging allows us to change how verbose this logging is. This is useful for session debugging.

Action Setting	Description
Remote Logging Enabled	Indicates whether the browser logs should also be sent to the Gateway to log in the Gateway logs. **Note:** For all messages to be visible, the **Perpective.Client** logger must be set to the same logging level as the level indicated in this action.
Set Logging Level	Dictates how verbose the session's logging should be. A logging level of **all** (or **trace**) shows all possible records, while a level of **off** shows none. Options are: - all - trace - debug - info - warn - error - fatal - off

Alter Dock

The following feature is new in Ignition version **8.1.19**
Click here to check out the other new features

Allows you to alter the configuration of a docked view. Any action settings left blank will remain unchanged.

<table>
<tr><th>Action Setting</th><th>Description</th></tr>
<tr><td>View</td><td>The currently selected view. Changing this will change which view is mounted to this position.</td></tr>
<tr><td>Display</td><td>This property allows you to show or hide the docked view. Options are:
<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>visible</td><td>The docked view is always expanded/displayed.</td></tr>
<tr><td>onDemand</td><td>The docked view is collapsed, but allows the user to display the view by clicking on the docked view's handle.</td></tr>
<tr><td>auto</td><td>Automatically shows or hides the docked view depending on how much space is available in the session: showing the view if the page is wider than the width specified in the auto-breakpoint setting. (Works in conjunction with the Auto Breakpoint property).</td></tr>
</table>
</td></tr>
<tr><td>Resizable?</td><td>Determines whether the docked view may be resized or not.</td></tr>
<tr><td>Modal?</td><td>Determines if the view should be modal, meaning users will not be able to directly interact with other views while the modal view is present. This property is only enabled when the <b>Display</b> property is set to <b>onDemand</b>.</td></tr>
</table>

<table>
<tr><td>Content</td><td>Determines how the docked view interacts with other views on the page.
<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>push</td><td>Opening or closing the docked view causes the content in the center to resize: the center view will be 'pushed' out of the way.</td></tr>
<tr><td>cover</td><td>When opening the dock, it slides in front of the center view, obscuring part of the center view: the dock will 'cover' part of the center view.</td></tr>
<tr><td>auto</td><td>Acts like the cover option when the viewport is smaller than the Auto Breakpoint value. Acts like the push option when the viewport is larger than the Auto Breakpoint value.</td></tr>
</table></td></tr>
<tr><td>Anchor</td><td>Allows you to make a view always visible while scrolling. Only available on North docked view configurations.
<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>fixed</td><td>The docked view will remain in a fixed position, relative to the page. Useful when a north-docked view should stay at the top of a page. Select this option if a docked view is acting as a header that should always be present.</td></tr>
<tr><td>scrollable</td><td>The docked view will not stay in a fixed position as the user scrolls down in the page. Select this option if the north dock should move along with the page as the user scroll down.</td></tr>
</table></td></tr>
<tr><td>Size</td><td>Determines the size, in pixels, of the view.
<ul>
<li>If the view is docked to the North or South edge, then size determines the height.</li>
<li>If the view is docked to the East or West edge, then size determines the width.</li>
</ul></td></tr>
<tr><td>Auto Breakpoint</td><td>Controls the minimum page width for Auto docked views to be visible. When the session is smaller than this width, these views will be hidden and able to be displayed on demand.
This property is enabled when the <b>Display</b> property is set to <b>auto</b>.</td></tr>
<tr><td>Dock ID</td><td>An optional arbitrary string that can be used to reference a docked view through other parts of Perspective such as in an action or as a scripting call.</td></tr>
<tr><td>Handle</td><td>Allows you to show or hide a handle for users to expand/collapse the view.
<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>Show</td><td>Show handle at all times.</td></tr>
<tr><td>Hide</td><td>Hide handle at all times.</td></tr>
<tr><td>AutoHide</td><td>Hide handle when page is not active.</td></tr>
</table></td></tr>
<tr><td>Handle Icon</td><td>Path to an icon used to identify the view when the view is hidden.</td></tr>
<tr><td>View Parameters</td><td>Allows specific parameter values to be passed to the docked view when navigating to the page.</td></tr>
</table>

Dock Action

Allows you to open or close a docked view. The view must be configured as docked in the Page Configuration section of the designer, and must be configured with a dock ID.

Action Setting	Description
Dock Action	There are three types of dock actions: - **Open**: Opens the docked view. - **Close**: Closes the docked view. - **Toggle**: Toggles the state of the docked view, so opens the view if it currently closed, or closes it when it is currently open.
Identifier	The ID of the docked view. Dock ID values can be set when you configure a view as docked.
Parameters	Parameters that can be passed into the docked view. The name of the parameters must match the name of the view parameters that are already set up.

Fullscreen Action

Enters full screen mode. Requesting to enter full screen mode only works with events that originate from user interactions. Some browsers may not support full screen requests.

Action Setting	Description
Enter	Enters full screen mode.
Exit	Exits full screen mode.
Toggle	Changes the Session's browser to whichever mode it is not currently in.
View	Enters full screen mode on a targeted view.
Page	Enters full screen mode on a targeted page.

Login and Logout Actions

Logs the current user in or out of the session. The only action property is the Enabled option which specifies whether the action should be used or ignored.

Action Setting	Description
Ask the IdP to re-authenticate users	Determines how re-authentication requests sent to the Identity Provider will be handled. Identity Providers can choose to ignore re-authentication requests, defaulting to their own behavior. Options are as follows: - **Project** - Use the re-authentication setting located in the General category of Project Properties. - **Enable** - Prompt the user to provide their credentials, even if they're already logged into the session. - **Disable** - When selected, the user will not have to provide their credentials if they're already logged in.

Authentication Challenge Action

The following feature is new in Ignition version **8.1.16**
Click here to check out the other new features

The Authentication Challenge action will navigate the user to an IdP, allowing a different user to authenticate against an IdP without logging out the currently logged in user. After the user successfully logs in, the IdP navigates them back to Perspective and triggers a session event onAuthChallenge Completed with contextual information about this second user's identity and their security levels. The session's authentication state will remain unchanged — in other words, the second user will only be logged in as the user executing the session's onAuthChallengeCompleted event handler script. This provides the opportunity for workflows that include requiring a supervisor's "e-signature" before performing certain actions.

Note: The second user will not be logged into the IdP or remembered in any way by the IdP when the IdP is an internal Ignition type, since the internal Ignition IdP is aware of “sessionless” authentication challenges. External IdP types (OIDC and SAML) are not aware of “sessionless” authentication challenges since there is no such concept in the respective open standards. To work around this, a separate IdP tailored specifically for the secondary users' authentication challenges may be specified under the **Identity Provider** action setting. So long as this special IdP is only used for these secondary user authentication challenges (and not used anywhere else), and so long as the **Ask the IdP to re-authenticate users** setting is enabled, secondary users should always be required to provide their credentials every time an authentication challenge is triggered.

Action Setting	Description
Identity Provider	Specifies the IdP to authenticate against. This may be different from the project's IdP. If set to None, the project default IdP is used.
Ask the IdP to re-authenticate users	Determines how re-authentication requests sent to the Identity Provider will be handled. Identity Providers can choose to ignore re-authentication requests, defaulting to their own behavior. Options are as follows: - **Project** - Use the re-authentication setting located in the General category of Project Properties. - **Enable** - Prompt the user to provide their credentials, even if they're already logged into the session. - **Disable** - When selected, the user will not have to provide their credentials if they're already logged in.
Timeout	An integer representing the number of minutes the system will wait in between the authentication request and the authentication response before timing out the request. If set to zero, the default of two minutes will be used as the timeout.
Payload	An opaque payload that may contain any information the user wants to pass to the Authentication Challenge Completed

<table>
<tr><td></td><td>session event.</td></tr>
<tr><td>Framing</td><td>How the challenge should be presented to the user’s current page:<ul><li><b>Same Tab / Window</b> - User is navigated away from the client and to the IdP in the same window/tab. Once authentication is complete at the IdP, they are redirected back to the client.</li><li><b>New Tab / Window</b> - A new tab is opened in the user’s browser, leaving the current client tab open while it waits for the new tab to complete the authentication challenge at the IdP.</li><li><b>Embedded Frame</b> - The redirect to the IdP is embedded in an Iframe within the current client’s page.</li></ul><b>Note:</b> Mobile and workstation clients do not support “New Tab / Window” and will fall back to “Same Tab / Window”. Mobile clients do not support “Embedded Frame” and will fall back to “Same Tab / Window”.</td></tr>
</table>

Navigation Action

The Navigation action allows you to navigate to different views, pages, or URLs from an event. The Navigation action has several modes. Each mode allows for a different type of navigation and different options. The following table lists the types of navigation:

<table>
<tr><th>Type</th><th>Description</th></tr>
<tr><td>Page</td><td>Navigates to a separate page.<table>
<tr><th>Setting</th><th>Description</th></tr>
<tr><td>Set Page</td><td>A string denoting the page URL of the target page. See Page URLs.</td></tr>
<tr><td>Open in new tab</td><td>Specifies whether the newly opened page should replace the current page, or open in a new browser tab.</td></tr>
</table></td></tr>
<tr><td>View</td><td>Replaces the current main View with a new main View.<table>
<tr><th>Setting</th><th>Description</th></tr>
<tr><td>Select View</td><td>The path to the view that should be used.</td></tr>
<tr><td>Parameters</td><td>A set oparameters to pass to the view. Add or remove parameters with the Add ✚ icon and <b>Delete</b> 🗑 icon. In the <b>Value</b> field for a given parameter, you can pass in a path to a property value using the <b>Parameter</b> ☰ icon</td></tr>
</table></td></tr>
<tr><td>Url</td><td>Navigates to an external web address.<table>
<tr><th>Setting</th><th>Description</th></tr>
<tr><td>Enter Url</td><td>The URL that the action should navigate to. Example: <code>https://inductiveautomation.com/</code></td></tr>
<tr><td>Open in new tab:</td><td>Specifies whether the newly opened page should replace the current page, or open in a new browser tab.</td></tr>
</table></td></tr>
<tr><td>Browser</td><td>The following feature is new in Ignition version <b>8.1.5</b>
Click here to check out the other new features

Navigates either forward or backward using browser history.<table>
<tr><th>Setting</th><th>Description</th></tr>
<tr><td>Navigate</td><td>Specifies whether to go forward or backwards in browser history when the action is triggered.</td></tr>
<tr><td>Prevent Default</td><td>Prevents the browser's default behavior from occurring. Useful when you want to prevent the browser's built-in right-click menu from showing.</td></tr>
<tr><td>Stop Propagation</td><td>Prevents events from higher up in the component hierarchy from triggering when the selected event triggers.</td></tr>
</table></td></tr>
</table>

Request Print Action

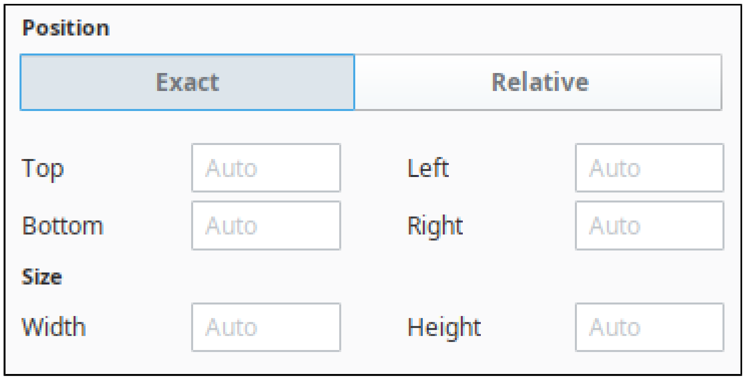
The following feature is new in Ignition version **8.1.28**
Click here to check out the other new features

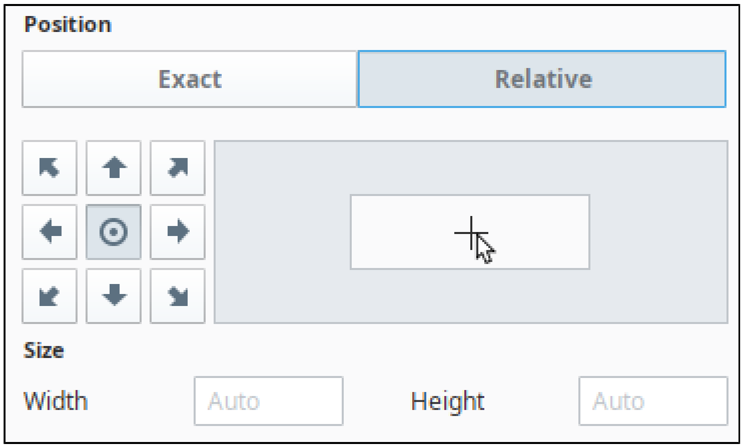
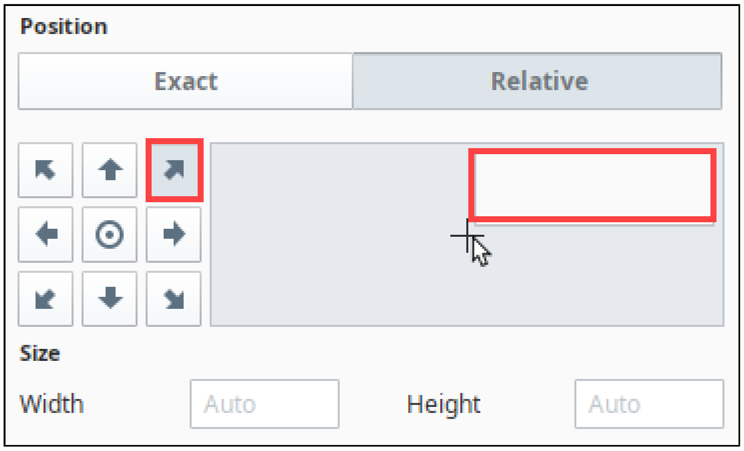
Prints the contents of the page, view, or component. To print content using Perspective component methods instead, see Requesting Print.

Action Setting	Description
Select Print Target	- **Page**: Print the contents of the page. - **View**: Print the contents of the view. - **Component**: Print the contents of the component.
Document Title	The name of the save file.

Popup Action

Opens a view as a popup, or closes an existing popup.

Action Setting	Description
Popup Action	- **Open**: Opens a view as a popup. - **Close**: Closes an existing popup. - **Toggle**: Opens a popup if it isn't open, and closes the popup if it is open.
Select View	The path to the view that should appear as a popup.
Parameters	A set of parameters to pass to the view. Add or remove parameters with the Add ✚ icon and **Delete** 🗑 icon. In the **Value** field for a given parameter, you can pass in a path to a property value using the **Parameter** ☰ icon.
Identifier	A string that specifies a unique popup identity. If you want to close an open popup from a popup action, you'll need to supply the identifier that was used to open it.
Title	A string of text to display in the titlebar. If omitted, no titlebar is used.
Show close button	A boolean indicating if a Close Icon should be displayed on the popup.
Draggable	A boolean indicating if the popup should be able to be dragged to new positions.
Resizable	A boolean indicating if the popup is allowed to be resized.
Modal	A boolean indicating if the popup should be modal, meaning it is the only view the user can interact with while open.
Background dismissible	A boolean indicating if the popup can be dismissed by clicking outside of it. This setting is only applied if the modal option is enabled. If omitted, defaults to false.
Position Exact	Exact Positioning controls where in the session the popup should appear, and how large it should be. If no specifications are given, the popup will open centered at its default size. - **Top**, **Left**, **Bottom**, and **Right** control how far the popup should be offset from each margin of the session. - **Width** and **Height** specify how large the popup should appear.  Position Exact | Relative Top: Auto — Left: Auto Bottom: Auto — Right: Auto Size Width: Auto — Height: Auto
Position Relative	

	Relative Positioning enables the popup to be positioned based off of the mouse cursor position when used with a Mouse Event. The Circle icon in the center represents the mouse location. The arrows icons represent the location the popup will appear in relation to the user's mouse cursor. Click on an arrow to position the popup window.  Position Exact Relative Size Width Auto Height Auto  Position Exact Relative Size Width Auto Height Auto
Viewport Bound	The following feature is new in Ignition version **8.1.3** Click here to check out the other new features If selected then Popups will be "shifted" on open to always open within the bounds of the viewport. If the Popup would have been larger than the viewport then Popup will be resized to fit within the bounds. This setting also prevents a Popup from being dragged even partially off-screen.

Refresh Action

Reloads the current browser tab.

Scan Barcode Action

Allows the user to scan a single barcode on their mobile device, which is then sent to the Gateway and handled by the Barcode Scanned Session Event.

This is a Native App Action, designed to help in gathering data from a mobile device. Each action needs to be configured in two parts:

1. The action is run on a mobile device, indicating that a type of data should *begin* to be gathered.
2. As the data is gathered, or once it's finished being gathered, it is sent to the Gateway to be handled by the corresponding *Session Event*.

> This action is designed specifically for the Ignition Perspective App for Android and iOS devices. If it is run in a browser session, it will be ignored.

Action Setting	Description
Barcode Type	Indicates the format of the barcode to be scanned. **Any** can be used to catch all barcodes.

Barcode Background Color	The following feature is new in Ignition version **8.1.3** Click here to check out the other new features This option enables the reading of barcodes with light or dark backgrounds, or previously white barcodes with black backgrounds couldn't be read. Options are Light, Dark, or Auto.
Context	Provides the opportunity to pass a custom object through to the Barcode Scanned Session Event. **Note:** A **session** object is already provided to that event script.

Scan Ndef NFC Action

Allows the Perspective app to catch any scans by the phone using the NFC data exchange format (Ndef), which is then sent to the Gateway and handled by the NFC Ndef Scanned Session Event. This suppresses any default behavior of the phone in catching the scan.

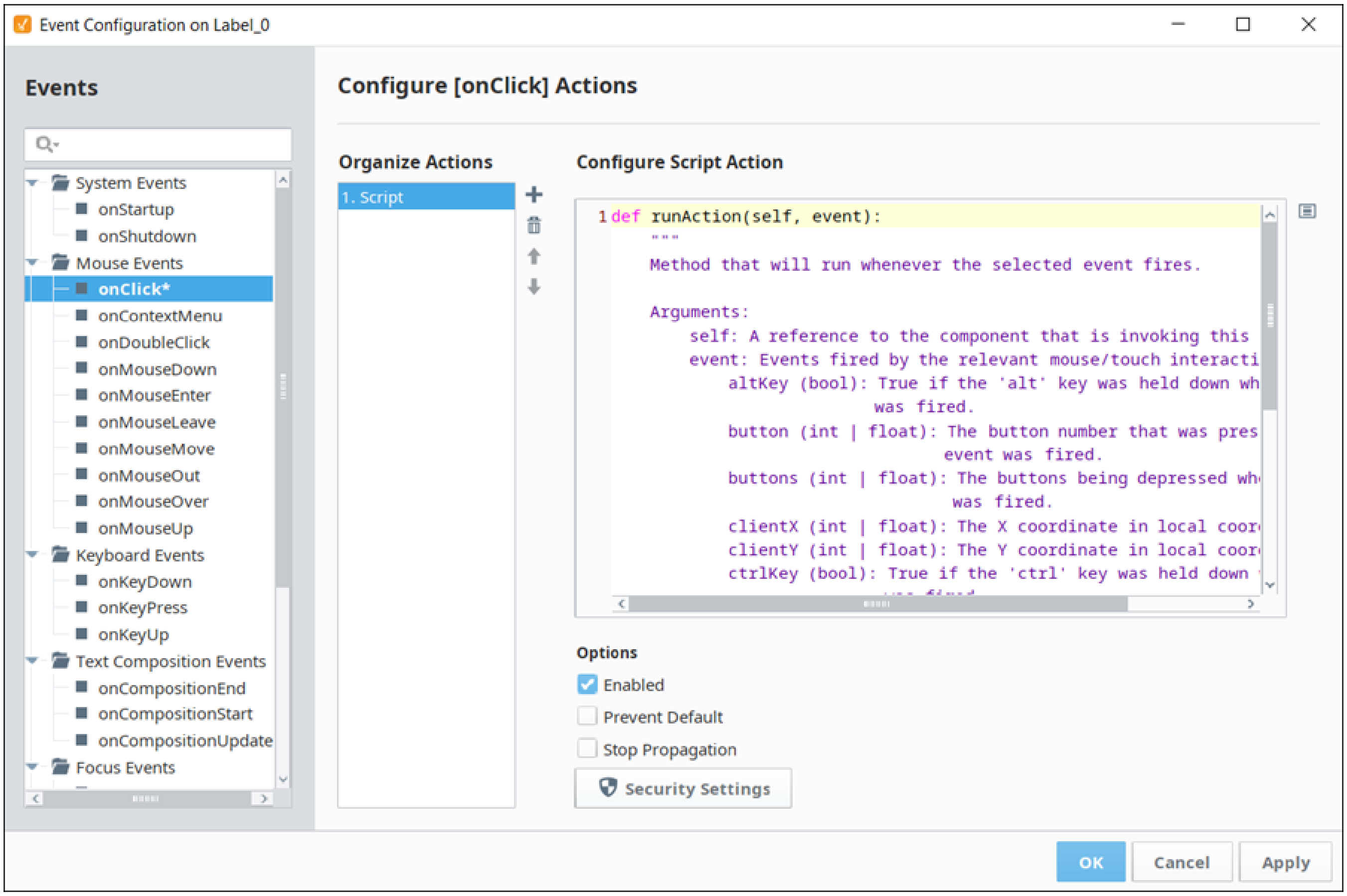
⚠ This action is designed specifically for the Ignition Perspective App for Android and iOS devices. If it is run in a browser session, it will be ignored.

Action Property	Description
Single Mode	Listens for a single NDEF scan to send.
Continuous Mode	Listens indefinitely for NDEF scans, which are sent to the gateway as they are received.
Off Mode	Turns off listening for NDEF scans.

Script Action

Write a script that happens on the event specified. See Perspective Component Methods and system.perspective Functions for more details on how to configure script actions.

Script actions contain a built-in "event" object, that further contains values pertaining to the underlying event. These values and descriptions are displayed in the docstring.



```
def runAction(self, event):
    """
    Method that will run whenever the selected event fires.

    Arguments:
        self: A reference to the component that is invoking this
        event: Events fired by the relevant mouse/touch interacti
            altKey (bool): True if the 'alt' key was held down wh
                           was fired.
            button (int | float): The button number that was pres
                                  event was fired.
            buttons (int | float): The buttons being depressed wh
                                   was fired.
            clientX (int | float): The X coordinate in local coor
            clientY (int | float): The Y coordinate in local coor
            ctrlKey (bool): True if the 'ctrl' key was held down
```

Options

☑ Enabled

☐ Prevent Default

☐ Stop Propagation

Theme Action

Changes the theme for the session.

Action Setting	Description
Select Theme	Theme that will be used, for example dark, light, light-cool etc. Dropdown list of all available themes.

Workstation Mode Action

The following feature is new in Ignition version **8.1.0**
Click here to check out the other new features

Note: This action will only take effect if called from a Session running in Workstation.

When called from a session running in Perspective Workstation, changes the mode of workstation from Windowed Mode to Kiosk Mode.

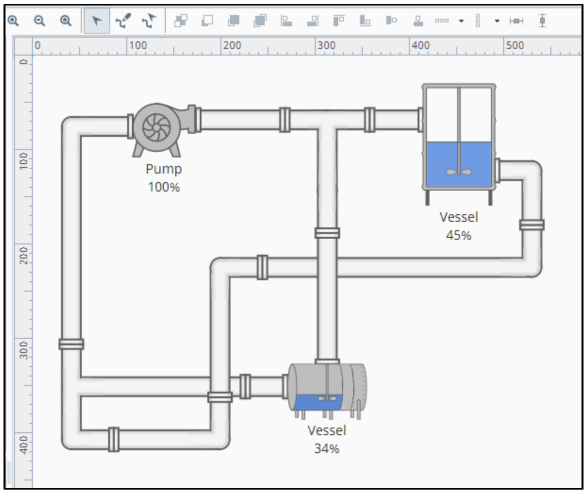
Action Setting	Description
Windowed	Enters Windowed Mode.
Kiosk	Enters Kiosk Mode
Toggle	Toggles the current mode between Windowed and Kiosk.

Perspective Pipes

The following feature is new in Ignition version **8.1.10**
Click here to check out the other new features

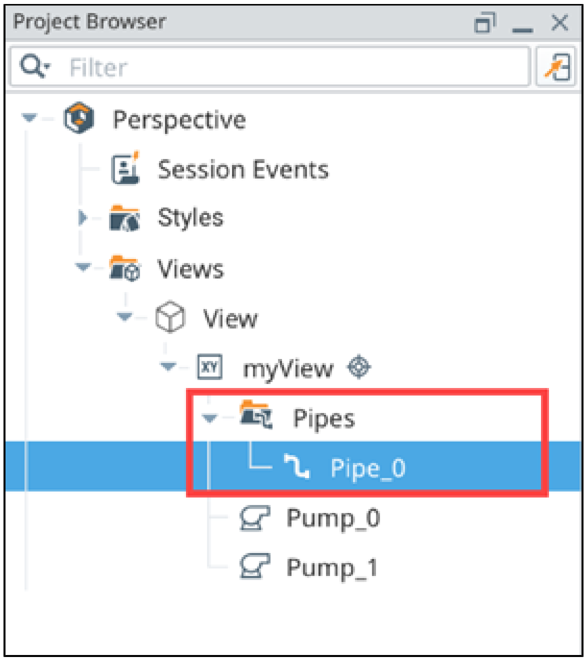
On this page ...

Pipes can be drawn within a coordinate container. These pipes can be used to visualize flow between separate objects on a view.

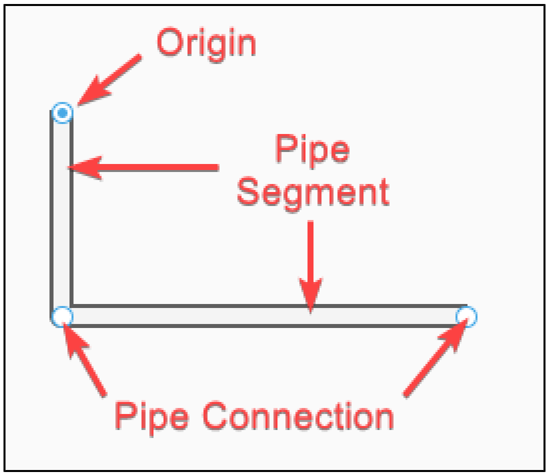


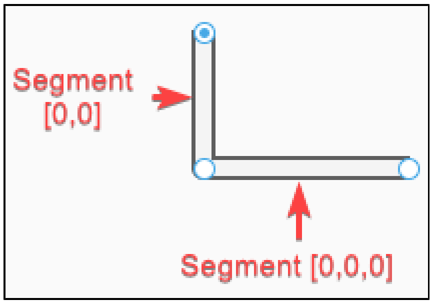
Pipes are created by a dedicated drawing tool on the toolbar, as opposed to being dragged onto a container like other components. Pipes are effectively drawn on the bottom of the z-order in a coordinate container, hence any components added into the container will always be placed above the pipes.

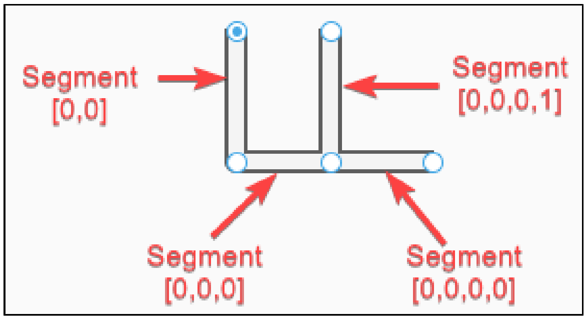
Each pipe consists of an origin, its connections, connections of connections, and so on. Each pipe is listed as a separate entry in the Project Browser, under "Pipes".



Terminology



Pipe	An object containing all the pipe connections and pipe segments. In addition, a Pipe contains a name, appearance, and appearance specific props.
PipeConnection	An XY coordinate within the pipe. Visualized as a circle.
Origin	The first Pipe Connection within a pipe. Note that origins differ visually from other Pipe Connections, in that they contain a dot at their center.
PipeSegment	The segment formed by a Pipe Connection, and one of its immediate child connections. A segment is a visual representation of relationship between other connections. Each connection has an array of connection indices (starting at 0), and segments can be represented as a series of indices. For example, [0,1,0] would represent the first pipe, second connection, then first sub connection. Consider the simple pipe below of three connections and two pipe segments. The top of Segment [0,0] is the origin and the bottom of Segment [0,0] is the pipe connection where Segment [0,0,0] begins, turning the pipe flow from a vertical to horizontal direction. The end of Segment [0,0,0] is the final connection.  We can add two new connections, which could look like the following diagram. Because of the connection locations, Segment [0,0,0,0] follows the existing naming structure, but Segment [0,0,0,1] is given the number 1 since it was added later and is the second child.



Note that segment indices are represented in the Project Browser.

Example JSON

```
[
  {
    "name": "pipe",
    "appearance": "auto",
    "flanges": true,
    "lineVariant": "solid",
    "start": "none",
    "end": "none",
    "fill": "",
    "stroke": "",
    "width": 10,
    "origin": {
      "x": 258,
      "y": 168,
      "connections": [
        {
          "x": 258,
          "y": 236,
          "connections": [
            {
              "x": 307,
              "y": 236,
              "connections": [
                {
                  "x": 360,
                  "y": 236
                },
                {
                  "x": 307,
                  "y": 168
                }
              ]
            }
          ]
        }
      ]
    }
  }
]
```

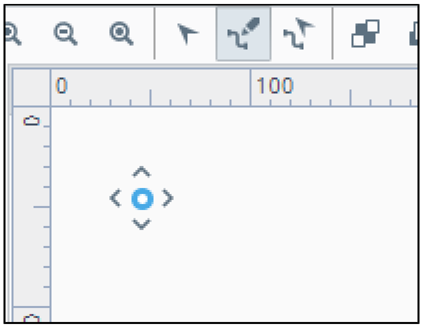
Creating Pipes

Click on the Pipe Drawing Tool toward the top of the Designer. While active, clicking anywhere inside a coordinate container will create a new Pipe with an **origin** at the mouse location.

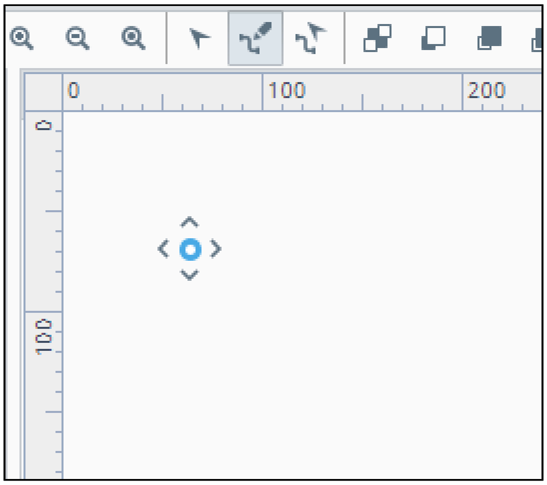
Note:

Pipes can only be drawn in a Coordinate Container. The Pipe Draw Tool and Pipe Move Tool will be disabled while other types of containers are selected.

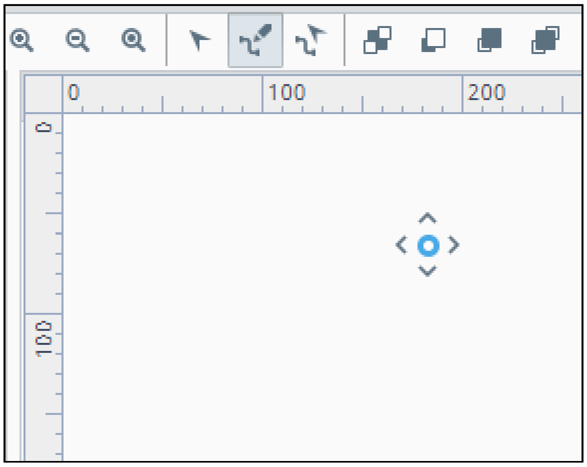
By default, a Coordinate Container's aspectRatio property does not have a value. As such, setting your container's aspectRatio to a value like 1:1 when creating Pipes is recommended. This helps Pipes remain anchored to components if the components change size in proportion with the container.



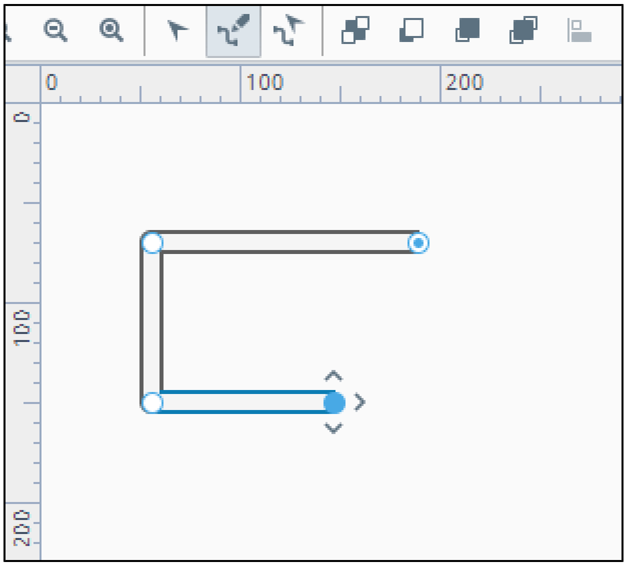
All connections, including the origin, can be repositioned by clicking and dragging.



New connections can be drawn by dragging the arrows around the connection. Letting go will create a new connection. By default, all segments are drawn orthogonally. Holding the **Alt/Option** key will allow the segment to be drawn in fixed angles at 15 degree increments. Alternatively, holding **Shift** while dragging allows the segment to be placed freely without snapping.

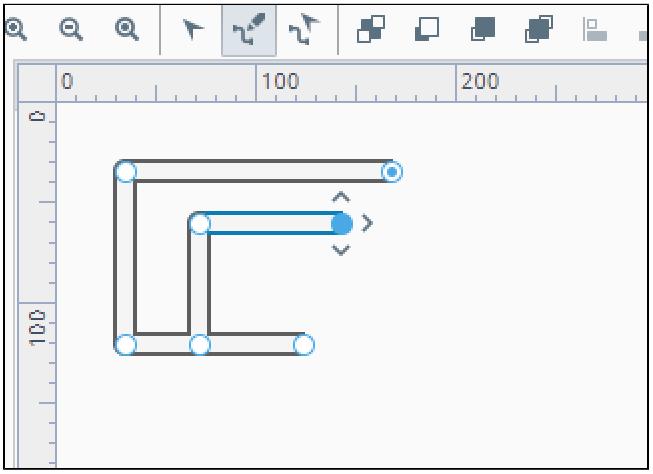


New connections can be added to existing segments. Doing so allows the segment to split off into a new segment.

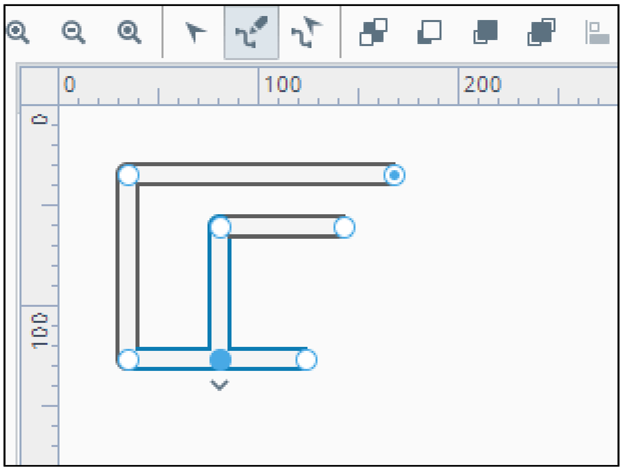


To delete a segment, click on a connection in the segment and press the **Delete** key, or by right-clicking and selecting "Delete".

Existing connections can be moved by dragging. All attached segments will move with the origin.

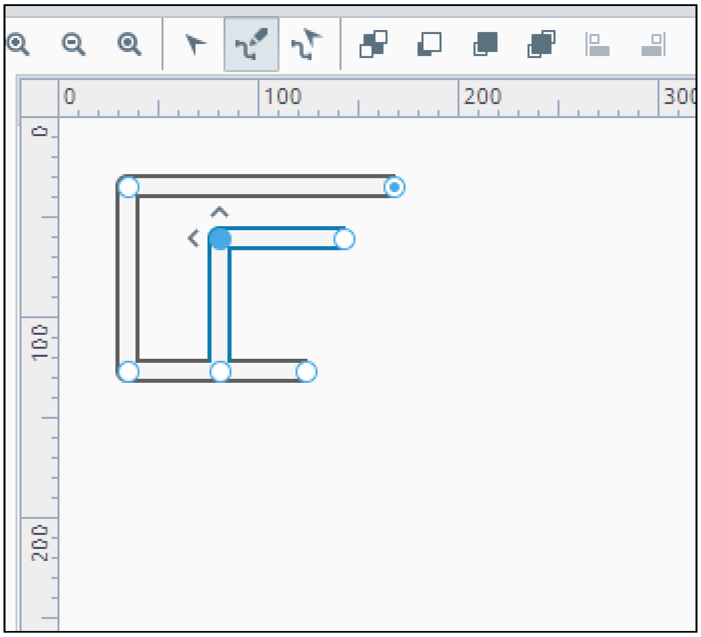


If **Shift** is held while repositioning a connection, then connecting segments will bend.



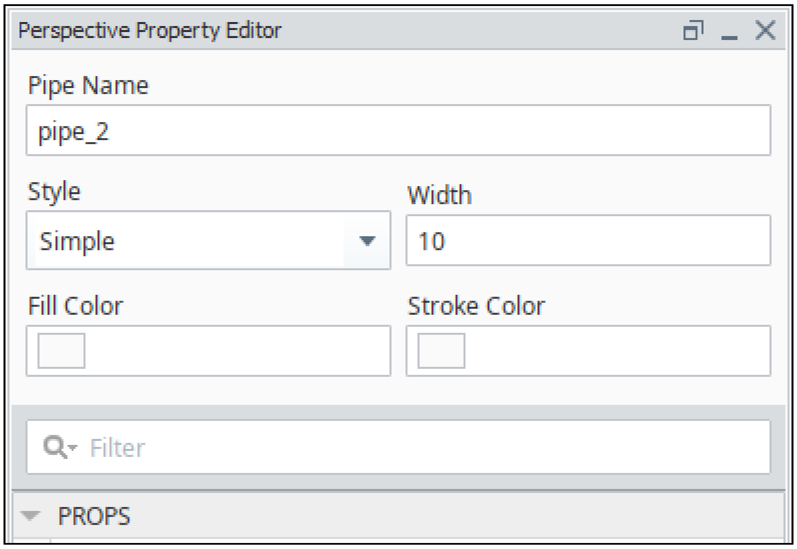
Moving Pipe Sets

The Pipe Move Tool can be used to move an entire set of pipes, as well as resize the set.



Pipe Appearance Editor

While either the Pipe Draw Tool or Pipe Move Tool are selected, the top of the property editor will display some additional fields that can be used to modify the appearance of the selected set of pipes. Additional fields are displayed depending on the selected Appearance of the pipe.



Perspective Property Editor

Pipe Name: pipe_2

Style: Simple

Width: 10

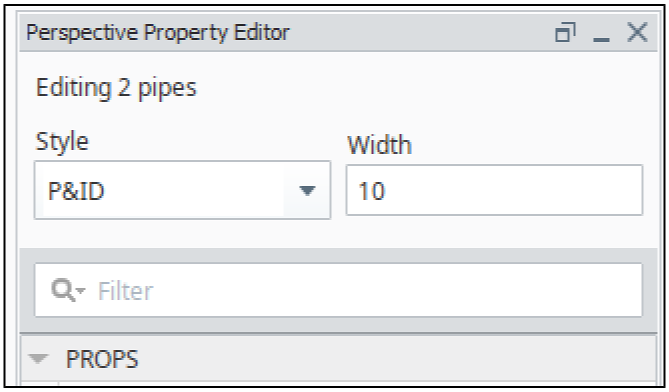
Fill Color

Stroke Color

Filter

PROPS

When multiple pipes are selected, only shared fields are displayed. For example, one pipe with an appearance of P&ID and another using simple appearance are selected, only the Appearance and Width properties will be shown since those are the only fields available to both apperances.


Perspective Property Editor

Editing 2 pipes

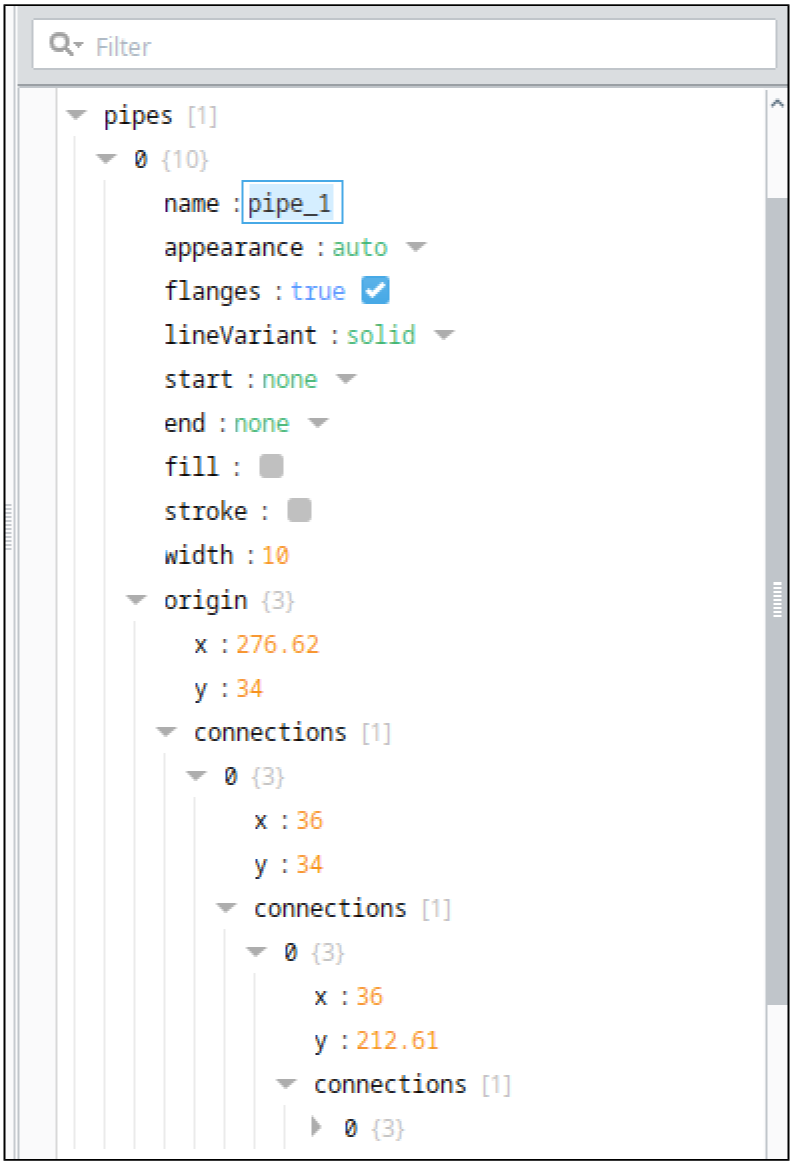
Style: P&ID

Width: 10

Filter

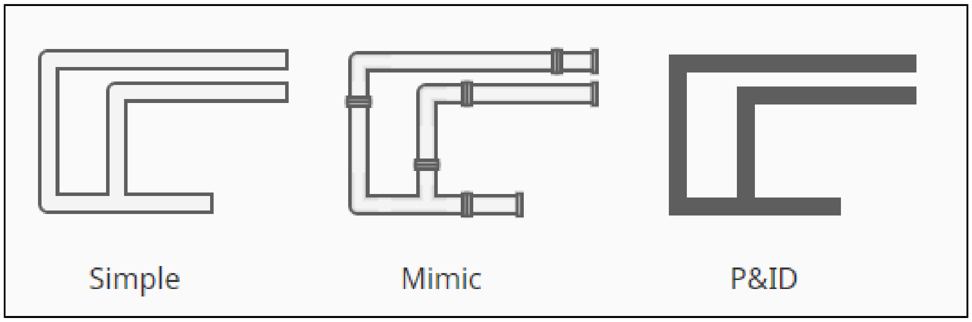
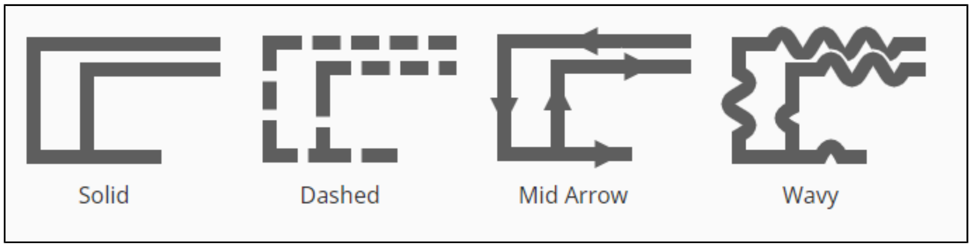
PROPS

In addition, a coordinate container that has one or more sets of pipes will display a pipes property in the Property Editor. Each origin is represented as an element in the pipes array. Each origin has X and Y coordinates, and an array of connections, which lead to other origins.


```
Filter
pipes [1]
  0 {10}
    name : pipe_1
    appearance : auto
    flanges : true ✓
    lineVariant : solid
    start : none
    end : none
    fill :
    stroke :
    width : 10
    origin {3}
      x : 276.62
      y : 34
      connections [1]
        0 {3}
          x : 36
          y : 34
          connections [1]
            0 {3}
              x : 36
              y : 212.61
              connections [1]
                0 {3}
```

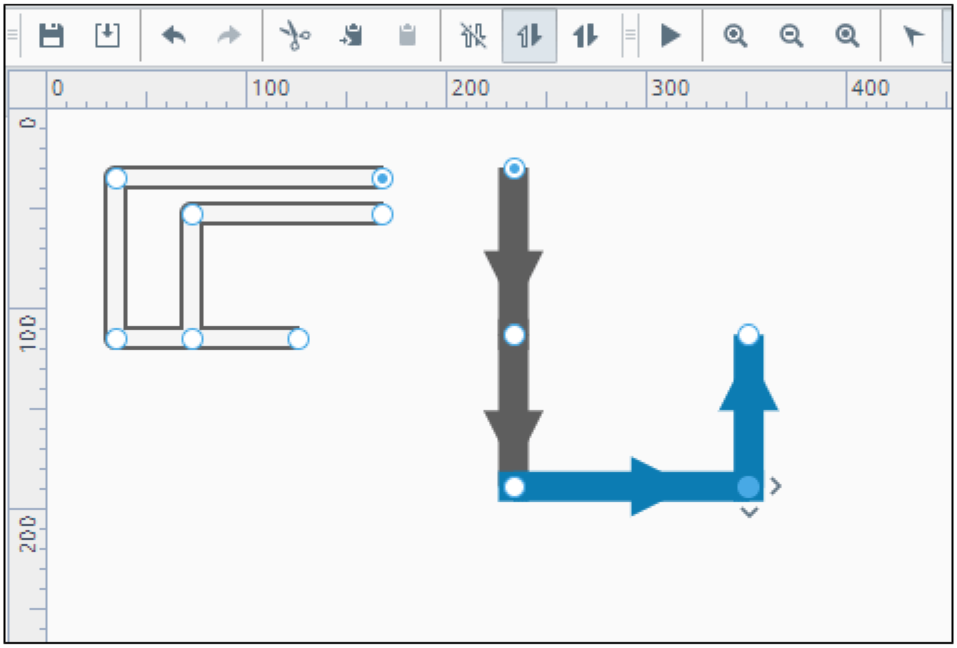
Pipe Properties

Property	Description
Pipe Name	The name of the selected set of pipes, as they appear in the Project Browser.

Appearance	Changes the look of the pipes. Note that different appearances have additional properties.  In addition to the appearance options, there is an **auto** value will will cause the set to use the appearance specified under `session.prop autoAppearance`.
Width	Determines the width of pipes in pixels within a single set.
Fill Color	The fill color to use when appearance is set to either **Simple** or **Mimic**. This setting is ignored when appearance is set to **P&ID**.
Stroke Color	The stoke color to use when appearance is set to either **Simple** or **Mimic**. This setting is ignored when appearance is set to **P&ID**.
Display flanges on pipe (Mimic Appearance Only)	Determines if the pipe set should show flanges.
Line Variation (P&ID Appearance Only)	Allows a unique variation to be applied to the set. 
Line Color (P&ID Appearance Only)	The color to use for the pipe set.
Start (P&ID Appearance Only)	Determines what the start of a pipe set look like. Applies to the starting origin of a set.
End (P&ID Appearance Only)	Determines what the end of a pipe set look like. Applies to all origins that don't lead into another origin.
Visible	The following feature is new in Ignition version **8.1.13** Click here to check out the other new features Determines whether the pipe set is visible or not. Default setting is True.

Combining Pipes

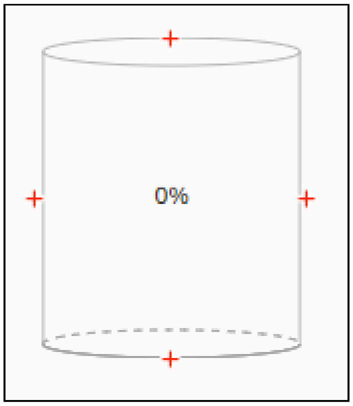
Dragging a pipe to another will combine the pipes. Combining pipes will cause the pipes to share the same appearance. However this combination can be overridden by pressing the Undo button (or by pressing **Ctrl-Z/Command-Z**) once. This allows different pipes to overlap without actually combining.



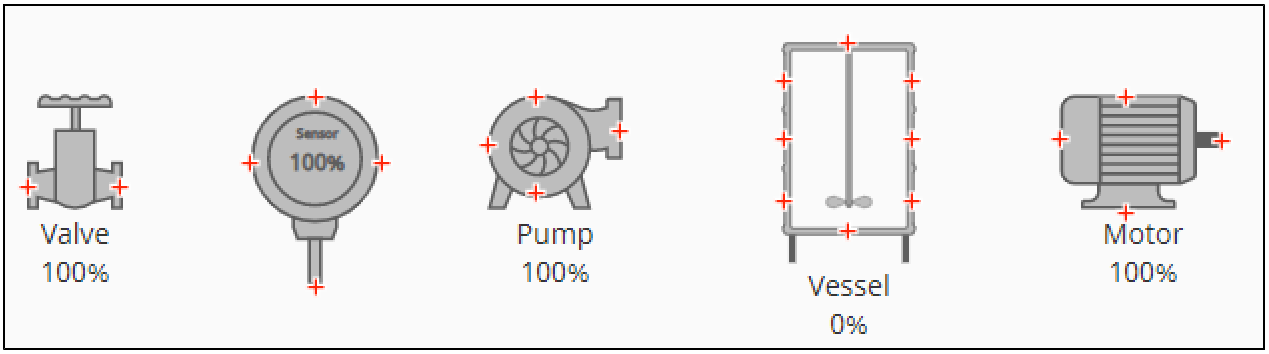
Additionally, a connection can be dragged onto it's immediate sibling or parent, which effectively removes the dragged connection.

Component Anchors

While using the Pipe Drawing Tool, other components in the container will display anchor points.



Symbol type components have custom anchors, positioned at appropriate locations.



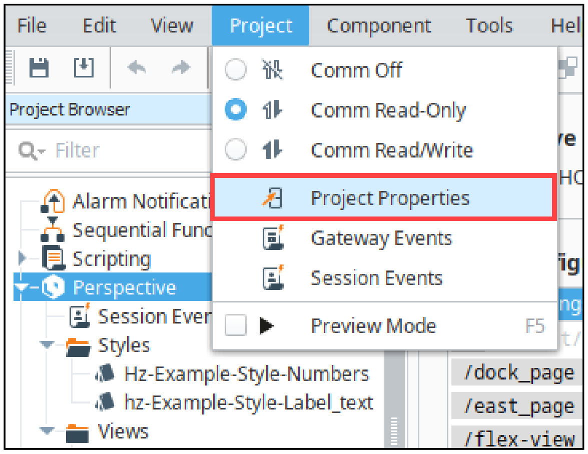
Dragging a segment near the anchor will cause the segment to snap to the anchor (unless the **Shift** key is currently being held). This can be useful when leading segments into different components. Note that anchors are only used to guide the segment in the designer. The pipes are not "connected" to the component, so moving the component later will not cause the pipe segment to move.

Perspective Project Properties

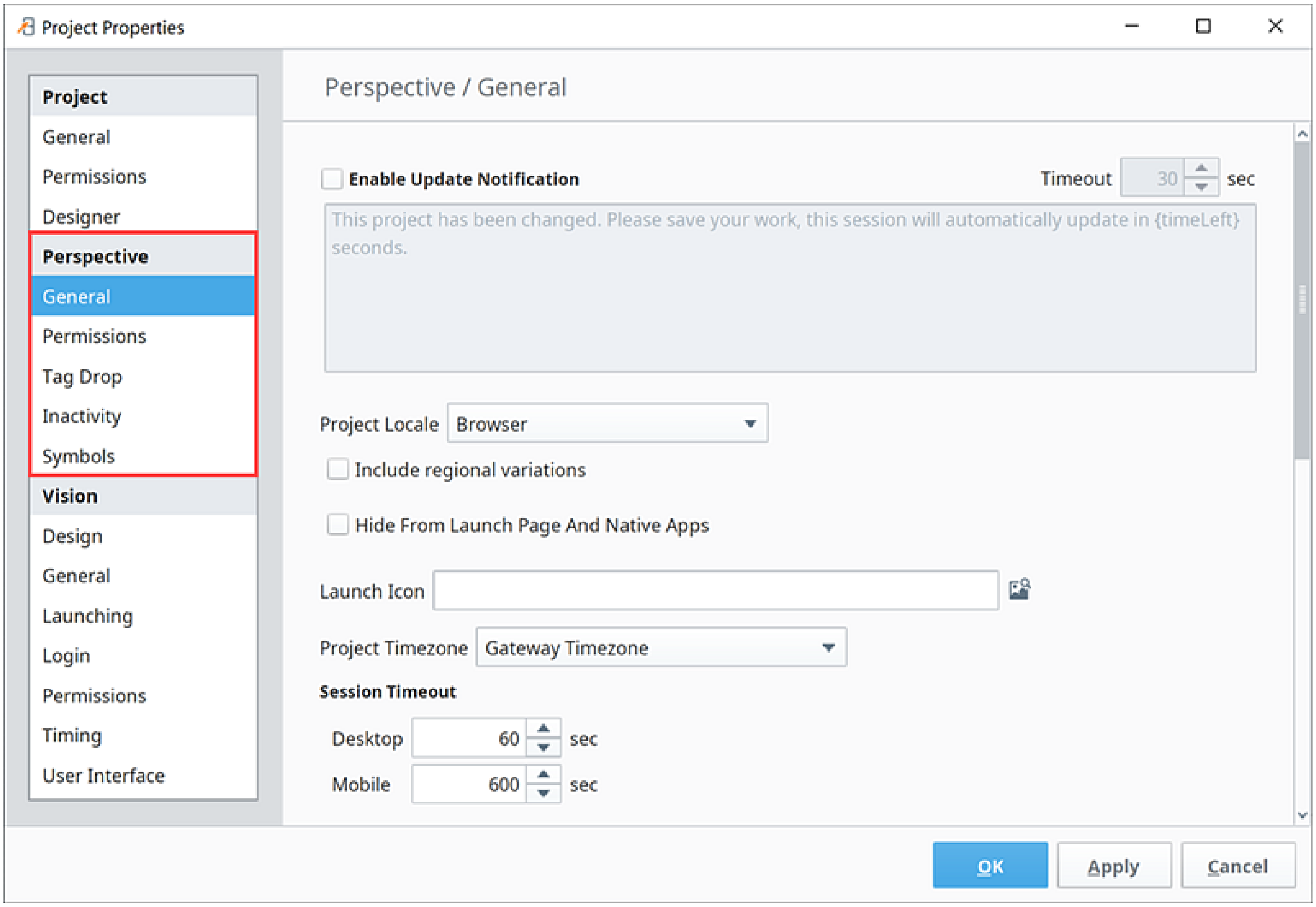
On this page ...

There are a number of properties you can set for your Perspective projects within the Designer. For example, there are properties for setting security levels, configuring how the sessions receive updates, and more.

To access the Project properties, in the Designer, click on **Project** tab on the menu bar. Then select **Project Properties**.



Project properties span several functional areas each containing settings applicable to that area. Scroll down to the Perspective section.



Perspective / General

□ **Enable Update Notification** Timeout 30 sec

This project has been changed. Please save your work, this session will automatically update in {timeLeft} seconds.

Project Locale: Browser

□ Include regional variations

□ Hide From Launch Page And Native Apps

Launch Icon:

Project Timezone: Gateway Timezone

Session Timeout

Desktop 60 sec

Mobile 600 sec

Perspective General Properties

These general properties apply to the Perspective Sessions.

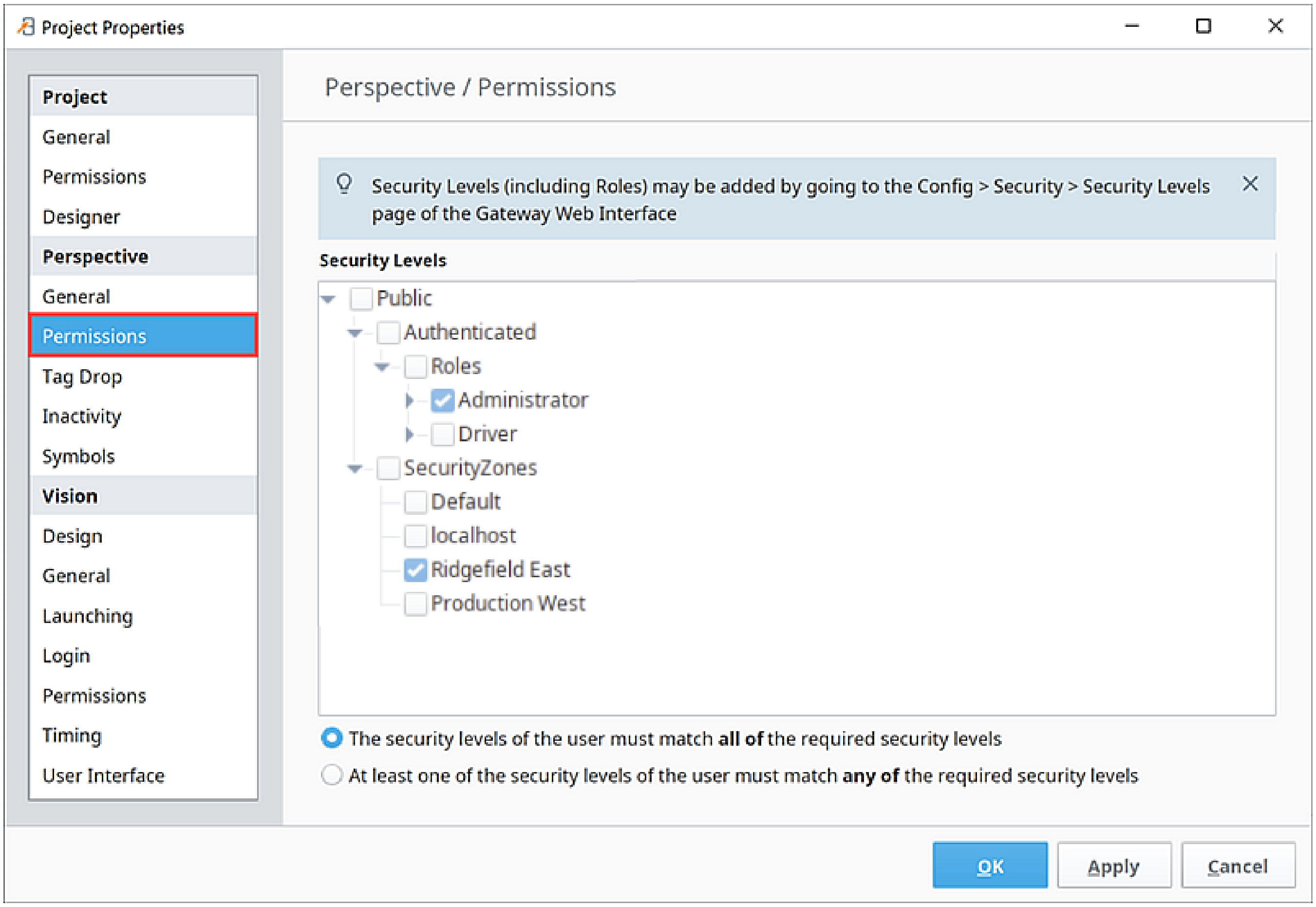
General	
Property	**Description**
Enable Update Notification	Enables notifications for sessions when the project is changed. When notification is enabled, this is the message that will be displayed. The token {timeLeft} will be replaced with the seconds remaining until update. If false, update will be immediate.
Project Locale	Select a project locale.
Include regional variations	Enables regional variations for the locale.
Hide from Launch Page and Native Apps	When selected, the project to be hidden from the Session Launcher and mobile app project listing.
Launch Icon	The image specified here is used to represent the project on the launch page and the Ignition Perspective mobile app. 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.
Project Timezone	Timezone for this project. Dropdown list includes the Gateway timezone, Session timezone, or specific timezones around the globe.
Identity Provider	Identity Provider (IdP) for this project if a Perspective Project had an IdP set for it prior to the release 8.0.6. **Note:** For new installations after 8.0.6, the Identity Provider property is now located in Project Properties > Project General > Security Settings The property will also move to the new location under Project Properties > Project General > Security Settings if you change set this property to none.
Session Timeout Desktop	The time, in seconds, for that the Gateway will wait for desktop devices to respond.
Session Timeout Mobile	The time, in seconds, for that the Gateway will wait for mobile devices to respond.
Session Closed Message	Message to be displayed if system.perspective.closeSession is invoked.
Page Closed Message	Message to be displayed if system.perspective.closePage is invoked.
Logged Out Message	The following feature is new in Ignition version **8.1.8** Click here to check out the other new features Determines the message displayed when a user logs out of a session. Only applies when the project requires authentication.

Perspective Permissions Properties

The Perspective Permissions properties restrict project access to specific security levels and security zones.

Permissions	
Property	Description
(Security Levels Tree)	Interactive tree that shows the current authenticated roles and security zones.
(First radio button)	If selected, then the security levels of the user must match **all of** the required security levels
(Second radio button)	If selected, then at least one of the security levels of the user must match **any of** the required security levels.

In the following example, a user must have the Administrator security level and Ridgefield East security zone to be able to access this project.



Perspective / Permissions

Security Levels (including Roles) may be added by going to the Config > Security > Security Levels page of the Gateway Web Interface

Security Levels

- □ Public
  - □ Authenticated
    - □ Roles
      - ☑ Administrator
      - □ Driver
  - □ SecurityZones
    - □ Default
    - □ localhost
    - ☑ Ridgefield East
    - □ Production West

● The security levels of the user must match **all of** the required security levels

○ At least one of the security levels of the user must match **any of** the required security levels

Perspective Tag Drop Properties

Tag Drop properties provide a way to link certain data types of Tags to commonly used components when dragging-and-dropping a Tag onto a view. This is done in **Project Properties** under **Tag Drop** in three steps: select the data type, select the component to create, and select the bindings that are created on that component.

- **Data Type Configuration:** This setting determines which components show up in the popup list when dragging a Tag into a view.
  You can switch between the different Tag data types to see what sorts of components are allowed to be dropped for each of those types. Add or remove new entries in the list using the **A dd** ✚ icon and **Delete** 🗑 icon. You can then double-click on the new blank cell and select a component from a dropdown.
- **Component Bindings Configuration:** This setting determines which bindings are created when a component is dropped into a view.
  Select a component, then set up or modify the bindings for the selected component (which is typically just the Tag value bound to the value/text prop on a component). Add or remove new binding entries using the **Add** ✚ icon and **Delete** 🗑 icon. Double-click the Tag Property and Prop Path cells to fill in properties.

INDUCTIVE UNIVERSITY

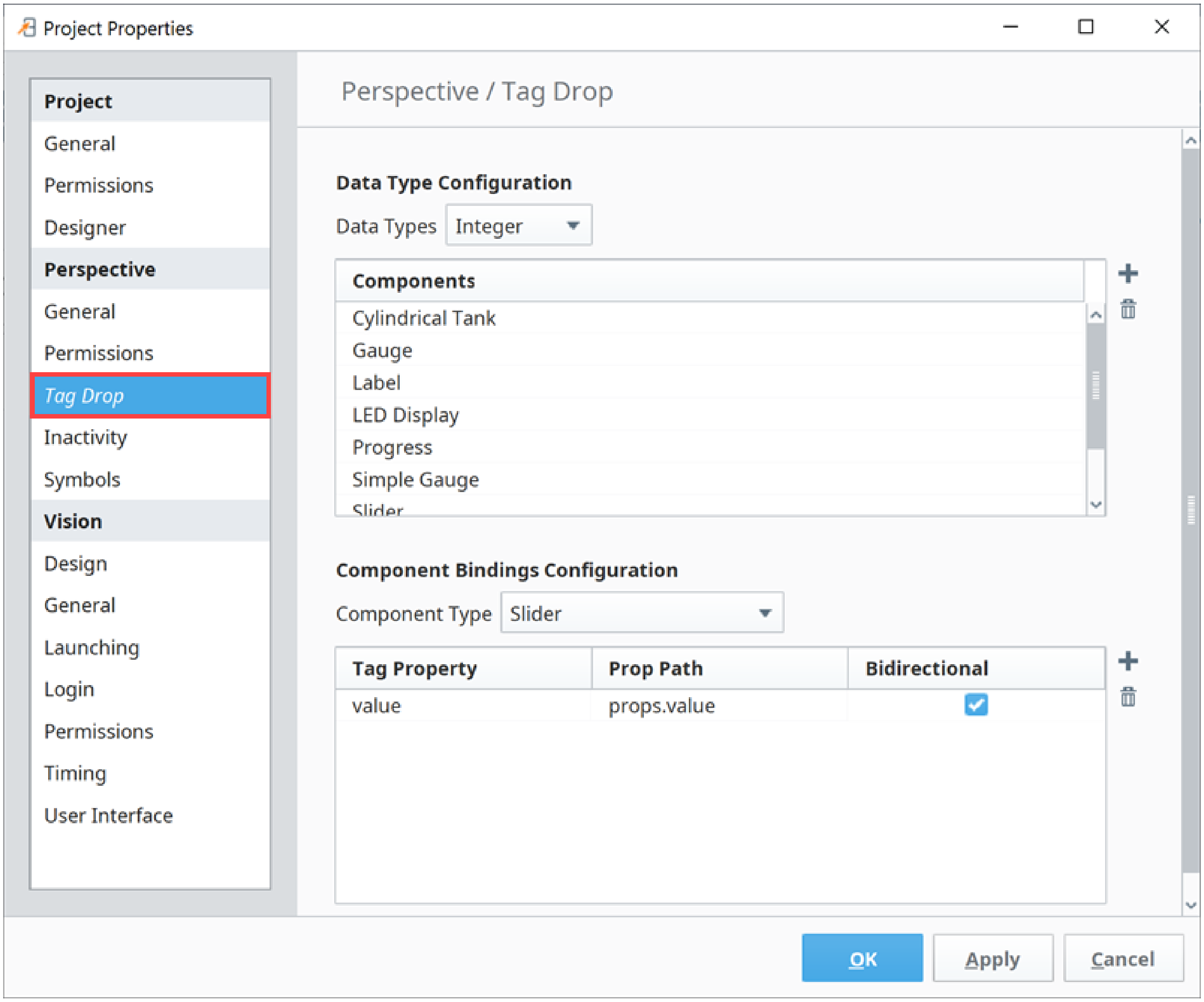
Tag Drop

Watch the Video

> **Note:** The Component Binding Configuration table values are tied to the component currently selected in the Component Type dropdown, they are not related to what is selected in the Data Type Configuration table above.

Dropping tags onto existing components can now create bindings on those components if the data type of the tag matches a Tag Drop configuration in the project. Tag Drop configuration is what determines if a binding is configured or not.

The Tag Drop interface enables the **Bidirectional** setting by default when adding Input Components.



Data Type Configuration

Data Types: Integer

Components
Cylindrical Tank
Gauge
Label
LED Display
Progress
Simple Gauge
Slider

Component Bindings Configuration

Component Type: Slider

Tag Property	Prop Path	Bidirectional
value	props.value	☑

Tag Drop	
Property	**Description**
DataType Configuration	This section is to set up a list of components to show for each data type that can be dragged onto the view.
Data Types	Dropdown list of the available datatypes in Ignition. Choose a datatype to see a list of components associated with it.
Components	List of components associated with selected data type.
Component Bindings Configuration	This section is to set up bindings when a Tag is dropped.
Component Type	Dropdown list of the available components in Perspective.
Tag Property	The name of a Tag's property to be used in a binding. "`value`" is most commonly used.
Prop Path	The property path on the component where the binding will be created. props.value or props.text are most commonly used.
Bidirectional	Check box to indicate if the binding should be bidirectional. The **Bidirectional** property is set by default when adding Input Components.

Perspective Inactivity Properties

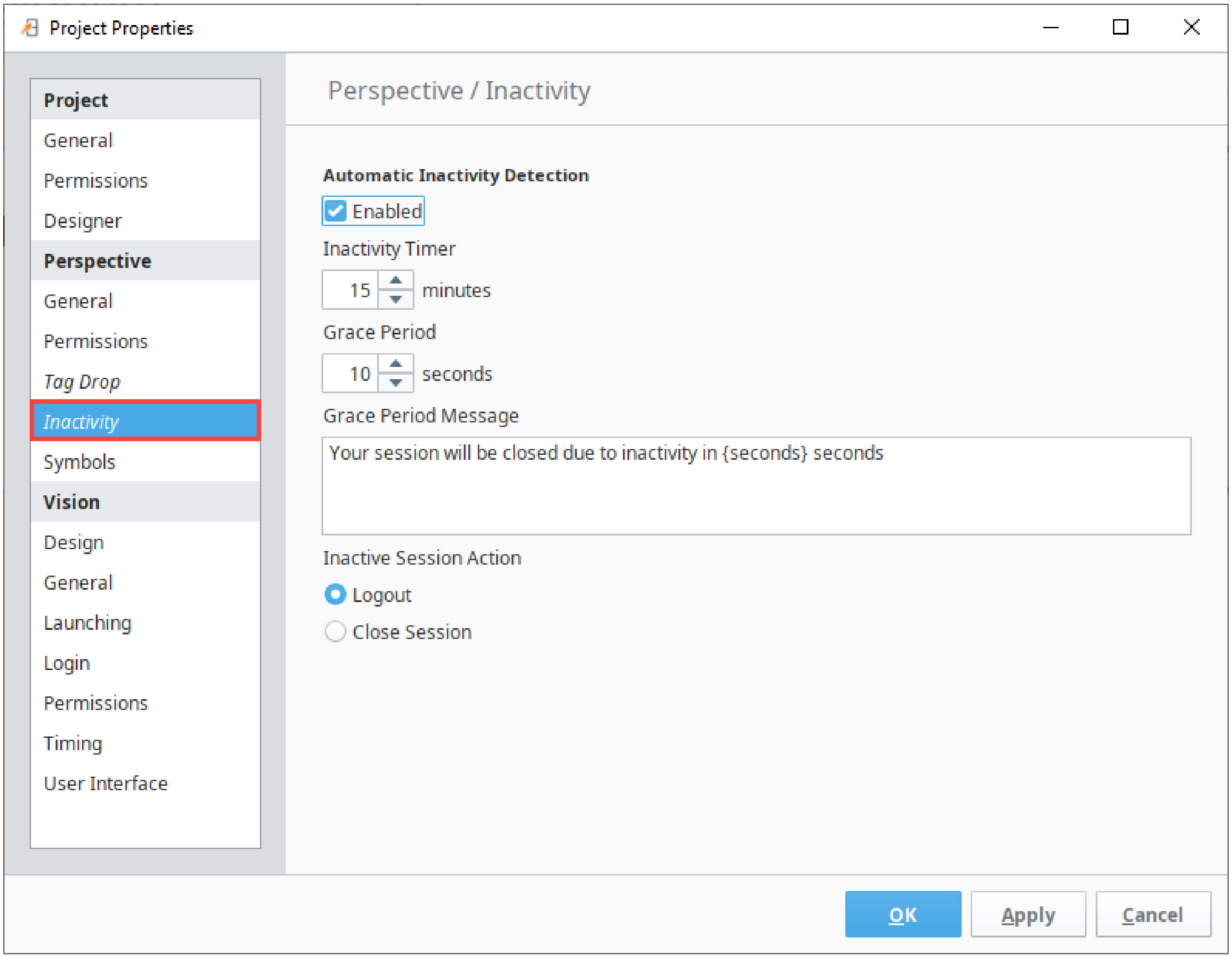
Perspective has Inactivity Timeout settings in Project Properties. You can set an Inactivity timer that either closes the Perspective session or logs the user out if no activity is detected after a specified number of minutes. Activity is considered opening a new tab, clicking, typing. If Perspective is running on a mobile device, activity is considered a swipe or tap. The Gateway is the "time keeper" for inactivity timer.

This inactivity feature pertains to perspective sessions only. When the Inactive Session Action occurs (regardless of whether "Logout" or "Close Session" is selected), the user will still be logged into the Identity Provider.

> **Note:** When enabling this inactivity feature, it is highly recommended that you also enable the "Always ask the IdP to re-authenticate users by default" setting under Project Properties, as this will require that any user attempting to log into a Perspective session will have to provide their credentials first.

The following are the properties for the Inactivity Timeout:

Inactivity	
Property	**Description**
Enabled	Whether the automatic inactivity detection option is enabled.
Inactivity Timer	Time (in minutes) in which a session can be inactive before the user is logged out or the session is closed. The maximum value is 2,147,483,647.
Grace Period	Grace period (in seconds) after the Inactivity Timer time has passed but before the user will be logged out or the session closed.
Grace Period Message	Message to be displayed before a session becomes inactive. Use the placeholder {seconds} to indicate remaining time. (Optional.)
Inactive Session Action	Action that occurs when the session becomes inactive. Options are: • Logout - Log the user out of the session. Note that the user will still be logged in with the • Close Session - Close the session.



Perspective / Inactivity

Automatic Inactivity Detection

☑ Enabled

Inactivity Timer

15 minutes

Grace Period

10 seconds

Grace Period Message

Your session will be closed due to inactivity in {seconds} seconds

Inactive Session Action

● Logout

○ Close Session

Perspective Symbols Properties

The following feature is new in Ignition version **8.1.26**
Click here to check out the other new features

The Symbols section displays built-in states for symbol visual theme configurations and allows you to create your own customized states. Selecting a state will bring up the Configure State settings, which display the state name and the Available and Applied Symbols lists. Selecting a visual theme state under the main state name will access the Edit State Style settings, which allow further color and function customization. Additionally, there are three icons on the right of the displayed States list to add and edit new states:

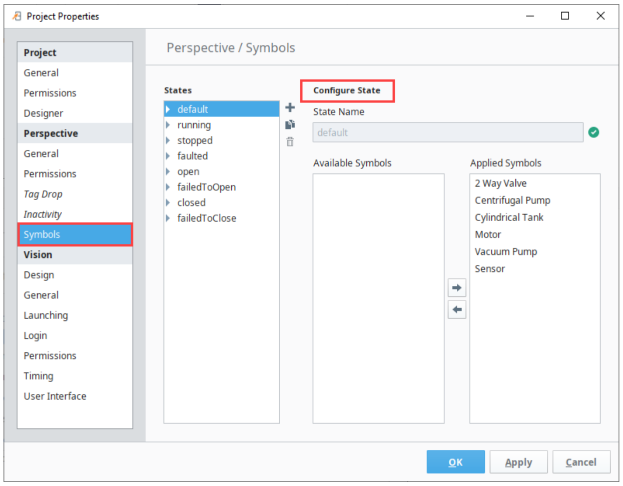
- Add ➕ icon: Creates a new state with full customization access. Color and function settings pre-populate with the built-in default state built-in settings, but will have no preset applied symbols.
- Duplicate ⧉ icon: Creates a new state based on a selected pre-existing state. Duplicated state names will automatically populate based on the original state name. Duplicated states have full customization access regardless of the state they were created from.
- Delete State 🗑 icon: Deletes selected custom states. A popup message will require you to confirm your selection before deleting. Built-in states can be edited, but not deleted.

When editing a built-in or custom state themes, you can use the Undo ↶ icon to replace changes with the last saved settings or the Reset Theme ⟲ icon to return all settings back to their original configuration when the state was first created.

The following table lists the properties for the Symbols States:

Symbols

Configure State (Only State Name Selected)	
Property	**Description**
State Name	The name of the state containing the light/dark visual state styles. Built-in state names cannot be changed.
Available Symbols	Lists the symbols that are available to use the selected state, but do not have it as a currently applied option.
Applied Symbols	Lists the symbols that will show this state as an option to apply when configuring the symbol. Add or remove applied symbols by selecting a symbol and using the Add or Remove Applied Symbol ➡ ⬅ arrows.
Edit State Style (State Name > Visual State Selected)	
Property	**Description**
Primary Color	Sets the symbol fill color.
Secondary Color	Sets the alternate symbol fill color. This color is used when Enabled Flashing is checked.
Tertiary Color	Sets the vessel fill color.
Stroke Color	Sets the symbol outline color.
Enable Animation	Enables symbol animation properties.
Enable Flashing	Enables symbol fill color to alternate between the set Primary and Secondary Colors.



Perspective / Symbols

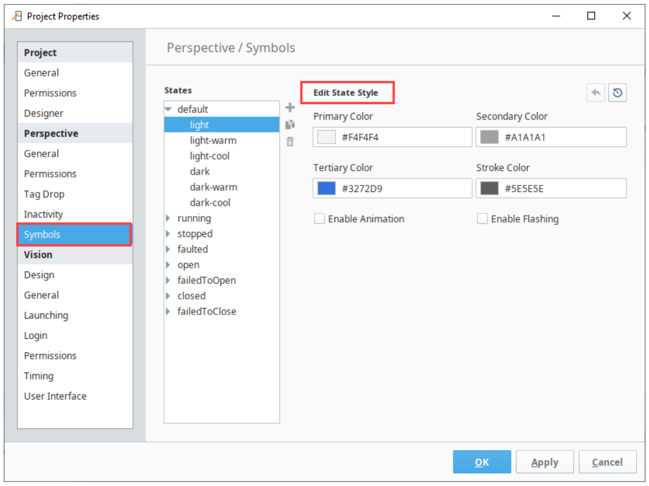
States: default, running, stopped, faulted, open, failedToOpen, closed, failedToClose

Configure State

State Name: default

Available Symbols

Applied Symbols: 2 Way Valve, Centrifugal Pump, Cylindrical Tank, Motor, Vacuum Pump, Sensor



Perspective / Symbols

States: default (light, light-warm, light-cool, dark, dark-warm, dark-cool), running, stopped, faulted, open, failedToOpen, closed, failedToClose

Edit State Style

Primary Color: #F4F4F4

Secondary Color: #A1A1A1

Tertiary Color: #3272D9

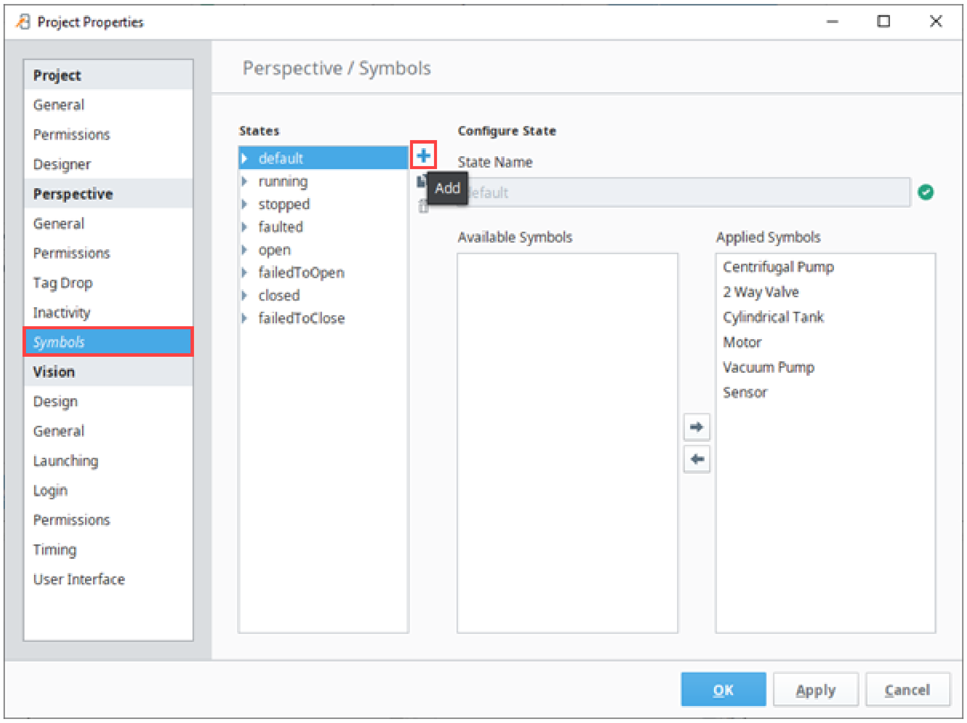
Stroke Color: #5E5E5E

□ Enable Animation

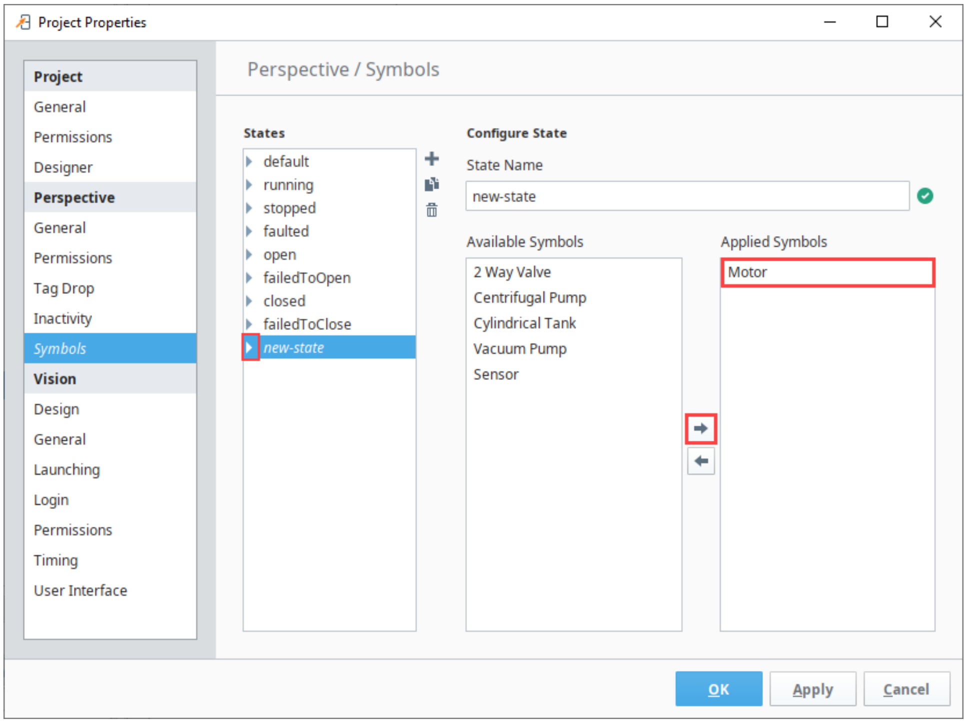
□ Enable Flashing

Create and Apply a Custom State

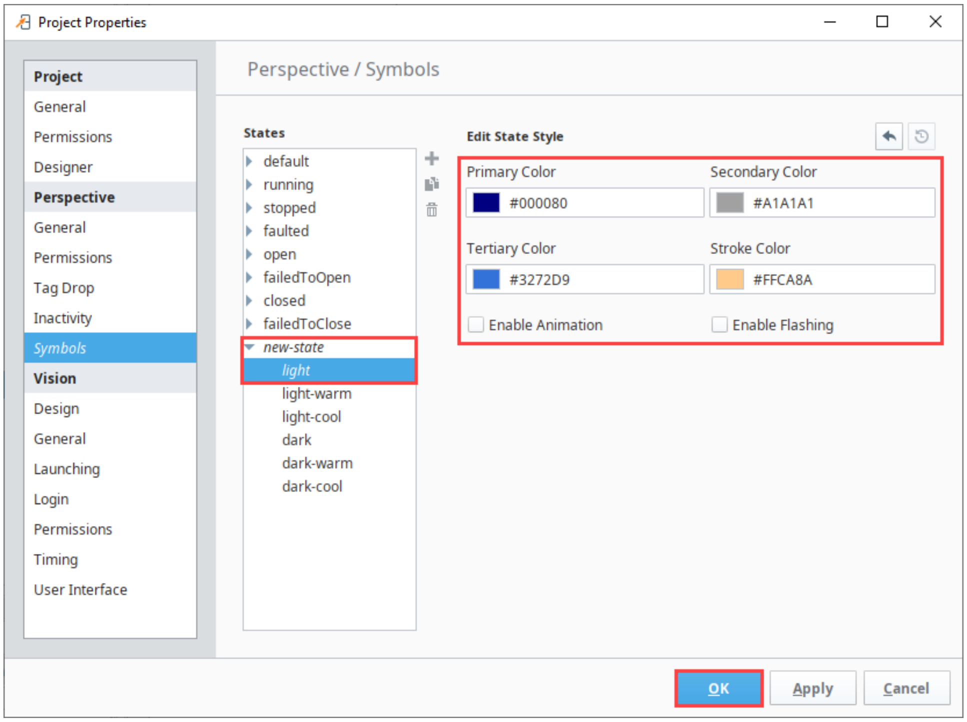
1. Open Project Properties.
2. Select **Symbols**.
3. Click the **Add** ➕ icon. Rename your new state if desired. In this example, it will be left as new-state.



4. Use the **Add Applied Symbol** ➡ arrow to select and apply your new state to desired symbols. This example will make this state applicable to the **Motor** symbol.
5. Use the state dropdown to select the visual themes you want to edit. This example will edit the light visual state.



6. Set your desired color, animation, and flashing configurations under Edit State Style.
7. Click OK.



Project Properties

Perspective / Symbols

States

Edit State Style

Primary Color: #000080

Secondary Color: #A1A1A1

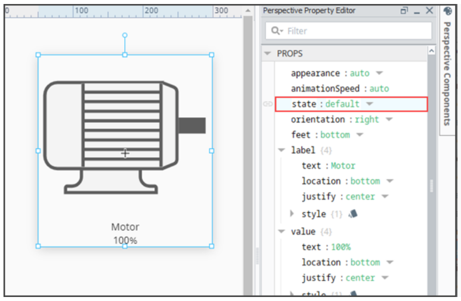
Tertiary Color: #3272D9

Stroke Color: #FFCA8A

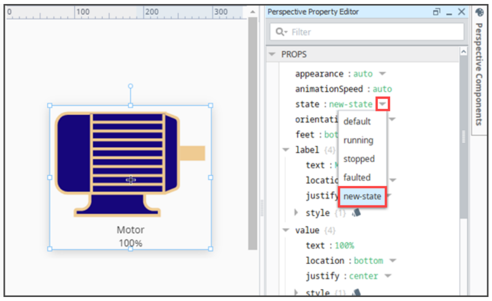
□ Enable Animation

□ Enable Flashing

8. Access a view and place a **Motor** symbol.
9. Select the Motor and change the state field from default to the custom state, in this case **new-state**. The new state configurations are immediately applied and displayed in the view.



```
appearance : auto
animationSpeed : auto
state : default
orientation : right
feet : bottom
label {4}
  text : Motor
  location : bottom
  justify : center
  style {1}
value {4}
  text : 100%
  location : bottom
  justify : center
  style {1}
```



```
appearance : auto
animationSpeed : auto
state : new-state
orientati
feet : bot
label {4}
  text : M
  locatic
  justify
  style {1}
value {4}
  text : 100%
  location : bottom
  justify : center
  style {1}
```

Styles

Perspective components have Style properties that enable you to customize the look and feel of components on the screen through various properties. These styles are based on Cascading Style Sheets (CSS), a style sheet language used for describing the presentation of a document or webpage. CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. The Perspective module uses the popular standard HTML5/CSS3 technology for its user interface (UI) layer.

Using CSS, the Perspective Style options for components are both detailed and flexible. Styles are used in multiple places, and components can accept style information from multiple sources. There are many CSS settings available through the user interface, they are described in the Style Reference.

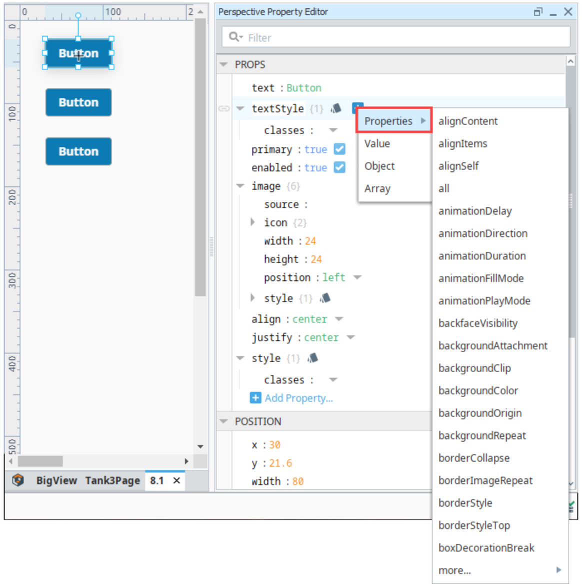
On this page ...

Styles

Watch the Video

The following feature is new in Ignition version **8.1.4**
Click here to check out the other new features

As of 8.1.4, CSS properties are also available through a properties dropdown menu. This menu provides access to all available properties in the schema.

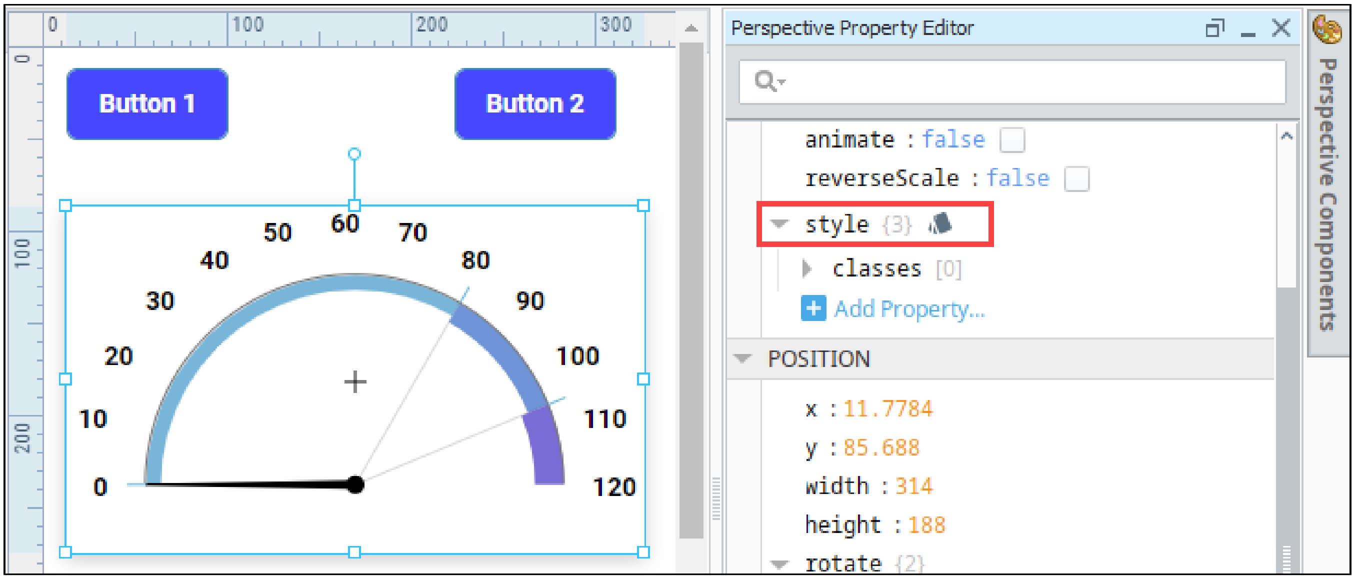


```
text : Button
textStyle {1}
    classes :
primary : true
enabled : true
image {6}
    source :
    icon {2}
    width : 24
    height : 24
    position : left
    style {1}
align : center
justify : center
style {1}
    classes :
x : 30
y : 21.6
width : 80
```

A group of Style settings can be saved together and given a name as a Style Class.

Style Editor

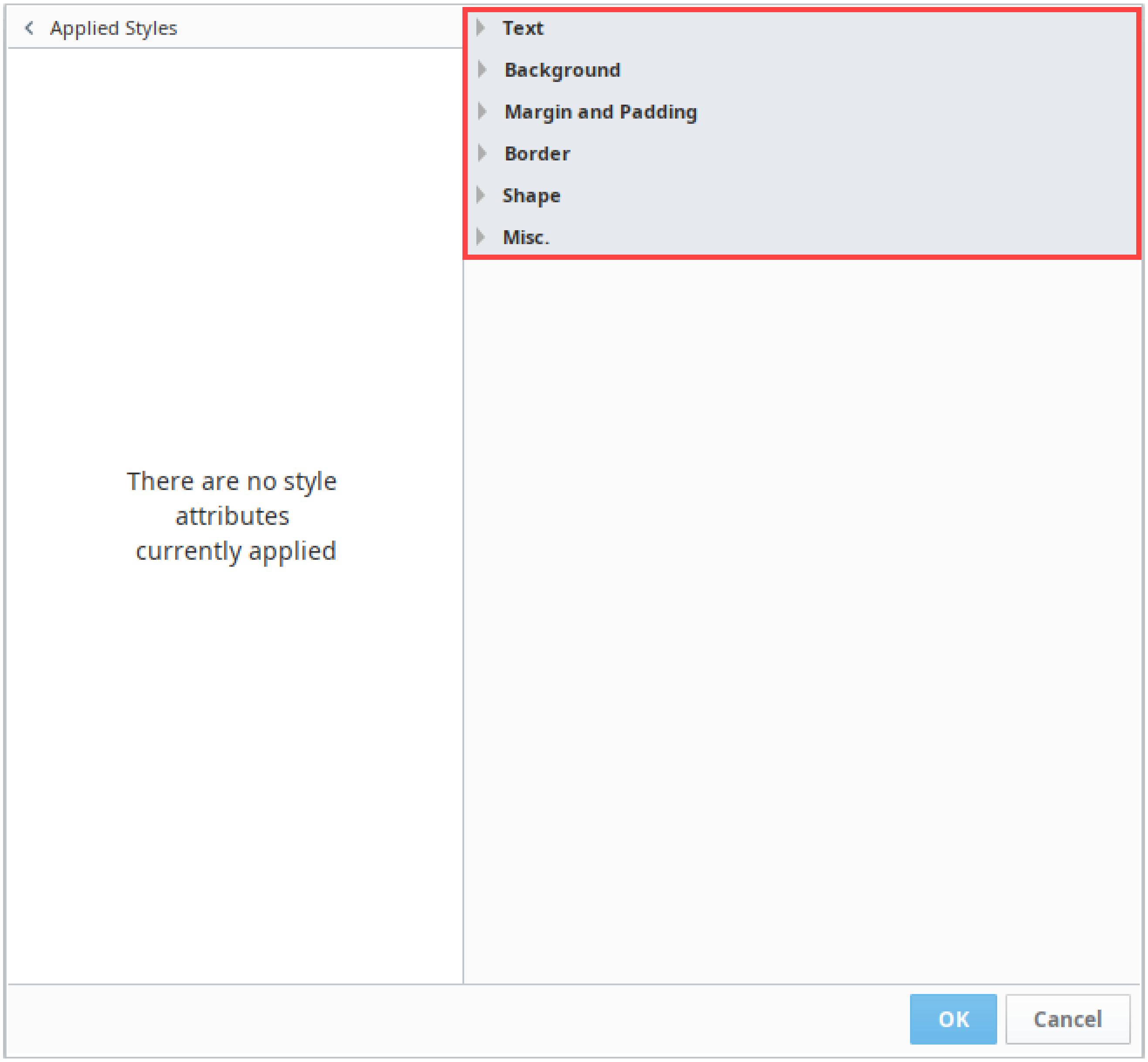
Wherever the style property and the **Styles** icon appear in the Property Editor, you can click on the icon to display the style editor.



animate : false □
reverseScale : false □
style {3}
classes [0]
Add Property...
POSITION
x : 11.7784
y : 85.688
width : 314
height : 188
rotate {2}

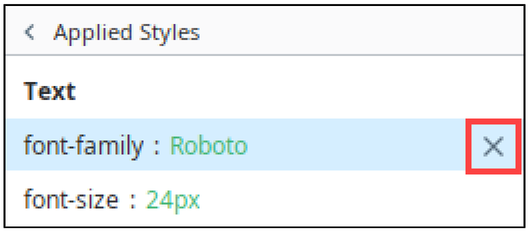
The following menus are available in the style editor:

- Text
- Background
- Margin and Padding
- Border
- Shape
- Misc



You can quickly make adjustments to the styles by clicking on them in the Applied Styles panel.

When you hover over a style listed in the panel, a **Delete** ✕ icon is displayed. Click on it to delete the style.

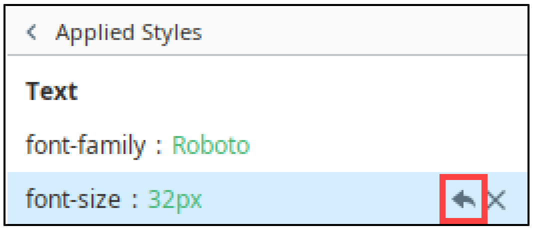

< Applied Styles

Text

font-family : Roboto

font-size : 24px

If you made new edits to an existing style attribute, you can hover over the style and an **Undo** ↶ icon is displayed. Click on it to revert to the previously saved setting.



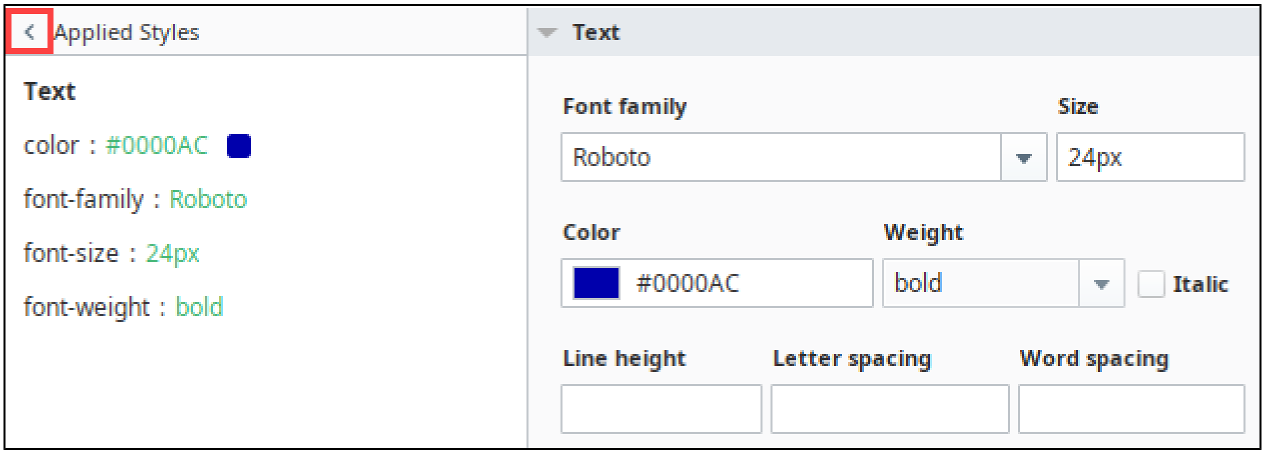
‹ Applied Styles

Text

font-family : Roboto

font-size : 32px

You can minimize the Applied Styles panel by clicking on the **Arrow** ‹ icon to the left of the name. Click the arrow again to reopen the panel.



‹ Applied Styles

Text

color : #0000AC

font-family : Roboto

font-size : 24px

font-weight : bold

Text

Font family: Roboto

Size: 24px

Color: #0000AC

Weight: bold

□ Italic

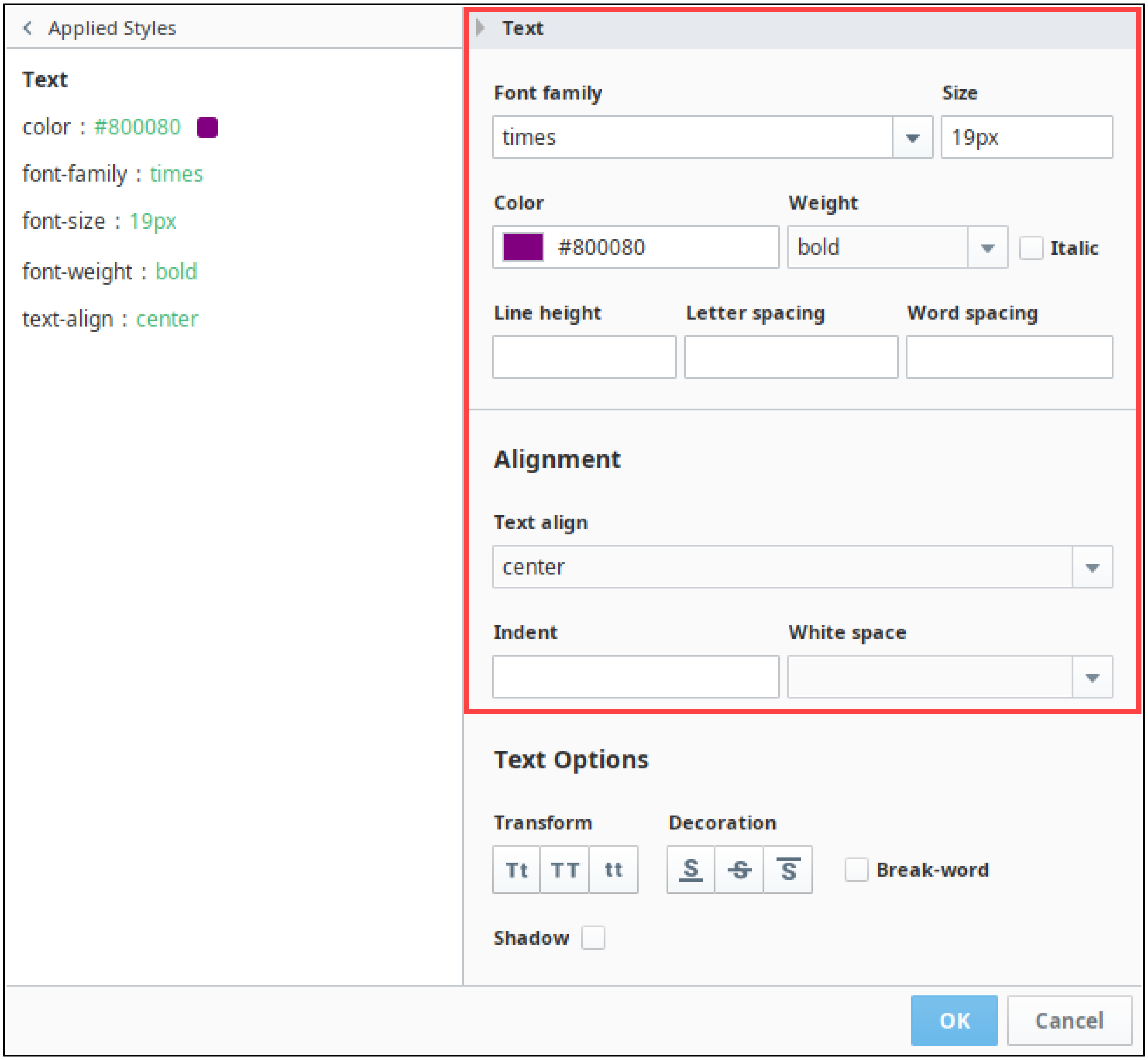
Line height:

Letter spacing:

Word spacing:

Expanding a category enables you to set the properties associated with it. Here is an example of the Style Editor with the Text menu expanded and a few options selected:

- color: #800080
- font-family: times
- font-size: 19px
- font-weight: bold
- text-align: center



Text

color : #800080 🟪

font-family : times

font-size : 19px

font-weight : bold

text-align : center

Text

Font family: times

Size: 19px

Color: 🟪 #800080

Weight: bold

□ Italic

Line height:

Letter spacing:

Word spacing:

Text align: center

Indent:

White space:

□ Break-word

Shadow □

Note:

Applying individual style elements to a component will overwrite the settings for the same style elements being applied from a Style Class.

Some components can have multiple individual Style elements, with each one focusing on a different part of the component. When these styles have a conflict, the more specific style wins out, and sets the style for that particular property.

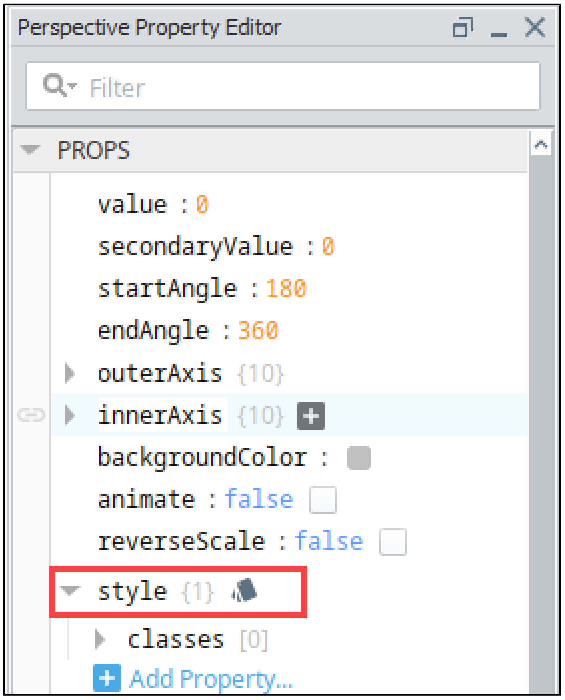
CSS Lengths

Styles, by default, make use of pixels (px) when it comes to length. However all CSS lengths are available and can be used by specifying the unit. For example, a font can be specified in points by appending the unit "pt". For example:

```
35pt
```

Creating a Style

1. In a coordinate view, drag a Gauge component and a Label component.
2. Select the Gauge component and click the **Styles** icon in the properties for the component.

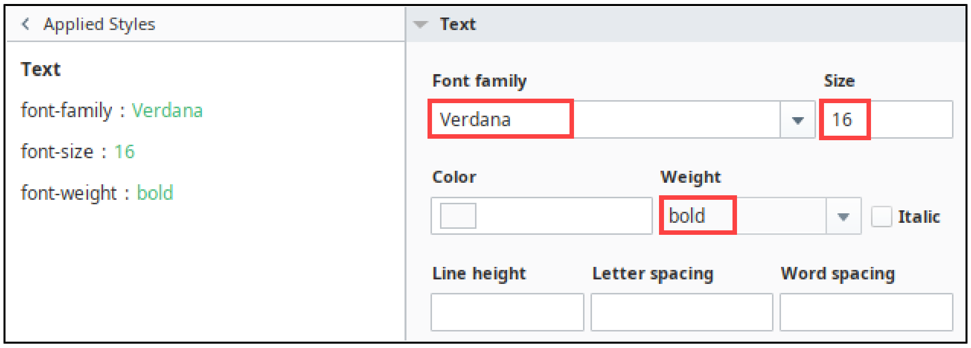


value : 0
secondaryValue : 0
startAngle : 180
endAngle : 360
outerAxis {10}
innerAxis {10}
backgroundColor :
animate : false □
reverseScale : false □
style {1}
classes [0]

The style editor is displayed. Each of the pull down menus has options. These options are described in the Style Reference.

3. Click on the **Text** menu then set the following style options:
   - **Font** Family: Verdana
   - **Size**: 16
   - **Weight**: bold

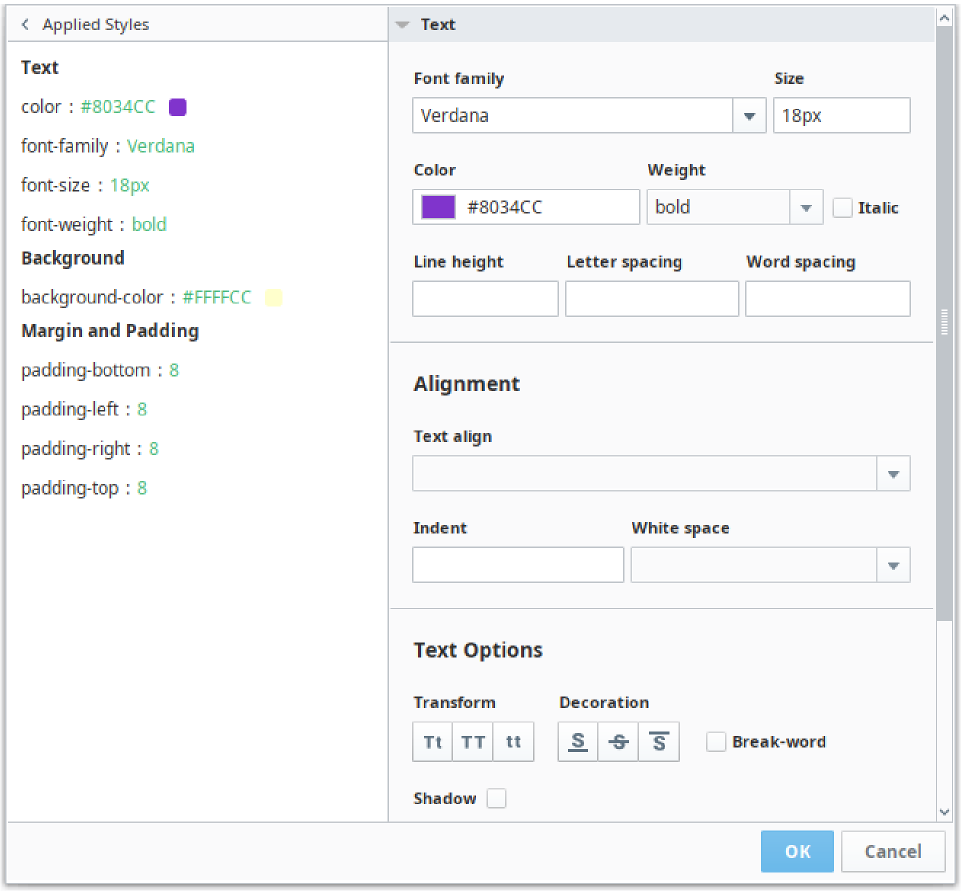
   As you select Style elements, they appear in Applied Styles column on the left.



Text
font-family : Verdana
font-size : 16
font-weight : bold

Font family: Verdana
Size: 16
Color:
Weight: bold
□ Italic
Line height:
Letter spacing:
Word spacing:

4. Click **OK**.
5. Next select the Label component and click the **Styles** icon.
6. Expand the **Text** menu and set the following style options:
   - Font family: **Verdana**
   - Size: **18px**
   - Weight: **bold**
   - Color: **#8034CC**



Applied Styles

Text

color : #8034CC 🟪

font-family : Verdana

font-size : 18px

font-weight : bold

Background

background-color : #FFFFCC

Margin and Padding

padding-bottom : 8

padding-left : 8

padding-right : 8

padding-top : 8

Text

Font family: Verdana

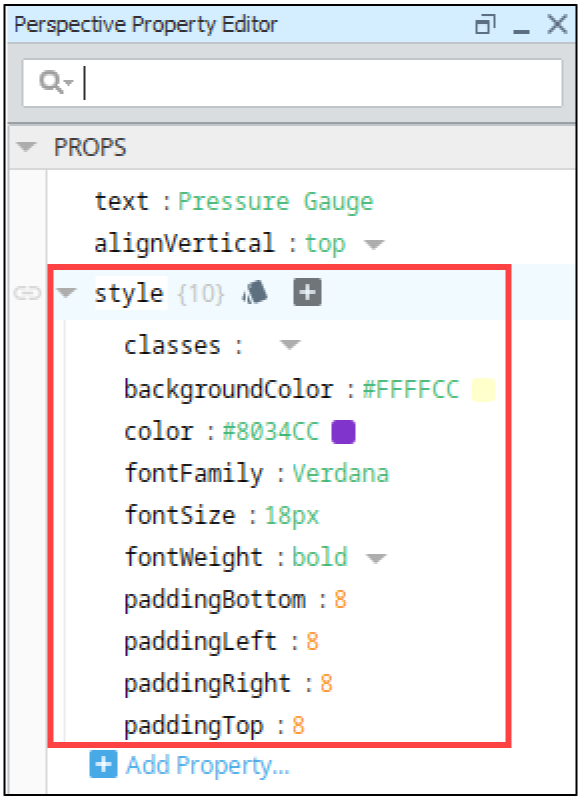
Size: 18px

Color: #8034CC

Weight: bold

□ Italic

7. Expand the **Background** menu and set the following style option:
   - Background Color: **#FFFFCC**
8. Expand the **Padding** menu and set the padding to 8 for all four sides**.**
9. Click **OK** to save the changes. You will notice those properties now appear in the components Style, letting you know that those particular elements have a style applied to them.



Perspective Property Editor

PROPS

text : Pressure Gauge

alignVertical : top

style {10}

classes :

backgroundColor : #FFFFCC

color : #8034CC

fontFamily : Verdana

fontSize : 18px

fontWeight : bold

paddingBottom : 8

paddingLeft : 8

paddingRight : 8

paddingTop : 8

Perspective Built-In Themes

Themes

Perspective comes with several themes, providing initial styling to all components. The active theme in a session is determined by a session property. Specifically, `session.props.theme` found on the home screen of the Perspective workspace. Changing the value of this property in a Perspective Session will change the active theme for the session.

Initial Theme

Ignition installations come with the following themes:

- light
- dark
- light-warm
- light-cool
- dark-warm
- dark-cool

Theme Colors

The built-in themes make heavy use of CSS variables for colors. For any of the default themes, colors are defined in the `variables.css` file. Defining your own variable is simple. Add a line with the following to the variables file:

```
--variable-name: #FF0000;
```

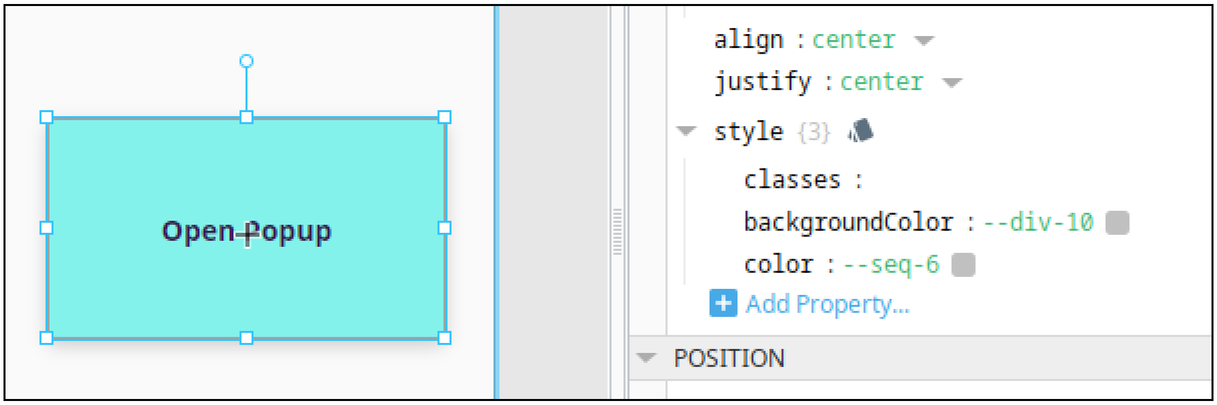
Caution:

Be aware that changes made to the built-in theme files will be replaced on Gateway start up (including restarts caused by a Gateway Restoration) and moved to a backup folder on upgrade. As a result, it is highly recommended that you create a custom CSS file that can then be imported into the entry point CSS files.

For more information, see the markdown README file located in the Gateway's installation directory: `%installDirectory%\data\modules\com.inductiveautomation.perspective\themes\README.md`

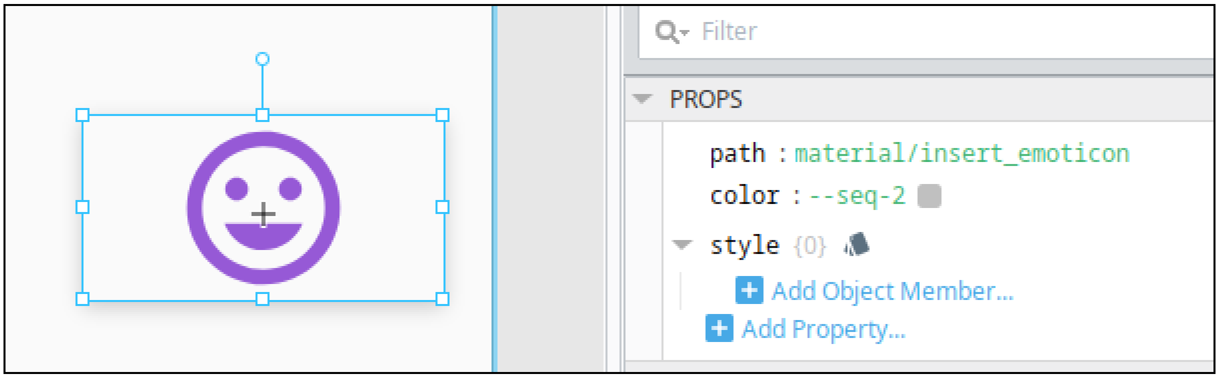
Using Theme Colors

Theme colors can be used on components by simply providing the variable name. For example, we can change the **backgroundColor** and **color** of a button component by just stating the variable name for the appropriate styling properties on the component's **style** object.



```
align : center
justify : center
style {3}
  classes :
  backgroundColor : --div-10
  color : --seq-6
```

If a component has a **color** property outside of a style object, such as the Icon component, the same rules apply; simply set the value of the color property to the name of the variable.



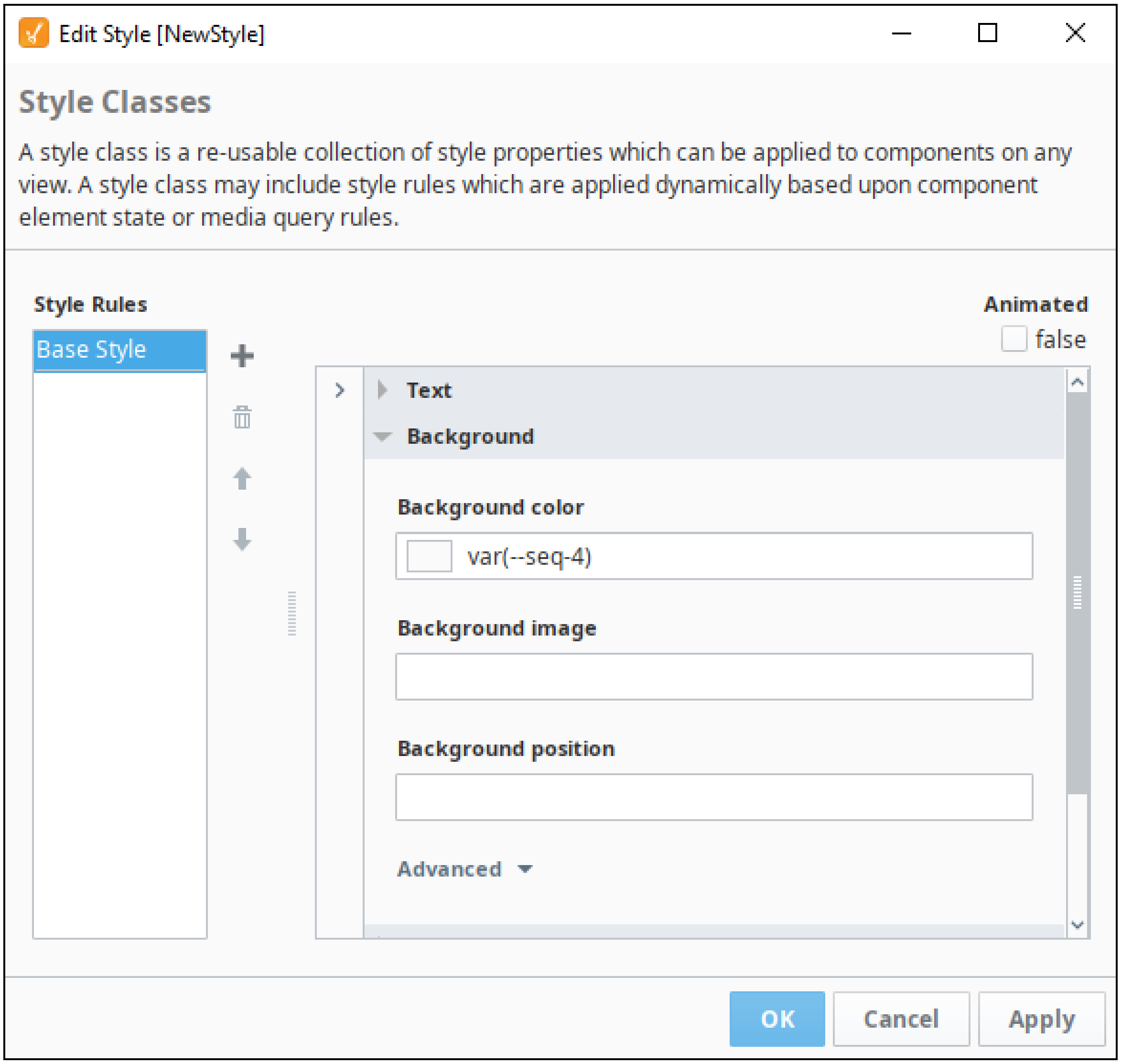
path : material/insert_emoticon

color : --seq-2

style {0}

Style Classes

When using a Theme Color in a Style Class, the variable must be wrapped in the `var()` method, as shown below.



Edit Style [NewStyle]

Style Classes

A style class is a re-usable collection of style properties which can be applied to components on any view. A style class may include style rules which are applied dynamically based upon component element state or media query rules.

Animated
□ false

Background color
var(--seq-4)

Background image

Background position

Built-in Theme Colors

The following color swatch represents the built-in color variables for each IA provided theme.

	light	light-cool	light-warm	dark	dark-cool	dark-warm

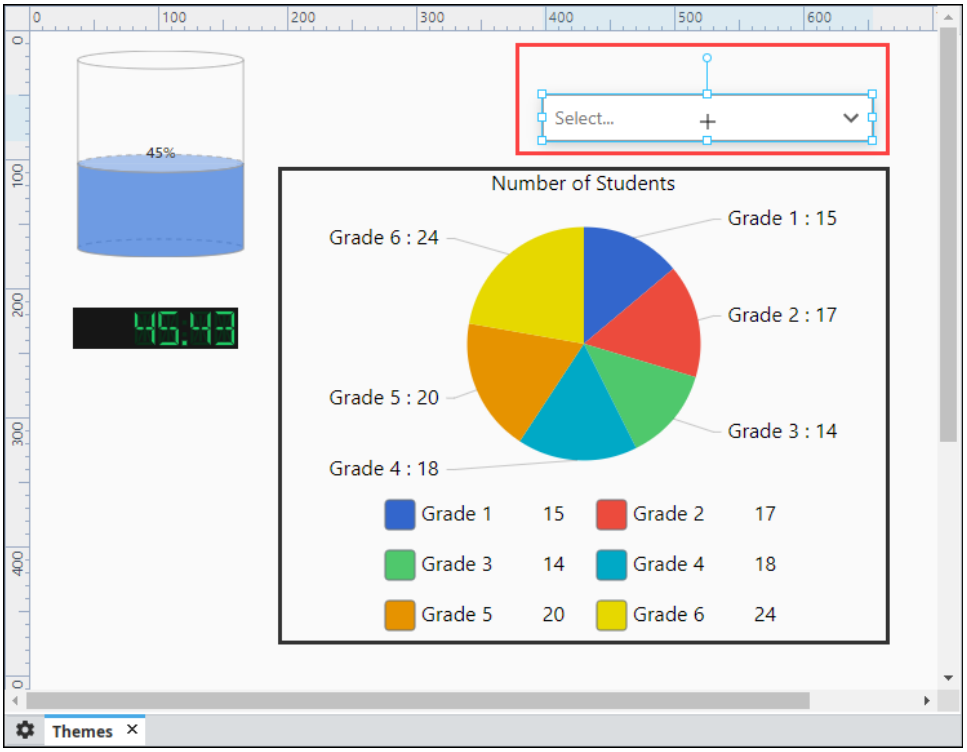
--neutral-10						
--neutral-20						
--neutral-30						
--neutral-40						
--neutral-50						
--neutral-60						
--neutral-70						
--neutral-80						
--neutral-90						
--neutral-100						
--seq-1						
--seq-2						
--seq-3						
--seq-4						
--seq-5						
--seq-6						
--div-1						
--div-2						
--div-3						
--div-4						
--div-5						
--div-6						
--div-7						
--div-8						
--div-9						
--div-10						
--div-11						
--div-12						
--div-13						
--div-14						
--div-15						
--div-16						
--qual-1						
--qual-2						
--qual-3						
--qual-4						
--qual-5						
--qual-6						
--qual-7						
--qual-8						
--qual-9						
--qual-10						
--callToAction						
--callToActionHighlight						
--callToAction--hover						
--callToAction--active						
--callToAction--disabled						

--error						
--info						
--infoSecondary						
--warning						
--warningSecondary						
--success						
--indicator						
--indicatorOff						

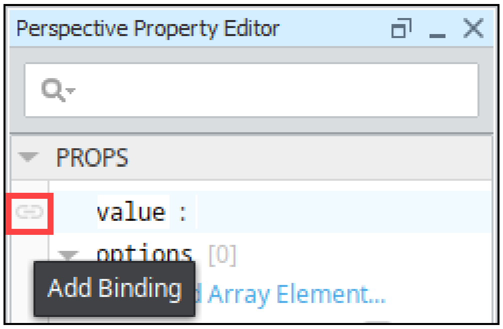
Changing the Theme From a Session

A theme can be set by directly writing to `session.props.theme` property. The property is writable from a session, so you can easily utilize component bindings in such a way that users can change the theme directly from the session. In this example, we will create a dropdown component, and configure a component binding that allows the dropdown to change the theme in the current session.

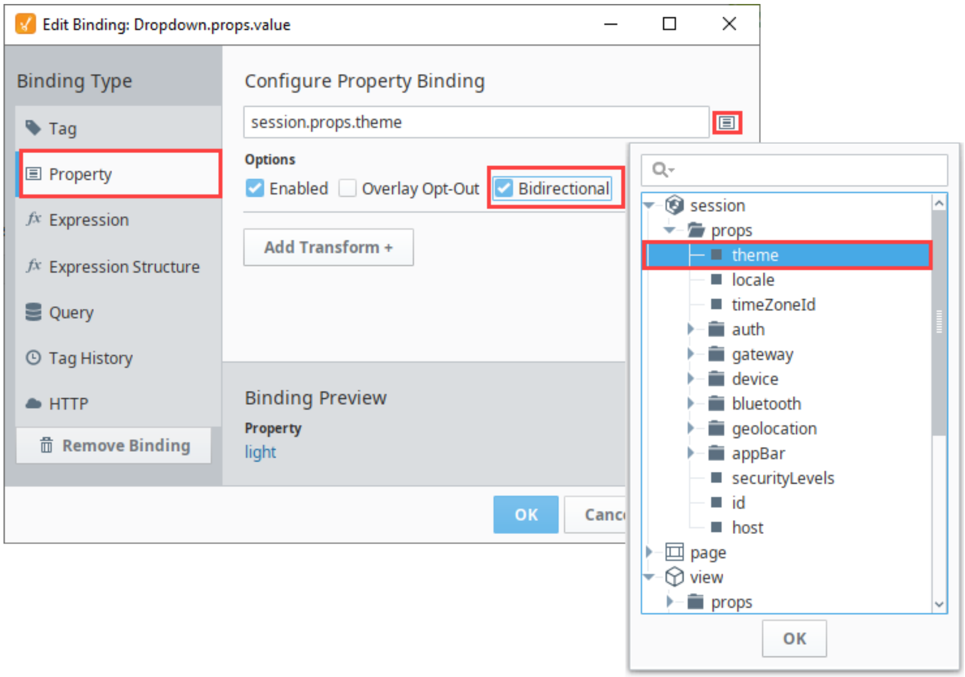
1. Open a view in the Designer that contains a few components and drag in a Dropdown component.



2. With the Dropdown component selected, click on the **Binding** 🔗 icon to create a property binding on the **value** property



3. The Edit Binding window will open. The theme property is a session property. Click on the **Property** ☰ icon and expand **session > props**, and select **theme**. Click **OK**.
4. Check **Bidirectional** and click **OK** to save your property binding.


Edit Binding: Dropdown.props.value

Binding Type: Property

Configure Property Binding

session.props.theme

Options: ☑ Enabled ☐ Overlay Opt-Out ☑ Bidirectional

Binding Preview

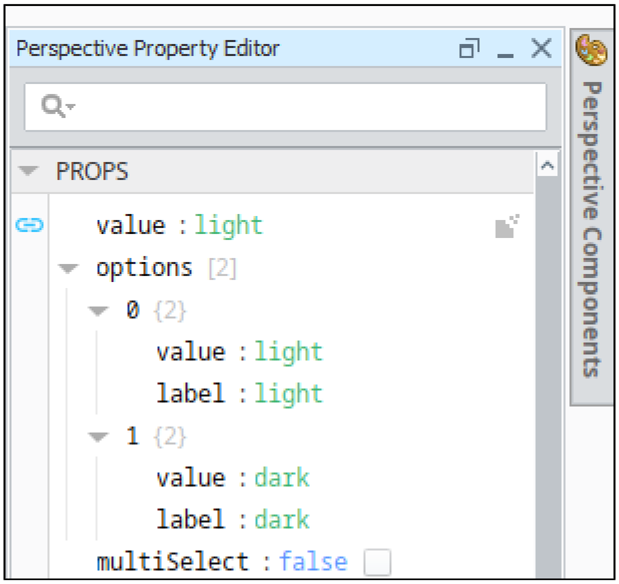
Property

light

5. With the Dropdown component still selected, locate the props.options property in the Property Editor.

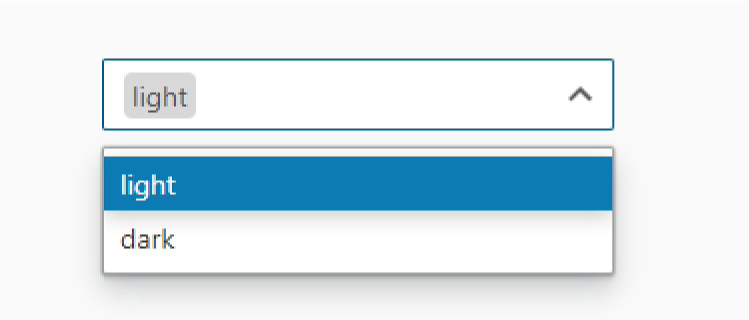
   Paste the following onto the "options" property of a dropdown component.

```
[
  {
    "value": "light",
    "label": "light"
  },
  {
    "value": "dark",
    "label": "dark"
  }
]
```


Perspective Property Editor

```
PROPS
  value : light
  options [2]
    0 {2}
      value : light
      label : light
    1 {2}
      value : dark
      label : dark
  multiSelect : false ☐
```

6. **Save** your project and open your view in a Session or enable Preview Mode in the Designer.
7. From the Dropdown component, select a desired theme (i.e., light).

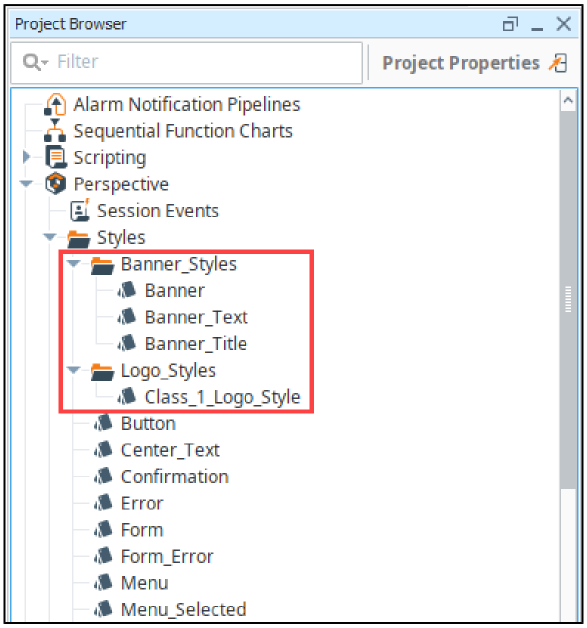


The selected theme will now be applied for the entire Perspective Session.

Style Classes

A Style Class is a group of Style settings that are saved together. Style Classes can be applied to multiple components to provide consistency in design. A Style Class enables you to define style elements in one place, and then quickly apply that style to different components. Style Classes are stored in the Styles folder in the Project Browser.

Within the Styles folder, Style Classes can be placed into folders in order to keep them organized. In the following example, we created separate folders for banner styles and logo styles in the Designer. As projects grow and increase in complexity, use folders to more easily manage your Style Classes.



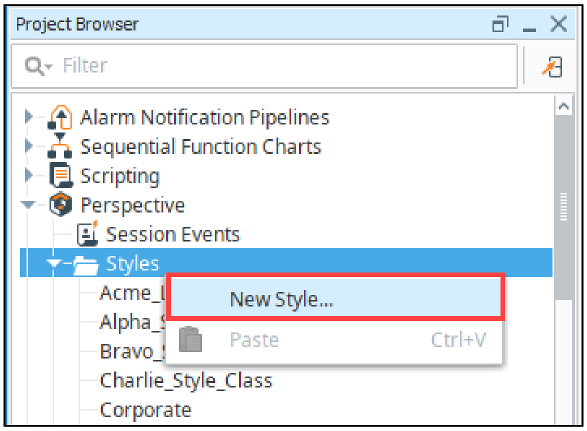
This feature was changed in Ignition version **8.1.22**:

The prefix `ia_` is reserved for the base, predefined styling that is built into Perspective, and should not be used when naming user-created Style Classes. Using the `ia_` prefix for user-created Style Classes may result in unintended behavior.

Creating a Style Class

In this example we'll create a 35 point Italic font, and center the text.

1. To create a Style Class, right-click on the Styles folder and select **New Style**.



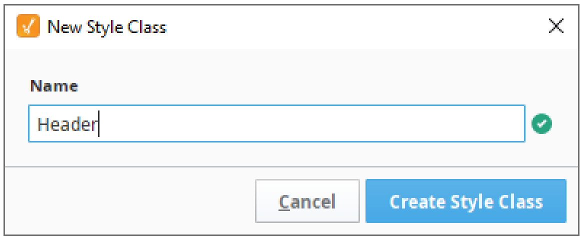
On this page ...

INDUCTIVE UNIVERSITY

Style Classes

Watch the Video

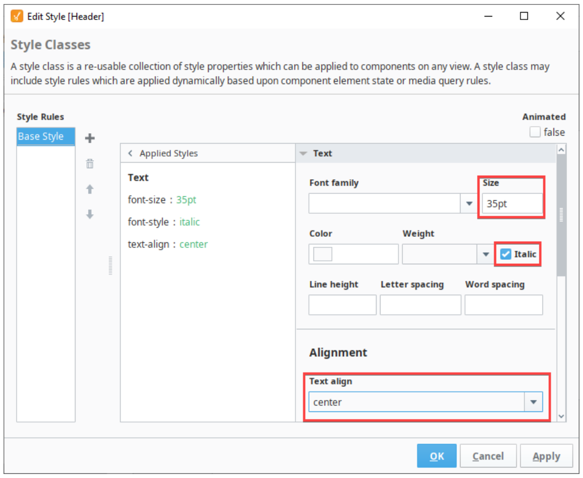
2. Enter a name for the new Style Class. Style Class names must be unique to the project. You will see a green check icon if the name you enter is acceptable. In this example we'll name ours "Header".



New Style Class

Name: Header ✓

3. Click **Create Style Class**.
4. On the Edit Style screen, use the style editor to define the style. In this case we'll configure values for the following properties under **Text**:
Size: 35pt
Italic: enabled
Text align: center



Edit Style [Header]

Style Classes

A style class is a re-usable collection of style properties which can be applied to components on any view. A style class may include style rules which are applied dynamically based upon component element state or media query rules.

Style Rules: Base Style

Animated: □ false

Applied Styles

Text
font-size : 35pt
font-style : italic
text-align : center

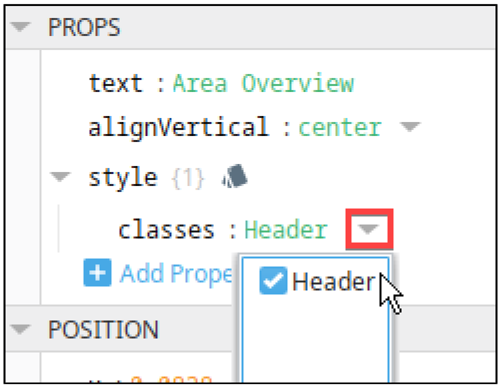
Text

Font family:
Size: 35pt
Color:
Weight:
☑ Italic
Line height:
Letter spacing:
Word spacing:

Alignment

Text align: center

5. Click **OK** to save the Style Class.
6. Once a Style Class has been created, it can then be applied to a component. In the component's Style property, there will be a classes property with a dropdown list of all available Style Classes that can be applied to the component.

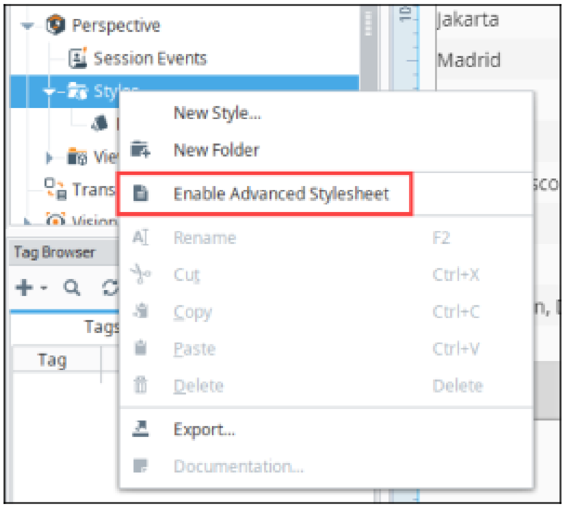


PROPS
text : Area Overview
alignVertical : center
style {1}
classes : Header
☑ Header

Enabling the Advanced Stylesheet

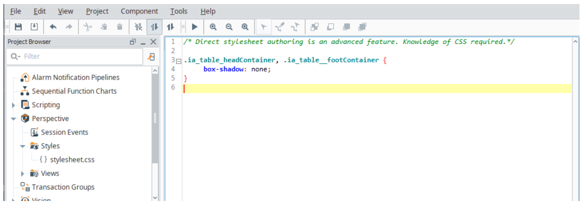
This feature was changed in Ignition version **8.1.22**:

Users with extensive background working with CSS are now able to fine tune styling by modifying the Advanced Stylesheet resource. The stylesheet is an advanced feature and is not placed by default. To enable the Advanced Stylesheet, right-click on the **Styles** folder and select **Enable Advanced Stylesheet**.



The ***stylesheet.css*** resource will now be available under Styles. The stylesheet allows users to insert CSS as in any normal CSS file. Note the editor does not support auto-complete CSS syntax.

Common examples of how this resource can be used include overriding or extending Perspective Style Classes or CSS classes used within a component's structure. CSS classes are not exposed in a component's Property Editor, where as Style Classes are exposed. Perspective uses the prefix `.psc-` for Style Classes before they are injected onto the page, so if you are trying to target a Style Class, you'll need to add this prefix to the name of your style declaration. Then add the Style Class name (without the prefix) to your Property Editor style field. Use the prefix .ia to edit CSS classes as shown. No further steps are required to activate edits to CSS classes.



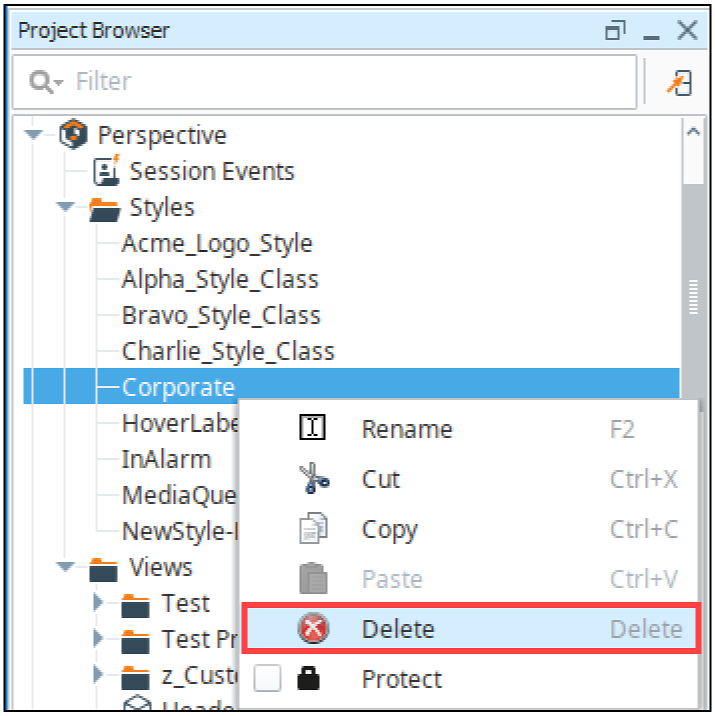
```
/* Direct stylesheet authoring is an advanced feature. Knowledge of CSS required.*/

.ia_table_headContainer, .ia_table__footContainer {
    box-shadow: none;
}
```

The rules defined in this resource are inserted into the Perspective project between the Theme files of the Gateway and the Named Styles of the project. This means the resource is scoped to a project, unlike Theming files being Gateway-scoped.

To disable this advanced feature, follow the **Delete a Style Class** process below.

Delete a Style Class

To delete a Style Class, right-click on the Style Class name in the Project Browser, then select **Delete**. When a Style Class is deleted, it is no longer applied to any components it was previously applied to. The component returns to the default style settings with the exception of any inline styles. Inline styles will remain applied to the component.



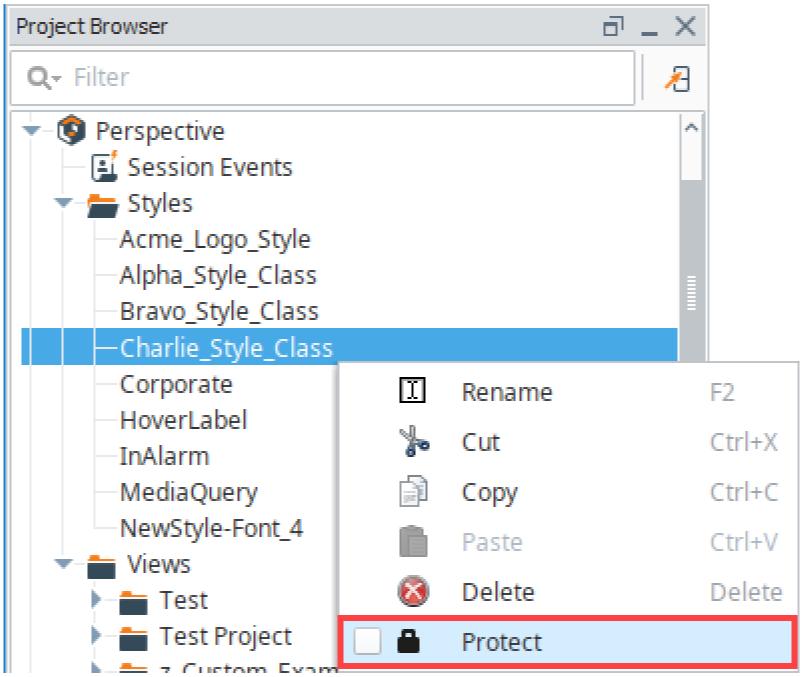
Rename a Style Class

To rename a Style Class, right-click on the Style Class name in the Project Browser, then select **Rename**.

Note: Once a Style Class is renamed, any component that was using the Style Class will lose reference to the newly named style class, meaning you'll need to reapply the newly named style class.

Protect a Style Class

You can lock a Style Class from inside Designer by opening the Project Browser, right-clicking on the Style Class, and selecting the **Protect** option to protect it. Once it's protected, it cannot be changed except by someone that has the permission to unprotect it and modify it. For more information on protecting project resources see Project Security in the Designer.

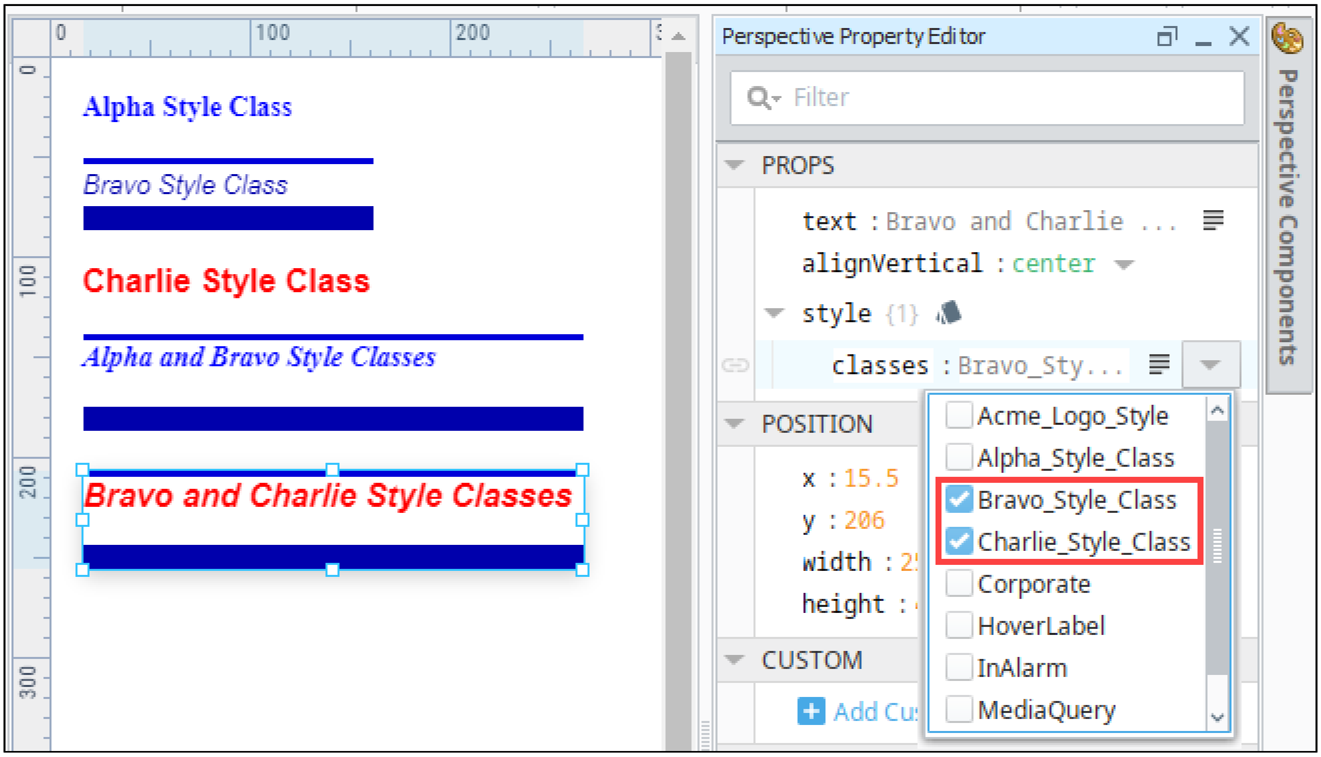


Multiple Style Classes

Perspective allows you to select more than one Style Class for a component. Multiple Style Classes to a component in alphabetical order. Style Classes further along in the alphabet will override earlier Style Classes.

In the example below, Bravo Style Class is blue, bold, italic, and 13px text with some borders. Charlie Style Class is red, bold, and 16px text.

When both styles are applied together, the color and text size in Charlie Style Class override Bravo. However, the italics and borders from Bravo remain because Charlie does not have those properties set.

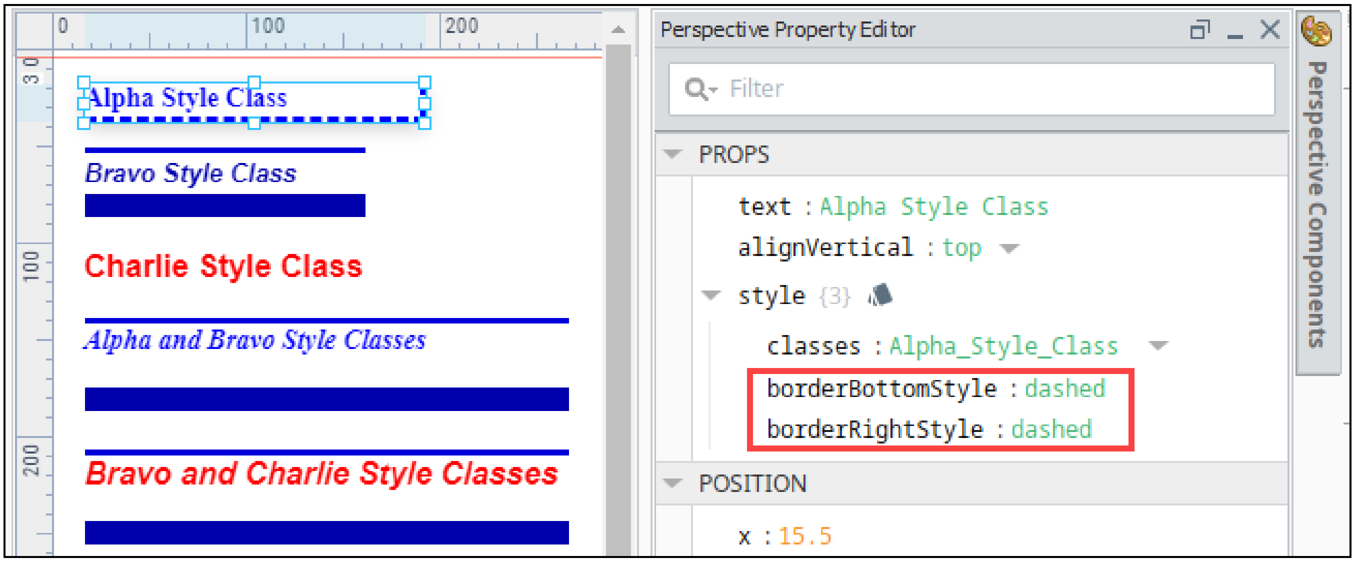


text : Bravo and Charlie ...
alignVertical : center
style {1}
classes : Bravo_Sty...

x : 15.5
y : 206

☐ Acme_Logo_Style
☐ Alpha_Style_Class
☑ Bravo_Style_Class
☑ Charlie_Style_Class
☐ Corporate
☐ HoverLabel
☐ InAlarm
☐ MediaQuery

You can also apply inline styles to components that have a Style Class. The inline style properties override any properties in the Style Class. For example, if the Style Class has a properties of 22pt bold text with a color of blue, but there is an inline style property of 18pt bold text with a color of orange, the component will receive the inline style properties. In this example, we applied the Alpha style to a label and the applied inline styles of a bottom and right border.



text : Alpha Style Class
alignVertical : top
style {3}
classes : Alpha_Style_Class
borderBottomStyle : dashed
borderRightStyle : dashed

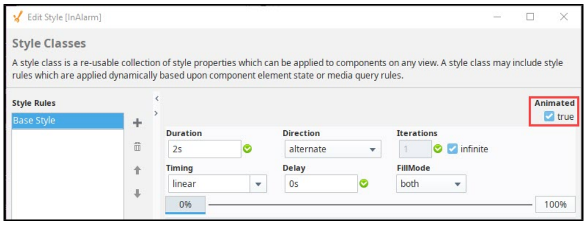
x : 15.5

Animated Style Classes

An animated style class transitions through two or more style configurations over some period of time. For example, using an animated Style Class can be powerful way to visually show data changes (such as an alarm state) on a component over time. When the Animated option is set to true on the Edit Style screen, several settings appear for customizing the animation.

Animated Style Classes

Watch the Video



Edit Style [InAlarm]

Style Classes

A style class is a re-usable collection of style properties which can be applied to components on any view. A style class may include style rules which are applied dynamically based upon component element state or media query rules.

Style Rules

Base Style

Animated ☑ true

Duration: 2s

Direction: alternate

Iterations: 1 ☑ infinite

Timing: linear

Delay: 0s

FillMode: both

0%

100%

Animated Settings

The following properties appear when the Animated checkbox is set to True.

Name	Description
Duration	Number of seconds for each animation stop.
Direction	Options are normal, reverse, alternate, and alternate reverse.
Iterations	Number of times you want the animation to run. Enter an integer, or check the infinite box to have the animation run continually.
Timing	Options are linear, ease, easeIn, easeOut, stepStart, stepEnd, and exaggerate.
Delay	Delay, in number of seconds.
FillMode	Options are none, forwards, backwards, or both.
0% to 100%	Animation stops. All of the animation and style settings can be set for each stop. Additional stops can be added by right-clicking on the bar in between 0% and 100%.
Style Settings	Sets the styles for an animation stop. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous.
Applied Styles	Displays the style names and settings as you add them to a component.

Bindings on Style Classes

In Perspective, you can dynamically change the Style Class on a component. Styles and Style Classes can have binding options. For example, a component could use a Style Class when a Tag has one value, and not use the class when the Tag has a different value.

Dynamic Style Classes

Watch the Video

Element States on Style Classes

Element States are used on Style Classes to change the style configuration on a component based on the state of the component. It is an additional styling configuration you can make on a Style Class. For example, you could have the border on a component change when it is disabled. Or you could have the background change when the user hovers the mouse over the component. See the table below for element states that are available. These states are based in CSS pseudo classes.

Element States

Watch the Video

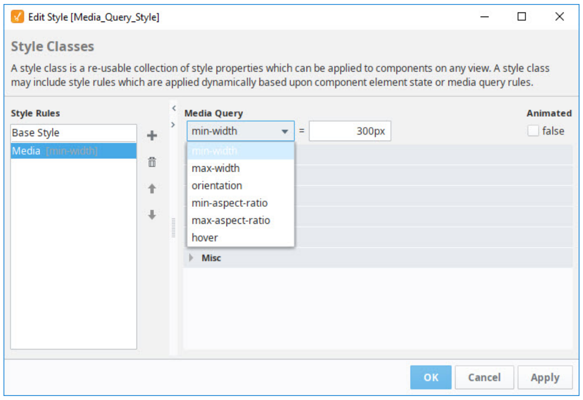
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>Element State</td><td>State of the component. Options are as follows:
<table>
<tr><th>State</th><th>Description</th></tr>
<tr><td>active</td><td>The component is being activated by the user. For example, clicking on it with a mouse.</td></tr>
<tr><td>checked</td><td>The component is checked or toggled to an "on" state. Applies to checkboxes, radio buttons, etc.</td></tr>
<tr><td>disabled</td><td>Component is disabled that is it cannot be selected, clicked on, typed into, or accept focus.</td></tr>
<tr><td>empty</td><td>Represents any element that has no children. Children can be either element nodes or text (including whitespace).</td></tr>
<tr><td>enabled</td><td>Component is enabled and can be selected, clicked on, typed into, or accept focus.</td></tr>
<tr><td>first-child</td><td>Changes the style for the first element among a group of sibling elements.</td></tr>
<tr><td>fullscreen</td><td>Automatically adjust the size, style, or layout of content when elements switch back and forth between full-screen and traditional presentations.</td></tr>
<tr><td>focus</td><td>The component receives focus. It is generally triggered when the user clicks or taps on an element or selects it with the keyboard's "tab" key.</td></tr>
<tr><td>hover</td><td>User hovers mouse over the component.</td></tr>
<tr><td>in-range</td><td>Element whose current value is within the range limits specified by the <code>min</code> and <code>max</code> attributes.</td></tr>
<tr><td>read-only</td><td>Component is read-only.</td></tr>
<tr><td>read-write</td><td>Component is available for read and write.</td></tr>
<tr><td>visited</td><td>Component links that the user has already visited.</td></tr>
</table>
<b><i>Editor notes are only visible to logged in users</i></b>
Per dev, some of these element states will not work yet in Perspective. There's a ticket. Some may end up being dropped/deleted and some may be implemented in the future.
<i>default-choice, invalid, last child, link, only-child, out-of-range, required, valid</i>
</td></tr>
<tr><td>Animated</td><td>If checked, the Element State can be animated. The animate options are as follows:
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>Duration</td><td>Number of seconds for each animation stop.</td></tr>
<tr><td>Direction</td><td>Options are normal, reverse, alternate, and alternate reverse.</td></tr>
<tr><td>Iterations</td><td>Number of times you want the animation to run. Enter an integer, or check the infinite box to have the animation run continually.</td></tr>
<tr><td>Timing</td><td>Options are linear, ease, easeIn, easeOut, stepStart, stepEnd, and exaggerate,</td></tr>
<tr><td>Delay</td><td>Delay, in number of seconds.</td></tr>
<tr><td>FillMode</td><td>Options are none, forwards, backwards, or both.</td></tr>
<tr><td>Style Settings</td><td>Sets a style animation stop. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous.</td></tr>
</table>
</td></tr>
</table>

Media Query on Style Classes

A Media Query can be applied to a Style Class in order to change the style of your Perspective components based on the device your session is running on. That is, you can change the style of your Perspective components based on the device your session is running on.

Media queries don't change anything about your layout, rather they react to those changes in the device and then choose from the various style rules you defined.

For example, you could make changes to your styles based on the width of the session. Media queries in Style Classes are a direct import of CSS rules. Depending on the particular media query, the selector will apply at less than or equal to, or vice versa.



A style class is a re-usable collection of style properties which can be applied to components on any view. A style class may include style rules which are applied dynamically based upon component element state or media query rules.

Media Query

Watch the Video

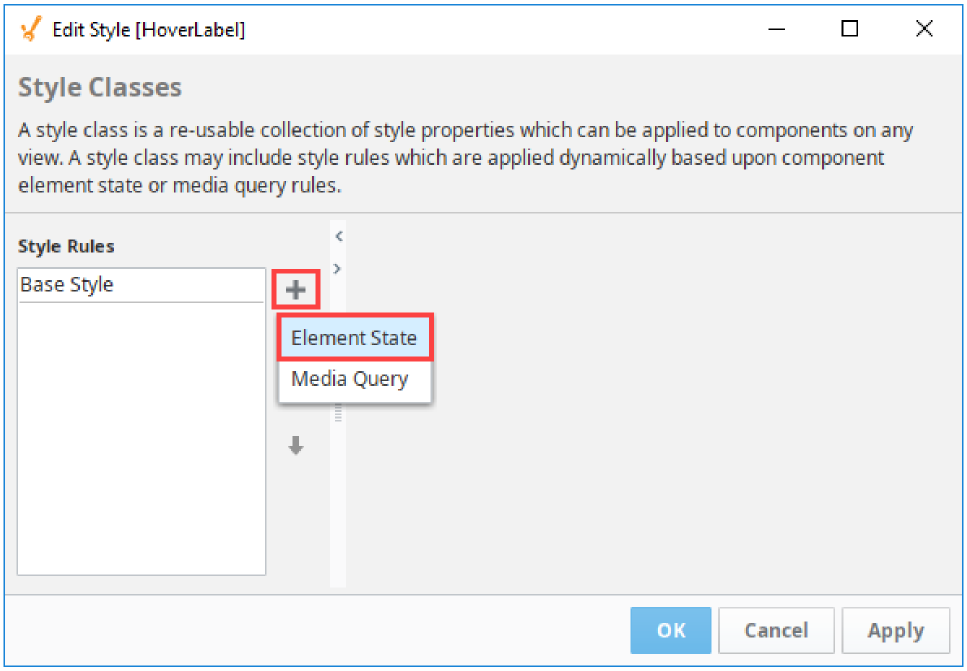
Media Query Settings

Name	Description
min-width	Sets the minimum width in pixels. If the viewport is larger than the specified width (in pixels), this rule will be applied. If the viewport is smaller than the minimum width, this rule has no effect. For example, a setting of min-width 361 means this rule is active if the screen is at least 361 pixels wide.
max-width	Sets the maximum width in pixels . If the viewport is smaller than the maximum width, it will automatically change the height of the element. If the viewport is larger than the maximum width, rule has no effect. For example, a setting of max-width 360 means this rule is active if the screen is at most 360 pixels wide or smaller.
orientation	This rule will apply based on whether the browser window is in landscape mode (that is, its width is greater than its height) or portrait mode (its height is greater than its width). Options are portrait or landscape.
min-aspect-ratio	Sets a minimum width-to-height aspect ratio. Enter value as a ratio or width-to-height, for example 8/5.
max-aspect-ratio	Sets a maximum width-to-height aspect ratio. Enter value as a ratio or width-to-height, for example 8/5. .
hover	Applies the style settings when the device supports hovering. Options are hover or none.
Style Settings	The style settings to apply during the media query. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous.

How to Change Style on Hover

Element states provide an opportunity to change the styling on a component when a user simply hovers the mouse cursor over a component. In this guide, we will create a Style Class with a hover Element State that changes the background color of a component using this Style Class.

1. Create a new style class, or edit an existing one. In this example, we will create a new style class called **HoverLabel**
2. Under Style Rules, click on the **Add** ✚ icon and select **Element State**.

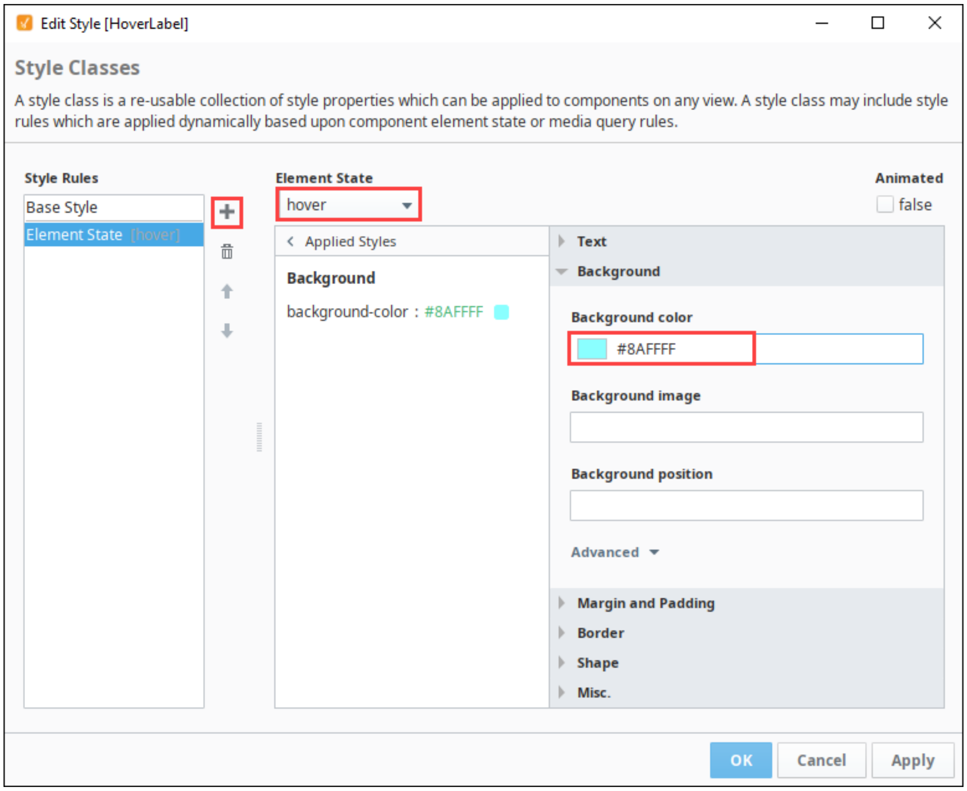


Edit Style [HoverLabel]

Style Classes

A style class is a re-usable collection of style properties which can be applied to components on any view. A style class may include style rules which are applied dynamically based upon component element state or media query rules.

3. On the Element State dropdown, scroll down and select **hover**.
4. Click on the **Expand** ▶ icon next to Background to see the Background settings.
5. In the **background color** field, enter a color code or click on the color wheel to select a background color. We chose **#8AFFFF**, a light blue.



Edit Style [HoverLabel]

Style Classes

A style class is a re-usable collection of style properties which can be applied to components on any view. A style class may include style rules which are applied dynamically based upon component element state or media query rules.

Style Rules

Base Style

Element State [hover]

Element State: hover

Animated: □ false

Applied Styles

Background

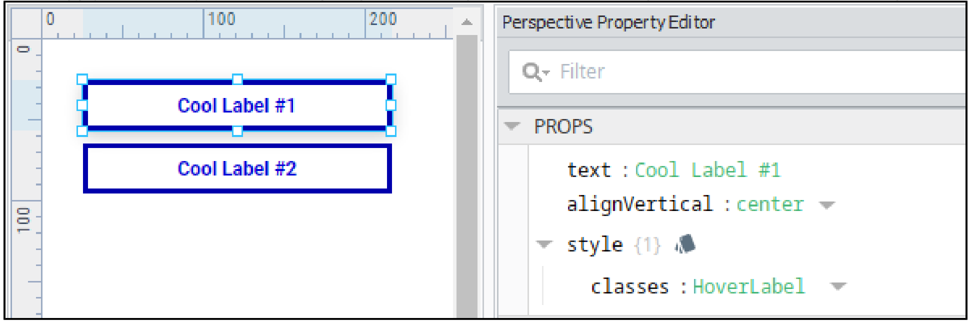
background-color : #8AFFFF

Background color: #8AFFFF

Background image:

Background position:

6. Click **OK** to finish configuring the Style Class.
7. Next, apply the the Style Class to a perspective component. In this example, we will use a Label component.



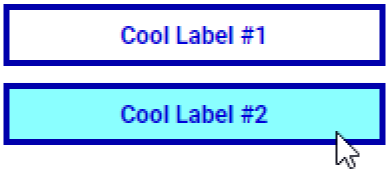
text : Cool Label #1

alignVertical : center

style {1}

classes : HoverLabel

8. Put the designer in preview mode, or launch a session. While hovering over the label, you will see that the background color of the component will change while the mouse cursor is hovering over the component.

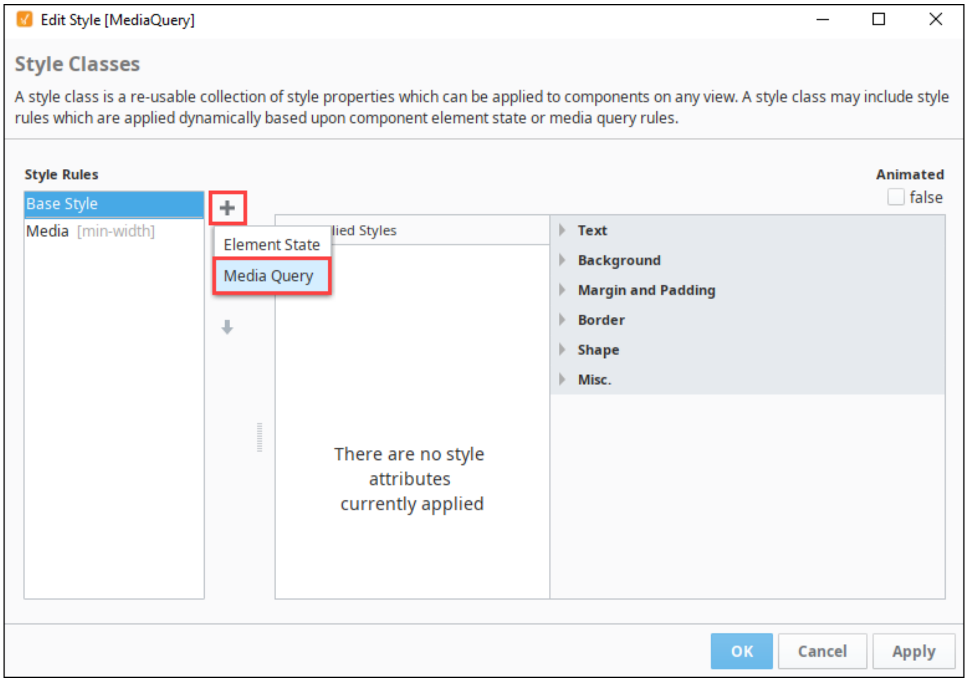


How to Change Text Size Based on Width

Since media queries react to viewport changes, we can use them to change the text size of a style class. Imagine a scenario where text needed to be larger on a smaller display, such as phone, to improve readability. That same text could likely be smaller on a desktop.

In this example, we will configure a media query that will change the text size for the Style Class when the screen is smaller.

1. Create a new Style Class. We called ours **MediaQuery**.
2. Under Style Rules, click on the **Add** ✚ icon and select **Media Query**.



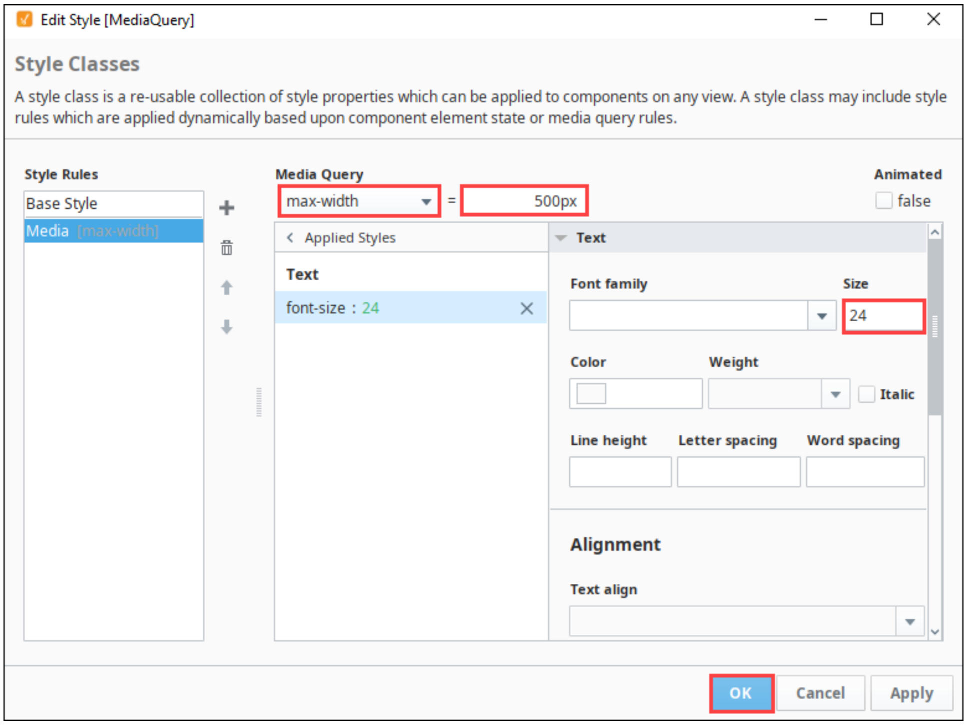
Edit Style [MediaQuery]

Style Classes

A style class is a re-usable collection of style properties which can be applied to components on any view. A style class may include style rules which are applied dynamically based upon component element state or media query rules.

There are no style attributes currently applied

3. With the Media Query selected, change Media Query to **max-width** and the pixel value to **500px**. We also set the Text Font size to **24px**.
4. Click **OK** to save the Style Class.

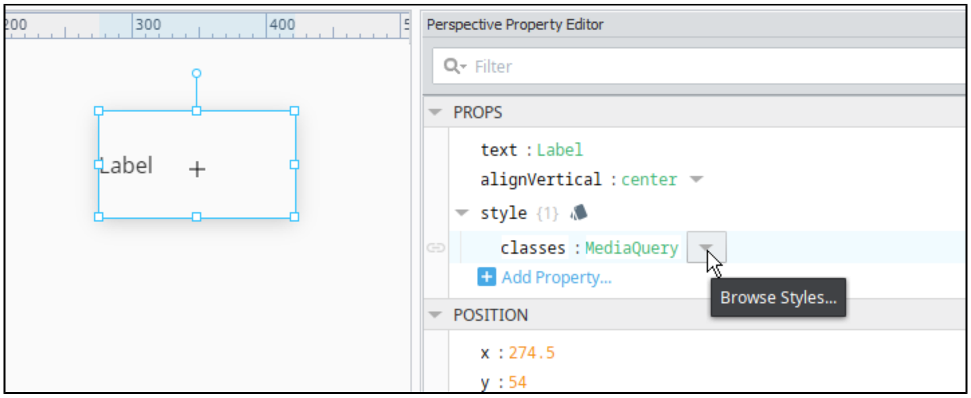


Edit Style [MediaQuery]

Style Classes

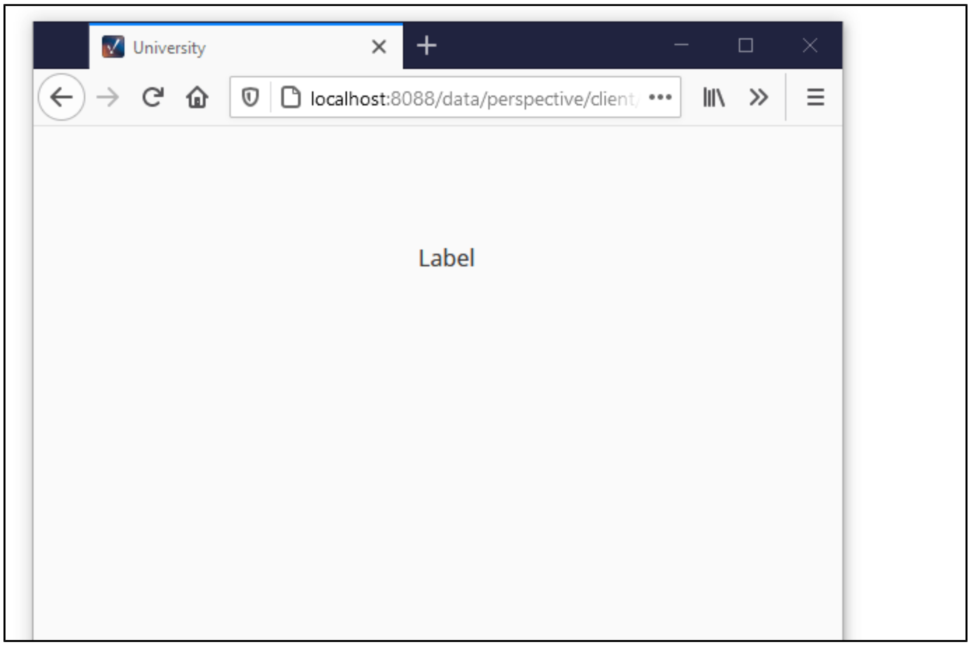
A style class is a re-usable collection of style properties which can be applied to components on any view. A style class may include style rules which are applied dynamically based upon component element state or media query rules.

5. Create a new component that contains text, or select an existing component within a container. In this example, we created a new Label component and placed it on a view.
6. Assign the **MediaQuery** Style Class we just created to the Label component using the props.style.classes dropdown to browse Styles.



7. **Save** the project.

8. Launch a Perspective Session, and navigate to view containing the label (you may have create a page for the view). You will see the font size on the label change as you adjust the size of your session window.



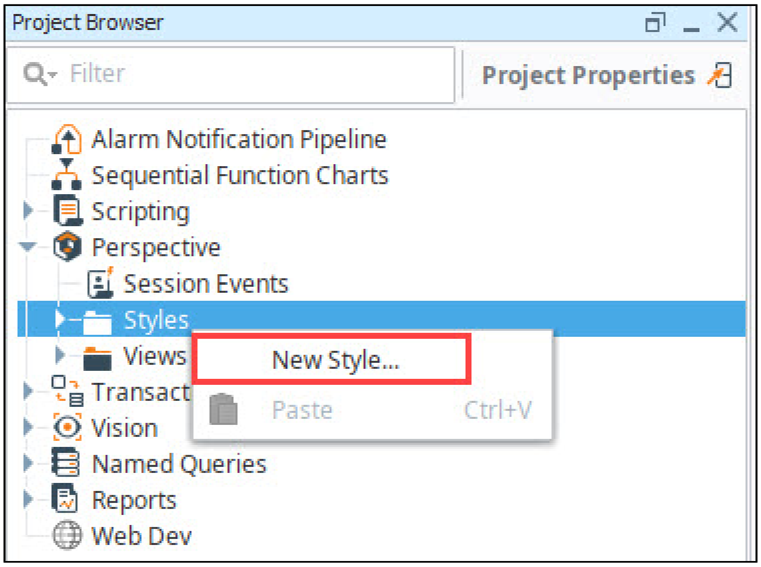
How to Create a Flashing Component

In some cases you may want to have a label or other display component flash, standing out from other components on the screen. This can easily be accomplished by creating an animated style class, as they have a built-in mechanism to change styling over a duration, allowing us to create a flash. However, you may might only want to have the component flash at certain times, such as in cases where an alarm is active on a tag, which can be handled with a binding. In cases where we don't want the component to flash, we can simply remove the style class from a component.

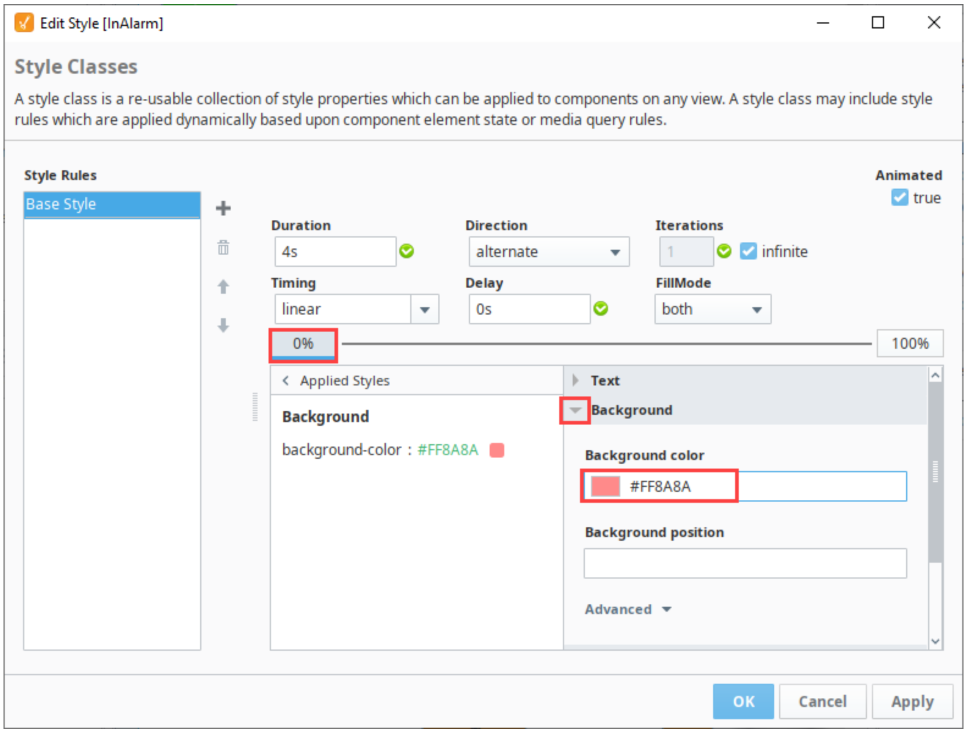
Part 1 of this guide will show how to create the animated style class, while Part 2 will demonstrate using a binding and transform to switch between different style classes on a component.

Part 1 - Creating an Animated Style Class

1. In the Project Browser, right-click Styles and select **New Style**.



2. Enter a Style Class name, for this example we used "InAlarm", and click **Create Style Class**.
3. On the Edit Style screen, check the **Animated** option in the upper right corner. For this example, we'll leave the default settings in the fields.
4. Next click on **0%** to set the style for the beginning of the animation.
5. Click on the **Expand** ▶ icon next to Background to see the Background settings.
6. In the Background color field, enter a color code or click on the color wheel to select a background color for the first animation stop. We chose #FF8A8A, a light red.


Edit Style [InAlarm]

Style Classes

A style class is a re-usable collection of style properties which can be applied to components on any view. A style class may include style rules which are applied dynamically based upon component element state or media query rules.

Style Rules: Base Style

Animated: ☑ true

Duration: 4s

Direction: alternate

Iterations: 1 ☑ infinite

Timing: linear

Delay: 0s

FillMode: both

0% — 100%

Applied Styles

Background

background-color : #FF8A8A

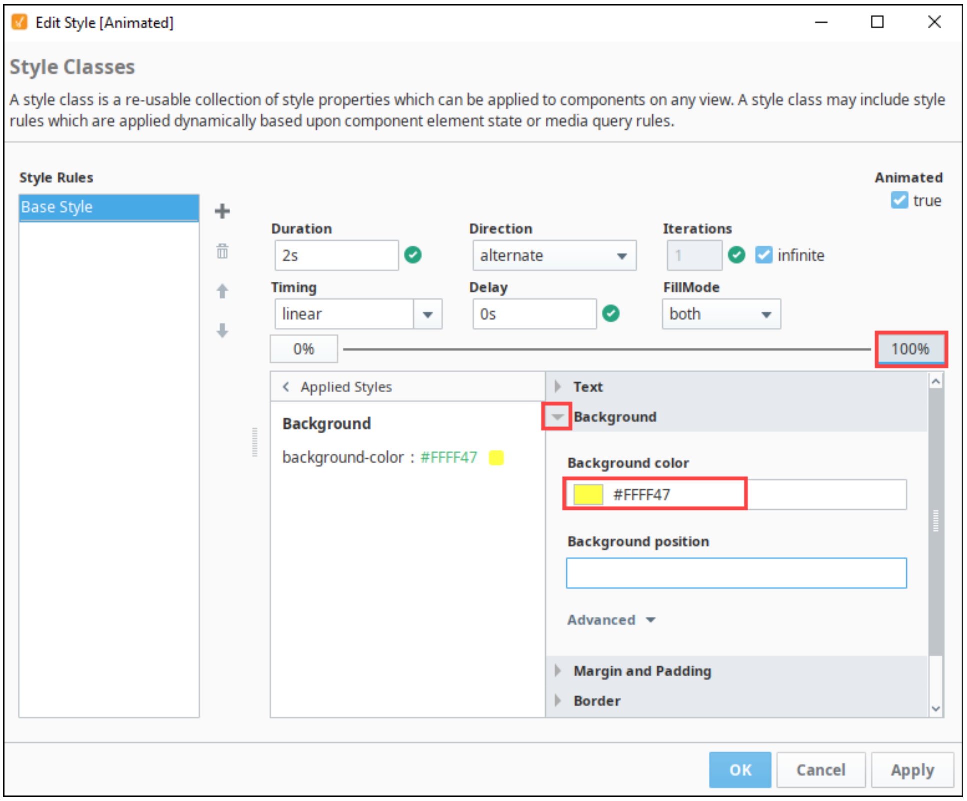
Text

Background

Background color: #FF8A8A

Background position

7. Next click on **100%** to set the style for the end of the animation.
8. Click on the **Expand** ▶ icon next to Background to see the Background settings.
9. In the Background color field, enter a color code or click on the color wheel to select a background color for the first animation stop. We chose **#FFFF47**, a bright yellow.


Edit Style [Animated]

Style Classes

A style class is a re-usable collection of style properties which can be applied to components on any view. A style class may include style rules which are applied dynamically based upon component element state or media query rules.

Style Rules: Base Style

Animated: ☑ true

Duration: 2s

Direction: alternate

Iterations: 1 ☑ infinite

Timing: linear

Delay: 0s

FillMode: both

0% — 100%

Applied Styles

Background

background-color : #FFFF47

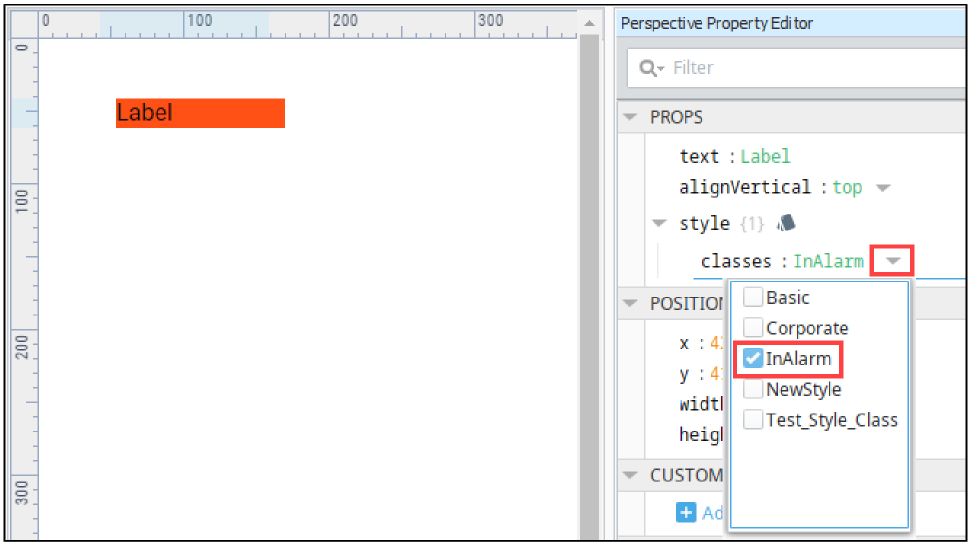
Text

Background

Background color: #FFFF47

Background position

10. Click **OK** to save the Style Class.
11. Next drag a Label component onto a view. Select the component.
12. In the Perspective Property Editor, click the Expand ▶ icon under Style Classes. Select the InAlarm class.

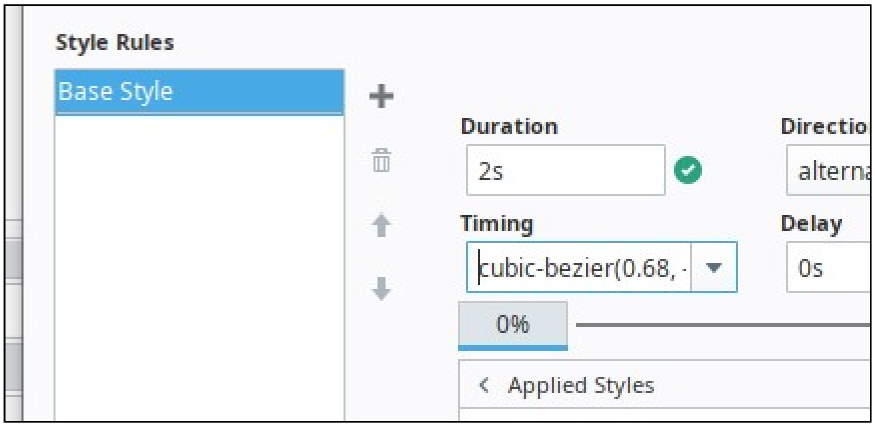


text : Label
alignVertical : top
style {1}
classes : InAlarm

☐ Basic
☐ Corporate
☑ InAlarm
☐ NewStyle
☐ Test_Style_Class

x : 4
y : 4
widt
heig

13. The Label component will immediately display the animated Style Class, transitioning between the light red and yellow colors we selected.
14. The color change will be linear initially, which creates a pulsing sort of effect. You can edit the style class again, and change the **Timing** to "exaggerated", which provides a more drastic switch between the two styles. Further more, selecting "exaggerated" will show the exact CSS cubic-bezier function the animation is using, allowing you to type on values manually.



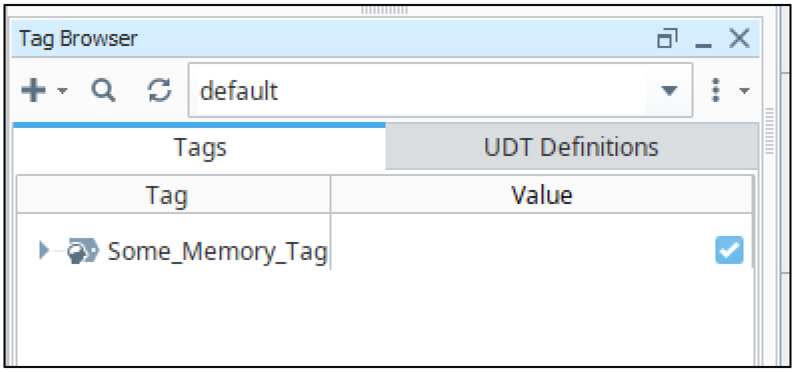
Duration: 2s
Directio: alterna
Timing: cubic-bezier(0.68, -
Delay: 0s

In addition, we can also reduce the overall duration of the animation, making the transition between the two colors occur more frequently. Regardless, we now have an animated style class.

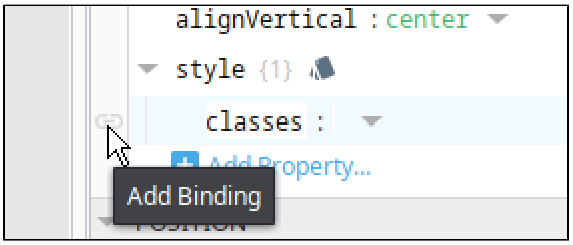
Part 2 - Dynamically Using the Style Class

In this part, we can use a binding to conditionally add or remove our animated style class. First, we need to identify what should cause the label to flash. In this example, we'll use a simple tag value on a Memory Tag set to a Boolean value. However, you could use some other criteria, such as a tag binding to the **HasActive** alarm property on a tag.

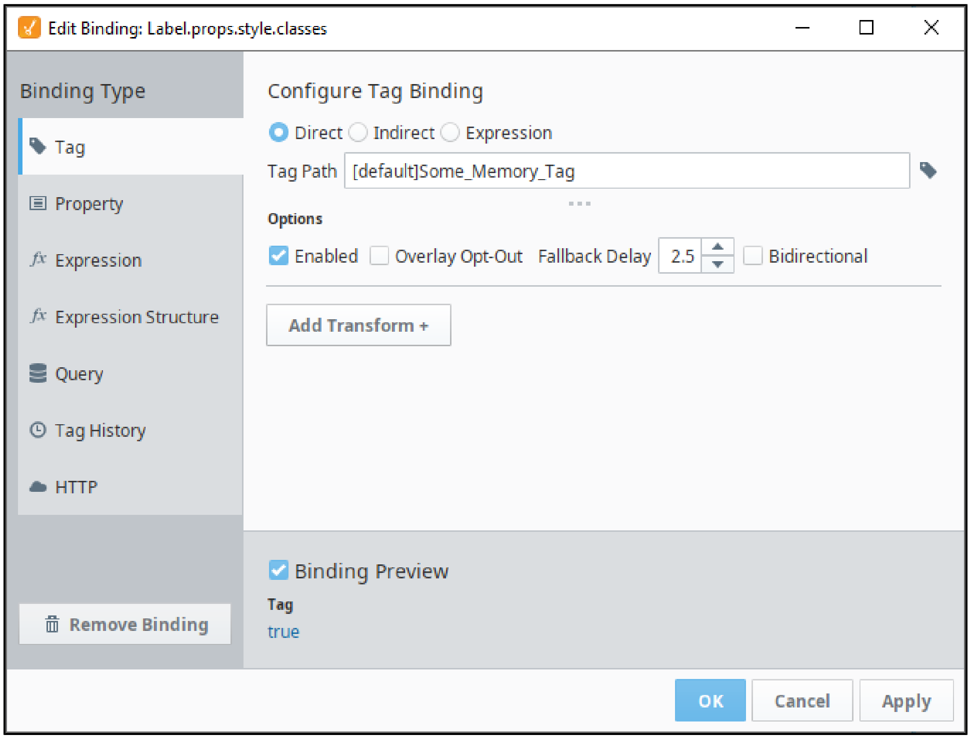
1. Identity the criteria that will cause our component to flash. In our case, we have a Memory Tag with a Boolean data type. When the value is True on the tag, we will cause the label to flash.



2. With our component selected, place a binding on the **style.classes** property.



3. Since we want a tag value to determine the flash, we'll select a **Tag Binding**, and set the path to our desired tag.


Edit Binding: Label.props.style.classes

Configure Tag Binding

● Direct ○ Indirect ○ Expression

Tag Path [default]Some_Memory_Tag

Options

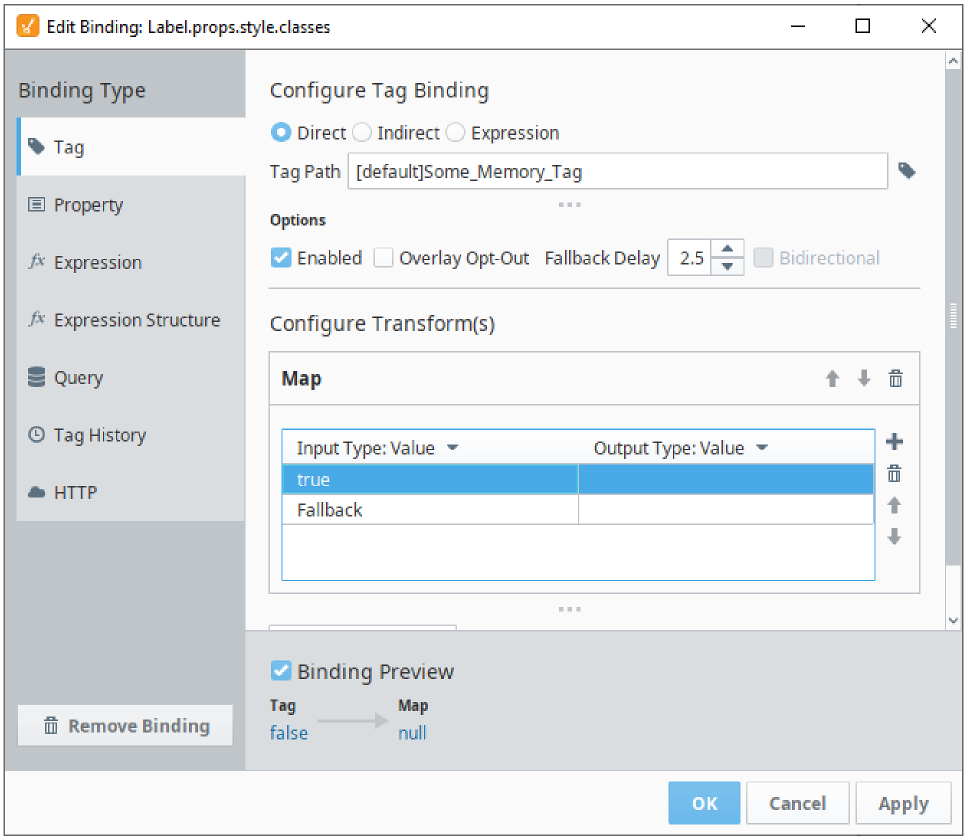
☑ Enabled □ Overlay Opt-Out Fallback Delay 2.5 □ Bidirectional

☑ Binding Preview

Tag

true

4. Next, click **Add Transform**. We'll configure a transform that will return a string to the property, and use this as a way to add or remove our animated style class.
5. Select a **Map** type transform, and click **Add Transform >** in the popup.
6. Add one new row to the Map table by clicking the Add + button.
7. Under the **Input Type** column. Double click into the blank cell and type "true", without the quotation marks, then press Enter on your keyboard to commit the new text.



Edit Binding: Label.props.style.classes

● Direct ○ Indirect ○ Expression

Tag Path [default]Some_Memory_Tag

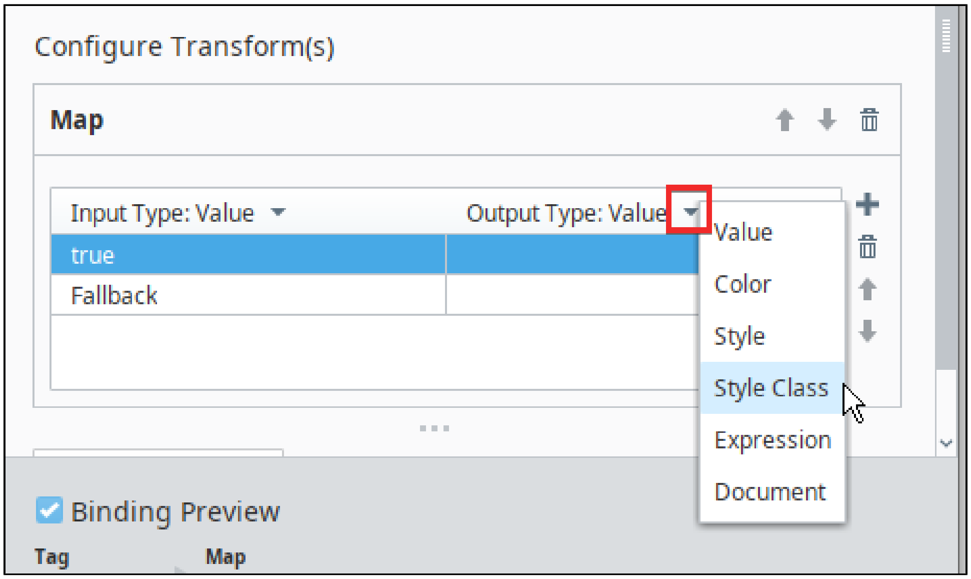
☑ Enabled □ Overlay Opt-Out Fallback Delay 2.5 □ Bidirectional

Input Type: Value	Output Type: Value
true	
Fallback	

☑ Binding Preview

Tag false → Map null

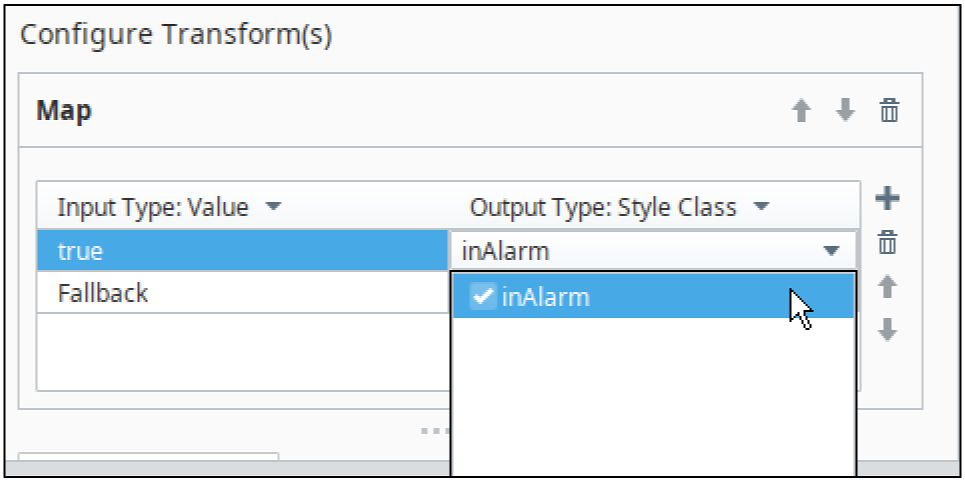
8. Change the Output Type column to **Style Class** by clicking on the header over the Output Type column, and selecting Style Class.



Input Type: Value	Output Type: Value
true	
Fallback	

Note: For this example, we could use a Value type of output since the classes property simply needs a string value that matches the name of our Style Class, but the Style Class output type is less error prone since you don't need to type the name of the Style Class.

9. Click the top cell under the Output Type column, and select the "inAlarm" Style Class from the popup. **Output Type** column, set the cell to the first row to "inAlarm".


Configure Transform(s)

Map

Input Type: Value	Output Type: Style Class
true	inAlarm
Fallback	

☑ inAlarm

10. Press **OK** to close the binding window.
11. Toggle the value on your tag. You'll notice that when the tag's value is "true", the animation will start playing. While the tag has a false value, the style class will be removed, and no animation can be shown.

This example can be extended by adding an additional style class when the value of our binding is false. Furthermore, instead of setting just a single style class, we can use the map transform to set multiple style classes on the component, allowing us to switch between multiple styles on demand.

Creating and Using Custom Perspective Themes

Theme Overview

On this page ...

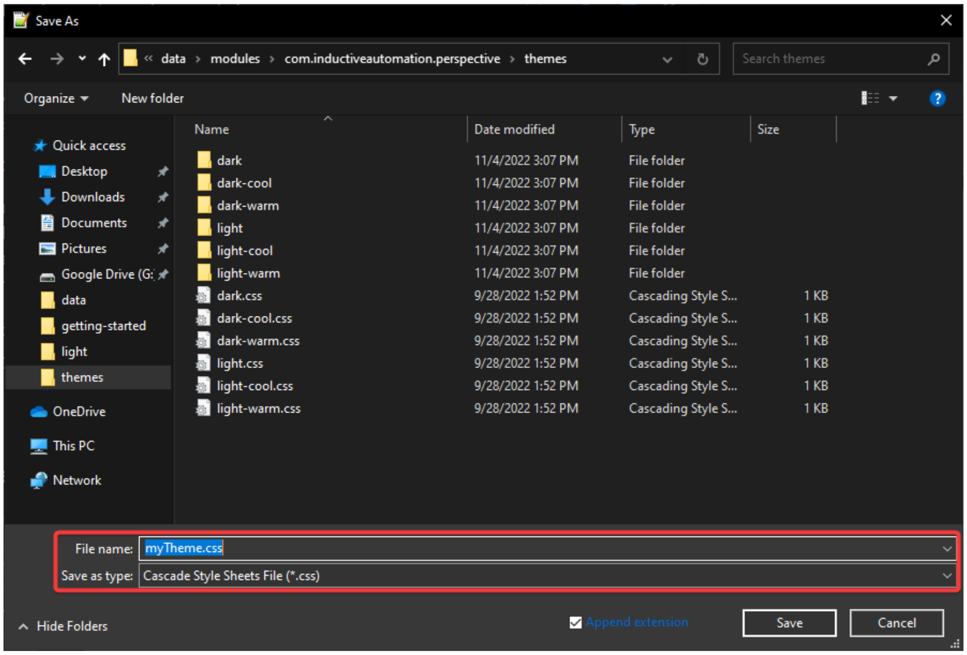
Themes help customize the look of components in your Perspective Sessions, acting as a foundation from which you can customize components even further using Style Classes. Perspective has some default themes already, which you can learn about on the Perspective Built-In Themes page.

By default, themes are stored as **cascading style sheets (CSS)** in `%installDirectory%\data\modules\com.inductiveautomation.perspective\themes`. Default themes use CSS variables for colors, which are defined in the `variables.css` file of every theme's folder.

In some cases, the default themes may not meet your requirements. For these situations, you can create a **custom theme**.

Creating a Custom Theme

1. Open a text editor, such as Notepad or Notepad++, as an **administrator**. Opening the text editor as a standard user may result in your operating system blocking you from saving your file to the themes folder.
2. Save the text file as a .css file in the themes directory, and give it a name. In this example, we are naming the .css file "`myTheme`".

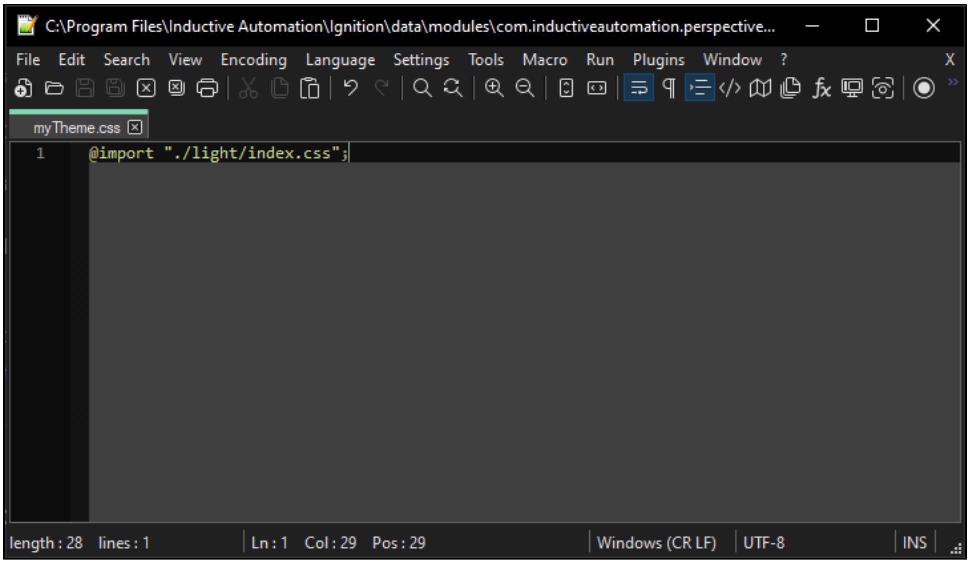


Name	Date modified	Type	Size
dark	11/4/2022 3:07 PM	File folder	
dark-cool	11/4/2022 3:07 PM	File folder	
dark-warm	11/4/2022 3:07 PM	File folder	
light	11/4/2022 3:07 PM	File folder	
light-cool	11/4/2022 3:07 PM	File folder	
light-warm	11/4/2022 3:07 PM	File folder	
dark.css	9/28/2022 1:52 PM	Cascading Style S...	1 KB
dark-cool.css	9/28/2022 1:52 PM	Cascading Style S...	1 KB
dark-warm.css	9/28/2022 1:52 PM	Cascading Style S...	1 KB
light.css	9/28/2022 1:52 PM	Cascading Style S...	1 KB
light-cool.css	9/28/2022 1:52 PM	Cascading Style S...	1 KB
light-warm.css	9/28/2022 1:52 PM	Cascading Style S...	1 KB

File name: myTheme.css
Save as type: Cascade Style Sheets File (*.css)
☑ Append extension

3. After you save the file, add the following line as the first line of the new .css file:

```
@import "./light/index.css";
```



C:\Program Files\Inductive Automation\Ignition\data\modules\com.inductiveautomation.perspective...
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
myTheme.css
length : 28 lines : 1
Ln : 1 Col : 29 Pos : 29
Windows (CR LF)
UTF-8
INS

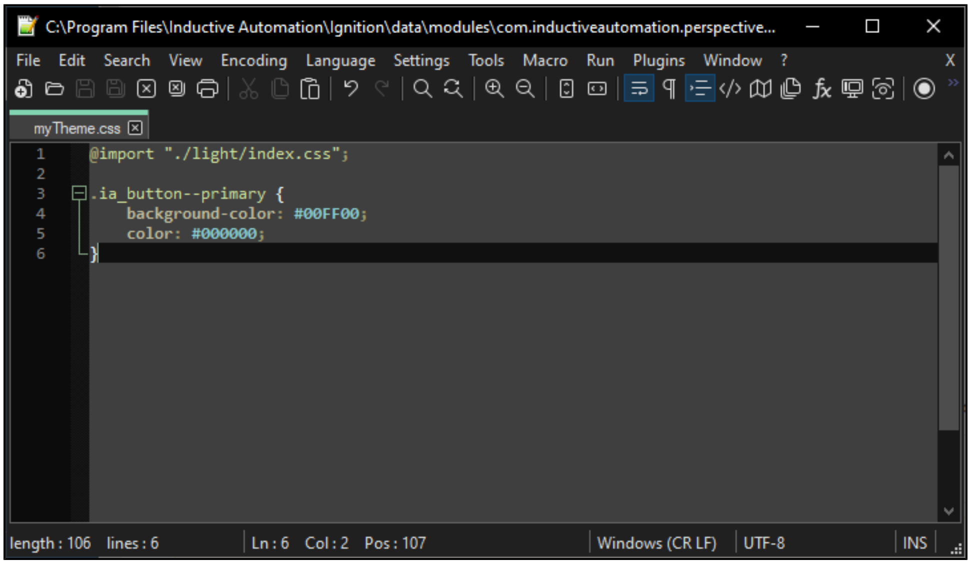
```css
@import "./light/index.css";
```

This will import the properties of the `light.css` file into your new .css file, acting as a base for your custom theme. You can change which . css file you want to reference by changing the folder name to another existing folder (such as `@import "./dark/index/css";`)

> **Note:** Referencing and importing settings from existing themes is recommended over directly referencing .css files. Many of the required . css files are nested deep within the various themes folders. Importing will allow you to skip over a few steps, reducing the chance of problems arising.

4. Once you add the initial import line, you can start adding your own changes to your custom theme file. In this example, we are using the default settings imported from the `light.css` theme, with the only changes being the background color and text color for all button components.

```css
.ia_button--primary {
        background-color: #00FF00;
        color: #000000;
}
```



C:\Program Files\Inductive Automation\Ignition\data\modules\com.inductiveautomation.perspective...
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
myTheme.css
length : 106 lines : 6
Ln : 6 Col : 2 Pos : 107
Windows (CR LF)
UTF-8
INS

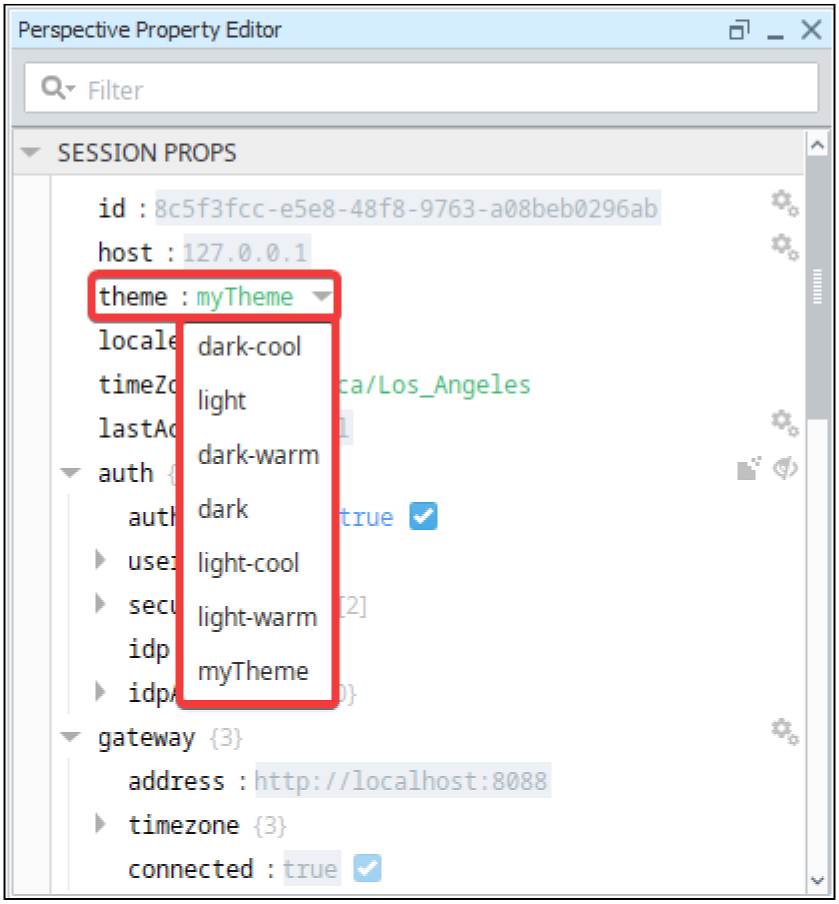
```css
@import "./light/index.css";

.ia_button--primary {
    background-color: #00FF00;
    color: #000000;
}
```

> **Note:** If you need a reference for all available Perspective component CSS class names, you can reference the various .css files located at `%installDirectory%\data\modules\com.inductiveautomation.perspective\themes\light\common`.

5. After you make your desired theme changes, save the .css file.

Using Your Custom Theme

To use your custom theme, select it from the Session Properties in the Designer using the `theme` property. You can select your custom theme from the dropdown menu.



Perspective Property Editor

Filter

SESSION PROPS

id : 8c5f3fcc-e5e8-48f8-9763-a08beb0296ab

host : 127.0.0.1

theme : myTheme

locale

timeZo ca/Los_Angeles

lastAc 1

auth {

auth true ✓

use

secu [2]

idp

idp/ 0}

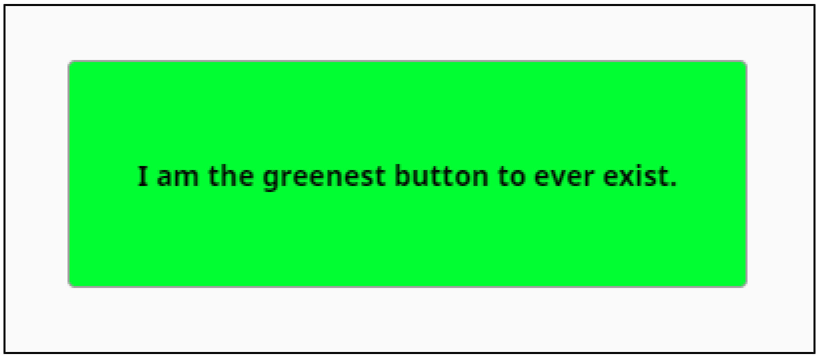
gateway {3}

address : http://localhost:8088

timezone {3}

connected : true ✓

Once you select your custom theme, your components will change their look according to your .css file.



I am the greenest button to ever exist.

There are some cases in which custom themes aren't enough to get your desired look. You can get around this by using Style Classes to fine-tune your components.

Scripting in Perspective

This section is designed to familiarize you with some of the basics of Python scripting in Perspective. Perspective scripting is particularly powerful, and can be used to control and fine-tune many aspects of project design.

For a more general view of scripting in Ignition, and an introduction to Python, see Scripting in our Ignition Platform section.

On this page ...

Perspective Scripting Fundamentals

Though Perspective uses the same basic platform (Jython) as other scripting environments in Ignition, interfacing with some unique features in Perspective might make it feel like a new scripting experience. Here are some key details that might be important to Perspective script writers.

Scopes

Perspective does not have a "Client" scope, because unlike Vision, Perspective does not have clients. All Perspective scripting is run on the Gateway, although session-specific functions (like navigation) will only affect a single session. Critically, this means that:

- Client-scoped scripting functions (`system.file`, `system.gui`, and `system.nav` functions) will *not* work in Perspective.
- Other scripting functions, like `system.util.getLogger()`, will run in a Gateway context.

Perspective Data Types

Component properties in Perspective are structured as JSON. However, interacting with them does not require any kind of specialized knowledge. Critically, every property in Perspective is one of three types:

Type	Description	Example

Value	A value is a simple piece of data, usually a number or string. Assigning a value to a value property is just like assigning a value to an ordinary Python variable.	`self.props.text = "My Text"`
		`self.props.startAngle = 5`

Object	An object is structured like a Python Dictionary, holding any number of elements. Each element can be a value, object, or an array.	`motorObject = {"motorNum":1,"motorState":"Running"}`
		`self.custom.myObject = {"operationNum":15,"motorObject":motorObject}`

Array	An array is structured like a Python List. Unlike an object where each element in the data type has an associated key, elements in an array only have a position, or offset. Elements in an array can be values, objects, or other arrays.	`rowObject1 = {"city":"Folsom","country":"United States","population":77271}`
		`rowObject2 = {"city":"Helsinki","country":"Finland","population":625591}`
		`self.props.data = [rowObject1,rowObject2]`

Object Traversal

In scripting, we can use component properties and methods to access related components, and view and session info. See Perspective Component Methods for details.

Scripting Transforms

Any property binding can make use of a Script Transform to apply any python script to the output value of the binding. For more information, see Transforms and Script Transforms.

Extension Functions

Extension Functions can be found in the Script Configuration window in certain Perspective Components. They allow users to add extended functionality to the component via scripting. These functions are generally more advanced and require better understanding of Python. Extension Functions differ from Event Handlers in that they are not event driven and are called by their component for a specific purpose when appropriate. Extension Functions are often called when the component first loads on a view or when it receives new input. For example, the Perspective Alarm Status Table has an Extension Function named **filterAlarm**. This function is called by the Alarm Status Table for each event before it is displayed in the table and it allows users to control what alarm events are displayed on the table or not.

From an object-oriented point of view, Extension Functions create a custom "subclass" of the base component type. Your subclass can then override and implement parts of the functionality of the component itself, in Python. Following Python object-oriented methodology, each extension function's first argument is called **`self`**. That is because these are methods that belong to the component's class itself, instance methods. The value of **`self`** will always be the component itself. Notice that this is different than Event Handler scripts where you are also given an **`event`** object in your scope. When you write an Extension Function, there is no **`event`** object so the component is given to you as the **`self`** object instead.

Each component Extension Function comes with its own documentation built-into the function's default implementation using a standard Python "docstring". The built-in documentation will contain descriptions for the arguments the Extension Function takes in as well as its return value, if any. You will find that you are unable to edit the function's signature or docstring. Changing the method's signature (arguments or function name) would prevent the component from calling it correctly. Changing the docstring could be misleading or confusing as you would lose the documentation for how your implementation of the function should work.

In This Section ...

Perspective Component Methods

On this page ...

A component method is a function that is defined on a component object. For example, this is how we would call a component method defined on the component object **self**:

```
output = self.myMethod(param1,param2)
```

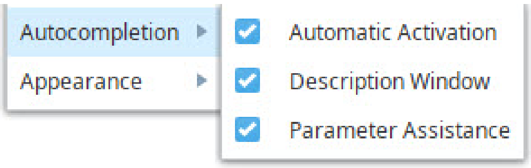
Perspective has a variety of component methods that are defined on *all* components, and it also offers you the ability to configure your own using custom methods.

Autocomplete

The following feature is new in Ignition version **8.1.18**
Click here to check out the other new features

A complete list of component methods (with their definitions) is available from the autocomplete popup. Starting in 8.1.18, the autocomplete popup is enabled by default and will appear after typing "."

To disable this feature, right-click anywhere within the Script Editor window and deselect **Automatic Activation.**



☑ Automatic Activation
☑ Description Window
☑ Parameter Assistance

Object Traversal

In most Perspective scripts, you are given a reference to a component object (often in the form of a **self** parameter). The object is given in component scripts, but has several methods and properties associated with it to help traverse to the other objects in a Perspective View or get values from the Session.

Object Traversal is limited to a single view. If a script needs to reference a component in a different view, or there is a possibility that the hierarchy of the view will change, then Message Handling should be utilized instead.

Component/Container		
Method /Property	**Description**	**Example**
.children	Returns all of the component's children.	`self.children`
. getChildren ()	Functionally similar to ".children" above.	`self.getChildren()`
.parent	Calling this property will move up the component hierarchy, accessing the parent container of the preceding object. Root containers will return the view, and views/sessions will return None.	`self.parent`
.getParent()	Functionally similar to ".parent" above.	`self.getParent()`

.getChild (string)	Method that looks for a child component of a given name. Returns None if not found.
	String can either be the name of a child object, or a path to an object delimited by a forward slash, allowing you to move through multiple items in the hierarchy in a single call.
		`self.getChild('Label_0')`
		`self.getChild`

		`('Container/Label_0')`
.getSibling (string)	Returns a reference to an object in the same container that the source component is located in. Similar to calling self.parent.getChild('component').	`self.getSibling('Label')`
.view	Calling this from anywhere within a view will return the parent view of the object.	`self.view`
.getView()	A method that will return the view, similar to ".view" above.	`self.getView()`

.page	Returns a page object associated with the page the current component is on.
	.close() can be called on the page object, and can accept a message string as a parameter.
		`page     = self.page`
		`pageID   = page.props.pageId`
		`pagePath = page.props.path`

.getPage()	A method that will return the page, similar to ".page" above.	`self.getPage()`

.session	Returns the current Perspective Session you are in. From this object you can get any of the existing properties of the Session, including custom properties. See the Session Objects sub table for more details.	`session = self.session`
		`sesName = session.props.gateway.address`
		`sesProp = session.custom.propertyName`

.getSession()	A method that will return the session, similar to ".session" above.	`self.getSession()`
Session Objects		

.close()
or
.close(message)
	When called will close the session.
	Optionally accepts a string message which will be displayed after the session closes.
		`session = self.getSession()`
		`session.close("The session has now closed")`

.getInfo()	Returns a PyJsonObjectAdapter object, which is functionally similar to a Python Dictionary. The object contain the same keys described on the system.perspective.getSessionInfo page.	`self.session.getInfo()["pageIds"]`
.getPages()	Returns a list of page objects.	`self.session.getPages()`

.getPage (ID)	Returns the page associated with the given string ID parameter, if it exists. Page ID values can be determined with the .getInfo() method.	`self.session.getPage("2b2eb647")`
.getProjectInfo()	The following feature is new in Ignition version **8.1.4** Click here to check out the other new features Returns a dictionary of project meta data, including name, title, description, lastModified, lastModifiedBy, views, and pageConfigs.	`self.session.getProjectInfo()`
View Objects		
.rootContainer	Returns the root container of the View.	`view.rootContainer`
.id	Returns a string that uniquely identifies the view instance. For example: Perspective/Views/path/to/view yields path/to/view@C	`view.id`

.session	Returns the current Perspective Session you are in. From this object you can get any of the existing properties of the Session, including custom properties. See the Session Objects sub table for more details.	`session = view.session`
		`sesName = session.gateway.address`
		`sesProp = session.custom.propertyName`

Object Traversal Examples

If you want to get other component properties in a view while scripting, you can use the above methods and properties to move around the View.

These examples assume you have the following structure/components in a View:

- View
    - Button 1
    - Text Field 1
    - Container 2
        - Text Field 2
    - Container 3
        - Text Field 3
        - Container 4
            - Button 4
            - Text Field 4

Scripting Example: Get component properties from a Button script

```
# this example code exists on the 'Button 1' in the above hierarchy.

# get to the view that the button is in
view = self.view

# get the text from 'Text Field 1'
text1 = self.getSibling('Text Field 1').props.text

# get the text from 'Text Field 2'
text2 = self.getSibling('Container 2').getChild('Text Field 2').props.text

# get the text from 'Text Field 3'
```

```
text3 = self.getSibling('Container 3').getChild('Text Field 3').props.text

# get the text from 'Text Field 4'. Either of these will work the same.
text4 = self.getSibling('Container 3').getChild('Container 4/Text Field 4').props.text
text4 = self.getSibling('Container 3').getChild('Container 4').getChild('Text Field 4').props.text
```

Scripting Example: Get component properties from a Button script

```
# this example code exists on the 'Button 4' in the above hierarchy.

# get to the view that the button is in
view = self.view

# get the text from 'Text Field 1'
text1 = self.parent.parent.getSibling('Text Field 1').props.text

# get the text from 'Text Field 2'
text2 = self.parent.parent.getSibling('Container 2').getChild('Text Field 2').props.text

# get the text from 'Text Field 3'
text3 = self.parent.getSibling('Text Field 3').props.text

# get the text from 'Text Field 4'
text4 = self.getSibling('Text Field 4').props.text
```

Built-In Methods

Perspective components contain several shared methods. This section details such methods. Some methods are only available to certain types of components. In these cases, the description for the method will state any limitations.

Refreshing Bindings

The `refreshBinding` function can be used to manually fire a binding, and is designed to be used on bindings that can poll (like query and Tag history bindings). In these instances, using refreshBinding in lieu of polling can save Gateway resources. The refreshBinding() function takes a string as a parameter, corresponding to the property that should be refreshed:

```
self.refreshBinding("props.data")
```

It is often useful to use refreshBinding() from a component message handler, since we can then refresh several applicable bindings via a single system.perspective.sendMessage call.

Requesting Focus

The `focus` method can be called by a component to request focus in a view. This is useful if you wish to control where keyboard input is directed after a particular action.

Due to the nature of focus, calling the `focus` method is only effective on components that can have focus. Input components such as the Text Field and Numeric Entry Field components can gain focus, but Display components like Labels and Images can not gain focus.

```
self.focus()
```

Requesting or Removing Tooltips

The following feature is new in Ignition version **8.1.10**
Click here to check out the other new features

The `requestTooltip` and `removeTooltip` methods can be used to display and remove tooltips from a scripting event. This is useful if you wish to allow users to request tooltips in a mobile session, where there is no `onMouseEnter` event.

You may call these functions for any component with meta tooltips enabled. Requesting a tooltip will have no effect if tooltips are not enabled for the component.

```
self.requestTooltip()
self.removeTooltip()

# enable and then call tooltip for 'Label 1' in the same view
self.getSibling('Label 1').meta.tooltip.enabled = True
self.getSibling('Label 1').requestTooltip()
```

Requesting or Removing Context Menus

The following feature is new in Ignition version **8.1.25**
Click here to check out the other new features

The `requestContextMenu` and `removeContextMenu` methods can be used to show or hide the context menu from a scripting event. As with the `requestTooltip` and `removeTooltip` methods, these methods are helpful to users in mobile sessions, to view or hide configured context menus. Requesting or removing a context menu will have no effect if the component context menu is not configured.

```
self.requestContextMenu()
self.removeContextMenu()
```

Requesting Print

The following feature is new in Ignition version **8.1.28**
Click here to check out the other new features

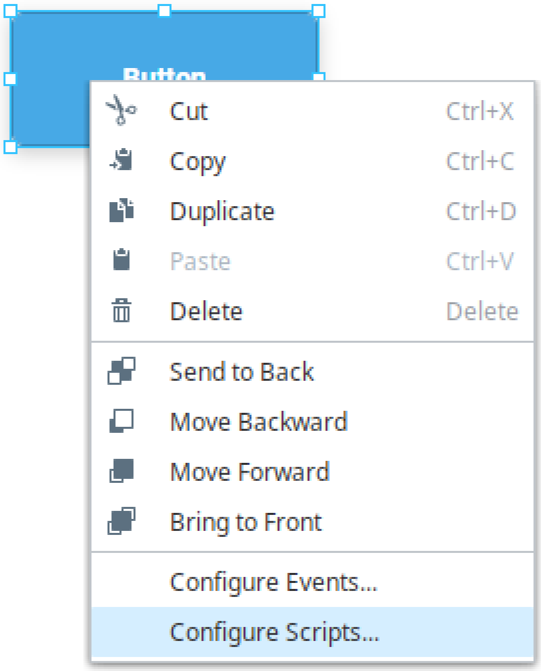
The `requestPrint` method can be called by a component, view, or page to print the contents of the selected target. If no target is specified, the component serves as the default target. Titles can also be assigned if saving the printed file. To request print using a Perspective component action instead, see the Request Print Action section on the Component Events and Action page.

```
self.requestPrint()
self.requestPrint('component', 'MyTitle')
```

Custom Methods

Perspective offers the option of configuring your own methods for a component. To configure a custom method:

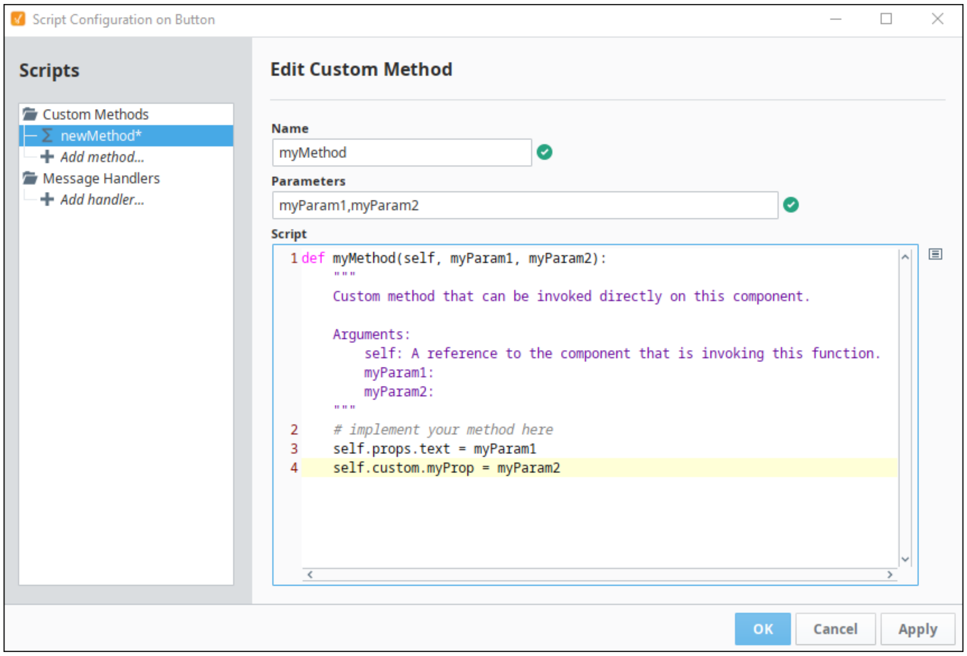
1. Right-click on the desired component in the Designer, and select **Configure Scripts...**



2. Under **Custom Methods**, double click on + **Add method...**
3. Enter a **Name** for your method, which will be used to call the method. We used **myMethod** in this example.
4. Enter any number of **Parameters** your method will need, separated by commas. For this instance, we have two parameters: **myParam1** and **myParam2**.
5. Add code to implement your method.

```
self.props.text = myParam1
self.custom.myProp = myParam2
```

> **Note:** A **self** object is provided in every custom method, but should *not* be provided as a parameter when calling the function.



Name: myMethod

Parameters: myParam1,myParam2

Script

```
def myMethod(self, myParam1, myParam2):
	"""
	Custom method that can be invoked directly on this component.

	Arguments:
		self: A reference to the component that is invoking this function.
		myParam1:
		myParam2:
	"""
	# implement your method here
	self.props.text = myParam1
	self.custom.myProp = myParam2
```

6. Click **OK** to commit your method.
7. To call this method, use:

```
self.myMethod(param1,param2)
```

For example:

```
self.myMethod("Hi!","This is a test")
```

Related Topics ...

- Component Message Handlers

Component Message Handlers

In Perspective, Component Message Handlers are the preferred way to pass parameters between components or Views. Doing this involves the system.perspective.sendMessage function. There are typically two steps involved:

1. Creating a message handler on the component that will listen for a particular call.
2. Create a script that will call the message handler.

This page will demonstrate how to prepare both steps. The goal of this example is to create a script that will cause a button to change the text property of a Label.

On this page ...

Object Traversal

When writing a component based script, Object Traversal should be avoided where possible. Object Traversal is the process of declaring hard-coded component paths in a script similar to how the Vision module uses component paths in scripting. While component paths exists in Perspective, they are brittle: changes to the hierarchy in the view (such as placing a component into a new container), or changing a component name will invalidate any paths defined before the change, since the component's relative location has changed.

Consider the following:

Pseudocode - Example Path

```
self.getSibling('Text Field').props.text
```

The path described above only works if there is component named "Text Field" in the same container as the component that is running this script. If the Text Field component is renamed at any point, this reference to the component will fail. Additionally, if the Text Field is placed in a different container, then the getSibling() call will no longer work. Components in other containers (in the same view) are available, but are not siblings.

Additionally, Object Traversal can't be used to reference a property in a separate view: i.e., a script in View A cannot reference something in View B.

We **strongly** suggest you utilize message handlers when a script is trying to interact with a component in another View.

Message Types

When sending a message, the Message Type field represents which Message Handler should respond. If a script sends a message with a type of "foo", then any handler listening for a Type of "foo" will execute. This means multiple components in the same window can all have Message Handlers with the same Type: e.g., if a view has a "reset" button to clear out multiple input components, then each input component can simply have a "reset" message type that clears the field, allowing a single message to trigger multiple handlers. Alternatively, if only a single handler should execute when sending the message, simply give that one handler a unique Type.

Message Handlers

Watch the Video

Component Paths

Watch the Video

Message Handler Scope

Message Handlers can be limited in scope, meaning the range of the sent message (or range of the listener) can be confined to a particular scope. The available scopes are:

Scope	Definition
View	Messages can only be heard by listeners within the same View that are listening for view-scoped broadcasts.
Page	Messages can only be heard by any listeners within the same Page that are listening for page-scoped broadcasts. This includes listeners both within the same View and those in other Views contained within the same page (including Docked Views and Popup Views).
Session	Messages will be heard by any listeners in any open tabs in your workspace that are listening for session-scoped messages.

For example, you can send a message that is scoped to just the View where the message originated, meaning only listeners in the same View will be able to respond. This is useful if you sent a message from a popup view, and didn't want any other views to respond.

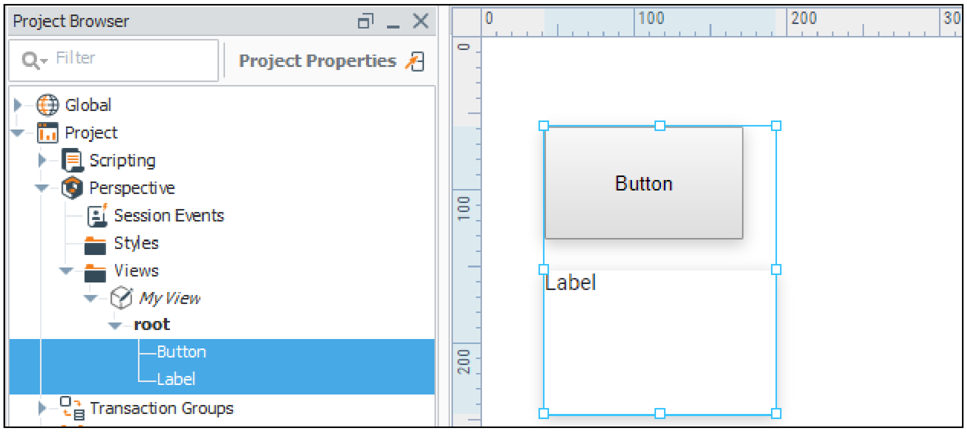
There are two ways to limit the scope of a message:

1. The system.perspective.sendMessage function contains a scope parameter that will restrict the range on the message being sent.
2. The Message Handlers have a **Listen Scopes** setting that can filter out messages from certain scopes.

Message Handler Example

Step 1 - Prepare Perspective Workspace

1. Open the Designer.
2. Switch over to the Perspective Workspace by clicking on **Perspective** in the Project Browser.
3. Right-click on the Views folder and select New View.
4. Give the new **View** a name and click the Create View button. The name of the view will not matter for this example.
5. Place a **Button** and a **Label** component on the View.



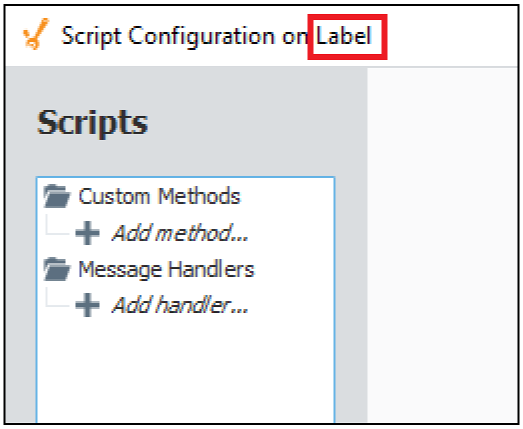
Step 2 - Create A Message Handler

Message Handlers are effectively user-created scripting events. A user can define a Message Handler that listens for a particular message. The idea being that some other component will broadcast a message, and if the type of the message matches what the Message Handler is listening for, the Message Handler will execute a script.

Message Handlers are useful because they can respond to messages from multiple components: if something sends a message with the correct type, then the Message Handler will execute. Additionally, message can be sent across separate views, pages, or throughout the entire session.

In our example, we want the Label's text property to change when something else in the view happens (in our case, our button is pressed), so it would make sense to configure a Message Handler (listener) on the Label. This way, if we relocate the Label component in the view, the script will still work.

1. Right-click on the Label, and select **Configure Scripts**.
2. The Script Configuration window appears. Make sure the title bar on the window states that it's the **Label** component.



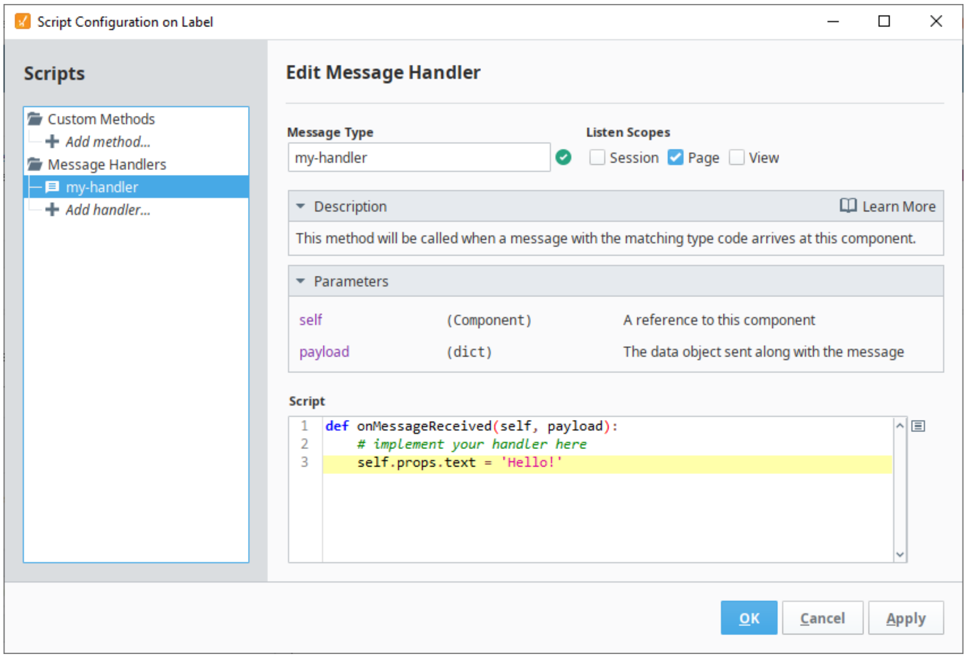
3. On the left side of the **Script Configuration** window, you will see a tree. Double-click on the **Add handler...** item.

4. A new handler named "type-name-here" will appear. Under **Message Type** type `my-handler`. Note that we're using all lowercase characters: **message handlers are case-sensitive**. The Message Type is effectively the name of the message handler. When sending a message, we will specify this message handler's type, which will cause it to respond by executing the script.
5. In our example, let's change the text on the Label to "Hello!". Under the script area, type the following script:

Python - Set the Text

```
self.props.text = 'Hello!'
```

6. This example is fairly limited to this one view. Thus, let's limit the scope so that it will only respond to messages from the same view the Label is on. Under the **Listen Scopes**, leave **Page** as the only selected scope.



Message Type: my-handler

Listen Scopes: ☐ Session ☑ Page ☐ View

Description

This method will be called when a message with the matching type code arrives at this component.

Parameters

self	(Component)	A reference to this component
payload	(dict)	The data object sent along with the message

Script

```
def onMessageReceived(self, payload):
    # implement your handler here
    self.props.text = 'Hello!'
```

7. Click the **OK** button. We just created a Message Handler. In the next step, we'll create a script that will call the Message Handler.

Step 3 - Send A Message

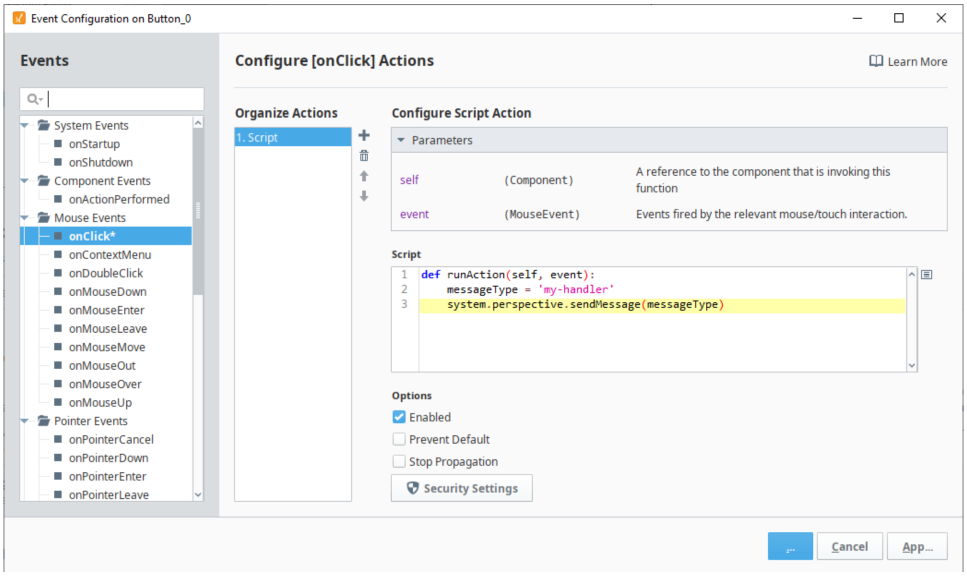
In this step, we place a script on the Button that will call our Message Handler.

1. Right-click on the Button, and select **Configure Events...** The **Event Configuration** window appears.
2. We want our script to trigger when the button is pressed. On the left side of the window, in the Mouse Events folder, select **onClick**.
3. The **Organize Actions** list will appear. Press the **Add** ➕ icon. A popup list will appear.
4. Select the **Script** action.
5. Add the following code:

Python - Sending a Message

```
messageType = 'my-handler'
system.perspective.sendMessage(messageType)
```

Make sure the script lines are indented one time.



self	(Component)	A reference to the component that is invoking this function
event	(MouseEvent)	Events fired by the relevant mouse/touch interaction.

```
def runAction(self, event):
    messageType = 'my-handler'
    system.perspective.sendMessage(messageType)
```

☑ Enabled
□ Prevent Default
□ Stop Propagation

6. Click **OK** to apply the script.

The example is now running. From the Designer, enable **Preview mode**, and then click on the Button component. The text on the label should update.

Note: In this simple example, the major issue you may run into is the Message Type. Recall from #4 in Step 1 of the example, that Message Type is case-sensitive, so make sure the script on the Button is correctly referencing the message type, and try again.

Passing Parameters Example

Let's make the previous example more complicated and pass some values with the message. The example above can be modified to determine the timestamp. When passing parameters in a script, the most direct approach is to include any parameters along with the message. Message Handlers have a built-in argument called a **payload**, which is used to transfer values to the handler. The payload is simply a Python Dictionary, so please see the Dictionaries page for more information.

Alternatively, we could create a session property to hold the value, and have the label reference the session property. However this would require that we either create a new property to hold the value or overwrite the value of another property. Thus, our next example will demonstrate how to utilize the payload.

Avoid Storing Values in Tags
Since Tag values are shared by all Perspective sessions, you may not want to write the parameters in Tags. Doing so would result in each session instance potentially trying to overwrite the same value.

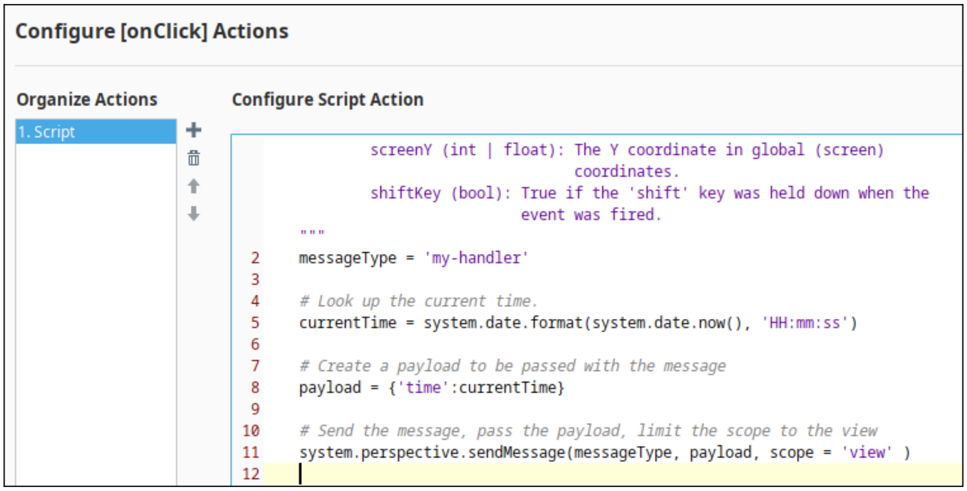
Step 1 - Update the Button Script Action

1. Right-click on the Button component and select **Configure events...**
2. Replace the code with the following:

Python - Check the Time, Send a Message

```
messageType = 'my-handler'

# Look up the current time.
currentTime = system.date.format(system.date.now(), 'HH:mm:ss')

# Create a payload to be passed with the message
payload = {'time':currentTime}

# Send the message, pass the payload, limit the scope to the view
system.perspective.sendMessage(messageType, payload, scope = 'view' )
```

3. Make sure the code is indented one time.



```
             screenY (int | float): The Y coordinate in global (screen)
                                    coordinates.
             shiftKey (bool): True if the 'shift' key was held down when the
                              event was fired.
     """
2    messageType = 'my-handler'
3
4    # Look up the current time.
5    currentTime = system.date.format(system.date.now(), 'HH:mm:ss')
6
7    # Create a payload to be passed with the message
8    payload = {'time':currentTime}
9
10   # Send the message, pass the payload, limit the scope to the view
11   system.perspective.sendMessage(messageType, payload, scope = 'view' )
12
```

Note: On line 8, we're creating a dictionary, creating a key called "time", and storing the current time with the "time" key. When our handler receives the payload, it can retrieve the value we passed by referencing the "time" key.

4. Once finished, click the **OK** button.

Step 2 - Update the Message Handler

Now that we're including a payload with the message, we need to modify our handler so that it will extract the time from the payload.

1. Right-click on the Label component, and select **Configure scripts...**
2. Replace the original code with the following:

```
# Access the time by referencing the 'time' key
self.props.text = payload['time']
```

3. Click **OK**.
4. To test it, enable **Preview mode**, and click the Button component. You will see the current time populate in the Label.

Related Topics ...

- Scripting
- Dictionaries

Perspective Property Change Scripts

With Perspective, individual component properties can have a property change script. When a change script is set up on a property, it will run when the property changes its value. Multiple different properties on the same component can each have different scripts configured. In Perspective, you can put a property change script on any component property.

A very common example of a property change script would be to take the dataset from a binding and modify it into a new dataset using other information on screen. This can be accomplished with a Script Transform instead.

> **Note:** Expression bindings are generally quicker and have less impact on session performance than a script. Before creating a property change script, consider if the goal of the script can be accomplished with an expression binding instead.

On this page ...

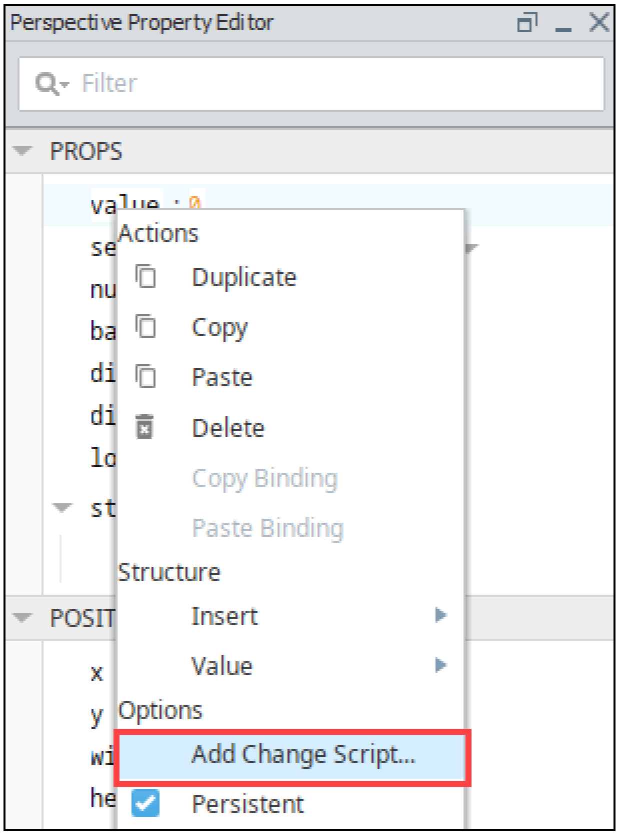
INDUCTIVE UNIVERSITY

Property Change Scripts

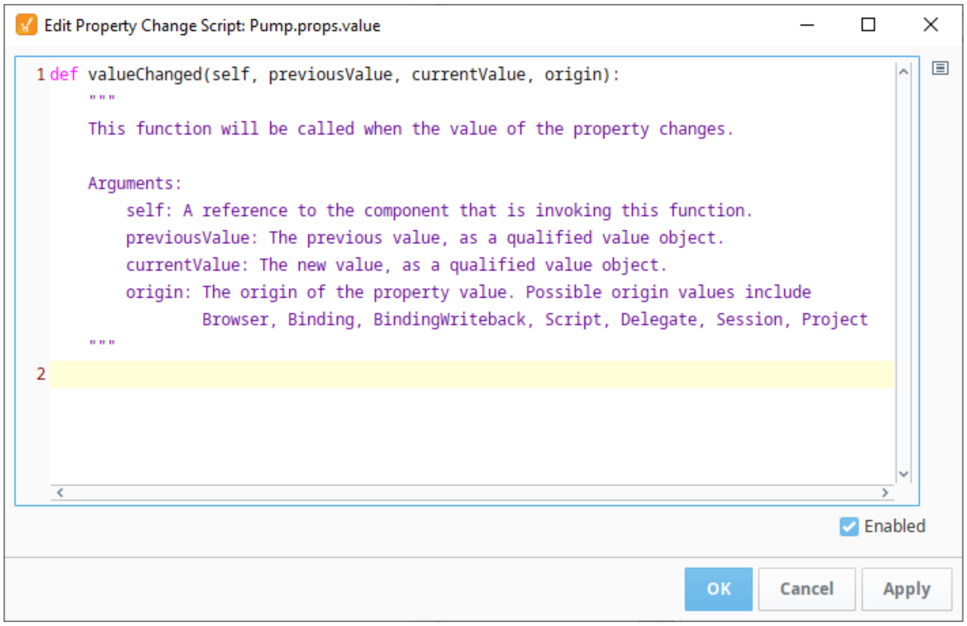
Watch the Video

Add a Property Change Script

1. To add a property change script to a property, right click on the property Property Editor and select **Add Change Script...**



2. The Edit Property Change Script screen is displayed.



Edit Property Change Script: Pump.props.value

```
def valueChanged(self, previousValue, currentValue, origin):
    """
    This function will be called when the value of the property changes.

    Arguments:
        self: A reference to the component that is invoking this function.
        previousValue: The previous value, as a qualified value object.
        currentValue: The new value, as a qualified value object.
        origin: The origin of the property value. Possible origin values include
                Browser, Binding, BindingWriteback, Script, Delegate, Session, Project
    """

```

☑ Enabled

3. Type in the script that you want to run and click **OK**.

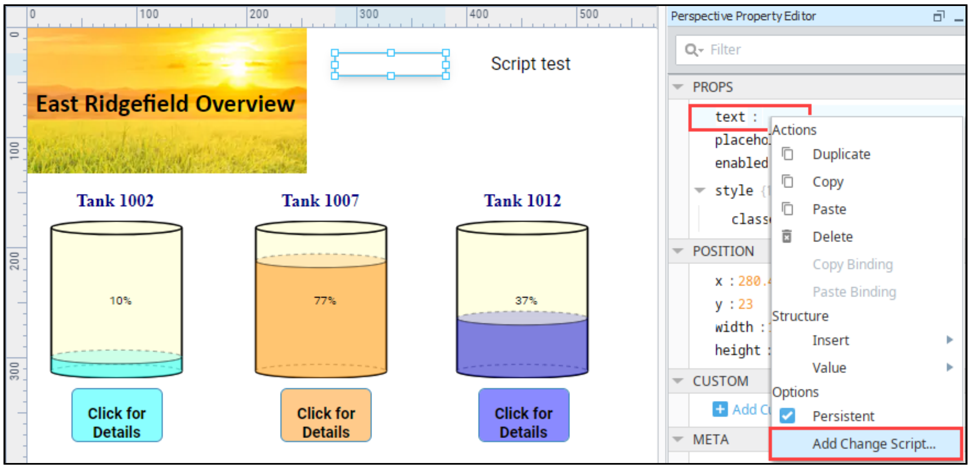
Property Change Arguments

<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td>self</td><td>A reference to the component that has the property in question. If the property change script is on a session property, the session object will be passed.</td></tr>
<tr><td>previousValue</td><td>The previous value, as a QualifiedValue object. QualifiedValue objects have a value, quality, and timestamp. See also Scripting Object Reference.</td></tr>
<tr><td>currentValue</td><td>The new value, as a QualifiedValue object. QualifiedValue objects have a value, quality, and timestamp. See also Scripting Object Reference.</td></tr>
<tr><td>origin</td><td>The origin of the property value, as a unicode string. The origin parameter will take on one of six types depending on how the property value is being updated:
<table>
<tr><th>Name</th><th>Description</th><th>Example</th></tr>
<tr><td>Browser</td><td>Used when the change comes from the Browser interface.</td><td>The user changes the <b>text</b> property on a text field by typing a word into the field.</td></tr>
<tr><td>Binding</td><td>Used when the change comes from a binding (or transform) generating a new value.</td><td>A Tag changes value, and a property with a binding to that tag is updated.</td></tr>
<tr><td>BindingWriteback</td><td>Used when the change comes from a bidirectional binding writing back to its source.</td><td>ComponentB's <b>value</b> property has a bidirectional binding to the <b>value</b> property on ComponentA. If ComponentB's <b>value</b> changes, then a property change script on ComponentA will have an origin of <b>BindingWriteback</b>.</td></tr>
<tr><td>Script</td><td>Used when the change comes from a script.</td><td>A user presses a button, and a script on the button assigns a new value to a custom property.</td></tr>
<tr><td>Delegate</td><td>Used when a change to a property comes from something intrinsic to the component's design.</td><td>A complex component that automatically fills itself with data, like the alarm status table component.</td></tr>
<tr><td>Session</td><td>Used when the session itself causes the property change.</td><td>A change in user privileges causes access to be revoked, resulting in a change in the <b>auth</b> session property.</td></tr>
<tr><td>Project</td><td>Used when the default property value is changed in the designer and saved.</td><td>A session property was set to a value of "A." The default value of that property was then changed in the designer to "B" and saved. The value for that property changes from "A" to "B" in the running sessions.</td></tr>
</table></td></tr>
<tr><td>missedEven</td><td></td></tr>
</table>

ts	The following feature is new in Ignition version **8.1.4** Click here to check out the other new features A flag indicating that some events have been skipped due to event overflow.

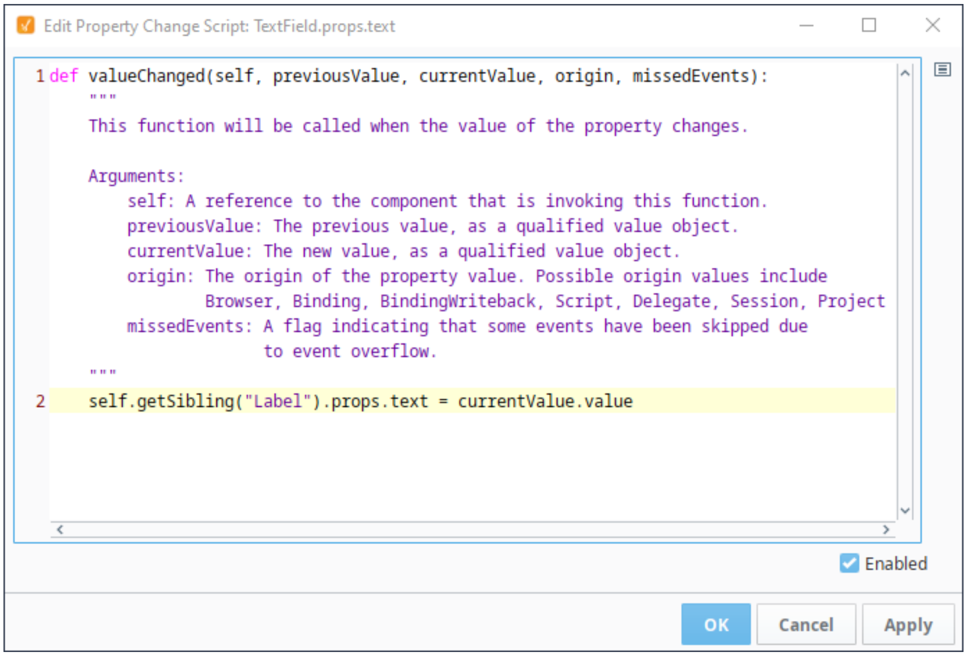
Change Script Example

1. Place a **Text Field** component and **Label** component on a Perspective view.
2. Select the Text Field component then right click on the **text** property.
3. Click on **Add Change Script**.



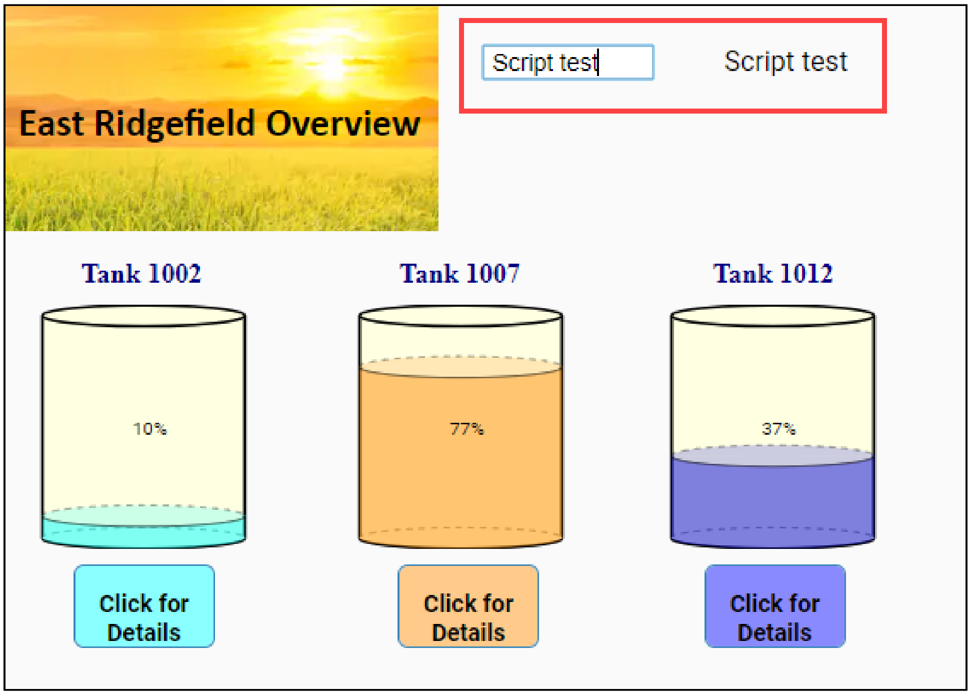
4. The Edit Property Change Script screen is displayed. Enter the following script, which will write the current value of the Text Field component to the Label component.

```
self.getSibling("Label").props.text = currentValue.value
```



```
def valueChanged(self, previousValue, currentValue, origin, missedEvents):
    """
    This function will be called when the value of the property changes.

    Arguments:
        self: A reference to the component that is invoking this function.
        previousValue: The previous value, as a qualified value object.
        currentValue: The new value, as a qualified value object.
        origin: The origin of the property value. Possible origin values include
                Browser, Binding, BindingWriteback, Script, Delegate, Session, Project
        missedEvents: A flag indicating that some events have been skipped due
                      to event overflow.
    """
    self.getSibling("Label").props.text = currentValue.value
```

5. Click **OK** to commit the script. You'll see that the Property Editor now shows a **Change Script** ▤ icon next to the text property.
6. **Save** your project.
7. In a Perspective Session, enter some text into the Text Field component and hit Return. You'll see that the contents are repeated to the Label component.


East Ridgefield Overview
Script test
Script test
Tank 1002
10%
Tank 1007
77%
Tank 1012
37%
Click for Details
Click for Details
Click for Details

Perspective Session Event Scripts

Perspective offers a collection of *Session Events* designed to allow the Gateway to track and interact with the Session at critical moments. Specifically, they are scripts that run in the Gateway when a Session starts up, shuts down, or runs a Native App Action.

There are nine configurable Session events in Perspective:

- Startup
- Shutdown
- Page Startup (new in 8.1.0)
- Authentication Challenge Completed (new in 8.1.16)
- Barcode
- Bluetooth
- NFC
- Accelerometer
- Message
- Keystroke

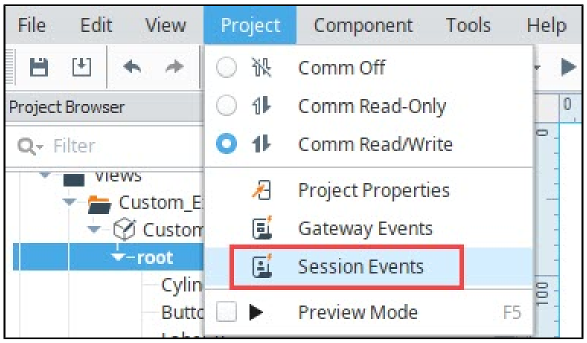
Note: Although they are designed to handle Session Events, the scripts that you write will be run in a *Gateway* scope, not a Session scope.

On this page ...

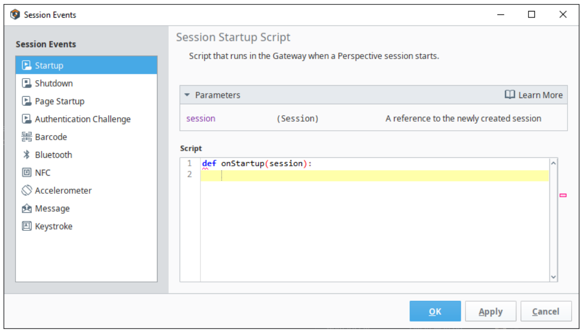
Configuring Session Events

To start working on a Session event script:

1. In the Project Browser, double-click on the **Session Events** section:



2. The **Session Events** dialog will appear:


Session Events

Session Events
- Startup
- Shutdown
- Page Startup
- Authentication Challenge
- Barcode
- Bluetooth
- NFC
- Accelerometer
- Message
- Keystroke

Session Startup Script

Script that runs in the Gateway when a Perspective session starts.

Parameters — Learn More

session	(Session)	A reference to the newly created session

Script

```
def onStartup(session):

```

OK Apply Cancel

3. Configure a script by selecting one of the events on the left-hand side.

Startup and Shutdown Event Scripts

Startup and **Shutdown** events run, naturally, whenever a Session starts or ends. In each case, the Gateway will have access to the **session** object associated with the section, complete with all Session properties.

When designing a startup or shutdown event script, custom Session properties can be used to pass any additional information to the Gateway, or, in the case of a startup script, to pass information to the newly opening Session. You can configure custom Session properties from the Page Configuration dialog, by clicking on the **Settings** ⚙ icon in the Designer.

A shutdown event script will run *specifically* when a Session is ending. This happens specifically when any of the following events occur:

- The Session closes due to a timeout. Session timeout is configurable in the **Perspective > General** section of **Project Properties** in the Designer
- The user is no longer authorized to run the Session.
- The redundancy system determines that the Gateway is inactive.
- The licensing system no longer permits the user to run the Session.
- The project is no longer runnable.
- The project is deleted.

> **Note:** Closing the browser tab with the Session will *not* immediately close the Session; the Session must first time out.

Startup and Shutdown Example

This example will record the Session start and end time to the database.

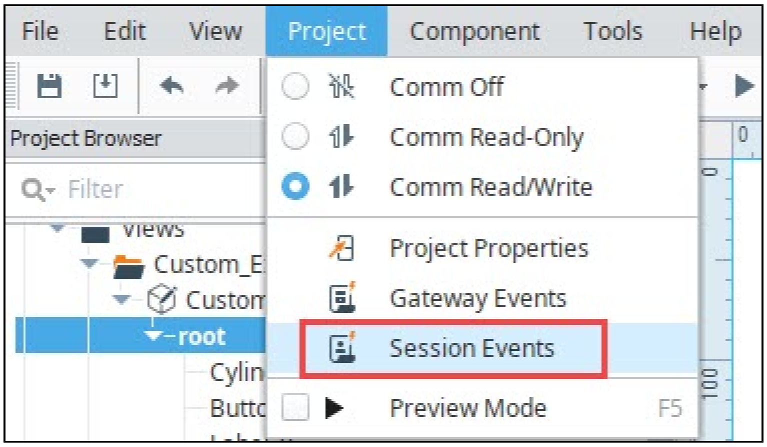
1. For this example, we need to set up a few queries that we can use to write our data to the database.
2. Make a new named query called **Startup Query**.
   a. Set the Query Type to **Update Query**.
   b. Set up a single Value type parameter with a name of **SessionID** and a datatype of string.
   c. Add the query:

```
INSERT INTO sessions (session_id, start_time)
VALUES (:SessionID, CURRENT_TIMESTAMP)
```

3. Make a second new named query called **Shutdown Query**.
   a. Set the Query Type to **Update Query**.
   b. Set up a single Value type parameter with a name of **SessionID** and a datatype of string.
   c. Add the query:

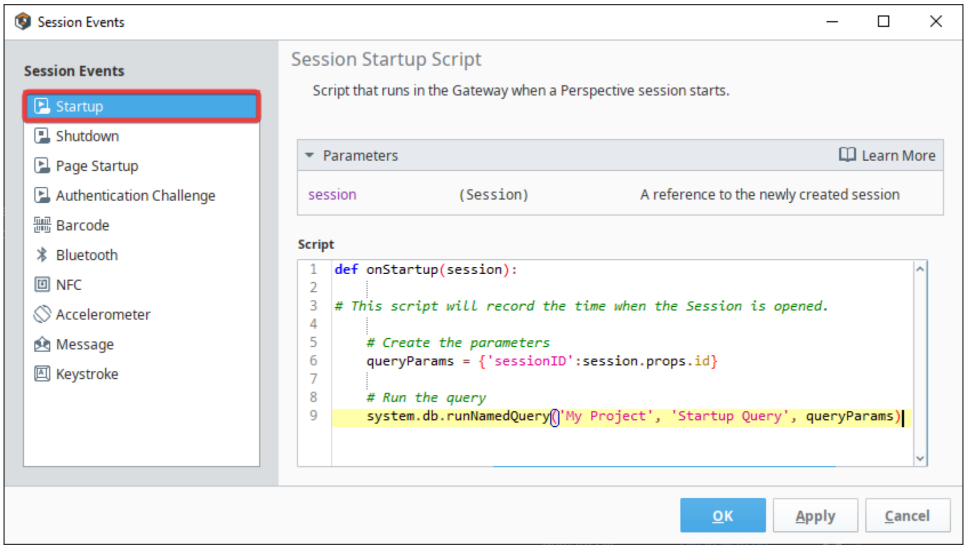
```
UPDATE sessions
SET end_time = CURRENT_TIMESTAMP
WHERE session_id = :SessionID
```

4. Next, we need to add Session Events so that Perspective knows to run those queries on startup and shutdown.
5. Under the **Project** tab, select **Session Events**.



6. On the **Session Events** screen, click the **Startup** icon.
7. Add the following script to the page:

```
# This script will record the time when the Session is opened.

        # Create the parameters
        queryParams = {'sessionID':session.props.id}

        # Run the query
        system.db.runNamedQuery('My Project', 'Startup Query', queryParams)
```



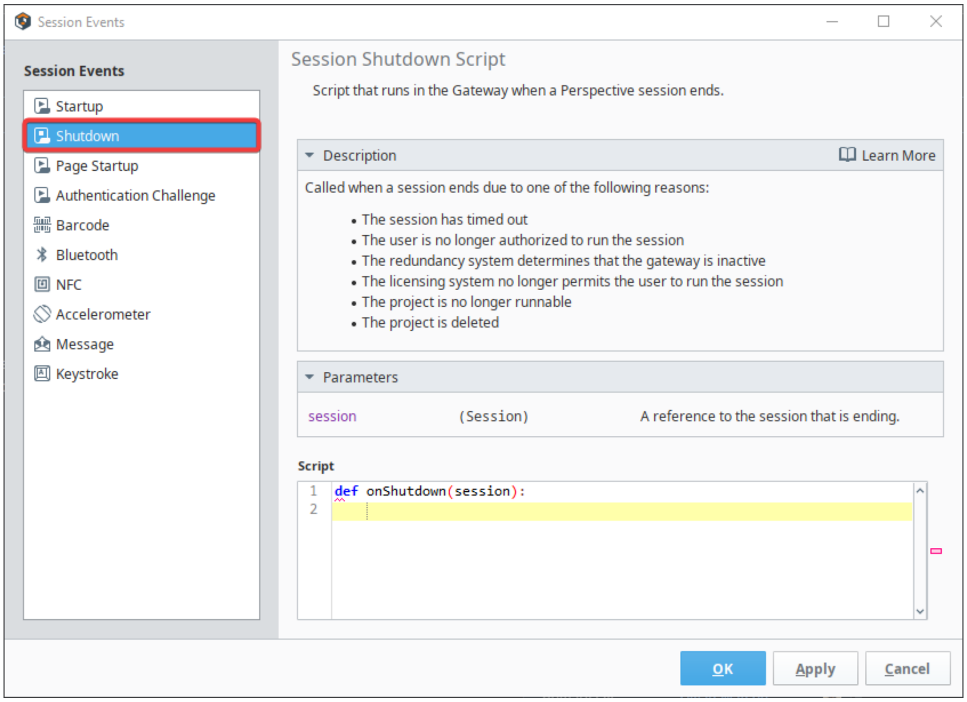
Session Startup Script

Script that runs in the Gateway when a Perspective session starts.

session	(Session)	A reference to the newly created session

```
def onStartup(session):

# This script will record the time when the Session is opened.

	# Create the parameters
	queryParams = {'sessionID':session.props.id}

	# Run the query
	system.db.runNamedQuery('My Project', 'Startup Query', queryParams)
```

8. Click **Apply** and then click the **Shutdown** icon.



Session Shutdown Script

Script that runs in the Gateway when a Perspective session ends.

Called when a session ends due to one of the following reasons:

- The session has timed out
- The user is no longer authorized to run the session
- The redundancy system determines that the gateway is inactive
- The licensing system no longer permits the user to run the session
- The project is no longer runnable
- The project is deleted

session	(Session)	A reference to the session that is ending.

```
def onShutdown(session):

```

9. Add the following script to the page:

```
# This script will record the time when the Session times out. Note that after the Session is closed,
# the Session won't time out until the time out period is reached.

        # Create the parameters.
        queryParams = {'sessionID':session.props.id}

        # Run the query.
        system.db.runNamedQuery('My Project', 'Shutdown Query', queryParams)
```

10. Click **OK**.
11. Save your project.

Test the Example

To test the example, open the Perspective App on your mobile device and load the project.

1. You should see a new entry in the database table with the time the time the Session was started as well as the session ID.
2. After the Session times out (such as after it is closed) you should see the original entry get updated to include the new shutdown time.

Page Startup

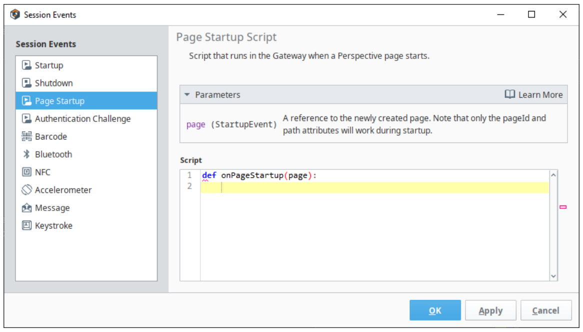
Page Startup

Watch the Video

The following feature is new in Ignition version **8.1.0**
Click here to check out the other new features

This script runs in the Gateway when a Perspective page *starts* in a new tab or window.

Note: Navigating to a page configuration in a tab that's already opened (such as using system.perspective.navigate) will not trigger this event.



Script that runs in the Gateway when a Perspective page starts.

page (StartupEvent) A reference to the newly created page. Note that only the pageId and path attributes will work during startup.

```
def onPageStartup(page):

```

<table>
<tr><th>Arguments</th><th>Description</th></tr>
<tr><td>page</td><td>A reference to the newly created page. Note that other page props will not return a valid value on page startup.
Accessing properties under this argument requires that you include "<code>.props.</code> " along the path. Demonstrations are included below:
<table>
<tr><th>Property</th><th>Description</th><th>Example</th></tr>
<tr><td>pageId</td><td>Identifier associated with the target page.</td><td><code># Provides the page id.
pageId = page.props.pageId</code></td></tr>
<tr><td>path</td><td>Represents the Page URL of the page, for example: / or /myPage.</td><td><code># Provides a path to the page.
pagePath = page.props.path</code></td></tr>
</table>
</td></tr>
</table>

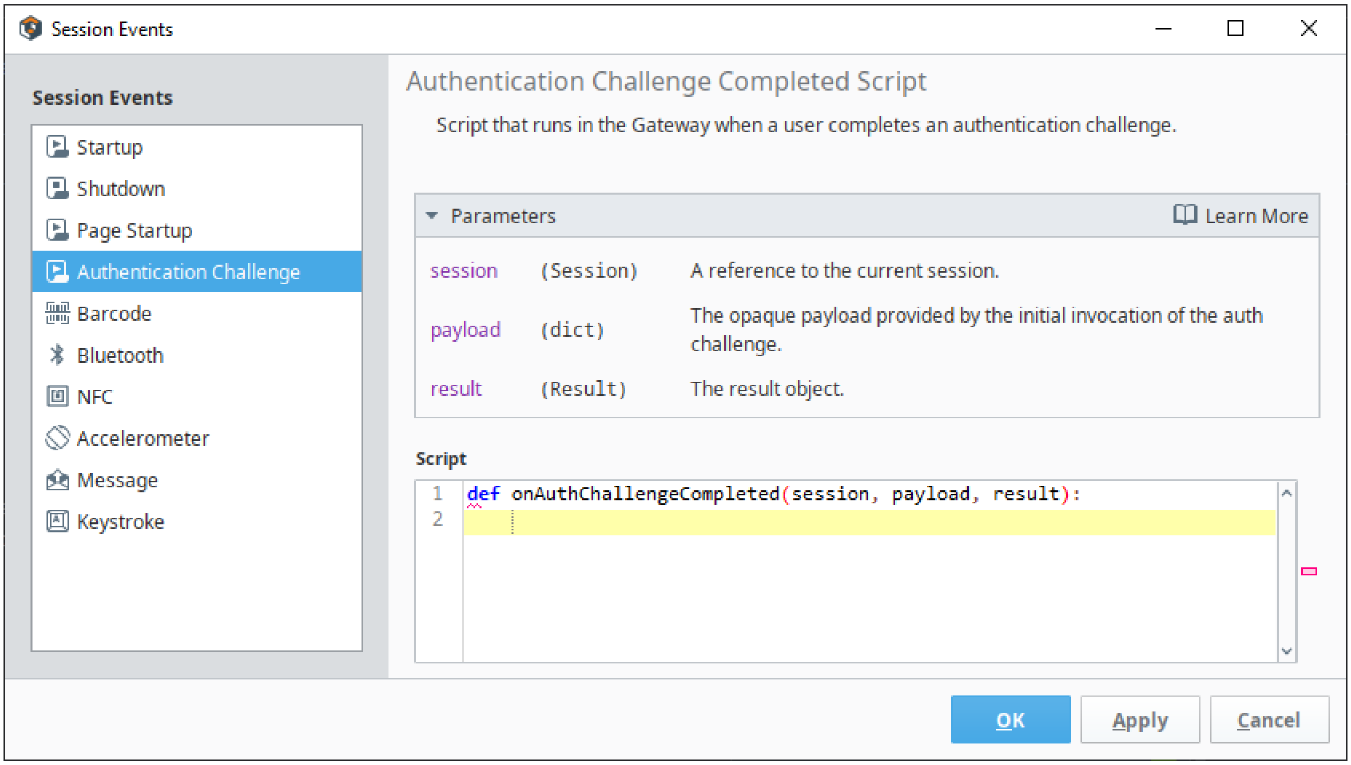
In addition to the arguments above, the Session is available from this event:

```
# For example, we can access the session id with the following:
sessionId = page.session.props.id
```

Authentication Challenge

The following feature is new in Ignition version **8.1.16**
Click here to check out the other new features

This event is fired when an authentication challenge is completed by the user.



Authentication Challenge Completed Script

Script that runs in the Gateway when a user completes an authentication challenge.

Parameters — Learn More

session	(Session)	A reference to the current session.
payload	(dict)	The opaque payload provided by the initial invocation of the auth challenge.
result	(Result)	The result object.

Script

```
def onAuthChallengeCompleted(session, payload, result):

```

<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td>session</td><td>An object that references the Project Session that triggered the authentication challenge. Use this to identify the specific Session that triggered the authentication challenge.</td></tr>
<tr><td>payload</td><td>The opaque payload provided by the initial Authentication Challenge Action or Scripting function invocation.</td></tr>
<tr><td>result</td><td>A Result object that can be parsed by the following functions:
<table>
<tr><th>Function</th><th>Description</th></tr>
<tr><td>isSuccess()</td><td>Returns true if the authentication challenge was a success. If true, optionally call <b>getAsSuccess()</b> to get the result as a <b>Result.Success</b> object for further parsing. If false, optionally call <b>getAsError()</b> to get the result as a <b>Result. Error</b> object for specific details pertaining to the error.</td></tr>
<tr><td>getAsSuccess()</td><td>Returns the result as a <b>Result.Success</b> object if the result is successful, otherwise throws an <b>UnsupportedOperationException</b> if the result is an error. Check <b>isSuccess()</b> or <b>isError()</b> before calling this function to ensure you get the correct result type.
<b>Note:</b> It is not necessary to call <b>getAsSuccess()</b> . If <b>isSuccess()</b> returns true, the result type is <b>Result.Success</b> , so all of the properties and functions of a <b>Result.Success</b> object are available on the original result reference.</td></tr>
<tr><td>isError()</td><td>Returns true if the authentication challenge resulted in an error. If true, optionally call <b>getAsError()</b> to get the result as a <b>Result.Error</b> object for specific details pertaining to the error.</td></tr>
</table></td></tr>
</table>

<table>
<tr><td>getAsError()</td><td>Returns the result as a <b>Result.Error</b> object if the result is an error, otherwise throws an <b>UnsupportedOperationException</b> if the result is successful. Check <b>isSuccess()</b> or <b>isError()</b> before calling this function to ensure you get the correct result type.
<b>Note:</b> It is not necessary to call <b>getAsError()</b> . If <b>isError()</b> returns true, the result type is <b>Result.Error</b> , so all of the properties and functions of a <b>Result.Error</b> object are available on the original result reference.</td></tr>
</table>

This object contains different properties depending on whether the authentication challenge result is a success or an error.

<table>
<tr><th>Object</th><th>Description</th></tr>
<tr><td>Result.
Success</td><td>An object that represents a successful authentication challenge result. Call <b>getContext()</b> to return a <b>WebAuthUserContext</b> object with the following properties:
<table>
<tr><th>Property</th><th>Description</th></tr>
<tr><td>idp</td><td>The name of the IdP which challenged the user.</td></tr>
<tr><td>securityZones</td><td>A string array containing the names of the security zones associated with the challenged user.</td></tr>
<tr><td>user</td><td>A <b>WebAuthUser</b> object that contains the following properties:
<table>
<tr><th>Property</th><th>Description</th></tr>
<tr><td>id</td><td>The unique identifier associated with the challenged user</td></tr>
<tr><td>userName</td><td>The challenged user’s username.</td></tr>
<tr><td>firstName</td><td>The challenged user’s first name, or null / None if not defined for this user.</td></tr>
<tr><td>lastName</td><td>The challenged user’s last name, or null / None if not defined for this user.</td></tr>
<tr><td>email</td><td>The challenged user’s email address, or null / None if not defined for this user.</td></tr>
<tr><td>roles</td><td>The challenged user’s roles as a string array.</td></tr>
</table></td></tr>
<tr><td>securityLevels</td><td>A string array of security level paths representing all of the security levels associated with the challenged user</td></tr>
</table></td></tr>
<tr><td>Result.
Error</td><td>An object that represents an error authentication result. It can be parsed by the following functions:
<table>
<tr><th>Function</th><th>Description</th></tr>
<tr><td>isGeneric()</td><td>Returns true if the error is not related to a timeout or user cancellation. Optionally call <b>getAsGeneric()</b> to get the <b>Result.Error</b> as a <b>Result.Error.Generic</b> object for specific details pertaining to the generic error.</td></tr>
<tr><td>getAsGeneric()</td><td>Returns the result as a <b>Result.Error.Generic</b> object if the error is generic, otherwise throws an <b>UnsupportedOperationException</b> . Check <b>isGeneric()</b> before calling this function to ensure you get the correct result type.
<b>Note:</b> It is not necessary to call <b>getAsGeneric()</b> . If <b>isGeneric()</b> returns true, the result type is <b>Result.Error.Generic</b> , so all of the properties and functions of a <b>Result.Error.Generic</b> object are available on the original result reference.</td></tr>
<tr><td>isTimeout()</td><td>Returns true if the error is due to a timeout. Optionally call <b>getAsTimeout()</b> to get the <b>Result.Error</b> as a <b>Result.Error.Timeout</b> object for specific details pertaining to the timeout error.</td></tr>
<tr><td>getAsTimeout()</td><td>Returns the result as a <b>Result.Error.Timeout</b> object if the error is due to a timeout, otherwise throws an <b>UnsupportedOperationException</b> . Check <b>isTimeout()</b> before calling this function to ensure you get the correct result type.
<b>Note:</b> It is not necessary to call <b>getAsTimeout()</b> . If <b>isTimeout()</b> returns true, the result type is <b>Result.Error.Timeout</b> , so all of the properties and functions of a <b>Result.Error.Timeout</b> object are available on the original result reference.</td></tr>
</table></td></tr>
</table>

isCancelled()	Returns true if the error is due to user cancellation. Optionally call **getAsCancelled()** to get the **Result.Error** as a **Result.Error.Cancelled** object for specific details pertaining to the cancellation error.
getAsCancelled()	Returns the result as a **Result.Error.Cancelled** object if the error is due to user cancellation, otherwise throws an **UnsupportedOperationException** . Check **isCancelled()** before calling this function to ensure you get the correct result type. **Note:** It is not necessary to call **getAsCancelled()** . If **isCancelled()** returns true, the result type is **Result.Error.Cancelled** , so all of the properties and functions of a **Result.Error.Cancelled** object are available on the original result reference.

This object contains different properties depending on the cause of the authentication error:

Object	Description
Result.Error.Generic	An object representing a generic error authentication result. Call **getMessage()** to return a diagnostic message pertaining to the error, or null if such a message does not exist.
Result.Error.Timeout	An object representing a timeout error authentication result. Call **getTimeout()** to return the integer timeout value (in minutes) that that was used to configure the timeout.
Result.Error.Cancelled	An object representing an error result due to user cancellation. This object has no functions or properties.

Authentication Challenge Example

This example script will determine whether to proceed with a specified action after a completed Authentication Challenge Action. In this case, the action is permitted only if the second user's roles include 'supervisor', If the action is not permitted, or the authentication challenge result is an error, a message box will pop up with relevant context.

```
if result.isSuccess():
        # Parse information about the successful result
        success = True
        context = result.getAsSuccess().getContext()
        user = context.user
        roles = user.roles

        # Determine if the e-signature is valid:
        if 'supervisor' in roles:
                # If the user is a supervisor, proceed with the specified action:
                proceedWithSomeAction()

        else:
                # If the user is not a supervisor, write to a logger:
                logger = system.util.getLogger("Authentication Challenge")
                logger.info("%s is not authorized"% user.userName)

# If the result is an error, write to a logger:
else:
        success = False
        error = result.getAsError()

        # Change the content of the logger message based on the type of error:
        if error.isGeneric():
                message = error.getAsGeneric().getMessage()
                 logger = system.util.getLogger("Authentication Challenge")
                logger.info(message)

        elif error.isTimeout():
                timeout = error.getAsTimeout().getTimeout()
                logger = system.util.getLogger("Authentication Challenge")
                logger.info("Authentication challenge timed out after %d minutes."% timeout)
        elif error.isCancelled():
                 logger = system.util.getLogger("Authentication Challenge")
                logger.info("Authentication challenge cancelled.")

        else:
                logger = system.util.getLogger("Authentication Challenge")
                logger.info("Unknown error occurred.")
```

Native App Event Scripts

When the Perspective App is running on a mobile device, it enables users to use tools available on the device, such as GPS location data, the camera, or the accelerometer. The remaining Session events are designed specifically to handle the three Native App Actions.

Barcode

The scanned barcode action can make use of a mobile device's built in camera.

Barcode Scanned

Watch the Video

Arguments	Description
session	An object that references the project Session that called the Barcode Scanned event. Use this to identify the specific Session that scanned the barcode.

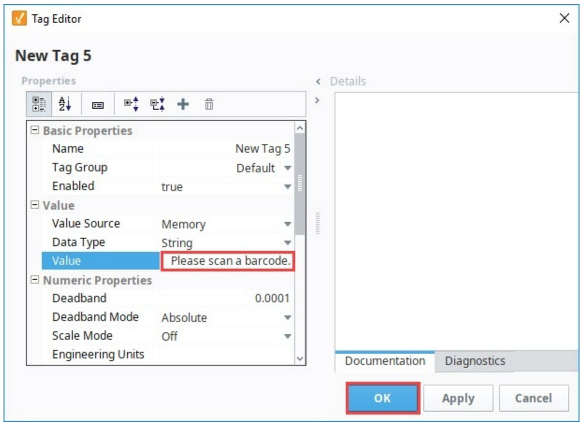
data	The data returned from the barcode scan. Access the underlying barcode data using:
	`# The value that was scanned`
	`data.text`
	`# An integer representing a unix timestamp of when the barcode was scanned`
	`data.timestamp`
	This feature is new in Perspective Mobile App version **0.98**
	Click here to check out the other new features
	Version 0.98 of the Perspective Mobile App added the following property:
	`# The type of barcode that was scanned.`
	`data.barcodeType`

context	The user defined context object that can be defined on the action.

Barcode Example

This example will scan a barcode, and write its value to a Tag.

1. For this example, drag a Button component and a Label component onto a view.
2. Create a new memory Tag with a data type of String. Set the value to "Please scan a barcode."


Tag Editor

New Tag 5

Basic Properties

Name	New Tag 5
Tag Group	Default
Enabled	true

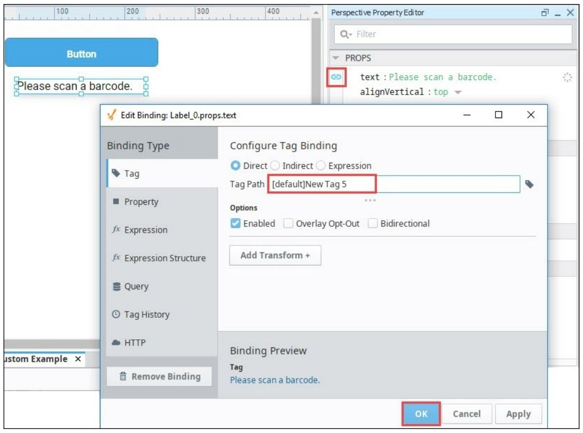
Value

Value Source	Memory
Data Type	String
Value	Please scan a barcode.

Numeric Properties

Deadband	0.0001
Deadband Mode	Absolute
Scale Mode	Off
Engineering Units	

3. Bind the text of the label to the new Tag.


Perspective Property Editor

PROPS

text : Please scan a barcode.
alignVertical : top

Edit Binding: Label_0.props.text

Binding Type: Tag

Configure Tag Binding

● Direct ○ Indirect ○ Expression

Tag Path [default]New Tag 5

Options

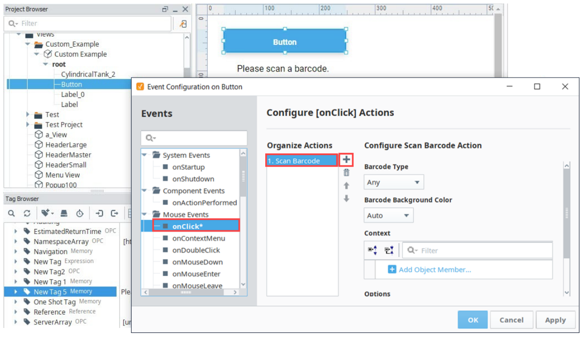
☑ Enabled □ Overlay Opt-Out □ Bidirectional

Binding Preview

Tag

Please scan a barcode.

4. Next right-click on the Button component and choose **Configure Events**.
5. On the Event Configuration screen, select **Mouse Events > onClick**.
6. Click the **Add** ✚ icon and select **Scan Barcode** action.


Event Configuration on Button

Configure [onClick] Actions

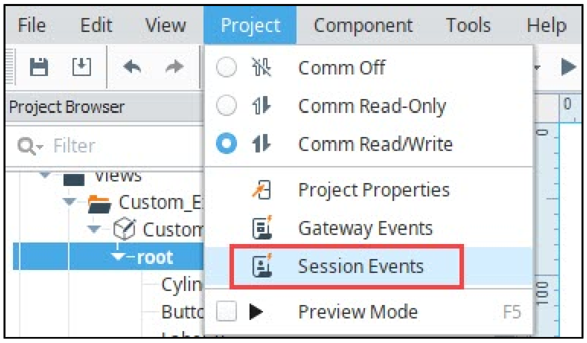
Organize Actions: 1. Scan Barcode

Configure Scan Barcode Action

Barcode Type: Any

Barcode Background Color: Auto

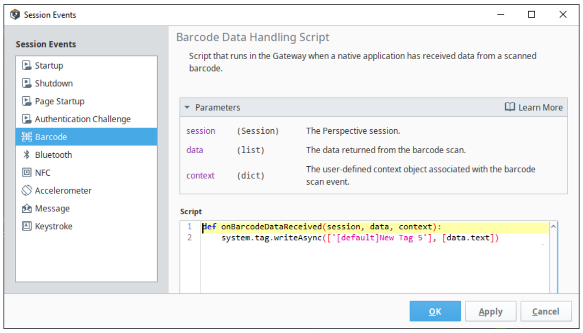
7. Click **OK**. Next we need to set up a session event so that Perspective knows how to interpret the scanned barcode data.
8. Under the **Project** tab, select **Session Events**.



9. On the Session Events screen, select the **Barcode** icon.
10. Add the following script to the page:

```
system.tag.writeAsync(['[default]New Tag 5'], [data.text])
```

> **Note:** We used the Tag created for this example, New Tag 5. You can enter your own Tag name if different.



Barcode Data Handling Script

Script that runs in the Gateway when a native application has received data from a scanned barcode.

session	(Session)	The Perspective session.
data	(list)	The data returned from the barcode scan.
context	(dict)	The user-defined context object associated with the barcode scan event.

Script

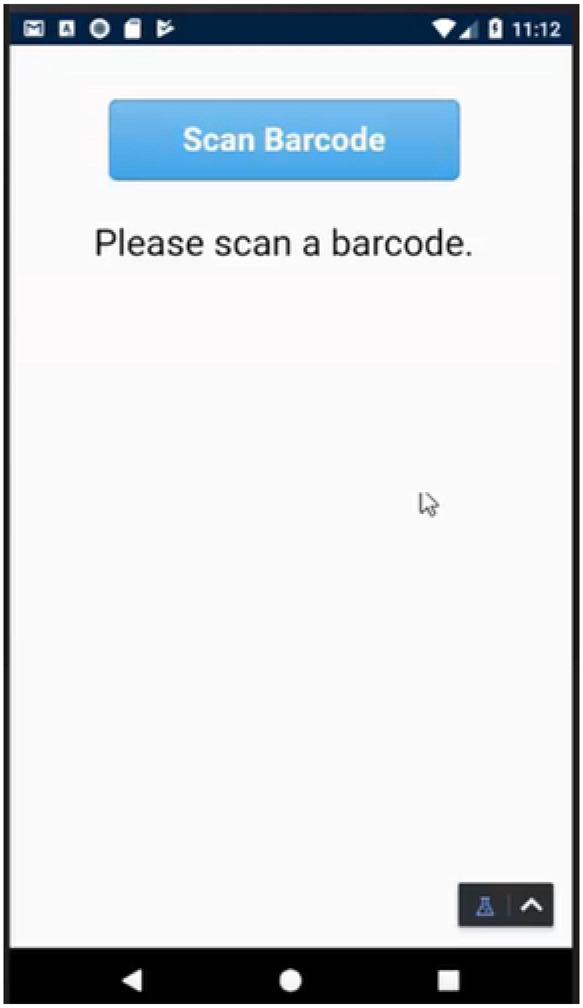
```
def onBarcodeDataReceived(session, data, context):
    system.tag.writeAsync(['[default]New Tag 5'], [data.text])
```

11. Click **OK** and save your project.

Test the Example

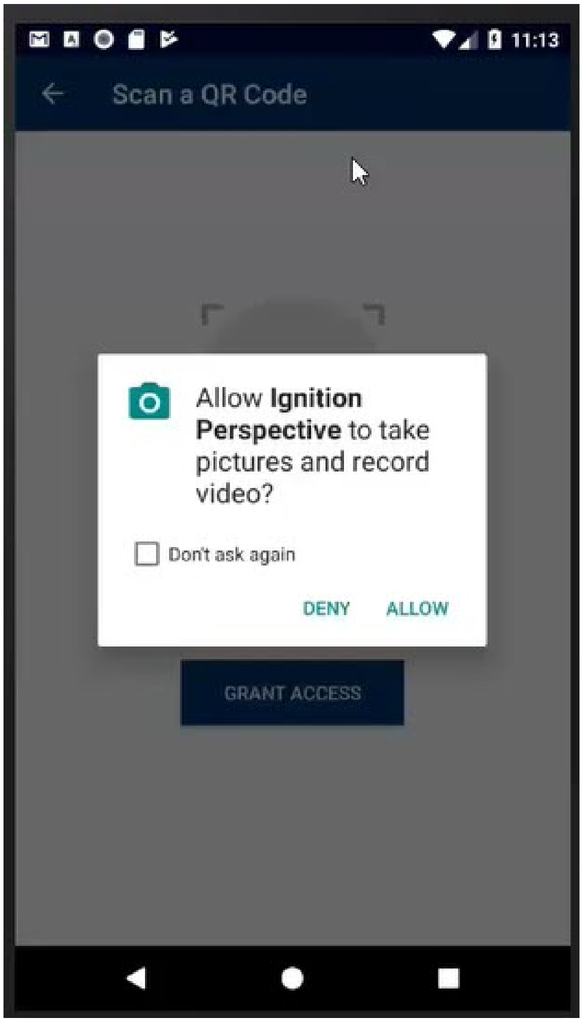
To test the example, open the Perspective App on your mobile device and load the project.

1. Click the **Scan Barcode** button.



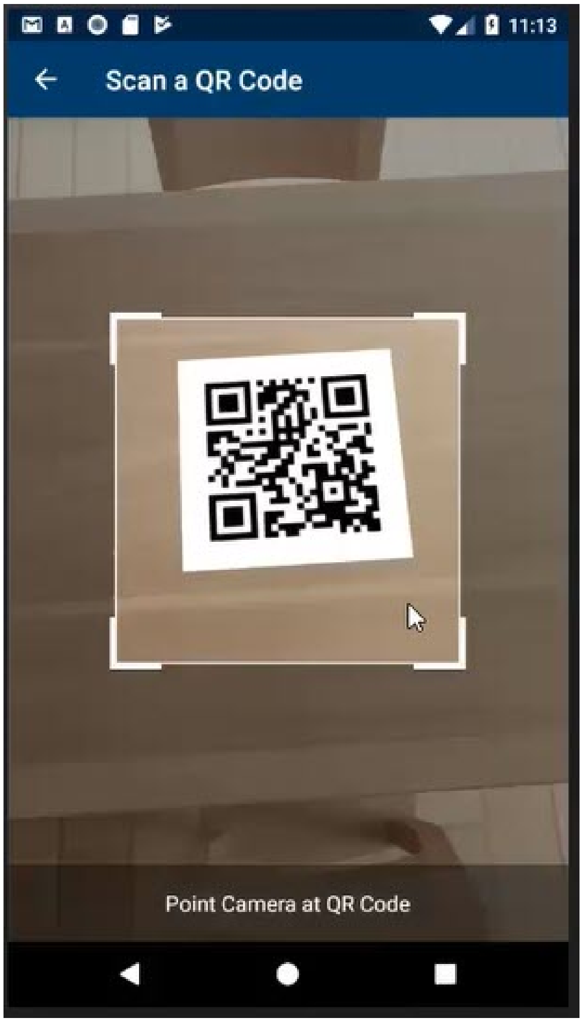
Please scan a barcode.

2. If this is the first time scanning a barcode, you'll get a message requesting permission for Perspective to take pictures and record.

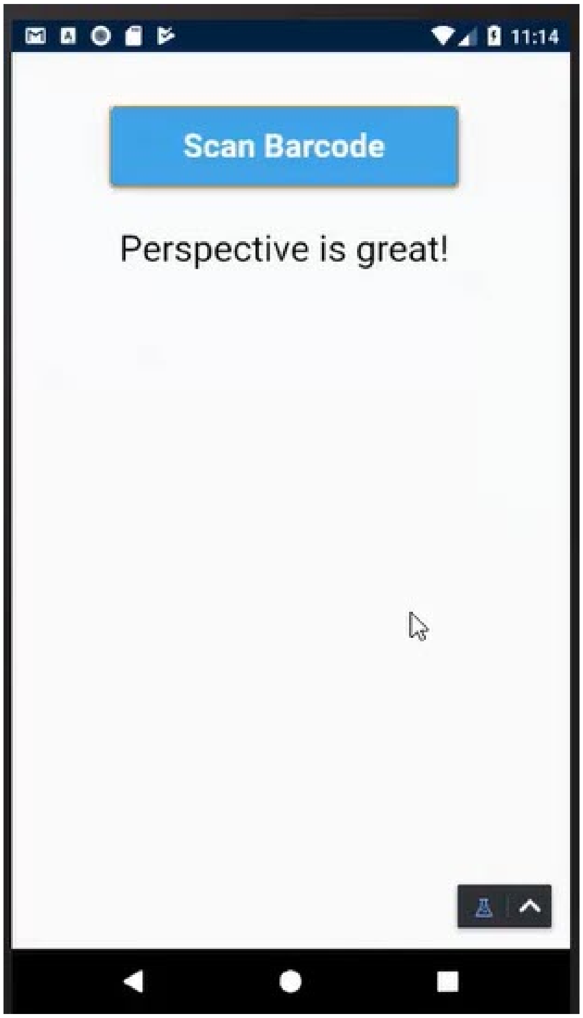


Allow **Ignition Perspective** to take pictures and record video?

3. Click **Allow**.
4. You can now use the camera on the mobile device to scan the barcode.


11:13
Scan a QR Code
Point Camera at QR Code

5. Ignition scans the barcode. Once it recognizes the barcode, the script will run and the text is written to the Tag. The label then shows the new Tag value.



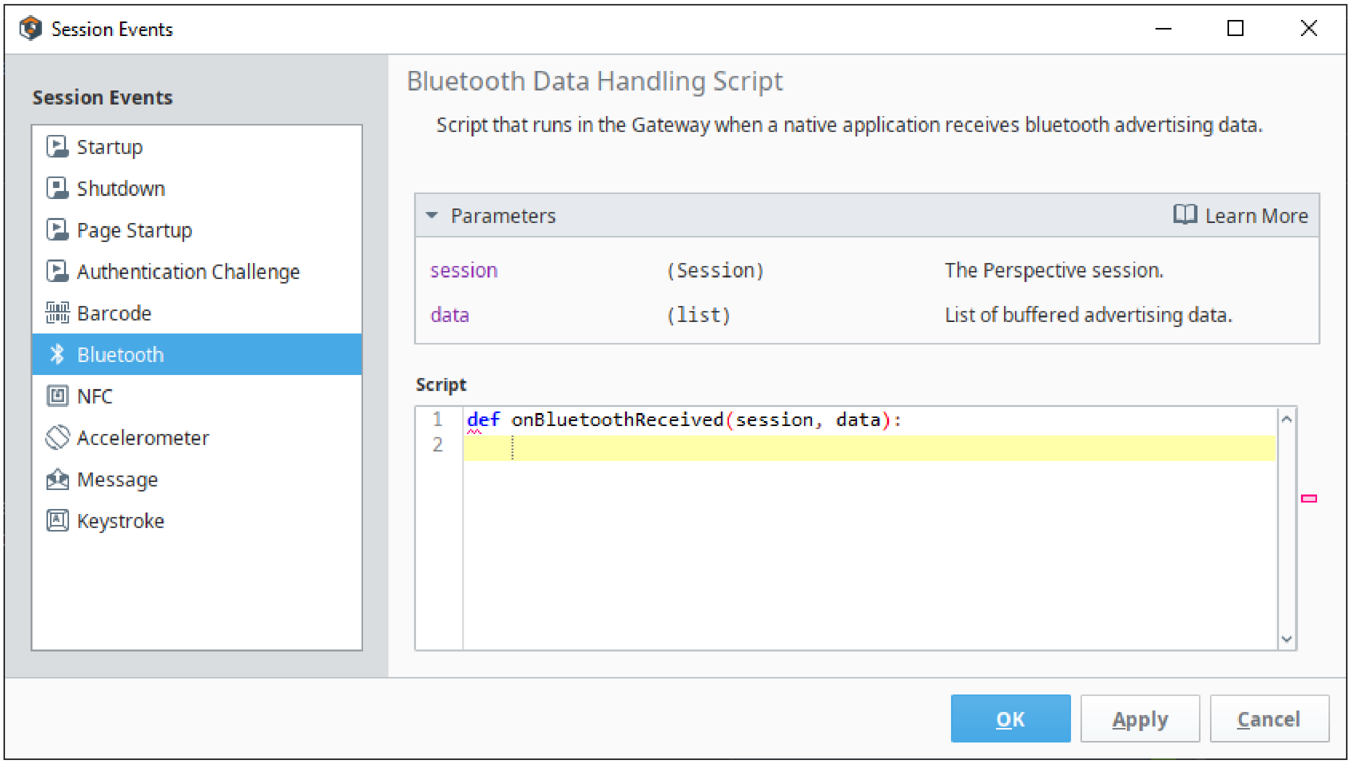
Perspective is great!

Bluetooth

The Bluetooth event is only used when a Session is running in a native application and it has received bluetooth advertising data. This Session event script sends Bluetooth advertising data to Perspective. It supports iBeacon and Eddystone formats.

- Eddystone will work on iOS and Android
- iBeacon on iOS requires user to specify the specific region (iBeacon UUID) located on Session props bluetooth.config.iBeaconRegion.

Note: Bluetooth "Advertising Data" is the name of the communication data according to the Bluetooth spec. There is no connection to advertising as an industry.



Bluetooth Data Handling Script

Script that runs in the Gateway when a native application receives bluetooth advertising data.

Parameters

session	(Session)	The Perspective session.
data	(list)	List of buffered advertising data.

Script

```
def onBluetoothReceived(session, data):

```

Arguments	Description
session	An object that references the Project Session
data	List of buffered advertising data. The data comes in as a data object, which has various parts. The following is example output.

```
{
        "values":
                [
                        {
                                "rssi":-48,
                                "timestamp":1570147485167,
                                "manufacturerData":{
                                        "companyId":6,
                                        "dataBase64Encoded":"AQkgAl_edccdnHClMvecMCiM--
aAkLHAPibd"
                                }
                        },
                        {
                                "rssi":-48,
                                "timestamp":1570147486012,
                                "serviceUUIDs":[
                                        "FEAA"
                                        ],
                                "serviceData":{
                                        "uuid":"FEAA",
                                        "dataBase64Encoded":"AOyqqqqqqqqqqqqqAAAAAAAA"
                                        },
                                "eddystoneUID":{
                                        "txPower":-20,
                                        "namespaceID":"AAAAAAAAAAAAAAAAAAAA",
                                        "instanceID":"000000000000"
                                        }
                        },
                        {
                                "rssi":-54,
                                "timestamp":1570147485919,
                                "manufacturerData":{
                                        "companyId":76,
                                        "dataBase64Encoded":"AhV88isfQjVLY4VnXXfYqpTyAAAAAL8="
                                        },
                                "iBeacon":{
                                        "uuid":"7CF22B1F-4235-4B63-8567-5D77D8AA94F2",
                                        "major":0,
                                        "minor":0,
                                        "txPower":-65
                                        }
                        },
                        {
                                "rssi":-54,
                                "timestamp":1570147485987,
                                "manufacturerData":{
                                        "companyId":65535,
                                        "dataBase64Encoded":"vqwSNFZ4EjQSNBI0EjRWeJASAAAAAOwA"
                                        },
                                "AltBeacon":{
                                        "manufacturerId":65535,
                                        "uuid":"12345678-1234-1234-1234-123456789012",
                                        "instance":"00000000",
                                        "txPower":-20,
                                        "manufacturerReserved":"00"
                                        }
                        }
        ]
}
```

Accelerometer

The Accelerometer event is only used when data is coming in from a batched Accelerometer Action.

<table>
<tr><th>Arguments</th><th>Description</th></tr>
<tr><td>session</td><td>An object that references the Project Session that called the Accelerometer Data Received event. Use this to identify the specific Session that triggered the batching of accelerometer data.</td></tr>
<tr><td>data</td><td>The data returned from the batched accelerometer. The data comes in as a data object, which has various parts. Access the various parts like:
<pre>
logger = system.util.logger('accelerometer')
for row in data.values.data:
        logger.info('X:' + str(row['x']))
        logger.info('Y:' + str(row['y']))
        logger.info('Z:' + str(row['z']))
</pre></td></tr>
<tr><td>context</td><td>The user defined context object that can be defined on the action.</td></tr>
</table>

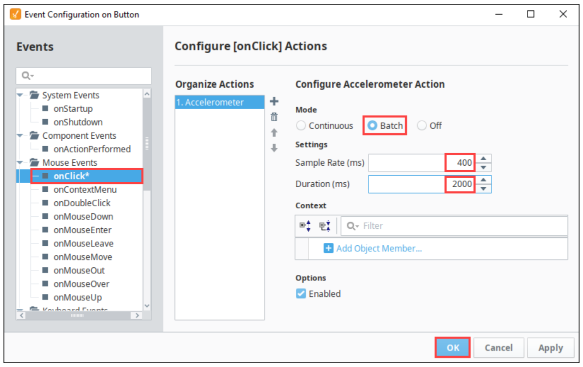
Accelerometer Data Received

Watch the Video

Accelerometer Example

This example will write accelerometer data to the Gateway logs.

1. For this example, put a Button component onto a view.
2. Next right-click on the Button component and choose **Configure Events**.
3. On the Event Configuration screen, select **Mouse Events > onClick**.
4. Click the **Add** ✚ icon and select **Accelerometer** action.
   a. Select **Batch** mode.
   b. Set a **Sample Rate** of 400.
   c. Set a **Duration** of 2000.



Mode

○ Continuous ● Batch ○ Off

Settings

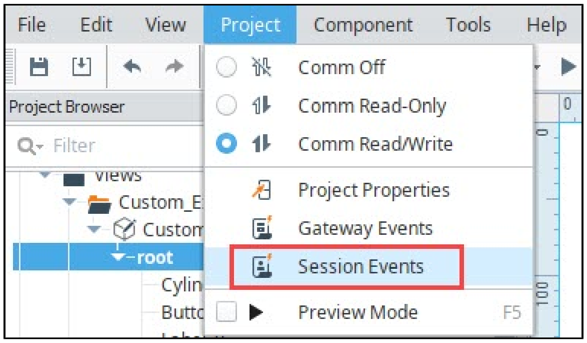
Sample Rate (ms) 400

Duration (ms) 2000

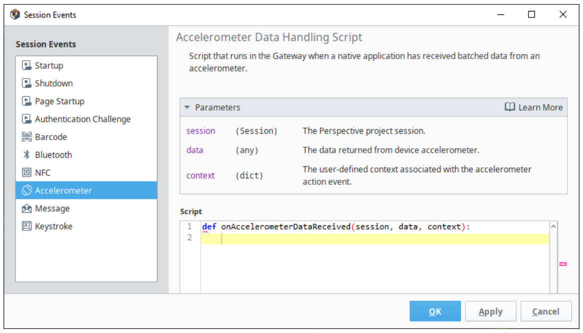
Options

☑ Enabled

5. Click **OK**. Next we need to set up a Session Event so that Perspective knows how to interpret the accelerometer data.
6. Under the **Project** tab, select **Session Events**.



7. On the **Session Events** screen, click the **Accelerometer** icon.



Accelerometer Data Handling Script

Script that runs in the Gateway when a native application has received batched data from an accelerometer.

session	(Session)	The Perspective project session.
data	(any)	The data returned from device accelerometer.
context	(dict)	The user-defined context associated with the accelerometer action event.

```
def onAccelerometerDataReceived(session, data, context):
    
```

8. Add the following script to the page:

```
# This script will take the accelerometer data and print it to the
Gateway logs.

# Create the logger.
logger = system.util.logger('accelerometer')

# Loop through the list of batched events and pull out the x, y, and
z values to print to the Gateway logs.
for row in data.values.data:
        logger.info('X:' + str(row['x']) + ', Y:' + str(row['y']) +
', Z:' + str(row['z']))
```

9. Click **OK**.
10. Save your project.

Test the Example

To test the example, open the Perspective App on your mobile device and load the project.

1. Click the Accelerometer button.
2. After clicking the button, the script will record accelerometer data for the next two seconds, so try moving the phone around.
3. After the two seconds, you should see the logged information appear in the Gateway logs.

NFC

The NFC event is used when the NFC Action is used and the mobile device scans an NFC Tag.

INDUCTIVE UNIVERSITY

NFC Ndef Scanned

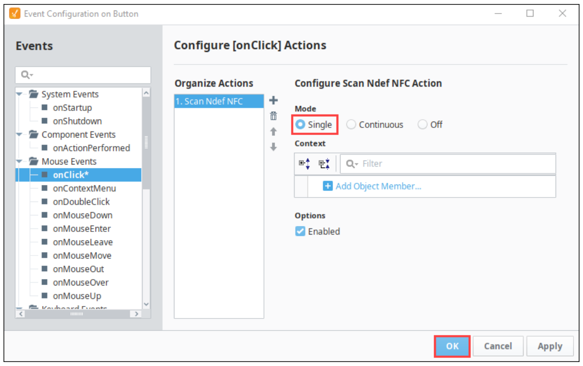
Watch the Video

<table>
<tr><th>Arguments</th><th>Description</th></tr>
<tr><td>session</td><td>An object that references the Project Session that called the NFC Ndef Scanned event. Use this to identify the specific Session that scanned the NFC Tag.</td></tr>
<tr><td>data</td><td>The data returned from the NFC Tag scan. The data object is a list which can contain multiple records from a single NFC Tag. Access the underlying NFC data using:
<pre>
logger = system.util.logger('NFC')
for row in data:
        logger.info('Type:' + str(row['type']))
        logger.info('Type Name Format:' + str(row
['typeNameFormat']))
        logger.info('Payload:' + str(row['payload']))
        logger.info('String:' + str(row['string']))
        logger.info('Bytes:' + str(row['bytes']))
</pre></td></tr>
<tr><td>context</td><td>The user defined context object associated with the NFC scan event.</td></tr>
</table>

NFC Example

This example will write the NFC Tag data to the Gateway logs.

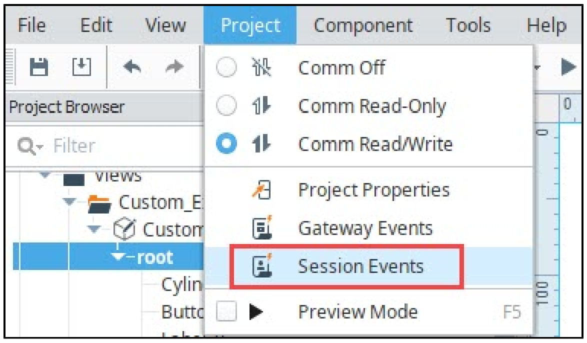
1. For this example, put a Button component onto a view.
2. Next right-click on the Button component and choose **Configure Events**.
3. On the Event Configuration screen, select **Mouse Events > onClick**.
4. Click the **Add** ✚ icon and select **Scan Ndef NFC** action.
5. Select **Single** mode, then click **OK**.



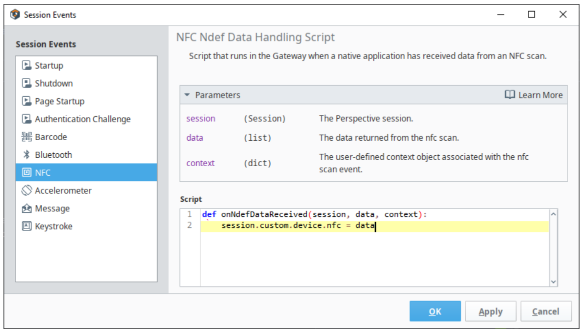
Mode
● Single ○ Continuous ○ Off

Options
☑ Enabled

6. Next we need to set up a Session Event so that Perspective knows how to interpret the NFC data. Under the Project tab, select **Session Events**.



7. On the Session Events screen, click the **NFC** ▣ icon.



NFC Ndef Data Handling Script

Script that runs in the Gateway when a native application has received data from an NFC scan.

Parameters

session	(Session)	The Perspective session.
data	(list)	The data returned from the nfc scan.
context	(dict)	The user-defined context object associated with the nfc scan event.

Script

```
def onNdefDataReceived(session, data, context):
    session.custom.device.nfc = data
```

8. Add the following script to the page:

```
# This script will take the NFC data and print it to the Gateway
logs.

# Create the logger.
logger = system.util.logger('NFC')

# Loop through the list of records stored in the NFC Tag and pull
out the type, type name format, payload, string data, and raw byte
data from each record and print it to the Gateway logs.
for row in data:
        logger.info('Type:' + str(row['type']) + ', Type Name
Format:' + str(row['typeNameFormat']) + ', Payload:' + str(row
['payload']) + ', String:' + str(row['string']) + ', Bytes:' + str
(row['bytes']))
```

9. Click **OK**.
10. Save your project.

Test the Example

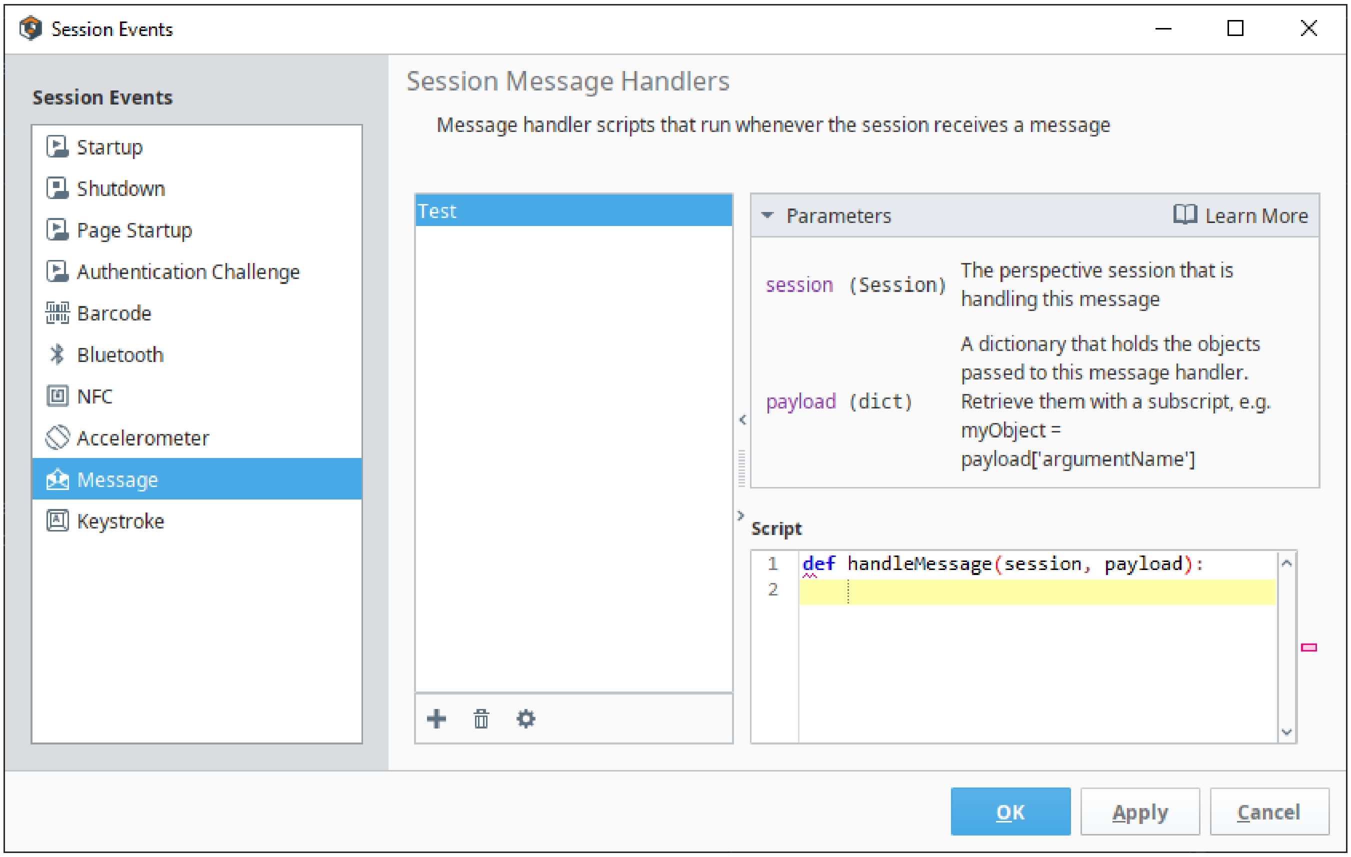
To test the example, open the Perspective App on your mobile device and load the project.

1. Click the **NFC button**.
2. After clicking the button, the script will pass the next NFC Tag scanned to the script to be handled.
3. After scanning an NFC Tag, you should see the logged information appear in the Gateway logs.

Message

The Message Handler scripts will run whenever the Session receives a message from system.util.sendMessage or system.util.sendRequest. Note that these types of message handlers are different than component-based message handlers, which are accessed with system.perspective.sendMessage. Session Message Handlers can not be called by system.perspective.sendMessage.

✓ An efficient way to use this message handler in cases with user driven operations is by incorporating system.perspective.getSessionInfo to lookup all sessions and then filter the sessions to target specific users.



Message handler scripts that run whenever the session receives a message

session	(Session)	The perspective session that is handling this message
payload	(dict)	A dictionary that holds the objects passed to this message handler. Retrieve them with a subscript, e.g. myObject = payload['argumentName']

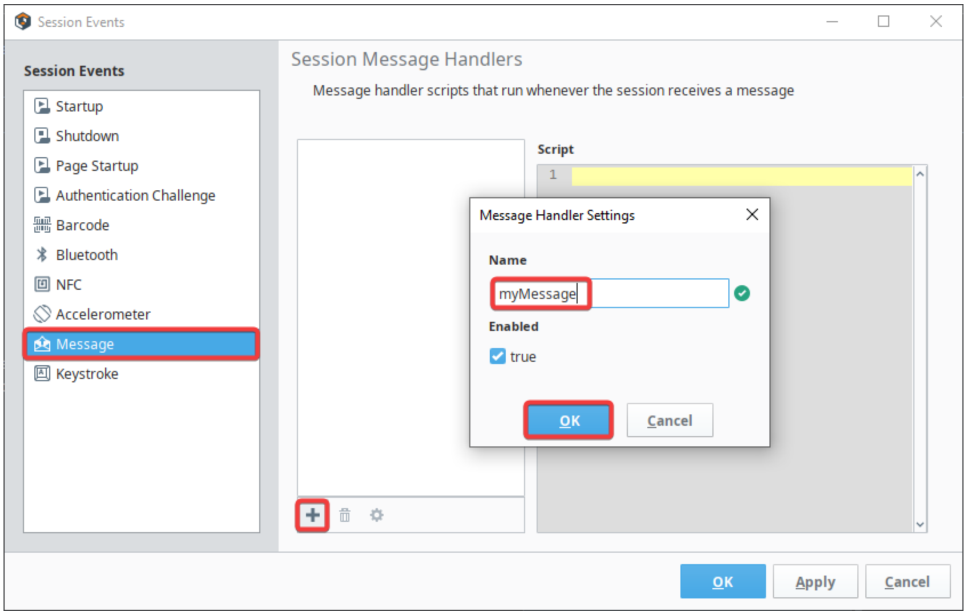
```
def handleMessage(session, payload):

```

Arguments	Description
session	The Perspective Session that is handling this message.
payload	A dictionary that holds the objects passed to this message handler. Retrieve them with a subscript, e.g., `myObject = payload['argumentName']`

Message Example

This example shows how to set up a message handler script to send a simple log message to the Gateway. We will use a button component to run the system.util.sendMessage command.

1. Under the Project tab, select **Session Events.**
2. On the Session Events screen, click the **Message** icon.
3. Click the **Add** ✚ **icon.**
4. Enter **myMessage** in the Message Handler Settings Name field.
5. Click **OK**.

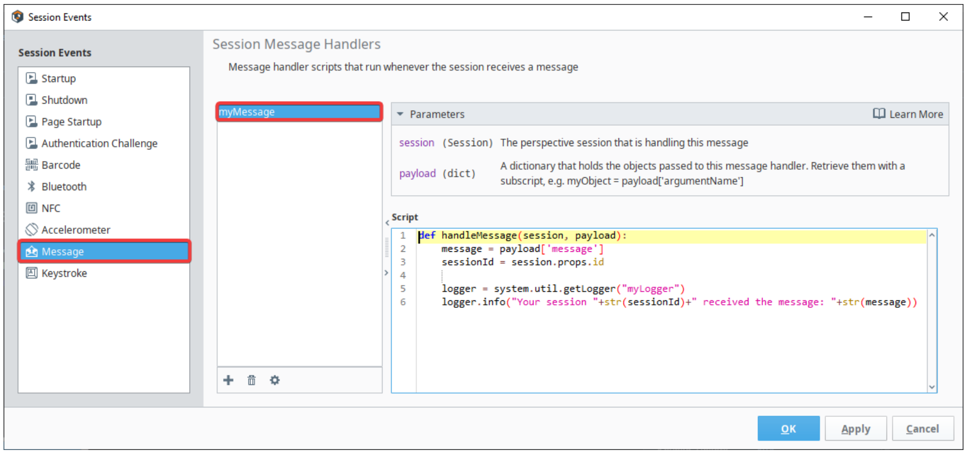


Name: myMessage

Enabled: ☑ true

6. We will first define the define the message and assign the sessionId to our current Perspective session. Then, we will use Logger functions to return a logger object to log message with level info. Add the following script to the page to complete these functions:

```
def handleMessage(session, payload):
        message = payload['message']
        sessionId = session.props.id

        logger = system.util.getLogger("myLogger")
        logger.info("Your session "+str(sessionId)+" received the message: "+str(message))
```



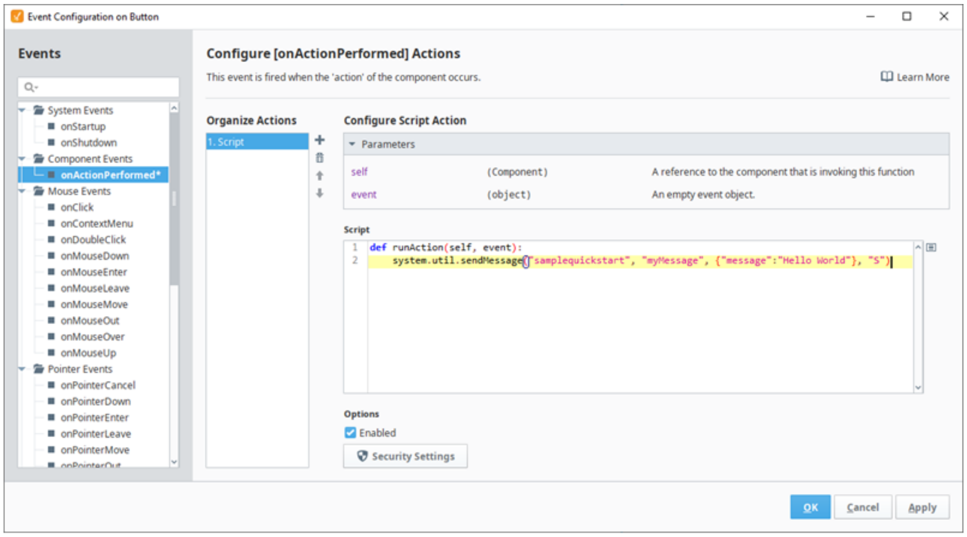
session (Session) The perspective session that is handling this message

payload (dict) A dictionary that holds the objects passed to this message handler. Retrieve them with a subscript, e.g. myObject = payload['argumentName']

```
def handleMessage(session, payload):
    message = payload['message']
    sessionId = session.props.id

    logger = system.util.getLogger("myLogger")
    logger.info("Your session "+str(sessionId)+" received the message: "+str(message))
```

7. Click **OK**.
8. Add a Button component onto a view.
9. Right-click on the Button component and choose **Configure Events**.
10. On the Event Configuration screen, select **Component Events > onActionPerformed**.
11. Click the **Add** + icon and select **Script** action.
12. For this example, we will use system.util.sendMessage, which means we need to add our project name, the message handler we created, the payload message we want to send, and limit the scope to Session. Add the following script to the page:

```
def runAction(self, event):
        system.util.sendMessage("samplequickstart", "myMessage", {"message":"Hello World"}, "S")
```



Configure [onActionPerformed] Actions

This event is fired when the 'action' of the component occurs.

Configure Script Action

Parameters

self	(Component)	A reference to the component that is invoking this function
event	(object)	An empty event object.

Script

```
def runAction(self, event):
	system.util.sendMessage("samplequickstart", "myMessage", {"message":"Hello World"}, "S")
```

Options

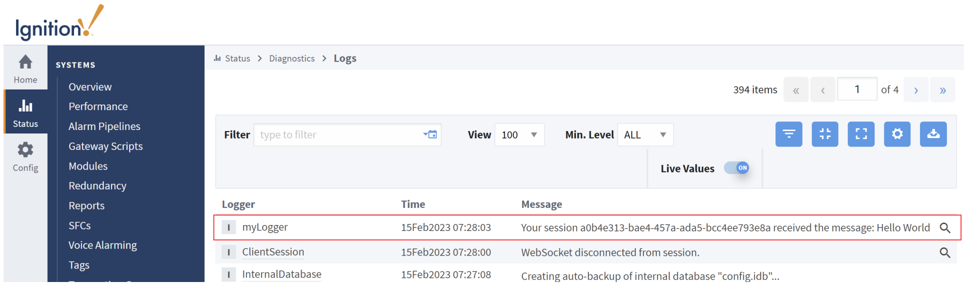
☑ Enabled

13. Click **OK**.
14. Save your project.

Test the Example

To test the example, launch your Perspective Session and click the button component we created.

1. After clicking the button, the script will send the "Hello World!" message to your session id.
2. Access your Gateway Status > Diagnostics > Logs page to confirm the message was received.



Logger	Time	Message
I myLogger	15Feb2023 07:28:03	Your session a0b4e313-bae4-457a-ada5-bcc4ee793e8a received the message: Hello World
I ClientSession	15Feb2023 07:28:00	WebSocket disconnected from session.
I InternalDatabase	15Feb2023 07:27:08	Creating auto-backup of internal database "config.idb"...

Keystroke

The following feature is new in Ignition version **8.1.10**
Click here to check out the other new features

Keystroke scripts will run when the session detects a KeyboardEvent matching either a single key or a regex pattern. A single key match can be made on either a key press (keyDown) or on a key release (keyUp). A regex match is always made on a key release. Key Event scripts can be triggered by keyboards and any other device that is capable of generating a KeyboardEvent, such as an HID barcode scanner.

Keystroke detection is only active inside of a Perspective session and is not active inside of the Designer.

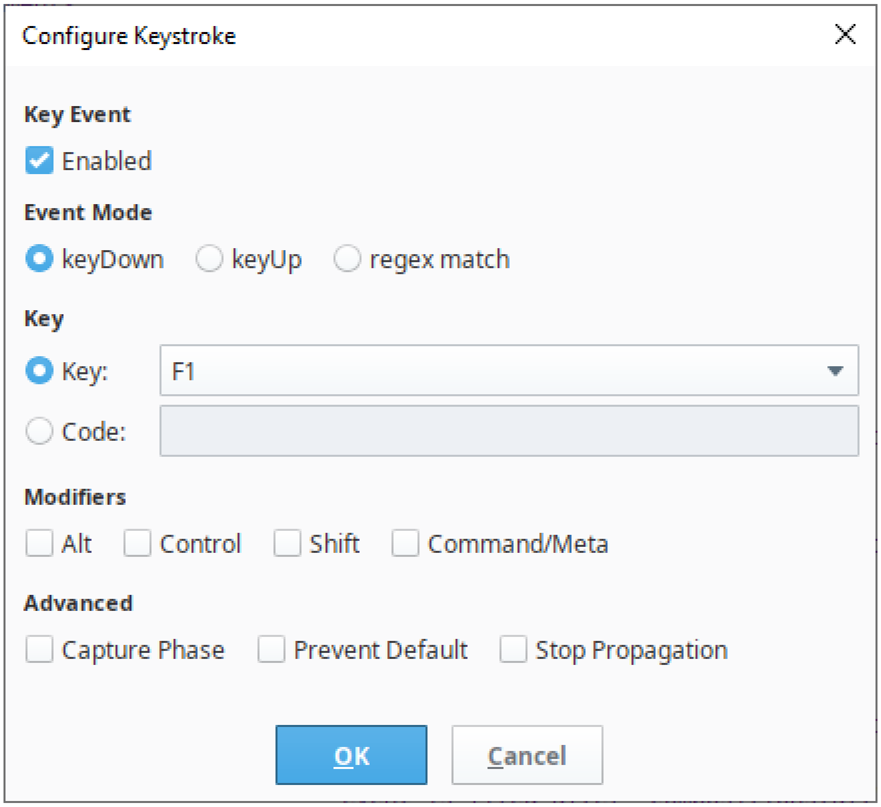
Note: The device being used to trigger key event scripts must be able to generate a KeyboardEvent. HID barcode scanners will generate KeyboardEvents because they mimic a user typing keys on a physical keyboard. Scanners and other input devices that do not create KeyboardEvents will not trigger this logic.

<table>
<tr><th>Arguments</th><th>Description</th></tr>
<tr><td>page</td><td>An object that contains information about the current page.
<table>
<tr><td>props</td><td>Properties of the "page" object.</td></tr>
<tr><td>session</td><td>An object that contains information about the current session.</td></tr>
</table></td></tr>
<tr><td></td><td></td></tr>
</table>

<table>
<tr><td>event</td><td>An object that contains information from the Key Event.
<table>
<tr><td>altKey</td><td>Returns a boolean that is true if the Alt (Option or ? on OS X) key was active when the key event was generated.</td></tr>
<tr><td>code</td><td>Returns a string with the code value of the physical key represented by the event.</td></tr>
<tr><td>ctrlKey</td><td>Returns a boolean that is true if the Ctrl key was active when the key event was generated.</td></tr>
<tr><td>isComposing</td><td>Returns a boolean that is true if the event is fired after "compositionstart" and before "compositionend".</td></tr>
<tr><td>key</td><td>Returns a string representing the key value of the key represented by the event.</td></tr>
<tr><td>location</td><td>Returns a number representing the location of the key on the keyboard or other input device.</td></tr>
<tr><td>metaKey</td><td>Returns a boolean that is true if the Meta key (on Mac keyboards, the ? Command key; on Windows keyboards, the Windows key) was active when the key event was generated.</td></tr>
<tr><td>repeat</td><td>Returns a boolean that is true if the key is being held down such that it is automatically repeating.</td></tr>
<tr><td>shiftKey</td><td>Returns a boolean that is true if the Shift key was active when the key event was generated.</td></tr>
<tr><td>matches</td><td>Returns a list where the first entry of the list is the exact text which registered as a match for the regex, and all subsequent entries in the list are the content which fell into capture groups.</td></tr>
</table></td></tr>
</table>

Single Key

For single key matches, the match can be a physical key typed or the code generated by the key. **Modifiers** are additional keys that can be held down during a KeyboardEvent and are detectable in the event data. These include Alt, Control, Shift, and Command/Meta.



Configure Keystroke

Key Event

☑ Enabled

Event Mode

● keyDown ○ keyUp ○ regex match

Key

● Key: F1

○ Code:

Modifiers

□ Alt □ Control □ Shift □ Command/Meta

Advanced

□ Capture Phase □ Prevent Default □ Stop Propagation

In addition, the resulting Key Event that is used for match detection has the following advanced features:

- **Capture Phase** - Allows execution during event capturing, which will let the resulting script execute *before* other interface events that may be in place. By default, events execute during an event bubbling phase which is dictated by the position of the event binding in the Document Object Model (DOM). In the case of Key Events, these are bound to the `document` layer of the DOM and will execute *after* other interface events if this option is unchecked.
- **Prevent Default** - Prevents any default browser functionality that would be associated with the event binding. This can be toggled on if there is any functionality that *should not* be run when a `keyDown/keyUp` event has been fired at the `document` level when a key match is made.

- **Stop Propagation** - Prevents the event from propagating beyond the point in the DOM where it is bound (the `document` level). This can be useful if combined with the *Capture Phase* option as it will prevent any interface `keyDown/keyUp` events from running that are bound deeper in the DOM than the `document` level.

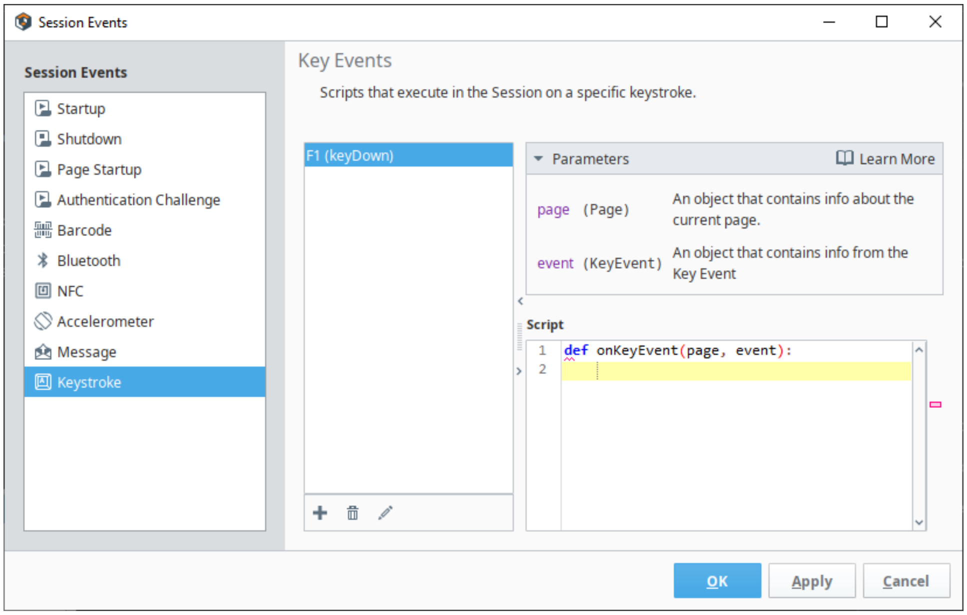
Regex Match

For a regular expression (regex) pattern match, a match will be based on a supplied regex pattern and a series of keys stored in a match buffer. This buffer will always be the most recently typed characters and should be the length of a "correct match" to the supplied regex pattern.

Regex Example

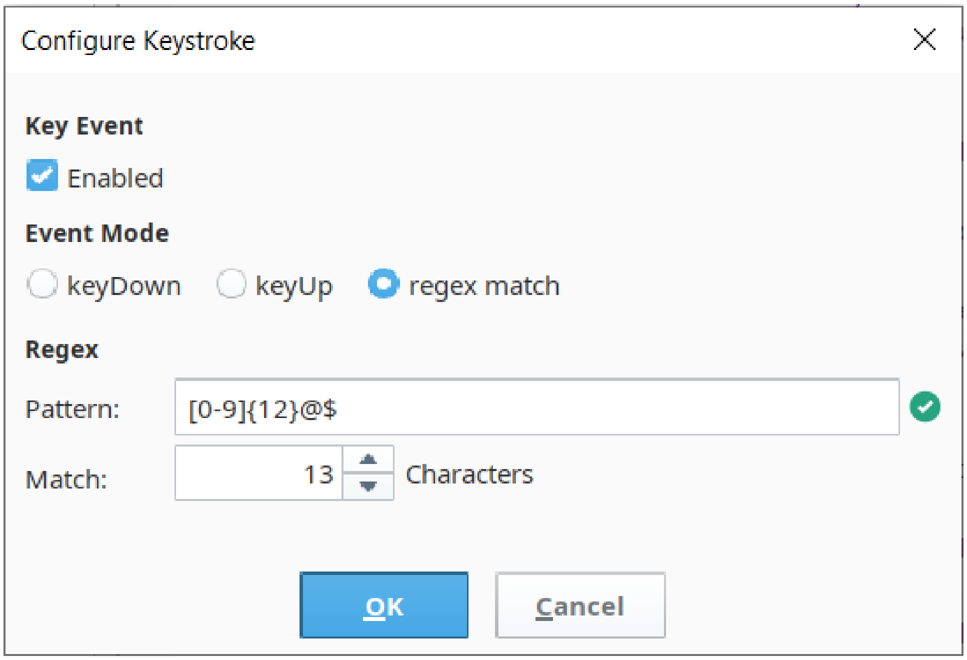
This example will update a custom Session property when a regex pattern match is found.

1. For this example, place a Label component onto a view.
2. Add a custom Session property called `scannerInput`.
3. Bind the text value of the Label to the custom Session property.
4. Under the **Project** tab, select **Session Events**.
5. On the **Session Events** screen, select **Key Events.**



```
def onKeyEvent(page, event):
```

6. Click on the **Add** ➕ icon and select **regex match**.



Configure Keystroke

Key Event

☑ Enabled

Event Mode

○ keyDown ○ keyUp ● regex match

Regex

Pattern: [0-9]{12}@$

Match: 13 Characters

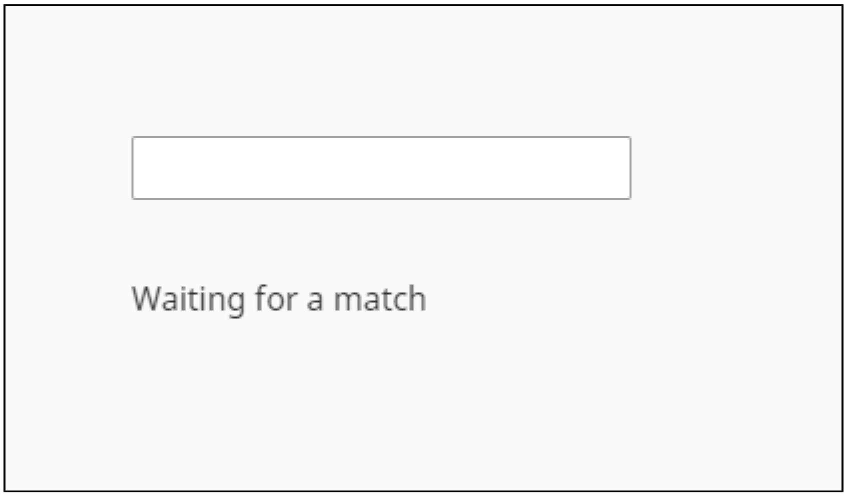
7. Enter the regex pattern you want the Session to watch for. In the above example, a match occurs for any sequence of 12 numeric characters followed by an "@"
8. Enter the length of the match buffer. The match buffer should equal the minimum number of characters required for a pattern match.
9. Add the following script:

```
page.session.custom.scannerInput = "Match found"
```

10. Click **OK**.
11. Save your project.

Test the Example

To test the example, launch a Perspective session and open the view containing your Label. Type a string of characters matching the regex pattern you configured above. You should see the Label's text change:



Security in Perspective

Security in Perspective is managed through Identity Providers (IdP). IdPs offers a way for users to log in to Ignition using credentials stored outside of Ignition. This level of security is set up through the Gateway. Setting up Security is covered in the Security section of the User Manual.

Once you have an IdP set up as well as Security Levels , Security Level Rules , and User Grants there are additional ways to control security for the following:

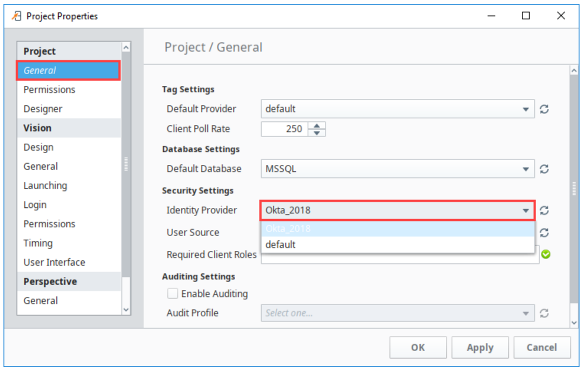
- Perspective Sessions
- Perspective Views
- Event actions on Perspective components

On this page ...

Perspective Sessions Security

For each Project, you can set the security for an associated Perspective Session. When you select the s ecurity levels, you are granting any user with that security level access to the Perspective Session for that Project.

1. In the Designer, select the **Project Properties** on the Project menu. Select Project > General.
2. In the Identity Provider field, use the dropdown to select the IdP you want to use or to select the default user source.



Project / General

Tag Settings
Default Provider: default
Client Poll Rate: 250

Database Settings
Default Database: MSSQL

Security Settings
Identity Provider: Okta_2018
Okta_2018
default
User Source
Required Client Roles

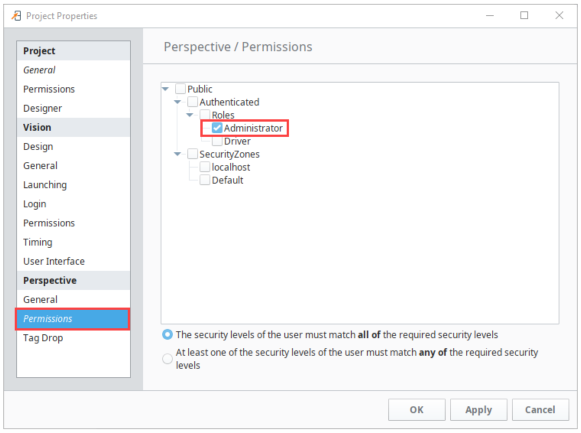
Auditing Settings
☐ Enable Auditing
Audit Profile: Select one...

3. Scroll down to select **Perspective** > **Permissions**.
4. Expand the tree to view the security levels you want to be able to access this project in a Perspective Session.
5. Click the check box next to each of the security levels you want to grant access.

INDUCTIVE UNIVERSITY

Requiring Authentication

Watch the Video



- □ Public
  - □ Authenticated
    - □ Roles
      - ☑ Administrator
      - □ Driver
  - □ SecurityZones
    - □ localhost
    - □ Default

● The security levels of the user must match **all of** the required security levels

○ At least one of the security levels of the user must match **any of** the required security levels

6. Click **OK** to save all of the Project Properties changes.

Perspective Views Security

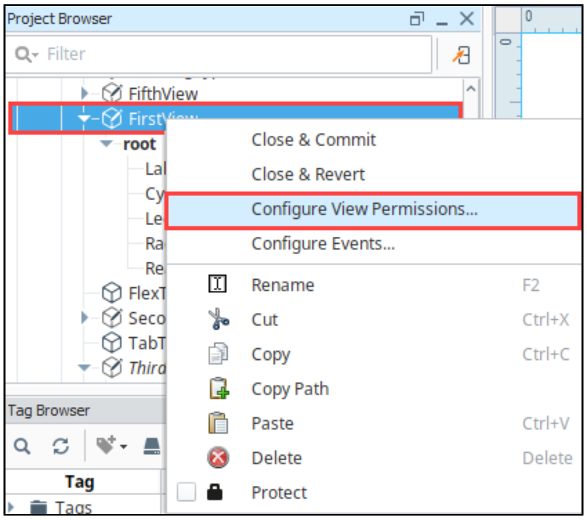
View Security

Watch the Video

You can set the security for an individual View in Perspective. When you select the security levels, you are granting any user with that access to the Perspective Session for that Project.

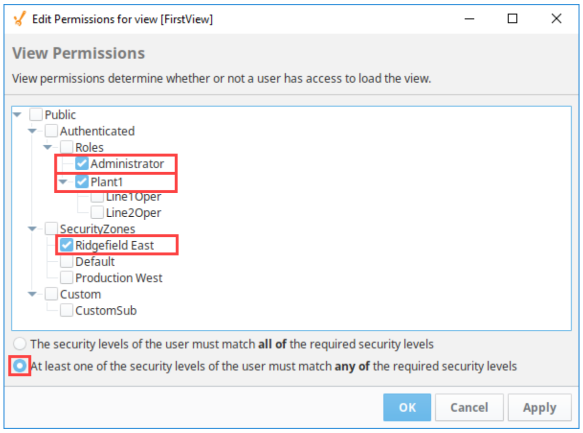
Note: Note that you must have the IdP selected in Project Properties > Project General.

1. In the Project Browser, right-click on the view and select **Configure View Permissions...**



2. On the Edit Permissions screen, click the check box next to the Roles that will be able to access this View.
3. Next, click the check box next to the Security Zones that will be able to access this View.
4. Finally, choose one of the radio buttons at the bottom of the screen to indicate whether the user much match **all** of the required security levels you've checked or if matching at least one is enough.

   In the example below, a user must have either the Administrator security level, Plant1 security level, or be in Ridgefield East to access this View.



Edit Permissions for view [FirstView]

View Permissions

View permissions determine whether or not a user has access to load the view.

- □ Public
  - □ Authenticated
    - □ Roles
      - ☑ Administrator
      - ☑ Plant1
        - □ Line1Oper
        - □ Line2Oper
    - □ SecurityZones
      - ☑ Ridgefield East
      - □ Default
      - □ Production West
    - □ Custom
      - □ CustomSub

○ The security levels of the user must match **all of** the required security levels

● At least one of the security levels of the user must match **any of** the required security levels

OK | Cancel | Apply

5. Click **OK** to save the permissions for this View.

Event Actions on Perspective Components

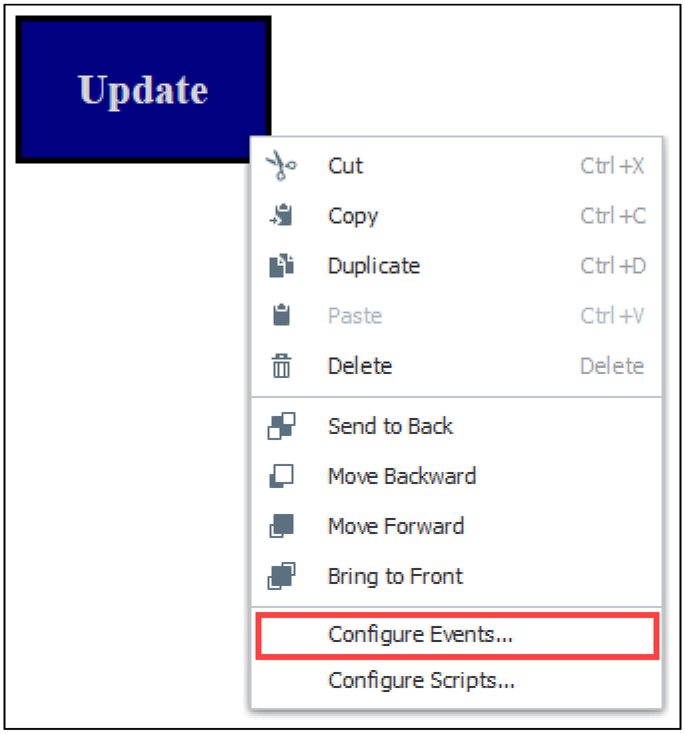
All Perspective components can have event scripts. These are scripts that run on an action, such as when the user clicks with the mouse on a component. For more information about event scripts see, Perspective scripting. Security can be configured on events. In the following example, set security for the action of clicking on a Button component in the Perspective View.

Note: Note that you must have the IdP selected in Project Properties > Project General.

Script Action Security

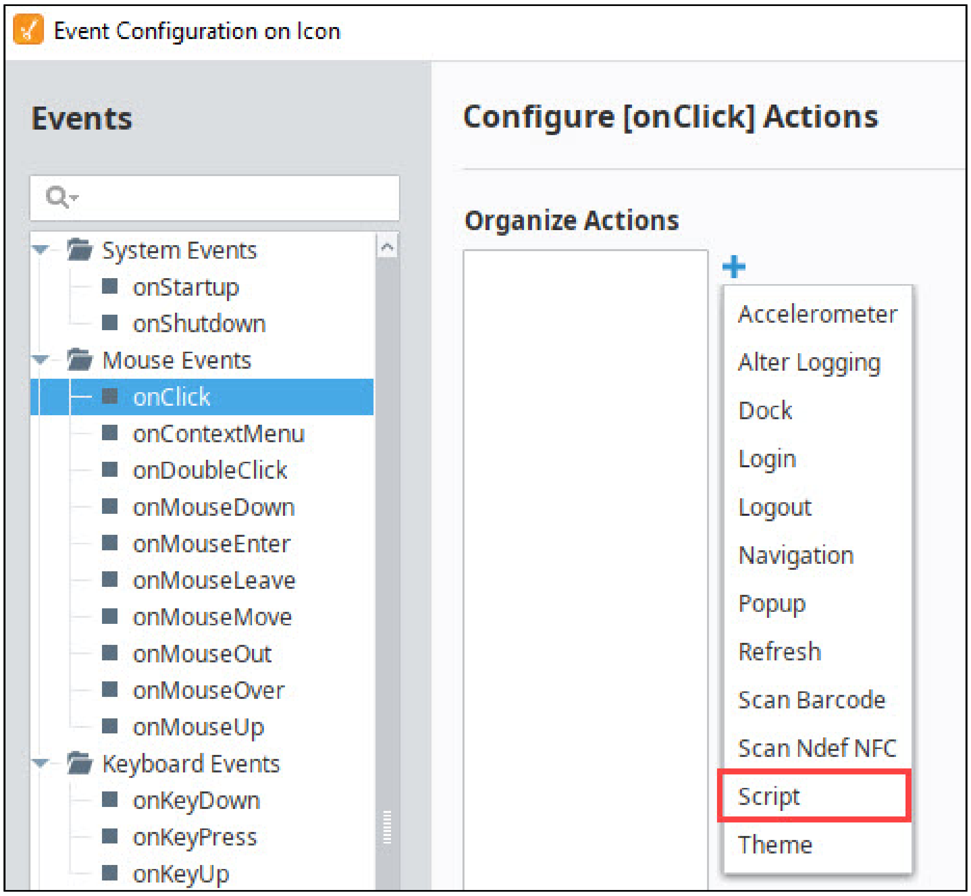
Watch the Video

1. To add security to an event on a component, right-click on the component then choose **Configure Events...**

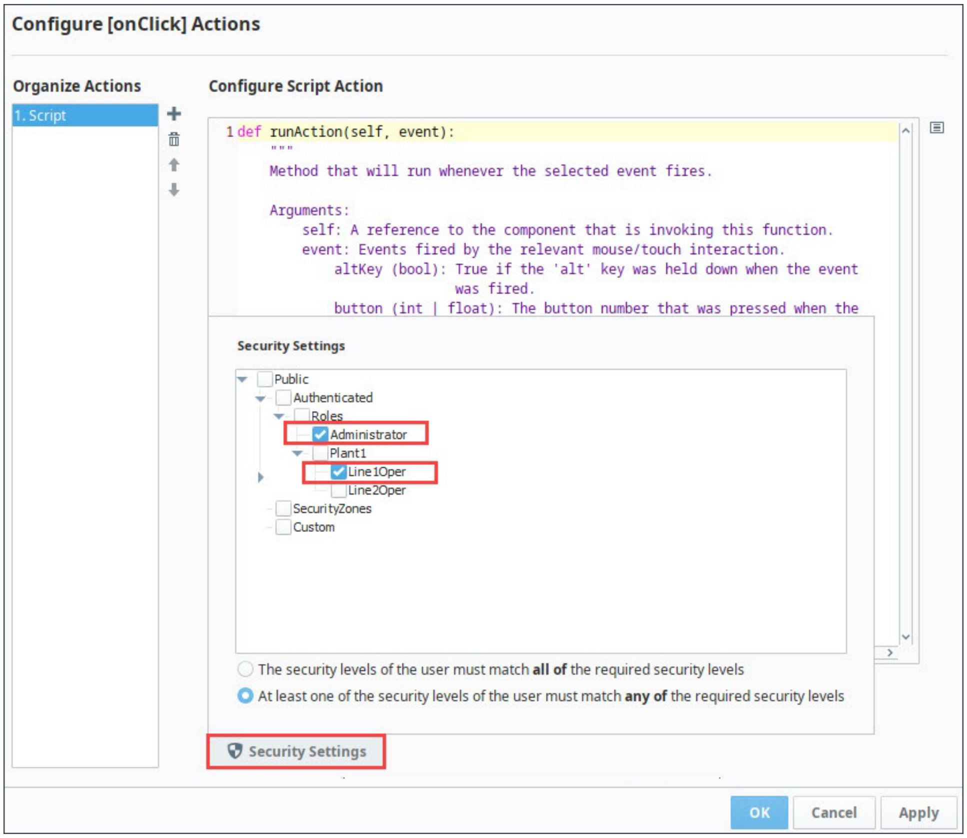


2. The Events Configuration screen is displayed. Many different types of events can be set for a component. For this example, choose **Mouse Events** > **onClick**.

3. Under **Organize Actions**, click the Add + icon, then select **Script** from the list.



4. Click the **Security Settings** icon near the bottom of the screen.
5. Click the check box next to the security levels you want to grant access. In the example, we checked Administrator and Line1Oper for the security Roles. Then, we chose the requirement option that at least one of the security levels for the user must match for access. This means anyone with Administrator or Line1Oper security levels will have permission to run the script associated with the **onClick** event on this button.



Configure [onClick] Actions

Organize Actions

1. Script

Configure Script Action

```
def runAction(self, event):
    """
    Method that will run whenever the selected event fires.

    Arguments:
        self: A reference to the component that is invoking this function.
        event: Events fired by the relevant mouse/touch interaction.
            altKey (bool): True if the 'alt' key was held down when the event
                           was fired.
            button (int | float): The button number that was pressed when the
```

Security Settings

- □ Public
  - □ Authenticated
    - □ Roles
      - ☑ Administrator
      - □ Plant1
        - ☑ Line1Oper
        - □ Line2Oper
  - □ SecurityZones
  - □ Custom

○ The security levels of the user must match **all of** the required security levels

● At least one of the security levels of the user must match **any of** the required security levels

6. Click the **Security Settings** icon to close the window, then click **OK**.

Using Component Bindings to Check Authorization

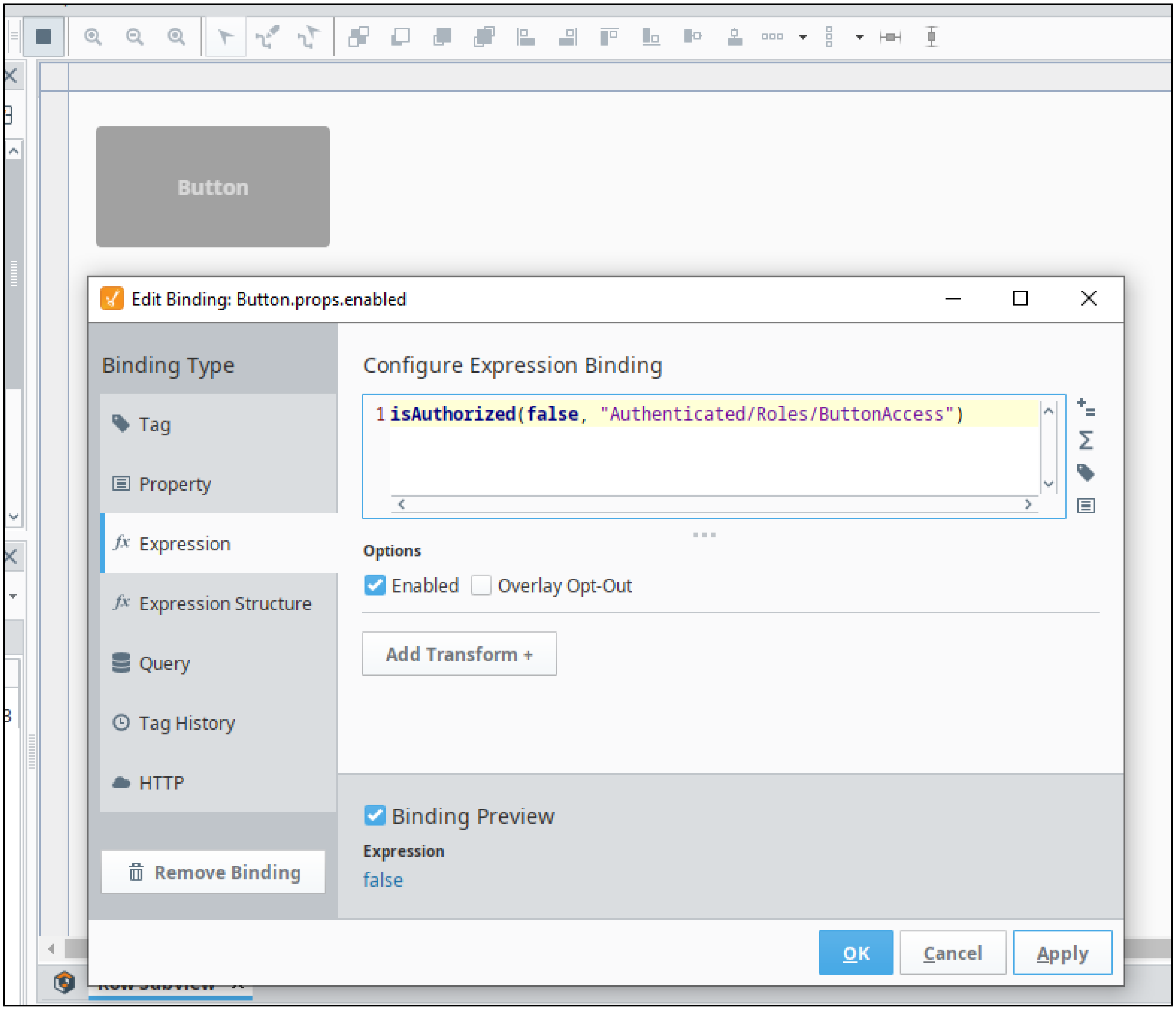
Component bindings can also be used to determine if a user should have access to a certain component. For example, if a component has an "Enabled" property that you want to toggle based off of security levels, then can simply apply one of these approaches.

Using isAuthorized in an Expression

The simplest method involves configuring an expression binding that uses the isAuthorized function.

```
isAuthorized(false, "Authenticated/Roles/YourRoleGoesHere")
```

In the image below, an isAuthorized function expression binding is configured on a Button's enabled property, which causes the property to change to "false" if a user doesn't have the specified security level. This has the additional benefit of preventing the onActionPerformed scripting event from triggering in cases where a user doesn't the security level.



```
isAuthorized(false, "Authenticated/Roles/ButtonAccess")
```

Options

☑ Enabled ☐ Overlay Opt-Out

☑ Binding Preview

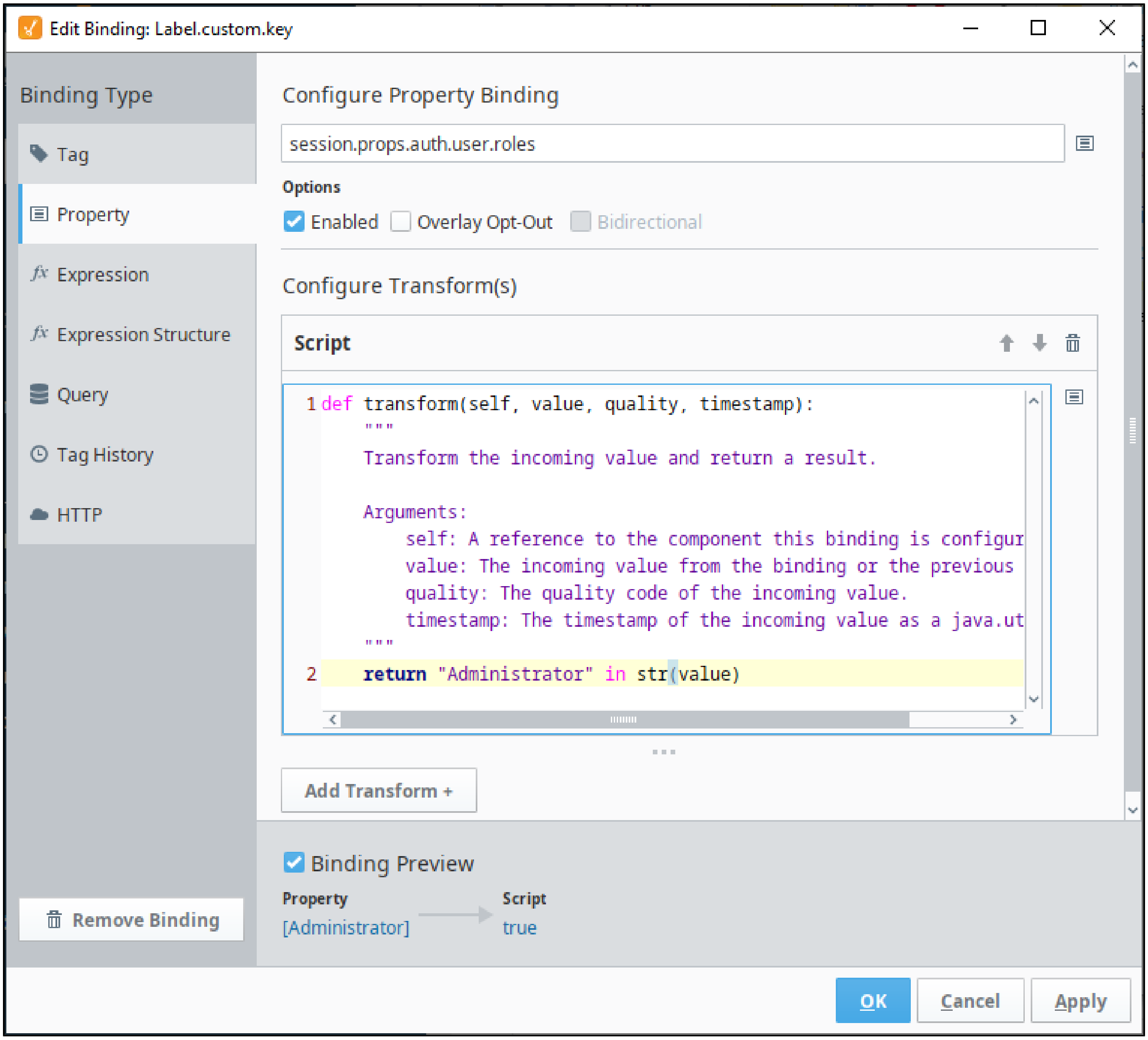
Expression

false

Using Session Props to Check Roles

Alternatively, a script transform can be used to examine the `session.props.auth.user.roles` property to determine if the current user has the desired role. In this case, we can configure a property binding to the roles session property, and use the following line in a script transform:

```
return "Administrator" in str(value)
```



Configure Property Binding

session.props.auth.user.roles

Options

☑ Enabled □ Overlay Opt-Out □ Bidirectional

Configure Transform(s)

Script

```
def transform(self, value, quality, timestamp):
    """
    Transform the incoming value and return a result.

    Arguments:
        self: A reference to the component this binding is configur
        value: The incoming value from the binding or the previous
        quality: The quality code of the incoming value.
        timestamp: The timestamp of the incoming value as a java.ut
    """
    return "Administrator" in str(value)
```

☑ Binding Preview

Property [Administrator] → Script true

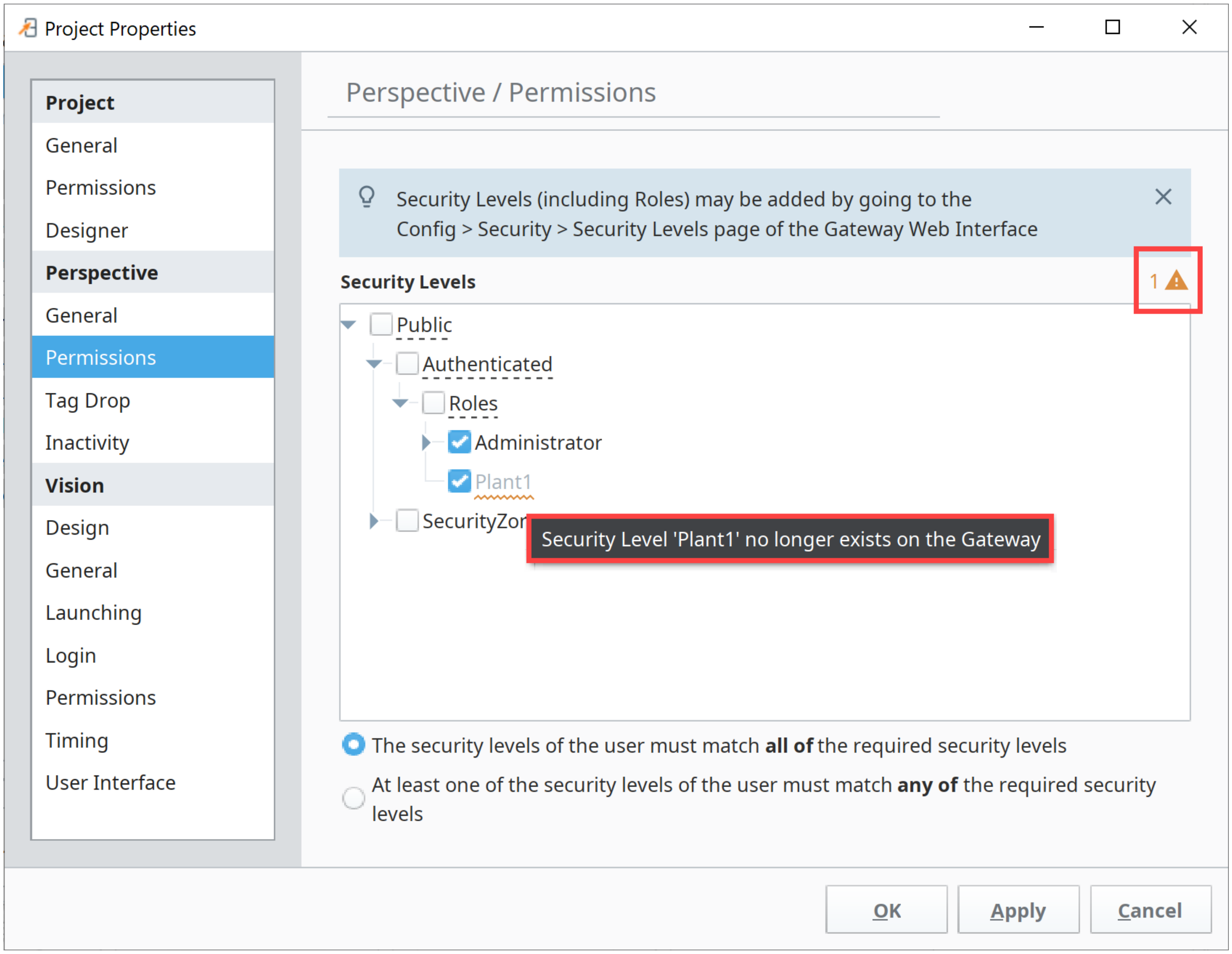
Deleted Security Level Indicators

The following feature is new in Ignition version **8.1.25**
Click here to check out the other new features

Since security levels are set through the Gateway, it may not be immediately obvious that a security level has been deleted. This can be checked quickly in the Designer by accessing the Project Properties and navigating to **Perspective** > **Permissions**. In the example image below, the **Plant1** role has been deleted. This is indicated by the security level, and any child levels, appearing grayed-out with a red warning underline. Hovering over the security level displays a tooltip informing the user the security level no longer exists on the Gateway. A warning indicator ⚠ icon will also appear in the upper right-hand corner with the number of selected security levels that no longer exist. If you follow the security levels tree up to the parent levels, you'll notice affected levels now include a dotted underline.

All warning indications are removed when deleted security levels are unchecked and new settings are saved. Additionally, the deleted security level will no longer be visible.

These security level indications are visible on the Event Configuration, Edit Permissions, and Tag Editor windows when applicable.



Security Levels (including Roles) may be added by going to the Config > Security > Security Levels page of the Gateway Web Interface

Security Level 'Plant1' no longer exists on the Gateway

● The security levels of the user must match **all of** the required security levels

○ At least one of the security levels of the user must match **any of** the required security levels

A Vision-Oriented Guide to Perspective

An Intro to Perspective

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

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

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

On this page ...

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

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

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

Perspective Design Considerations

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

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

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

Designing Your Main Views

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

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

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

If you want to make your project **mobile responsive**, there are other types of containers that you can use. The breakpoint container allows you to show two completely different views based on the width of the session.

Docked Views and Pages

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

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

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

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

INDUCTIVE
UNIVERSIT

Docked Views in Perspective

Watch the Video

JSON Component Properties

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

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

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

Bindings and Transforms

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

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

Components

Because of all the differences between Perspective and Vision, the two modules necessarily have a different set of components. There is no plan to ever have a one-to-one parity between components, and not all of the Vision components will make their way into Perspective components. Some components are new to perspective like the Menu tree and the Link components. The Link component was not feasible in Vision, but with Perspective Sessions living in a browser it makes sense to have that component.

Comparison of Perspective and Vision Functionality

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

Perspective Concept	Similar Vision Concept	Perspective Description
Session	Client	A Perspective Session is a running instance of an application, much like a Vision client. Whereas Vision clients run as independent Java programs on a user's machine, a Perspective session runs natively in a web browser (or the Ignition Perspective App). Notably, a session can run across multiple pages in the same web browser. This is similar to how sessions function elsewhere on the Internet: log into your favorite shopping website and then open several new tabs, each one knows about your current shopping cart.
Page	Desktop (A.K.A multi-monitor clients)	A page in Perspective equates to a single page in a web browser. Pages are the main navigational unit in a session, and consist of one or more views. Each page is associated with a URL, which means the Forward and Back buttons in a web browser can be used to navigate to pages that have already been visited. Multiple pages can be open as part of the same Session, similar to how multiple desktops in a Vision Client may be open simultaneously.

		Views can be docked to specific edges of the page, or be used as a "Main View". Each page consists of at least a main view, but multiple views can be configured on a page. A page has specific regions where you can place instances of your views.
View	Window and Template	A View is the primary unit of design for Perspective. Perspective Views fill the roles of both Windows and Templates in Vision. Thus, you could think of a view as a window that can be nested inside other windows, or as a template that can be maximized or docked. Views have a root container, much like windows in Vision. Nesting one view inside another requires the Embedded View component. Parameters may be defined on the view, and then passed into the embedded view There are also components that may dynamically create view instances (like the Flex Repeater component). In this way, views act like templates just as they do in the Vision module. Parameters can be passed into a view from an external source.
Container	Container + Layout Constraints	Containers are objects that contain components. You can nest one container inside of another. In Perspective, the way that components reposition and resize is managed on the container, and every component contains only simple information about its size and position, which is interpreted based on the container's specifications. There are several container types, and each type is defined by the layout type it uses.
Component	Component	Components, like in Vision, are displays, buttons, charts, labels, and other objects that display information to the user viewing the session. Components are the elements displayed in the Component Palette in the Designer.
Property	Property	Properties serve as a place to change how a component looks or behaves, or store specific pieces of information. In Perspective, the property tree of a component is a JSON object, and as a result, there are only three configurable data types: **value**, **object**, and **array**. Rather than characterize the expected format of the property's data (e.g., integer, boolean, or string), these data types control the structure of the property tree. No further configuration of data types is required.
Session Property	Client Tag	Every session has a configurable collection of properties that can be managed from the designer. They can be referenced from any view in the project, both from property/expression bindings and from scripting.
Events and Actions	Component Scripting	Event and Actions are configured similarly to the Component Scripting section in Vision. However, Perspective offers additional configurable events along with possible responses to these events (Actions).
Styles	Component Style Properties /Customizer	Styles in Perspective use CSS, and are configurable from the properties on each component. However, Perspective also offers the ability to configure style classes, which allow you to reuse configured styles across many different components. Styles can also be configured to change based on more advanced properties, like whether a mouse is hovering over the component, or how wide the viewport of the viewing device is.

New Features in Perspective

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

Perspective Term	Description
Transform	A transform lies between a binding and the property it modifies, and provides an opportunity to change the value or format of the binding's output. For example, if a tag binding yields 0 for Normal and 1 for Faulted, we could use a transform to map 0 to Green and 1 to Red.
Component Messaging	Perspective offers the ability to communicate between *components*, using a similar style of messaging that one might use to communicate between clients in Vision. You can send messages from any component using the system.perspective.sendMessage function, and you can configure message handlers on any component. This is useful for controlling the behavior of one component from another in a different view.

Perspective Co-Branding

The following feature is new in Ignition version **8.1.20**
Click here to check out the other new features

On this page ...

Perspective Co-Branding is a Gateway setting that allows users to customize the look their Perspective Sessions with logos, pictures, and colors. Various aspects of Perspective can be customized, such as the login screen, the desktop icon, and more. The table below lists various properties and elements that can be modified. These settings will not affect the internal contents or styles of the Perspective Session:

Property	Description
Custom Branding	Choose to toggle custom branding on or off. Default is off.
Background Color	Choose what the background color will be.
Text Color	Choose what the text color will be.
Button Color	Choose what color buttons will appear as.
Button Text Color	Choose what color button text will appear as.
Logo	Graphic logo used on the Perspective Session login and loading screen. Supports .jpg or .png formats. Suggested logo size is 160x160 pixels.
Favicon	Favicons will appear on your desktop browser tab after Session refresh. Supports .png format. Suggested logo size is 180x180 pixels.
App Icon	The iOS app icon is used when saving the page to the home screen. Supports .png format. Suggested logo size is 180x180 pixels. Note that the icon used on Android devices when creating a project shortcut is set by "Launch Icon" project property.
Preview	Shows a preview of the login and loading screens with custom branding applied.

Note: Perspective Co-Branding is currently only available for Standard and Cloud Edition.

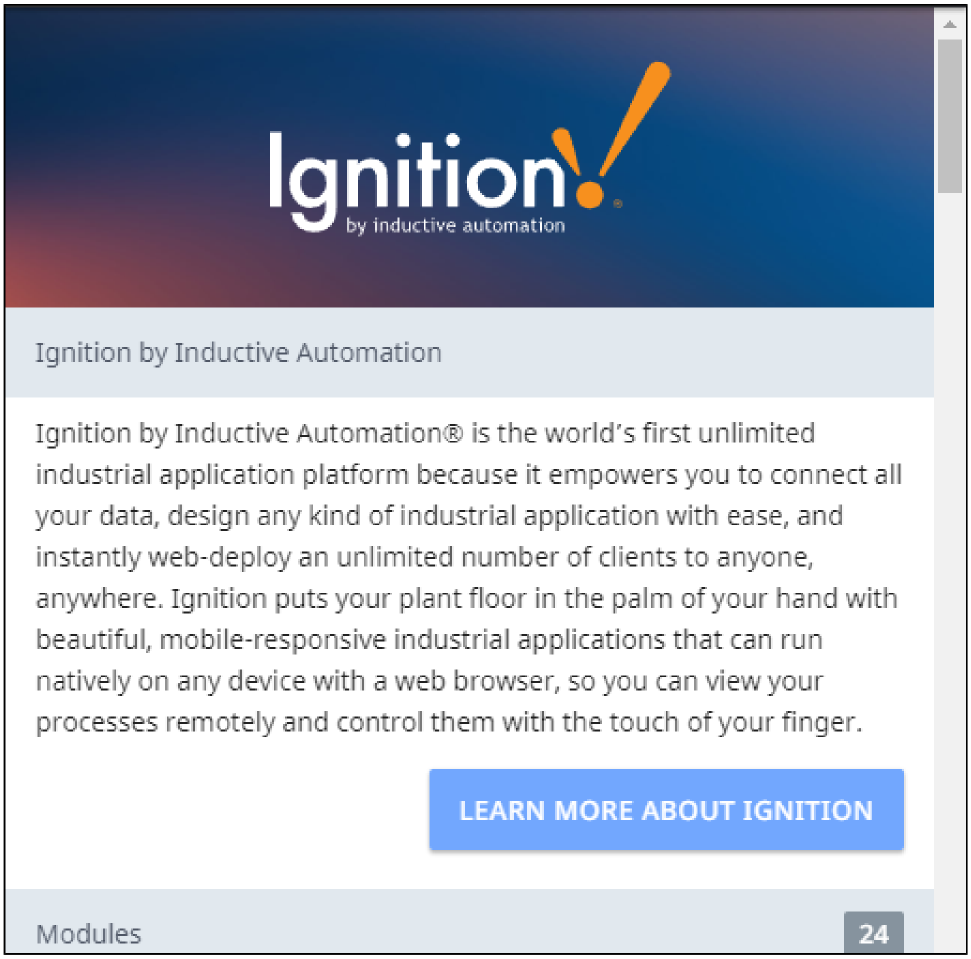
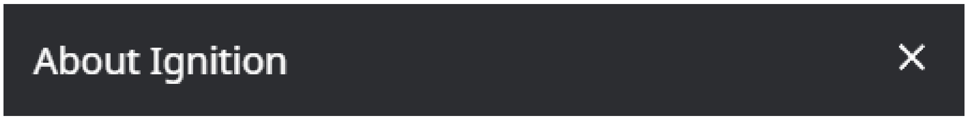
App Bar Customization

In addition to the login, loading, and terminal screens, you can also personalize the bottom-docked App Bar by:

- Choosing to toggle the App Bar off or on.
- Changing the icon for the About button.
- Changing the view that displays the about modal.
- Changing the title of the about modal.

The table below lists the customizable elements of the App Bar, as well as with what they look like by default. For more information about individual Session properties that affect the App Bar, click here.

Name	Default
Show	False
About Icon	✓
About View	

	 Ignition by Inductive Automation Ignition by Inductive Automation® is the world's first unlimited industrial application platform because it empowers you to connect all your data, design any kind of industrial application with ease, and instantly web-deploy an unlimited number of clients to anyone, anywhere. Ignition puts your plant floor in the palm of your hand with beautiful, mobile-responsive industrial applications that can run natively on any device with a web browser, so you can view your processes remotely and control them with the touch of your finger. LEARN MORE ABOUT IGNITION Modules 24
About Title	 About Ignition ✕

Best Practices (Favicons and App Icons)

There are some considerations to take when selecting or designing a logo to use as a favicon or app icon. While these considerations are not mandatory, they may give some insight into optimal logo design.

Here are a few tips that applies to both favicons and app icons:

- Use a higher resolution of the final icon you have selected, if possible.
  - Since your image resolution may be downscaled, using a higher resolution picture will help retain image quality.
- If bandwidth or some other bottlenecking factor is an issue, using smaller favicons may help reduce data load.

Favicons

Favicons are generally used in browser tab icons, desktop icons, and Android icons. The bullet(s) below contain tips on designing favicons:

- (Chrome on Android specific) If your favicon is too small, then the favicon will not be used when saving to the home screen.
- For typical uses in desktop environments, either 16x16px or 32x32px are recommended.
  - Modern browsers downsize images, so larger images will also work.
- For multi-platform environments, either 180x180px or 192x192px are recommended.
  - Many Android devices will not use favicons smaller than a certain size.

App Icons

App icons are used by Apple devices and may follow some different rules than favicons. For Apple's official documentation detailing best practices and creating an app icon for iOS and other Apple devices, click here.

One caveat to keep in mind is that while the favicon and logo fields will accept and render icons with transparent designs without issue, attempting to use an icon with transparency as an app icon will result in the icon appearing all black.

Best Practices (Design)

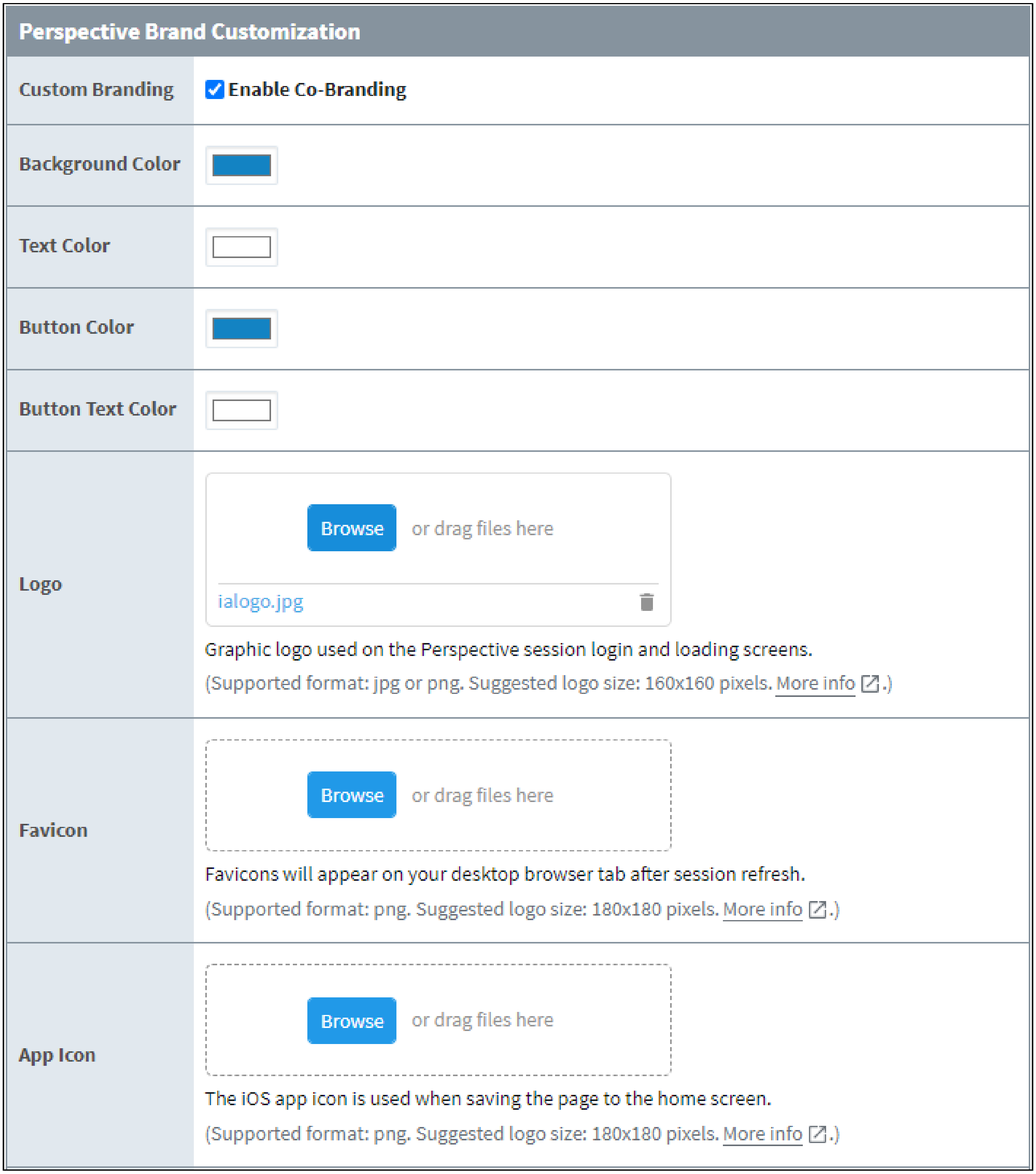
While not exactly white labeling, the co-branding feature allows you to customize the look of your projects. Listed below are a few basic tips for designing your Perspective login, loading, and terminal screens. These tips are not mandatory, as styles, personalization, and customization preferences differ from person to person and organization to organization. Additionally, these tips may extend to apply during general Perspective Session design.

- Use text colors that can be easily seen against their backgrounds.
- Keeping designs simple will help minimize cognitive load.
- Use logos and icons that are easily identifiable.

Examples

Example 1 (Login and Terminal Screens)

The settings below will modify the look of the Perspective login, loading, and terminal screens:



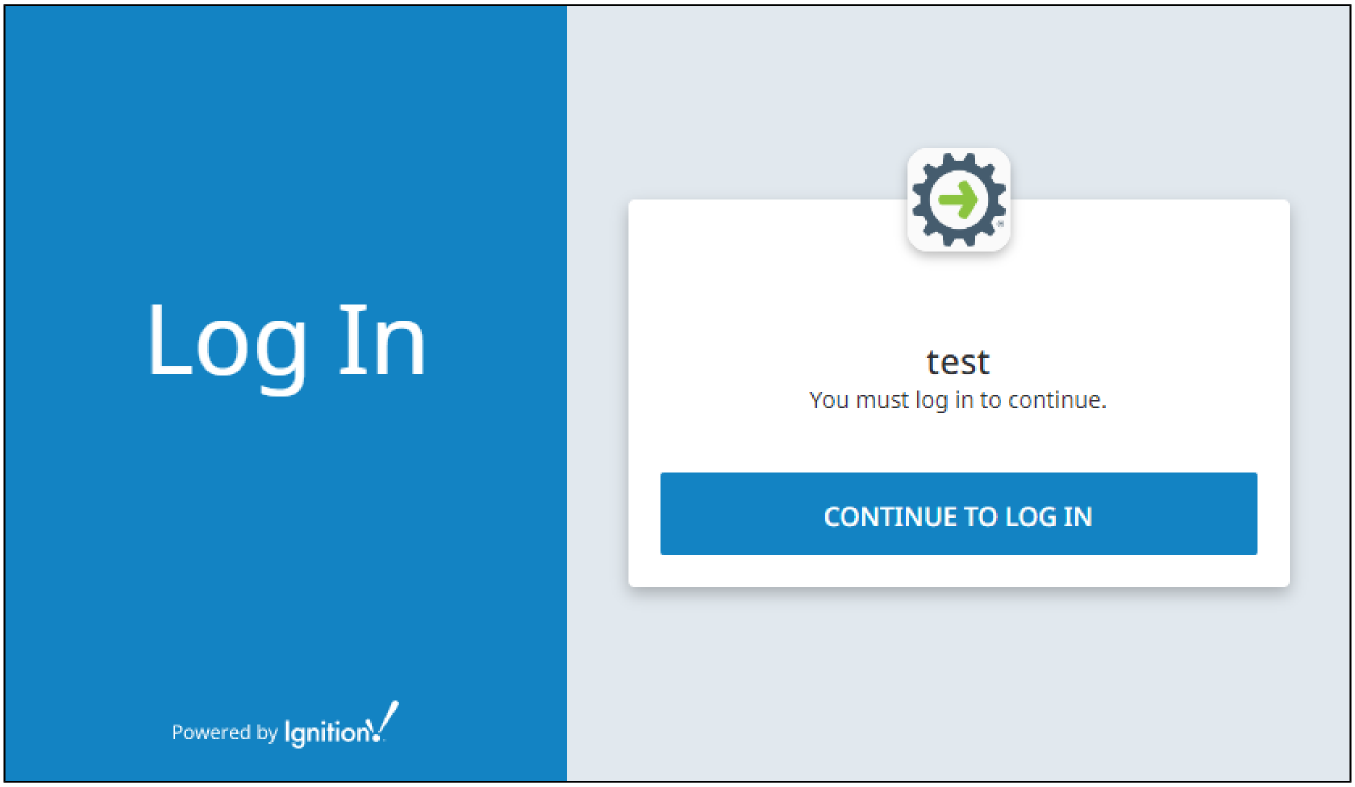
Perspective Brand Customization

Custom Branding	☑ Enable Co-Branding
Background Color	🟦
Text Color	⬜
Button Color	🟦
Button Text Color	⬜
Logo	Browse or drag files here ialogo.jpg Graphic logo used on the Perspective session login and loading screens. (Supported format: jpg or png. Suggested logo size: 160x160 pixels. More info.)
Favicon	Browse or drag files here Favicons will appear on your desktop browser tab after session refresh. (Supported format: png. Suggested logo size: 180x180 pixels. More info.)
App Icon	Browse or drag files here The iOS app icon is used when saving the page to the home screen. (Supported format: png. Suggested logo size: 180x180 pixels. More info.)

Property	Value
Custom Branding	True
Background Color	#1383C3
Text Color	#FFFFFF
Button Color	#1383C3
Button Text Color	#FFFFFF

Logo	Inductive Automation Logo

The following is the rendered output:



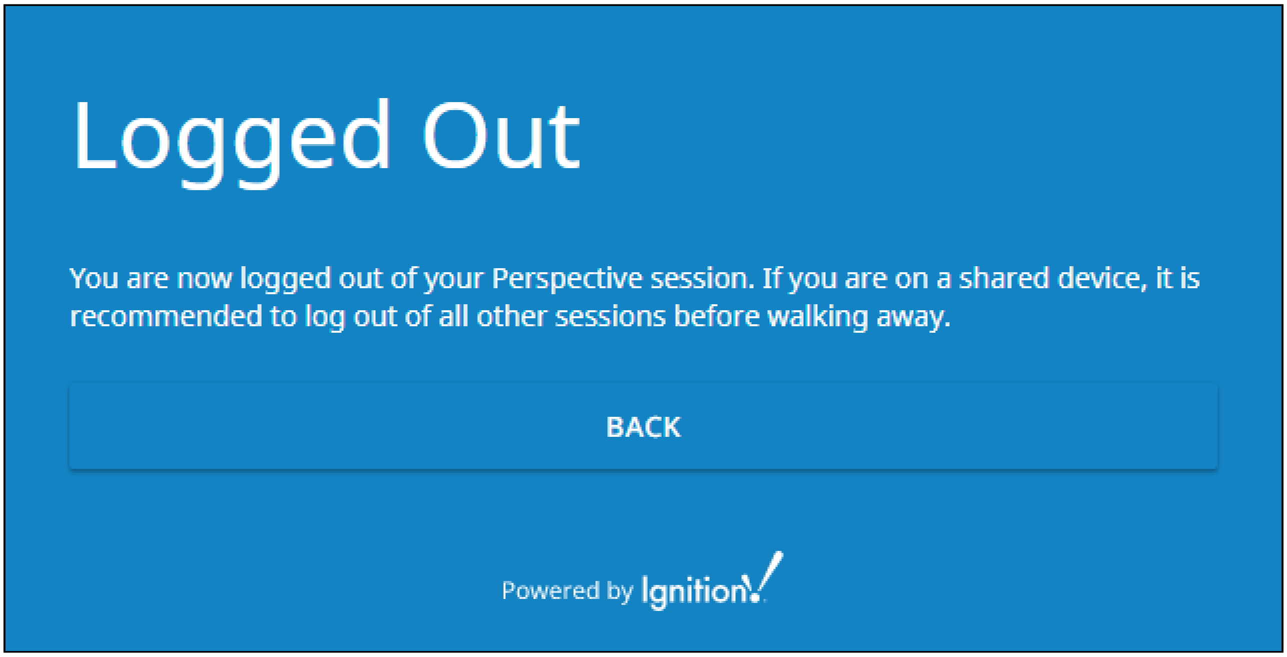
Log In

test

You must log in to continue.

CONTINUE TO LOG IN

Powered by Ignition



Logged Out

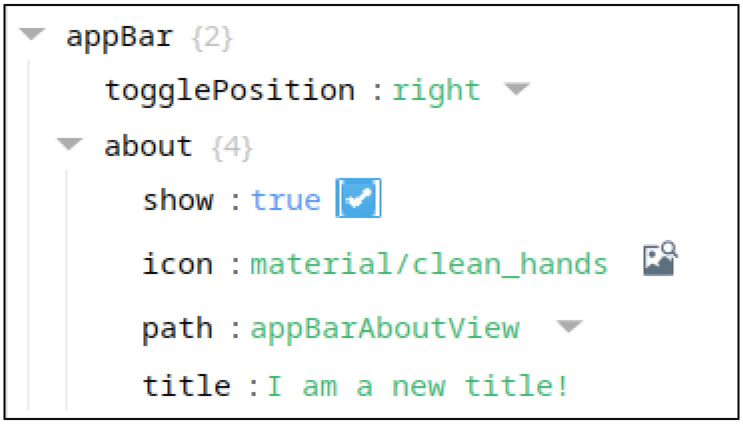
You are now logged out of your Perspective session. If you are on a shared device, it is recommended to log out of all other sessions before walking away.

BACK

Powered by Ignition

Example 2 (App Bar)

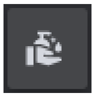
The settings below will modify the look of the Perspective Session's App Bar:



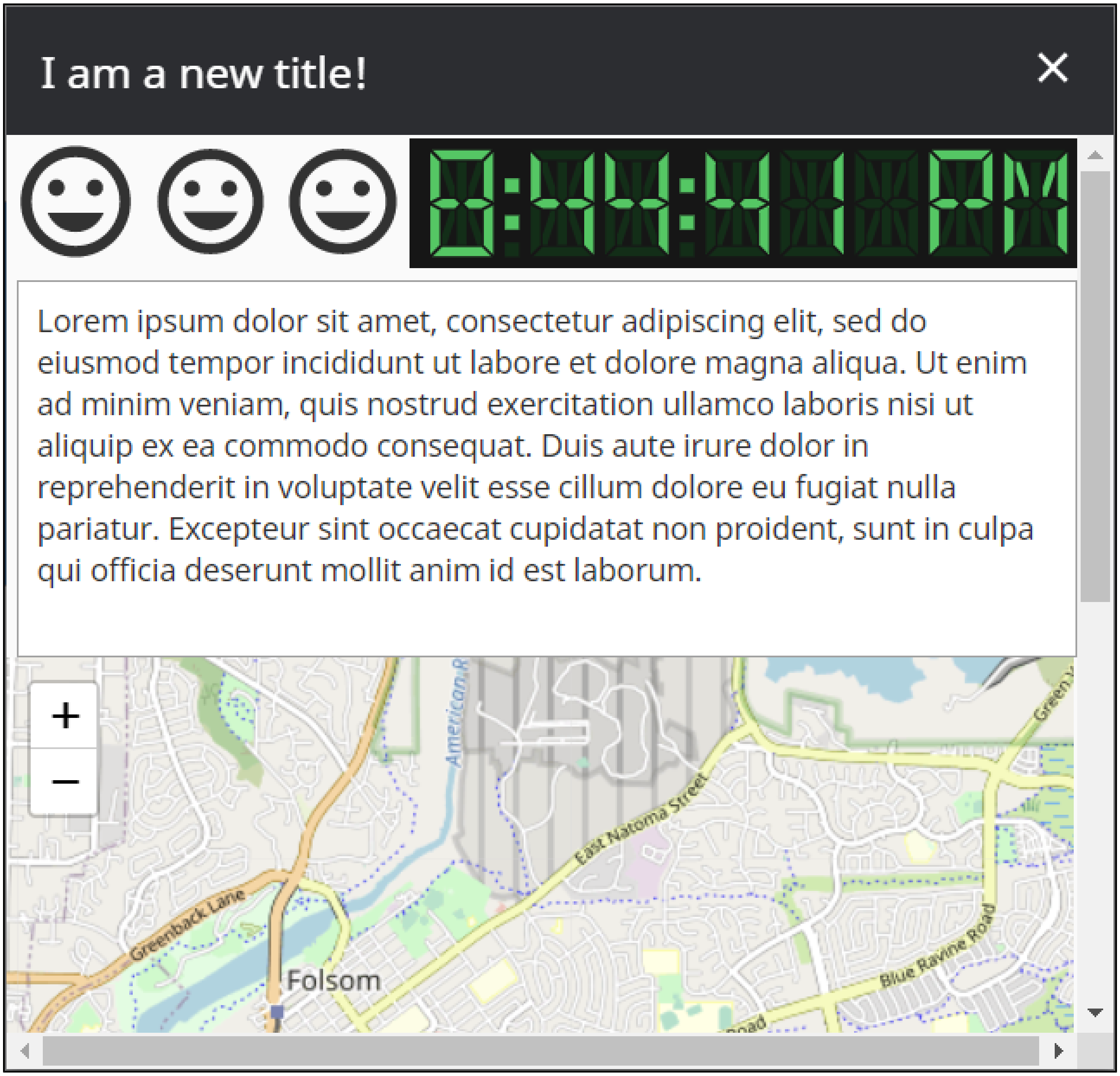
```
▼ appBar {2}
    togglePosition : right ▼
  ▼ about {4}
      show : true ☑
      icon : material/clean_hands
      path : appBarAboutView ▼
      title : I am a new title!
```

Property	Value
show	True
icon	material/clean_hands
path	appBarAboutView
title	I am a new title!

The following is the rendered output affected by the icon property:



The following is the rendered output affected by the path and title properties:



I am a new title!

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.