

Ignition 8.3 User Manual — Perspective Module

Generated 2026-10-12 from
<https://www.docs.inductiveautomation.com/docs/8.3/>

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.



INDUCTIVE
UNIVERSITY

Perspective Project Elements

[Watch the video](#)

Views and Containers



INDUCTIVE
UNIVERSITY

Anatomy of a View

[Watch the video](#)

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 within a View.

The screenshot shows the Inductive University Perspective design tool interface. The top menu bar includes File, Edit, View, Project, Component, Tools, and Help. Below the menu is a toolbar with various icons for design and development. The left sidebar contains two panels: the Project Browser and the Tag Browser.

Project Browser: Shows a hierarchical tree structure. The 'FlexContainer' is selected, which contains a 'MultiStateButton' and a 'Label'.

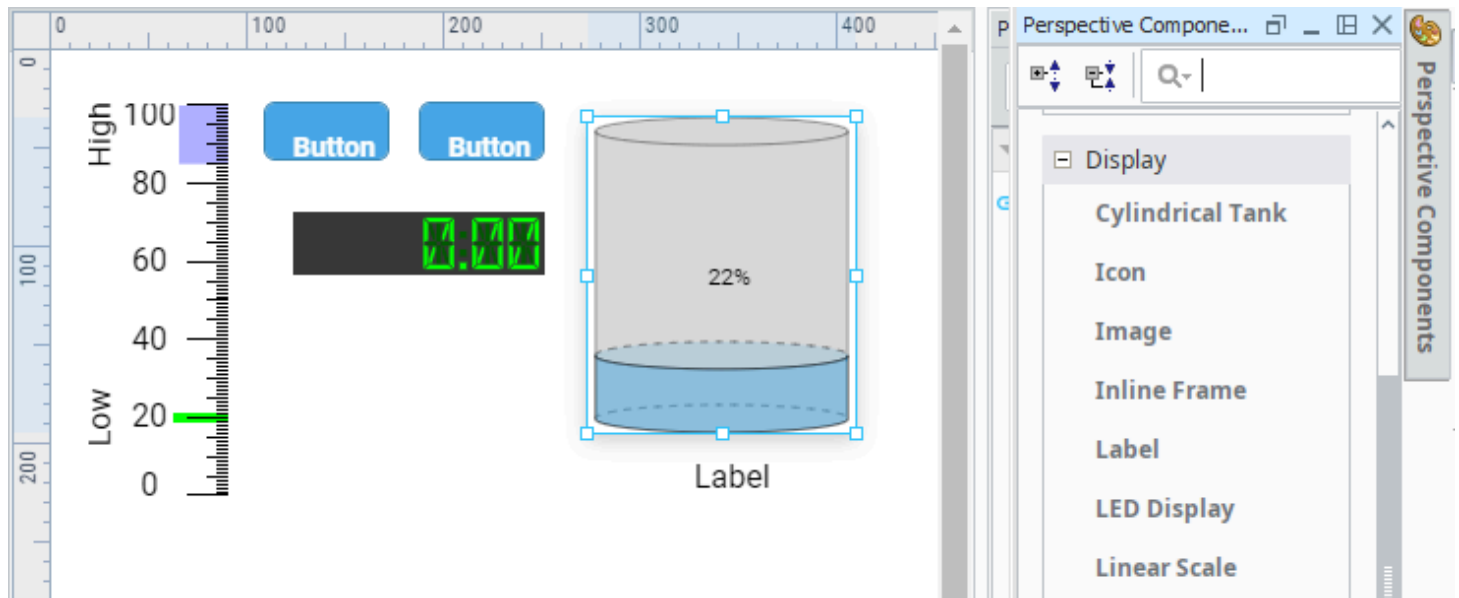
Tag Browser: Displays a table of tags with columns for Tag, Value, Data T..., and Traits.

Tag	Value	Data T...	Traits
New Tag	Memory	97	Int4
zztag	Memory	55	Int4

The main design area on the right shows a HMI screen layout. At the top, there is a digital display labeled 'Temp' showing '00:00'. Below it, a 'Multi-state Button in a Flex Container' is shown, which includes three states: 'Manual' (grey), 'Off' (red), and 'Auto' (grey). The bottom status bar shows 'View1' and 'New View' tabs.

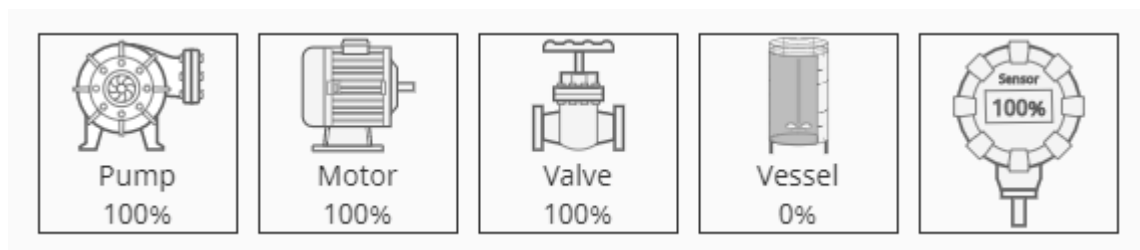
Components

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**.



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.



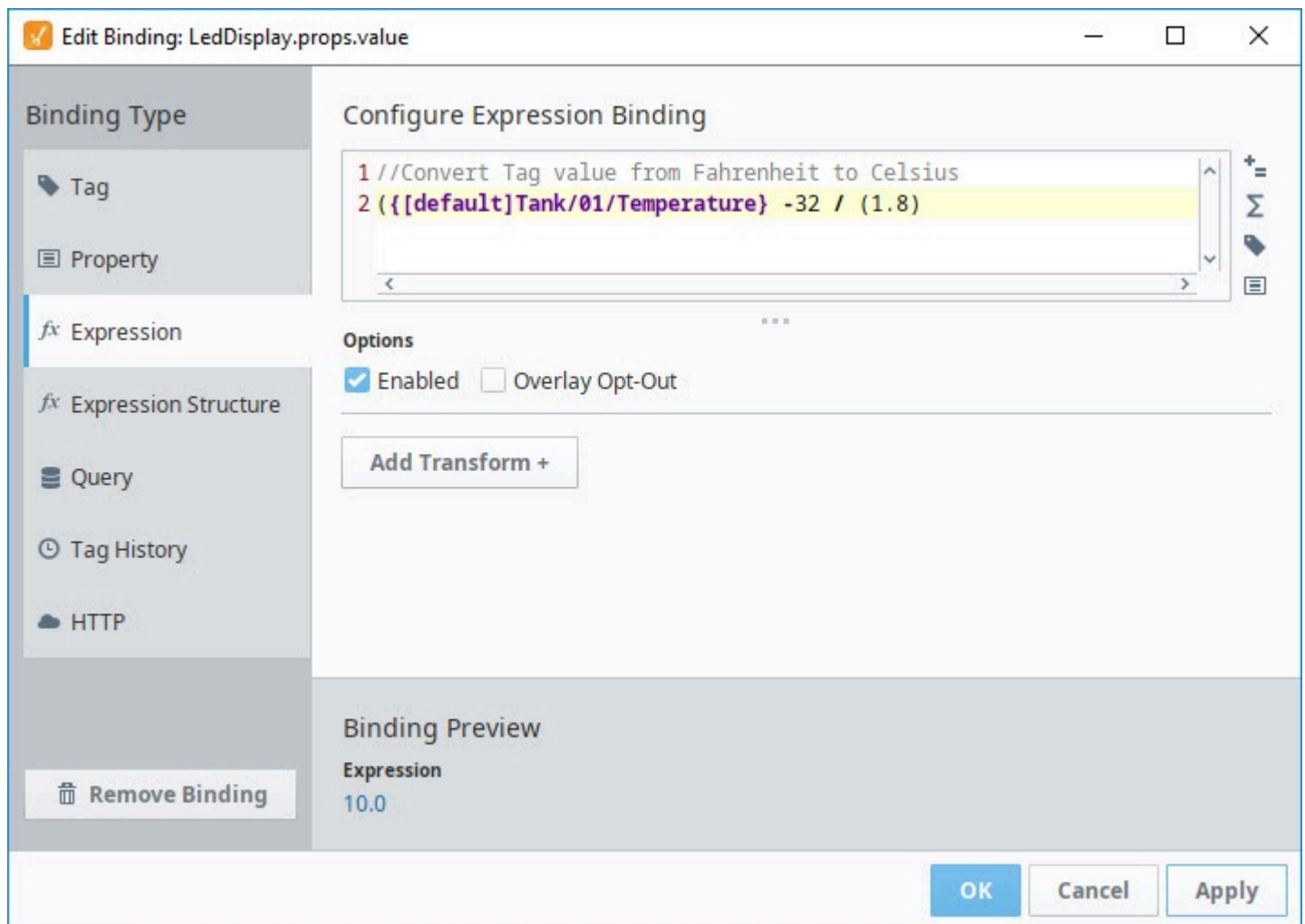
Bindings

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:

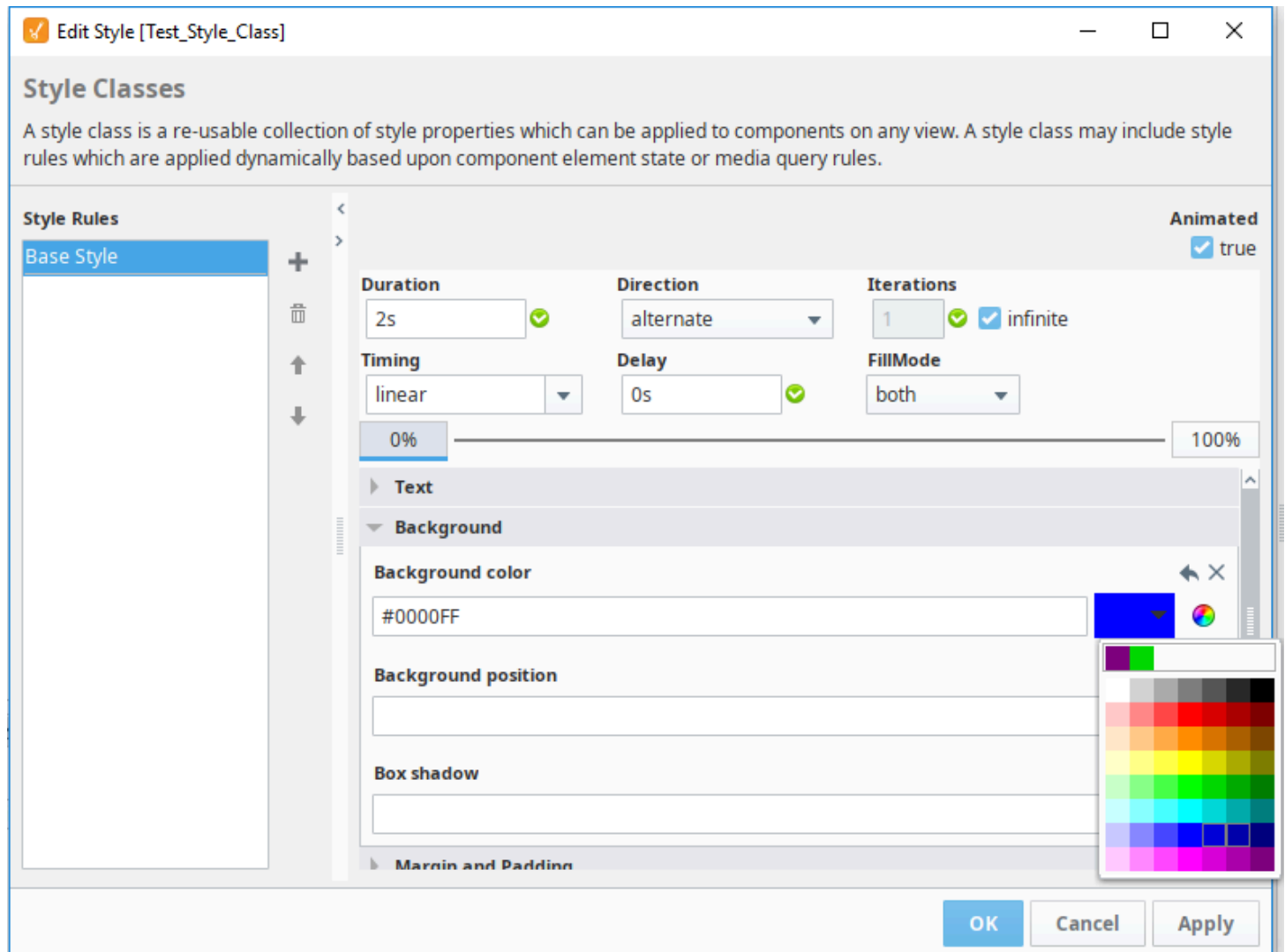
- **Tag** - Binds a property directly to a tag, which sets up a tag subscription for that tag.
- **Property** - Binds one property to another.
- **Expression** - A powerful type of property binding that uses simple expression language to calculate a value.
- **Expression Structure** - A property binding 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 Historian.
- **HTTP** - Used for passing data directly to and from a URL link.



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.



Security in Perspective

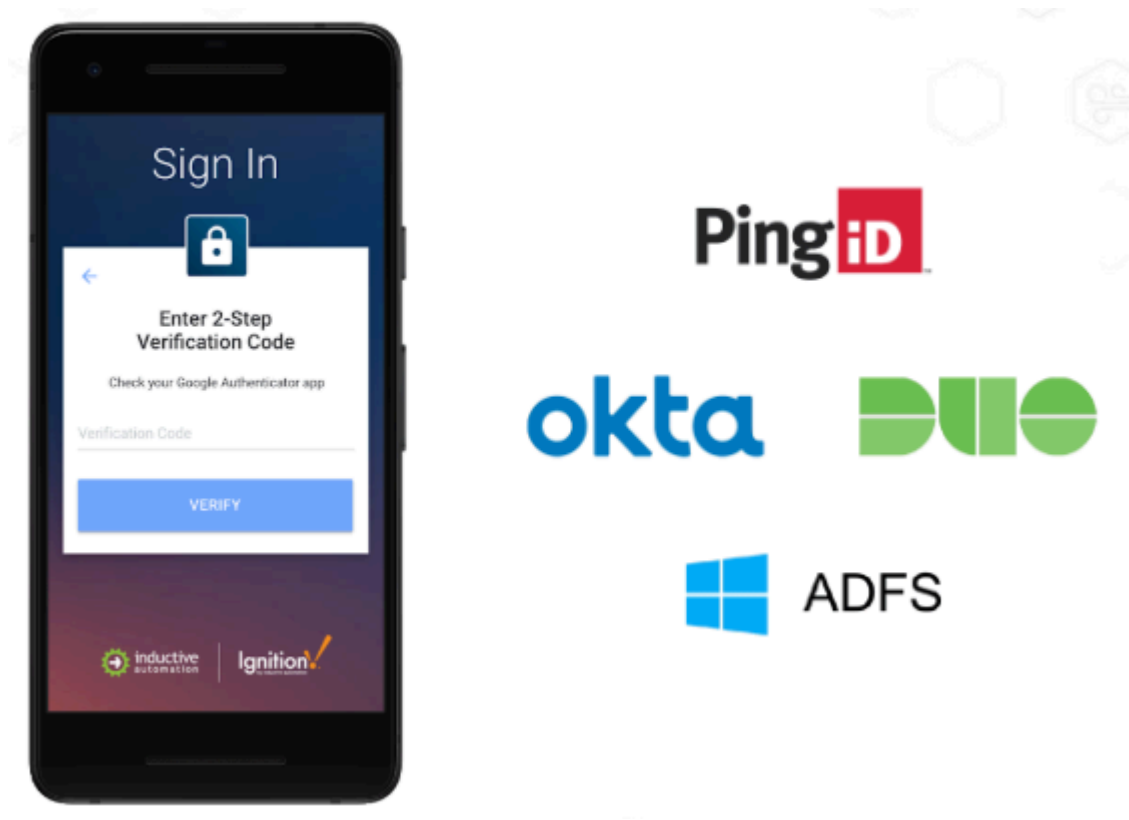
Perspective's approach to security covers a wide array of topics, including authentication, authorization and permission modeling, and transport layer security (TLS).

- 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 SAML, OAuth, OpenID, and others.

For more information, see [Security in Perspective](#).



Perspective Co-Branding

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](#).

Perspective Designer Interface



INDUCTIVE
UNIVERSITY

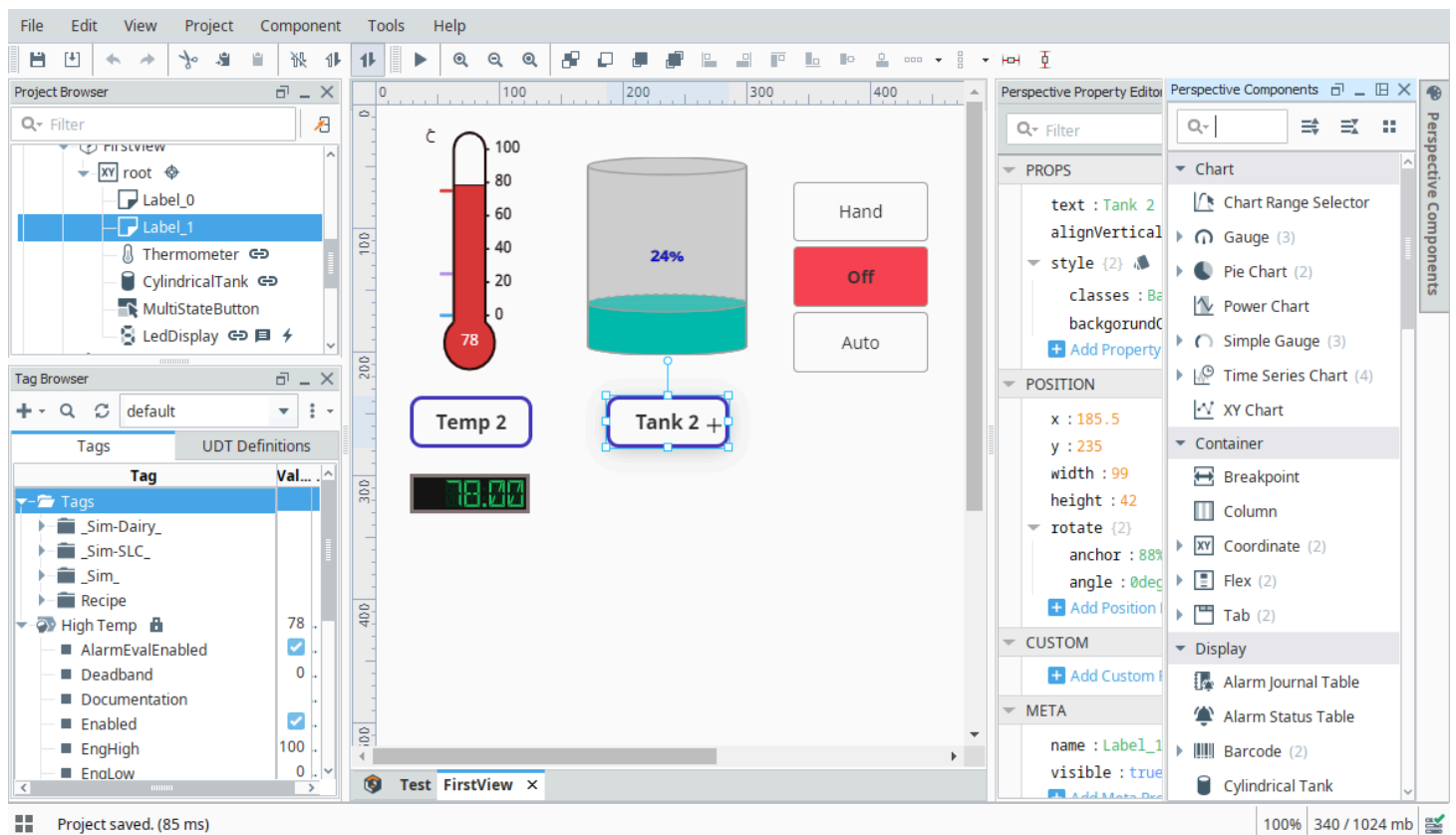
The Designer User Interface

[Watch the video](#)

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 the 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**.



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

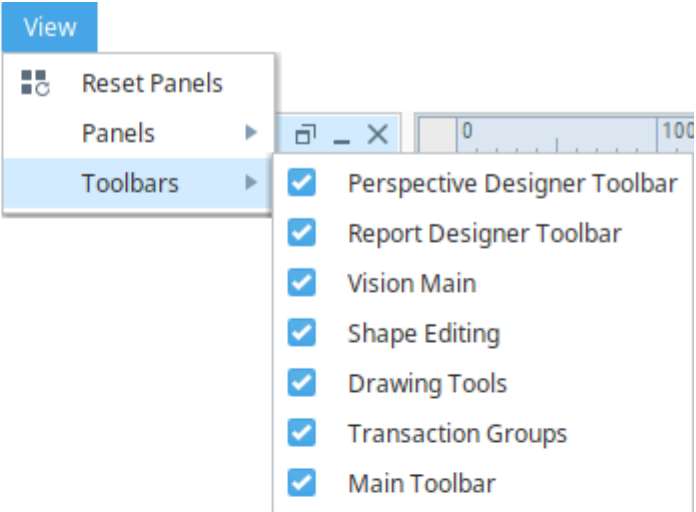
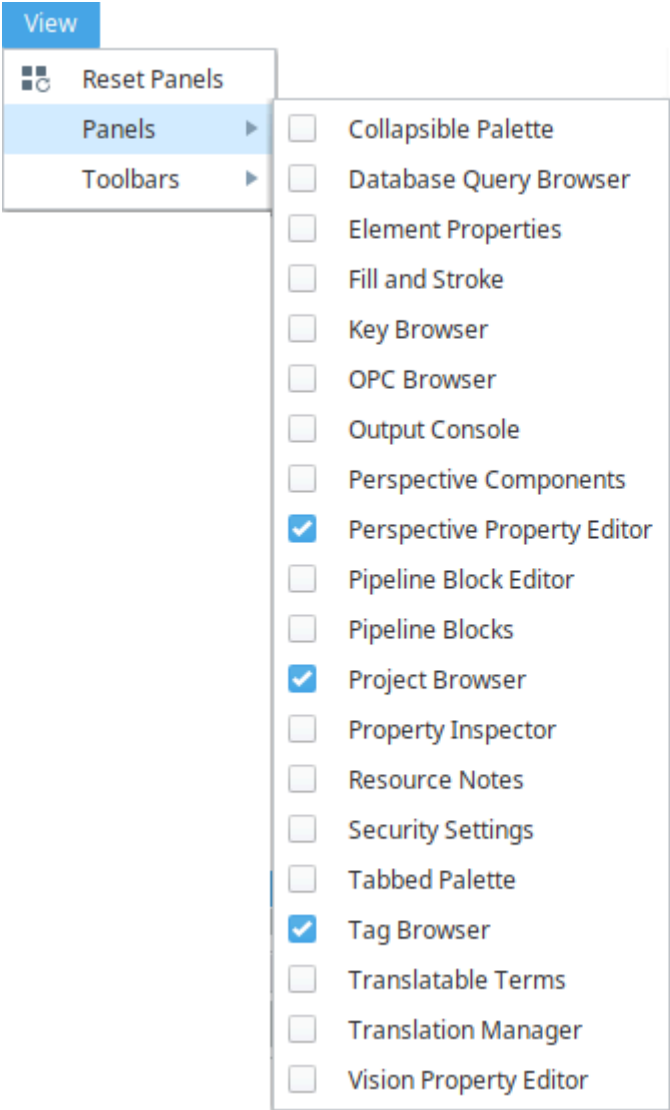
The **Edit Menu** is similar to other applications edit menus in that it provides much of the basic copy/paste functionality. You can also right-click on an item in the browser to access this menu.

Edit		
	Undo	Ctrl+Z
	Redo	Ctrl+Y
	Cut	Ctrl+X
	Copy	Ctrl+C
	Paste	Ctrl+V
	Duplicate	Ctrl+D
	Find/Replace	Ctrl+F
	Delete	Delete

Function	Description
Undo and Redo	Can be used to revert to the previous state, essentially removing the last change, or redoing it again after having been removed. This has a large queue that can be traversed, but does not include every change (i.e., Tag edits cannot be undone).
Cut/Copy/Paste/Duplicate	These functions 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** accesses configuration popups where you can view and add events, actions, and message handlers to individual components.

Component	
Configure Events...	Ctrl+J
Configure Scripts...	Ctrl+K
Configuration Explorer	Ctrl+R

Function	Description
Configure Events...	Displays the Event Configuration popup. For more information, see Perspective Events and Actions .
Configure Scripts...	Displays the Script Configuration popup. For more information, see Component Message Handlers .
Configuration Explorer	Displays the Configuration Explorer popup that lists the dynamic configurations for views and components. For more information, see Configuration Explorer .

Tools Menu

Perspective's workspace makes use of the [general Tools Menu options](#). However, the **Launch Project** item holds several unique sub options.

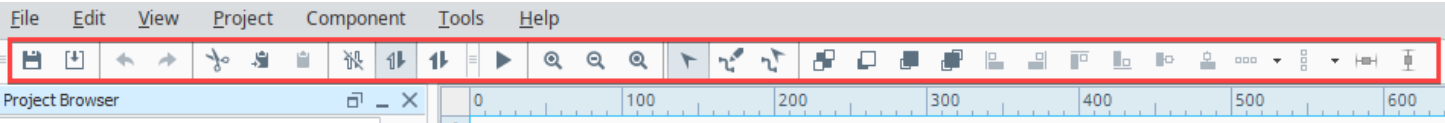
Function	Description
Launch Session	Launches the current project, using the system's default web browser.
External Debugger	Opens Dev Tools , via the designer's JxBrowser, inspecting whichever view was currently being viewed in the designer.

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.



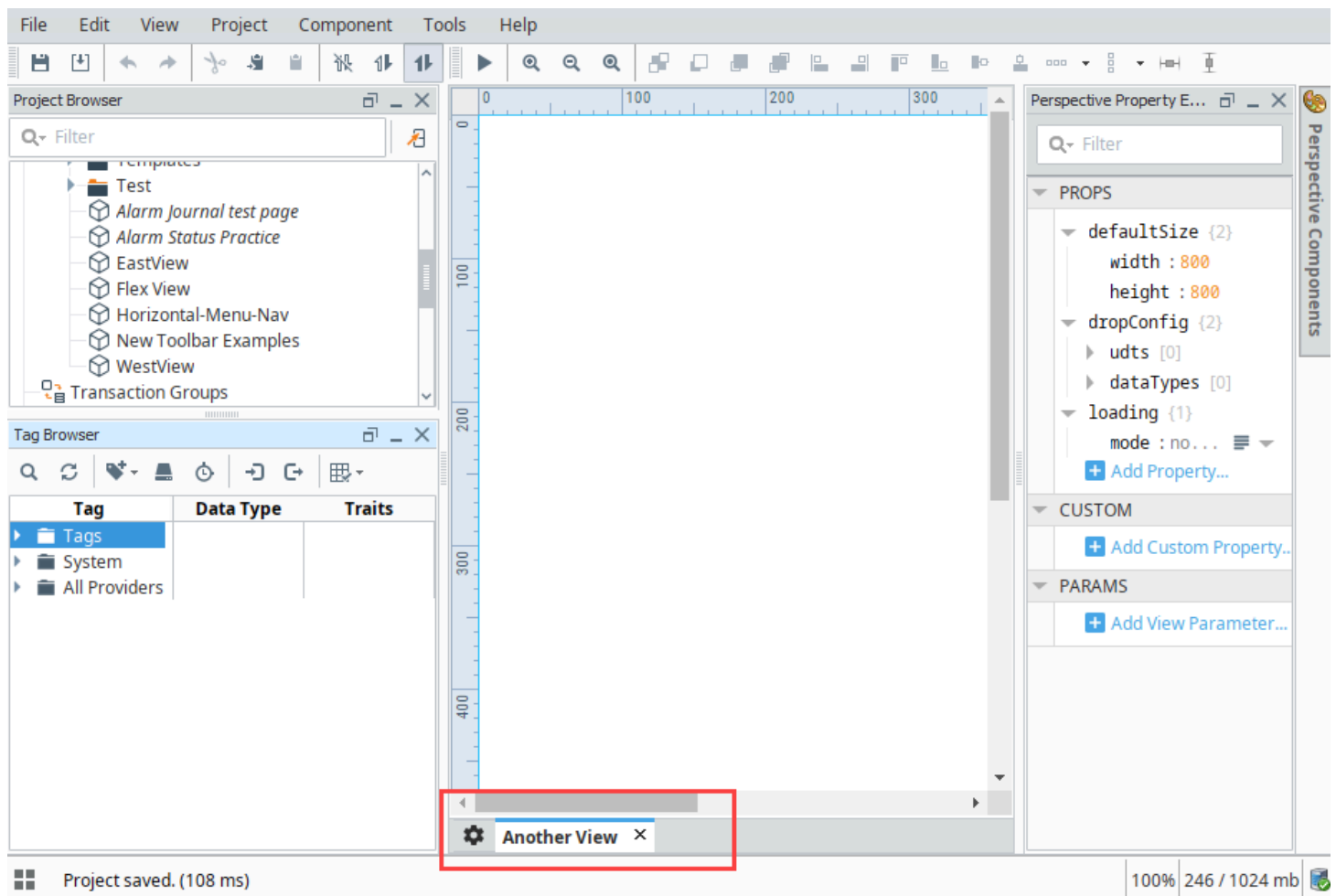
Icon	Function/Description
	Save all outstanding project changes in Ignition Gateway.
	Merges any new changes on the Gateway into the open project.
	Undo the last action.
	Redo the last undo action.
	Cuts the current selection into the clipboard.
	Copies the current selection into the clipboard.
	Pastes 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. Design mode is the default mode where you can manipulate components and adjust their properties. In Preview mode, you are interacting with a "live" version of the view. Property bindings and event handlers will run just like in the Session.
	Zoom into the currently open window.
	Zoom out of the currently open window.
	Zoom reset to 100%.
	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.

Icon	Function/Description
	Toggles the Pipe Draw Tool, allowing the mouse to draw pipe connections. Disabled when a container other than a coordinate container is deeply selected.
	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.

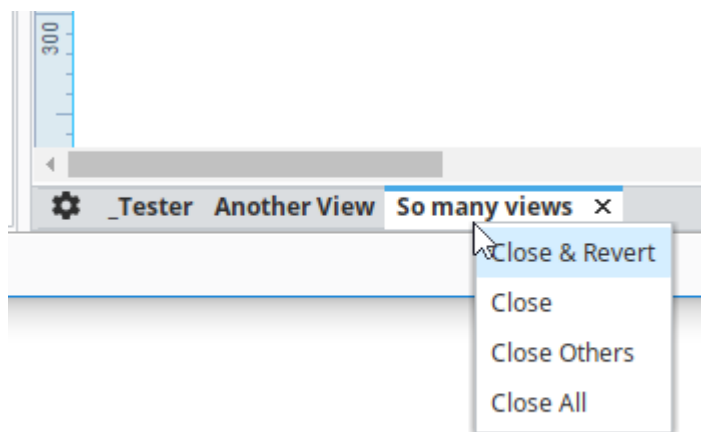
Resource Tabs

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.



Right-Click Menu

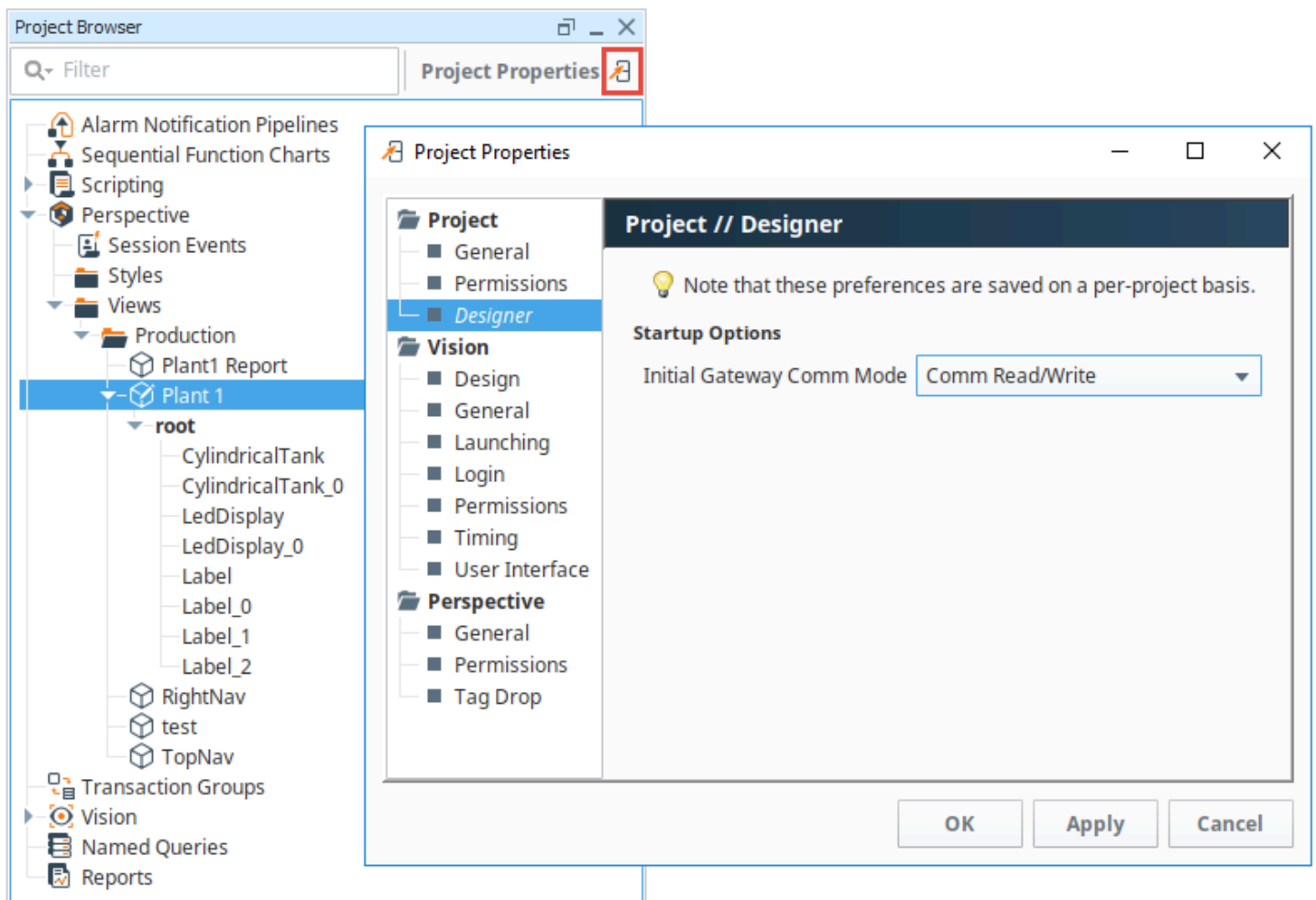
The right-click menu provides multiple ways to close one or more views.



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.

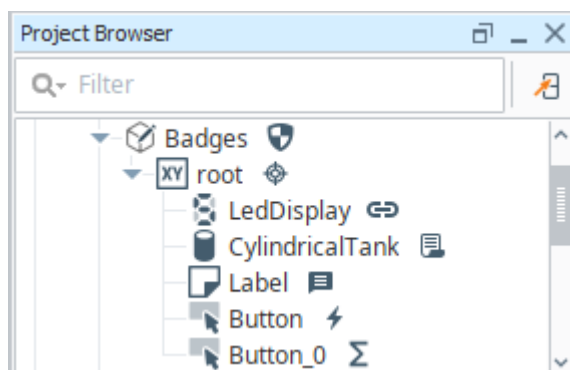
Project Browser

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.



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.



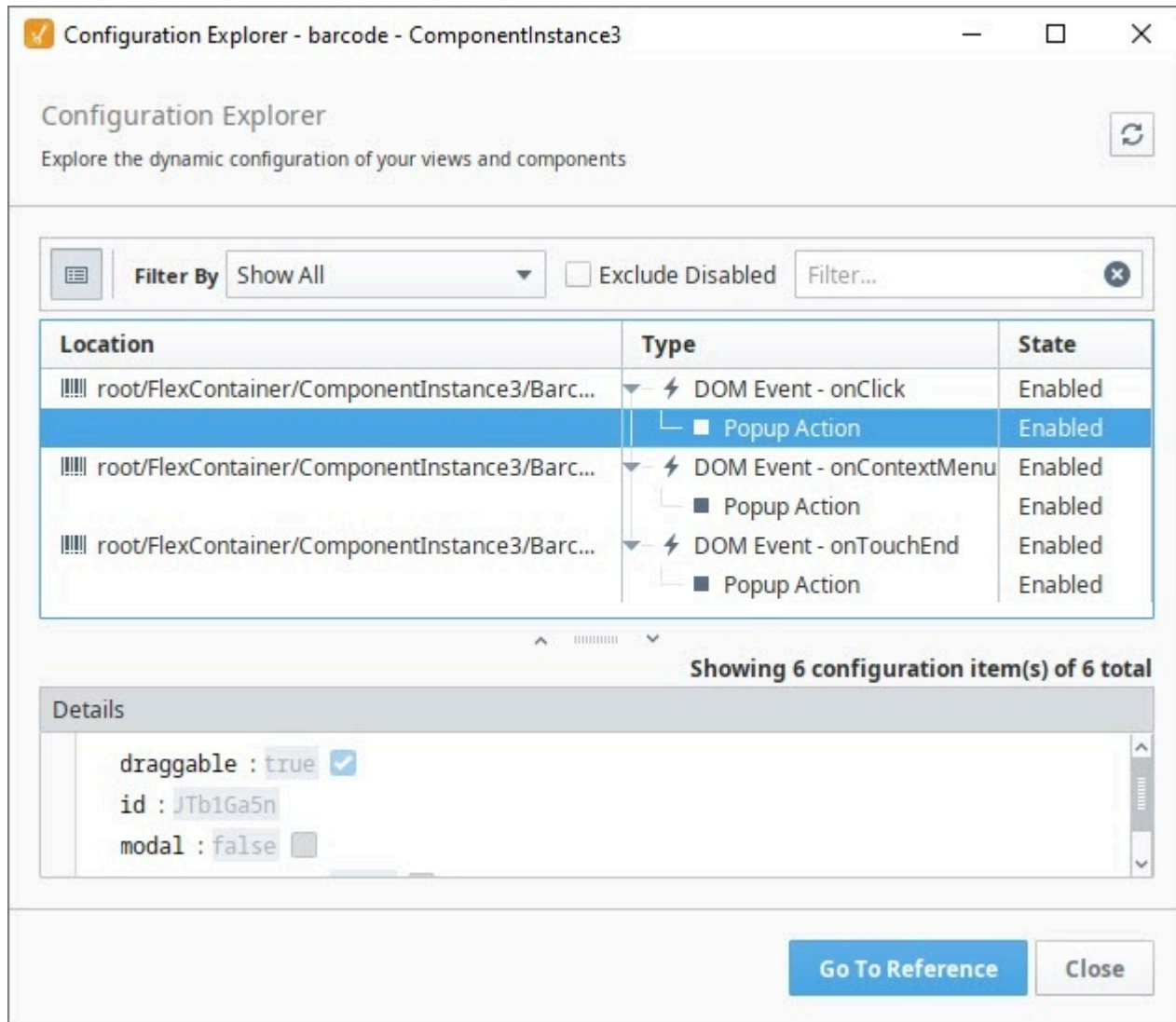
Badge Icon	Name	Description
	Binding	Indicates the component has a binding . Appears next to the name of the component.
	Custom Method	Indicates the component has a custom method . Appears next to the name of the component.
	Deep Select	Shows a selected component within a selected container. Indicates 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

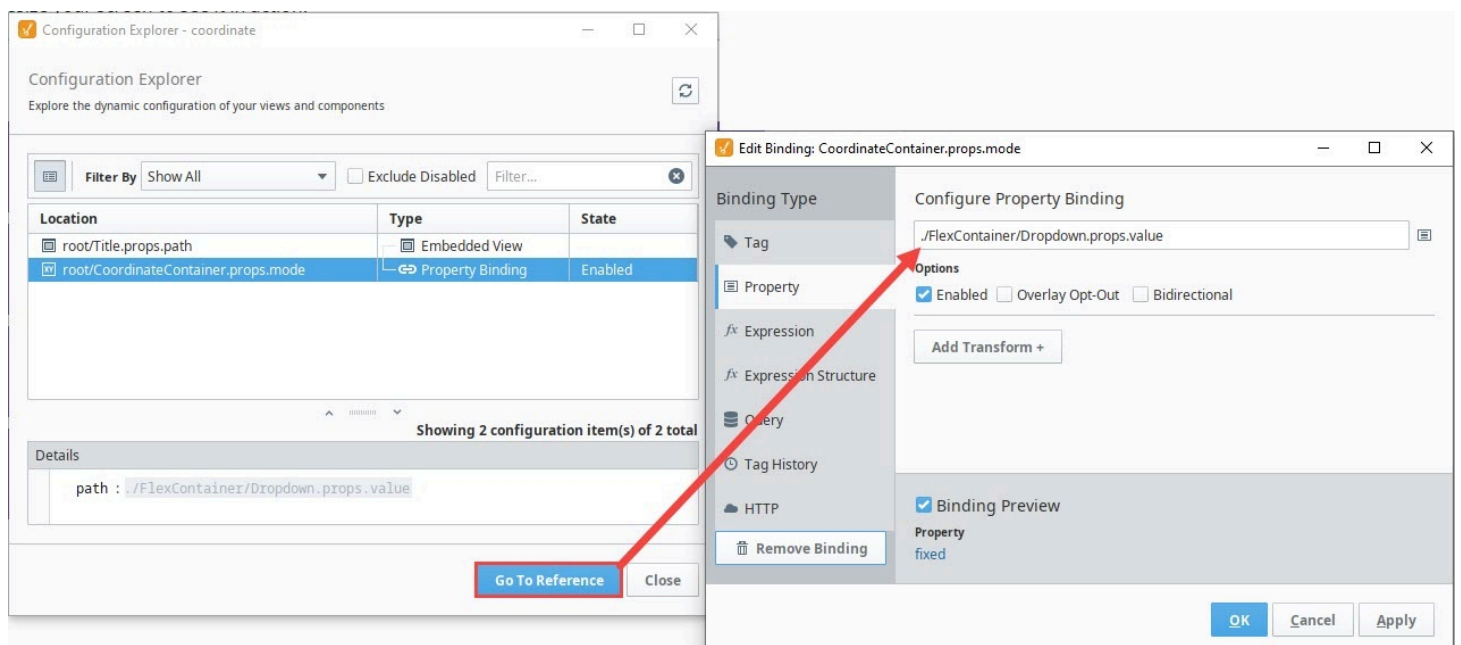
The Configuration Explorer allows you to view the dynamic configuration of all of your views and components. Right-click on a view or component in the Project Browser and select **Configuration Explorer** to examine the following configurations:

- Bindings and Transforms
- Custom Methods
- Embedded Views
- Events and Actions
- Extension Functions

- Message Handlers
- Property Change Scripts

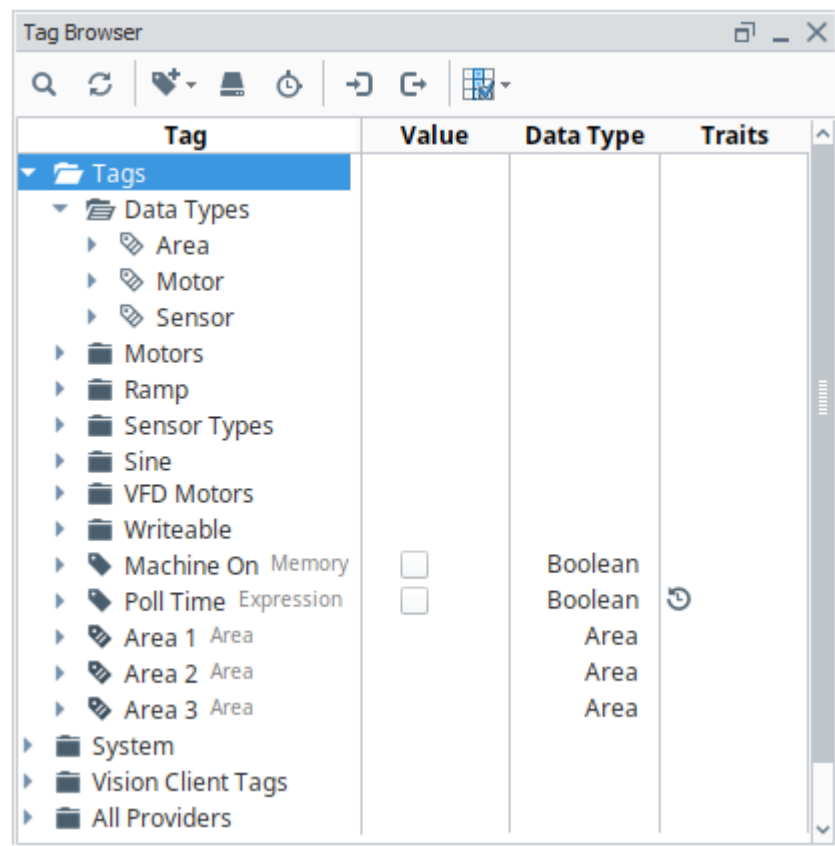


Select a configuration and click **Go To Reference** to highlight the relevant view or component in the Project Browser and, if applicable, open an Edit window to view details about the selected configuration.



Tag Browser

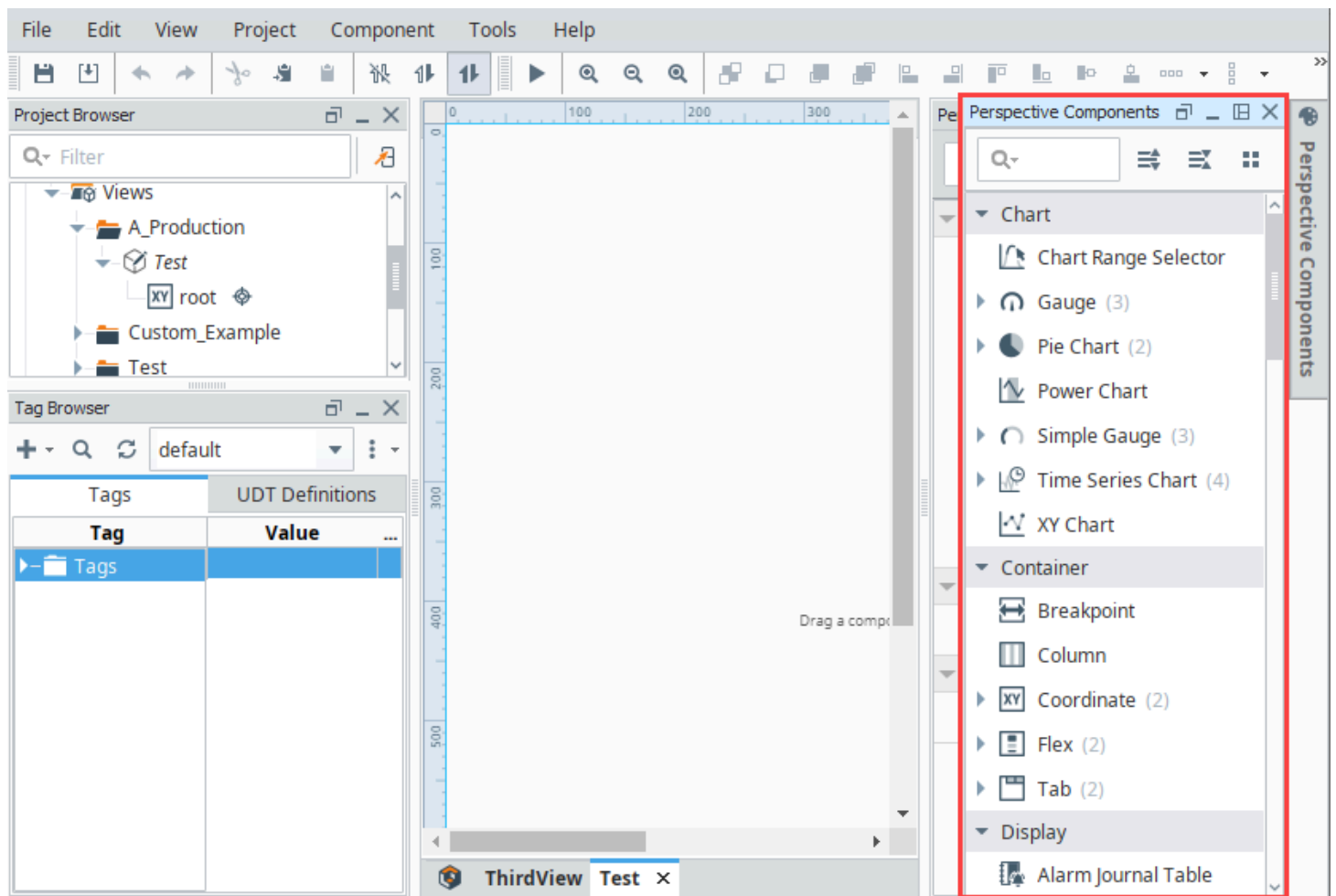
The Tag Browser allows you browse tags in the Designer and OPC servers. In addition, tags can be created, edited, exported and imported directly from the Tag Browser. For more information on tags, the different tag types, and how they work, see [Tag Browser](#).



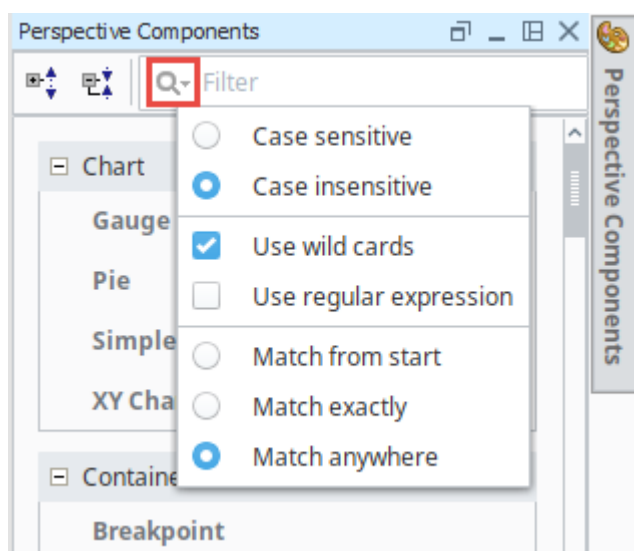
Perspective Component Palette

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.



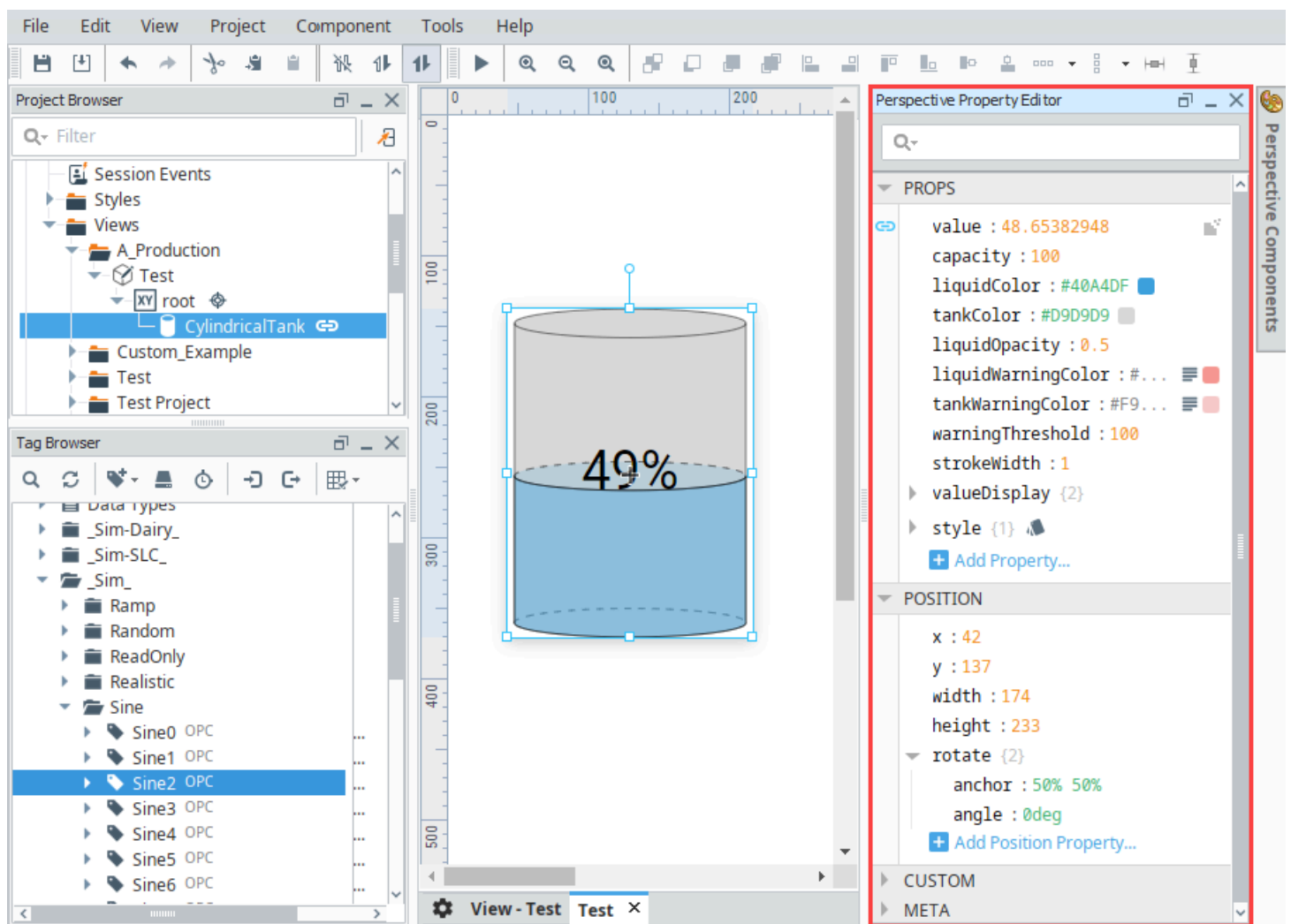
Perspective Property Editor

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

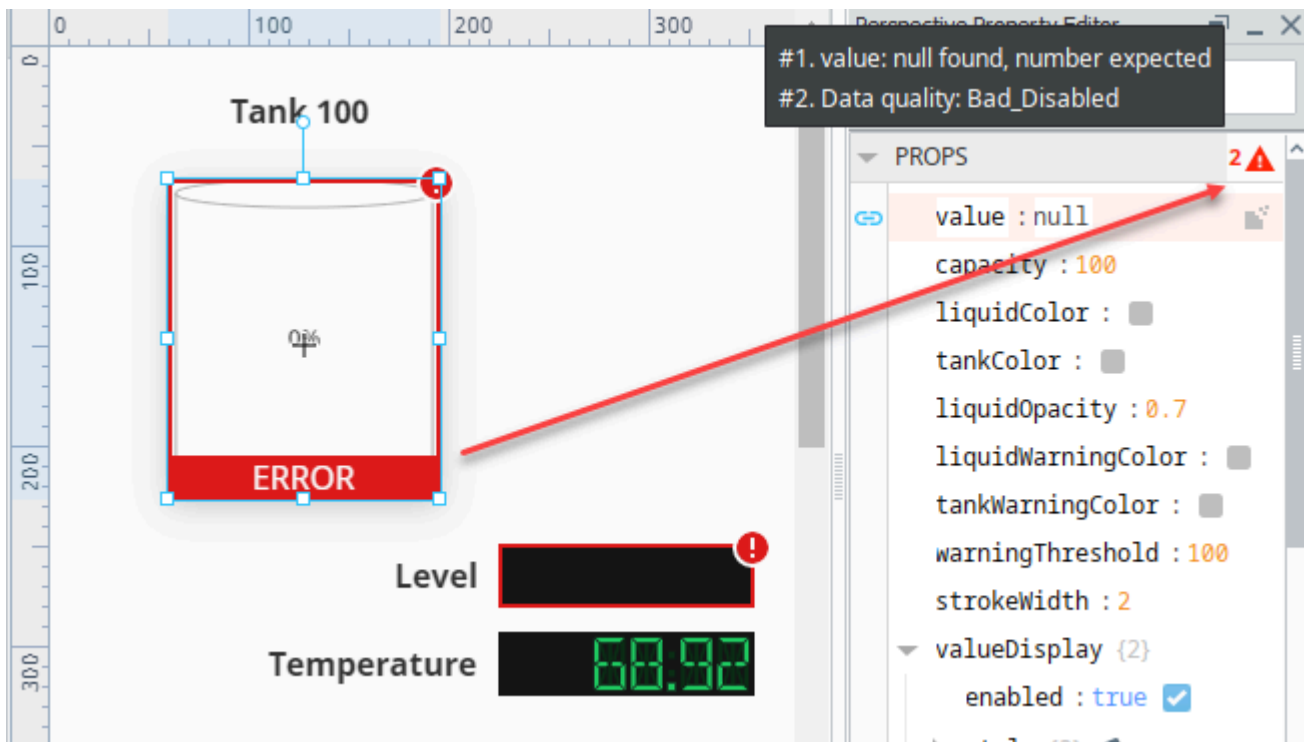
properties available are dependent on which object or component is selected.

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.



The Property Editor will allow show warnings if there is 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.



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 session 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 **Corner 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.

Page Configuration

Shared settings

/ → Production/Plant 1

/plant1-report → Production/

/test → Production/test

+

<

+

>

TopNav

+ Add

Corner Priority

left-right

top-bottom

+ Add

+ Add

+ Add

Recently Modified Views

TopNav

Last edit: admin

05/03/19 11:44 AM

Plant 1

Last edit: admin

05/03/19 10:05 AM

test

Last edit: admin

05/03/19 09:52 AM

Day Shift Staff

Production Manager

Shift Supervisor

Operator 1

Operator 2

Operator 3

RightNav

Last edit: admin

04/03/19 03:51 PM

Plant1 Report

Last edit: admin

04/03/19 03:40 PM

Plant1 Report

RightNav

Plant 1

TopNav

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 horizontal 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 disappear.

You can add multiple vertical and horizontal guides.



The Perspective Designer toolbar also has options for [aligning components](#).

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.



INDUCTIVE
UNIVERSITY

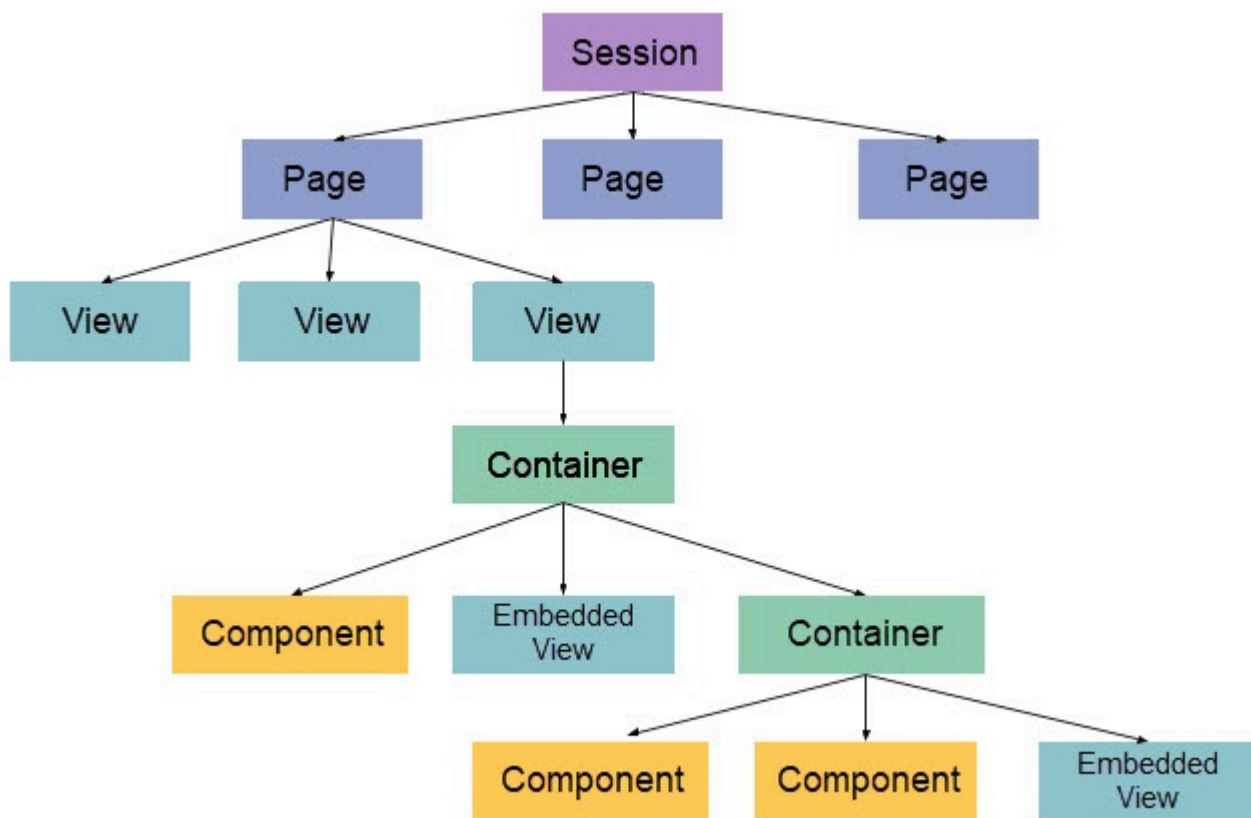
Perspective Project Elements

[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



INDUCTIVE
UNIVERSITY

Launching a Session

[Watch the video](#)

There are several ways to launch a Perspective Session:

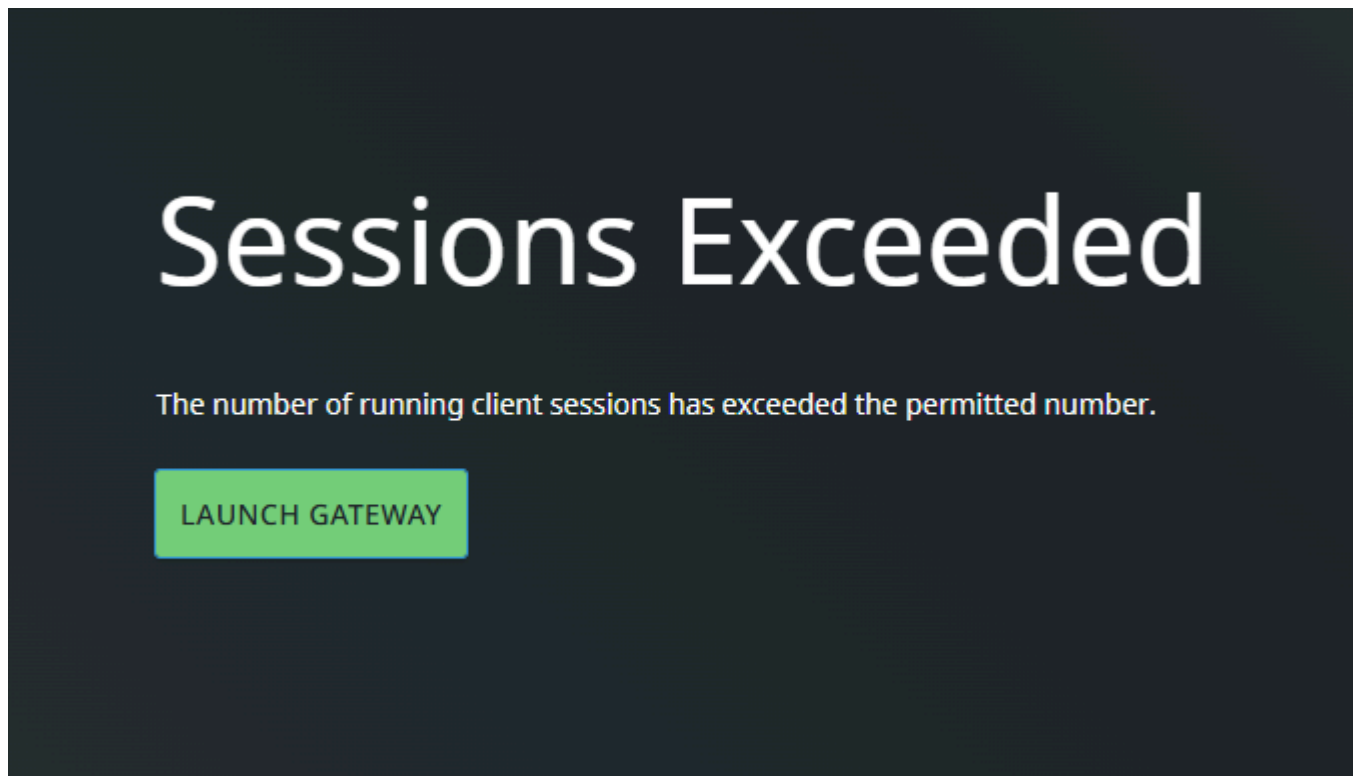
- From the Designer
- From the Gateway Webpage
- By entering the web address of the project in your web browser
- Through [Perspective Workstation](#)

- From the [Ignition Perspective Mobile App](#)

The Perspective Mobile App is a dedicated mobile launcher that opens Perspective Sessions on Android and iOS devices. When **Offline Mode** is enabled in a project, Sessions launched in the app can remain usable even when the device is no longer connected to the Gateway.

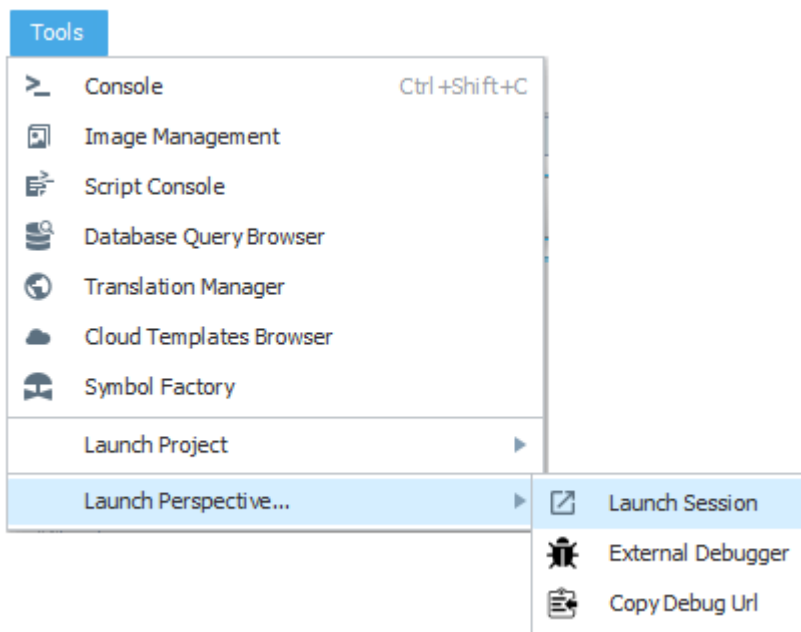
For more information on accessing Perspective projects remotely, refer to the [Accessing Projects Remotely](#) section.

Note that if you are using **Edge Panel**, the maximum number of concurrent Sessions you can have is two. Attempting to open a third will cause a message to appear saying that the number of running Sessions has been exceeded.



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 from the Gateway Webpage

On the Gateway Webpage, navigate to the Home section and click **Perspective > Sessions**. Under the Perspective logo, click **View Projects**. The page will list available Perspective projects. Find your project and click **Launch**. This opens your project in a browser tab.

Session App Bar

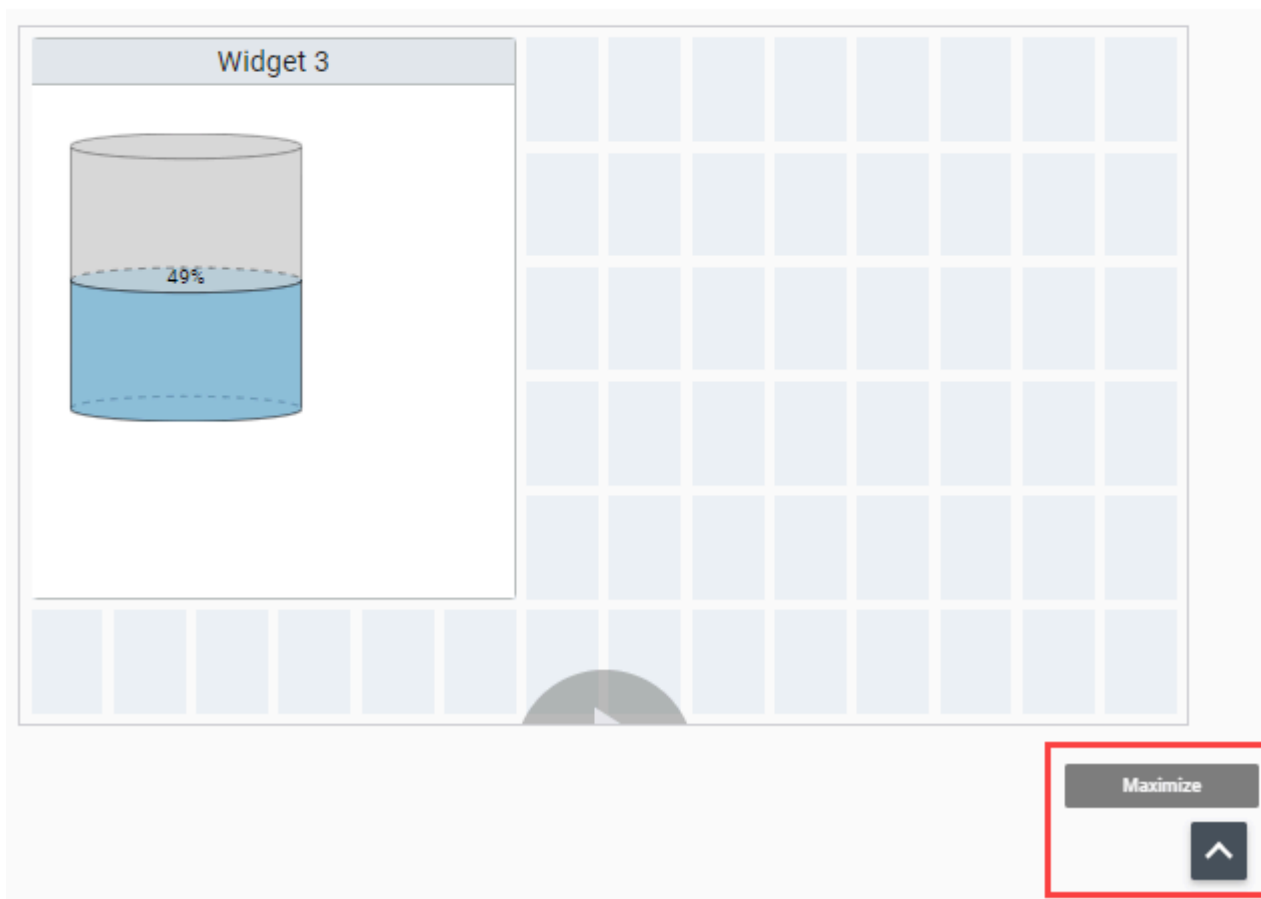


INDUCTIVE
UNIVERSITY

Session App Bar

[Watch the video](#)

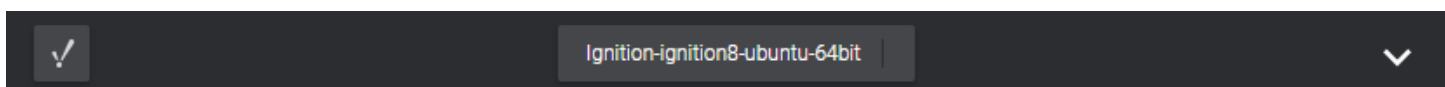
At the bottom of every Perspective Session, there is an expandable app bar with a few options that provide information about Ignition and the current Session. To display the app bar, click the **Maximize**  icon in the lower right corner of the screen.



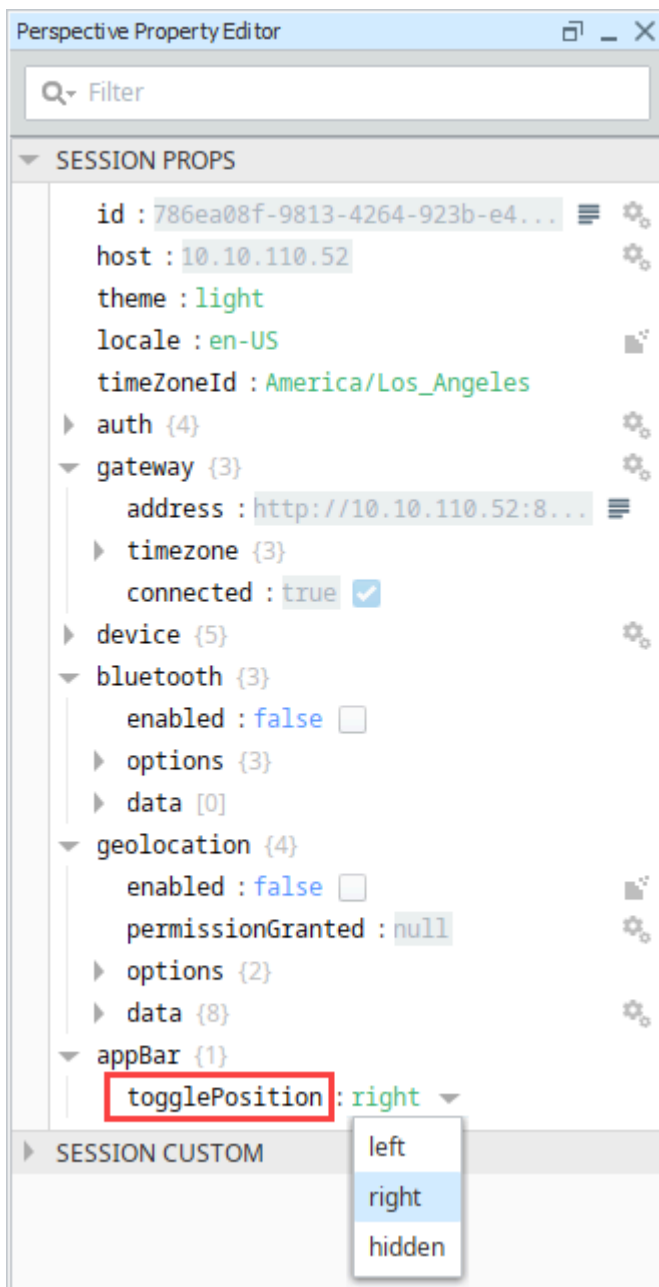
📘 NOTE

If you are using Trial Mode, there will also be a Timer  icon that when clicked displays the [Trial Period timer](#) in the Session Status Popup.

Once expanded, the app bar is displayed at the bottom of the screen.

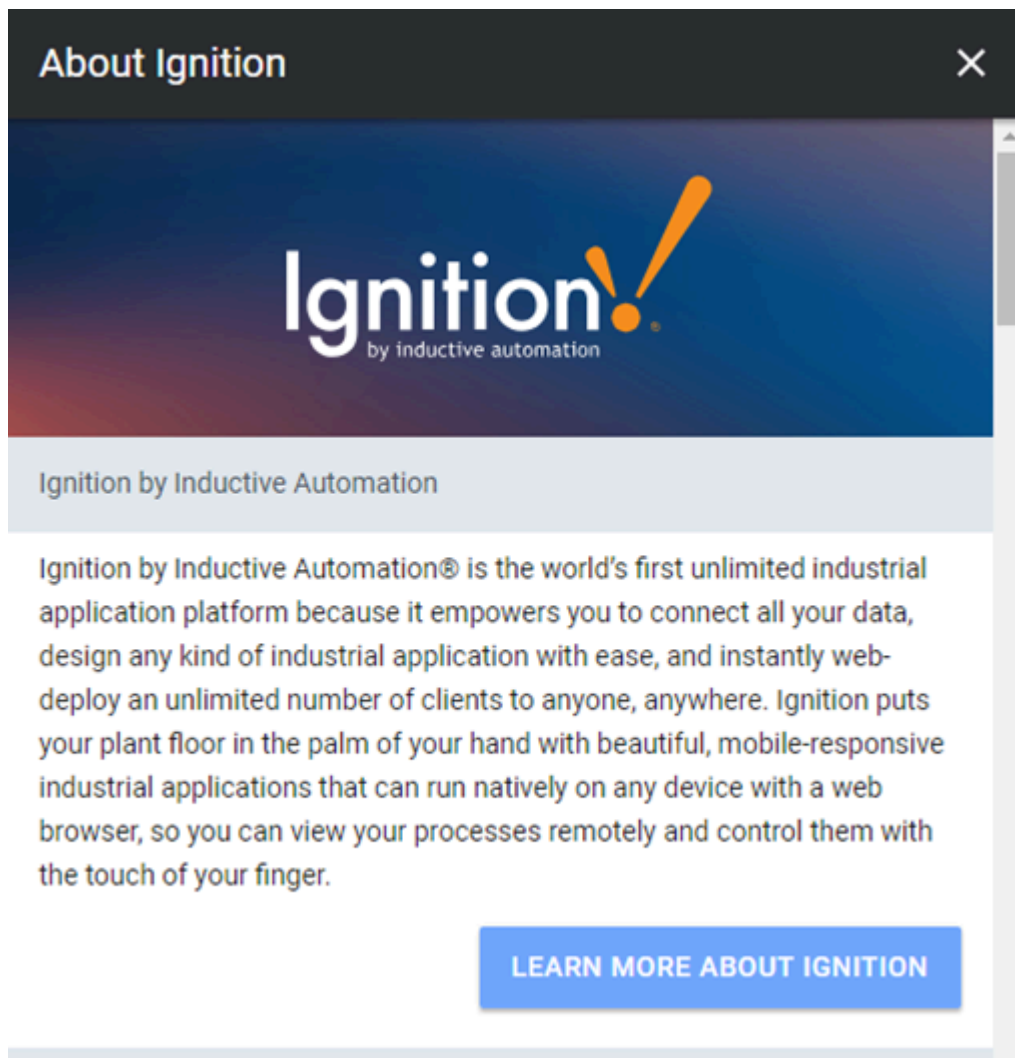


You can change the location of the Maximize/Minimize icon on the app bar in the Property Editor. Scroll down the list of properties in the Perspective Property Editor and expand the **appBar** property. Right-click on the **togglePosition** property and select one of the three options: **left**, **right**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

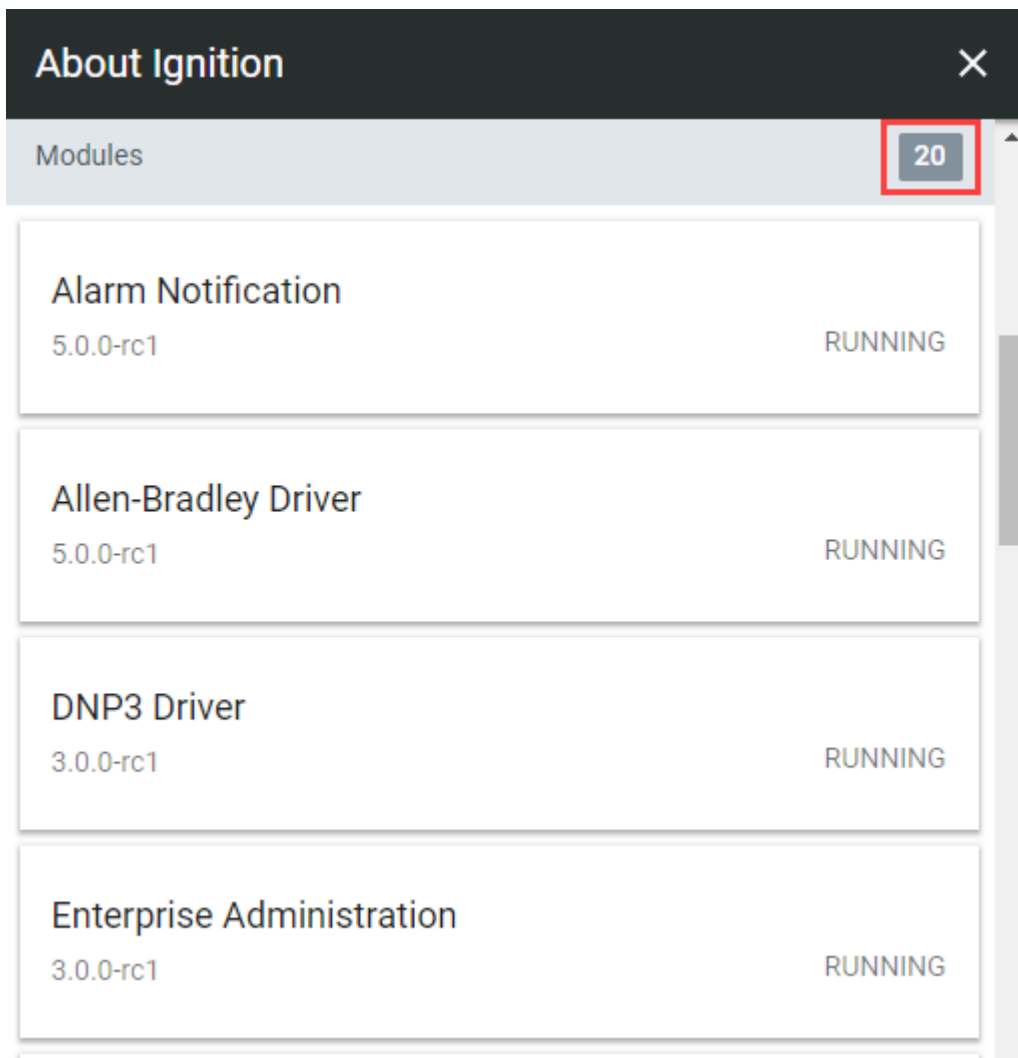


About Ignition Information Popup

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.



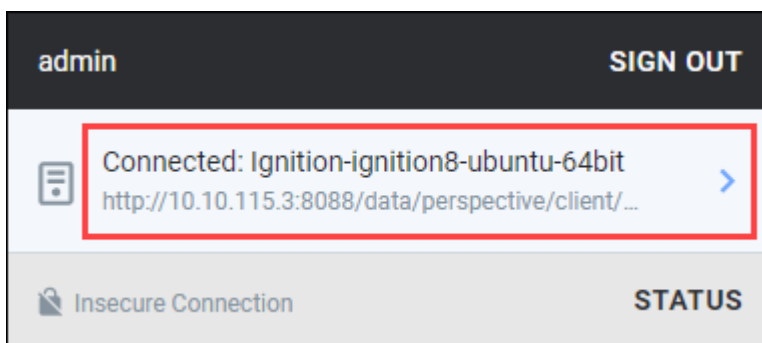
Here you'll see the number of modules installed on the Gateway, along with a list of those a modules, their version numbers, and whether they are currently running.



Session Status Popup

In the middle of the app bar at the bottom of the window, the current Gateway name 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.



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 Details. The Page Details section includes the Page ID and data for the number of view resources, components, and property changes, as well as the Latency regarding the time taken for the Gateway to receive a response from a request

and Up Time since the page was last refreshed or loaded. The **Visit Gateway** button is a shortcut to the Gateway Homepage.

Session Status

GATEWAY

PROJECT



Ignition-ignition8-ubuntu-64bit

✓ CONNECTED

Gateway URL

http://10.10.115.3:8088

VISIT GATEWAY

Session ID

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

Page Details

Page ID: 3a341949

Number of View Resources

10

Number of Components

17

Number of Property Changes

0 /sec

Latency

89 ms

Up Time

5 minutes 50 seconds

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.

Session Status

GATEWAY

PROJECT

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, 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**.

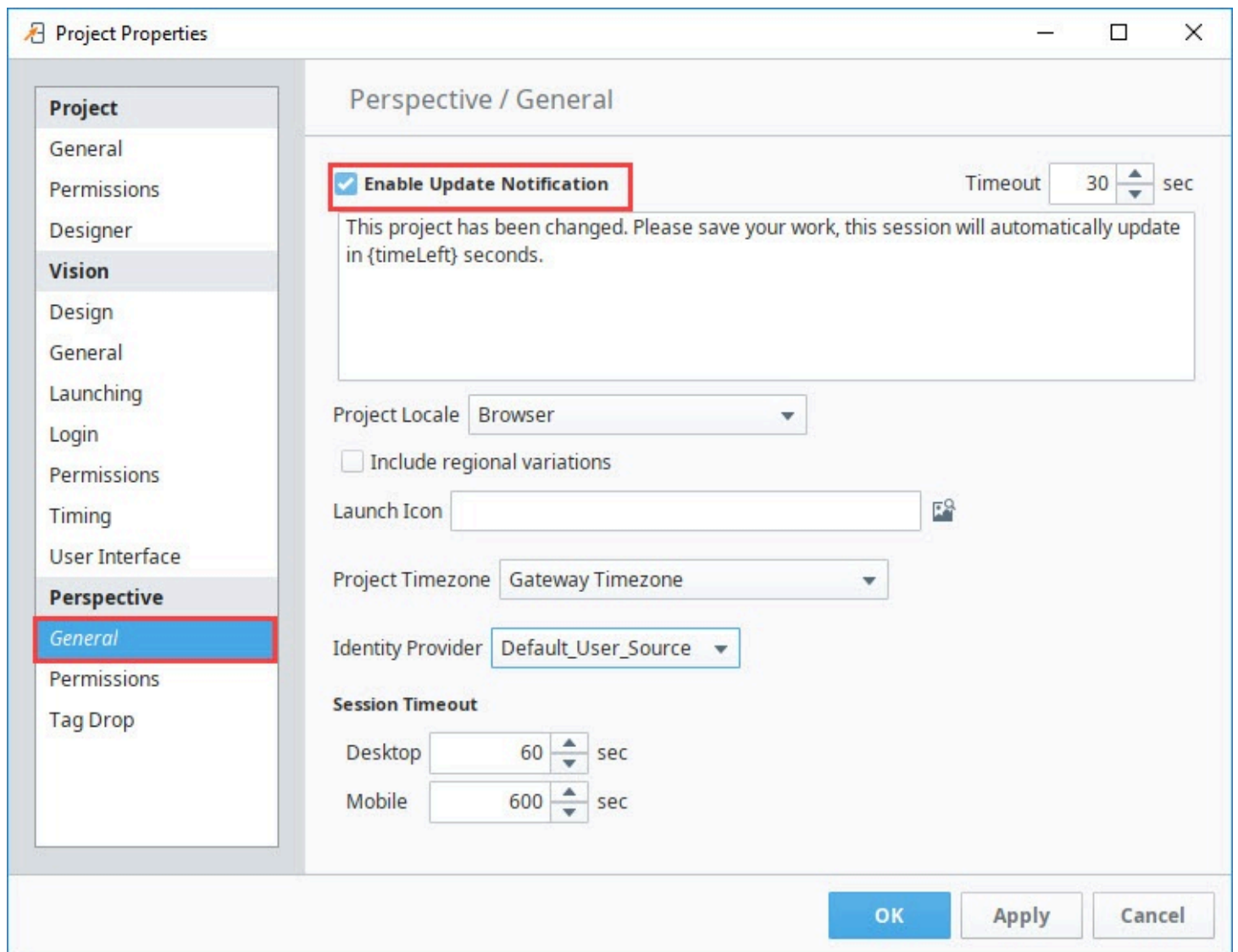


Project Update • a few seconds ago ^

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**.



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 version 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 use features that have been supported and standardized for some time. Overall, we recommend that you keep your browser up to date to take advantage of browser features, fixes, and most importantly for security updates.

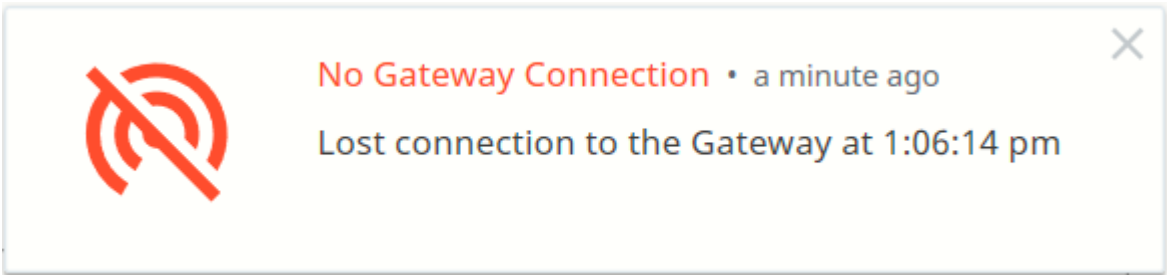
Additionally, Ignition (as well as browsers) is constantly evolving and leveraging new APIs to provide you with the best experience possible, so it's difficult to pin down minimum versions for 100% of Perspective's functionality. We do take care to ensure Perspective Sessions will always work on the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

Basic Session Functionality Requirements for Ignition 8.3.0

Browser	Minimum Version Required
Chrome	92
Firefox	90
Safari	15.4
Edge	92

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.



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.

Ignition Perspective Mobile App



INDUCTIVE
UNIVERSITY

Perspective Mobile App

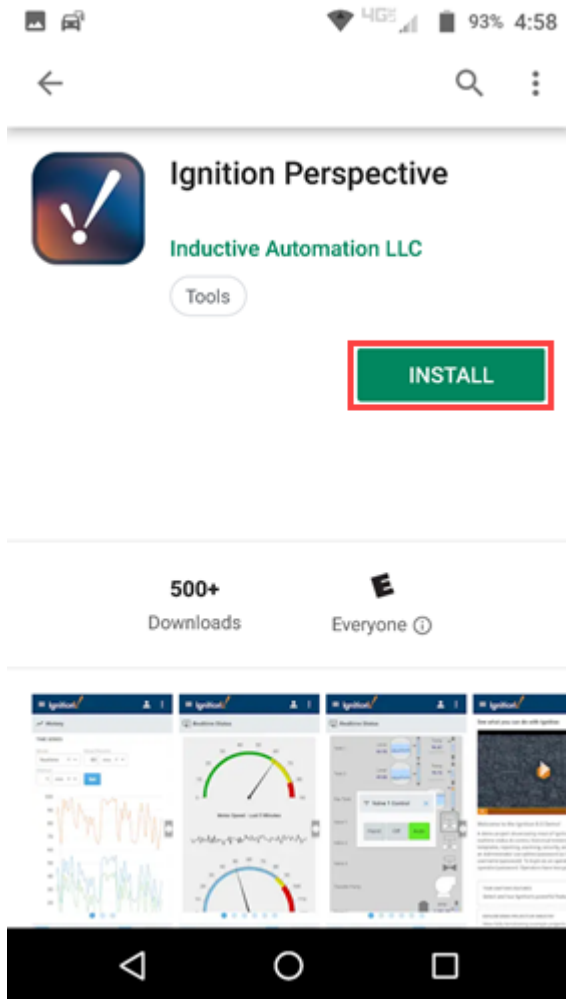
[Watch the video](#)

The Ignition Perspective Mobile 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 Ignition Perspective Mobile 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](#).

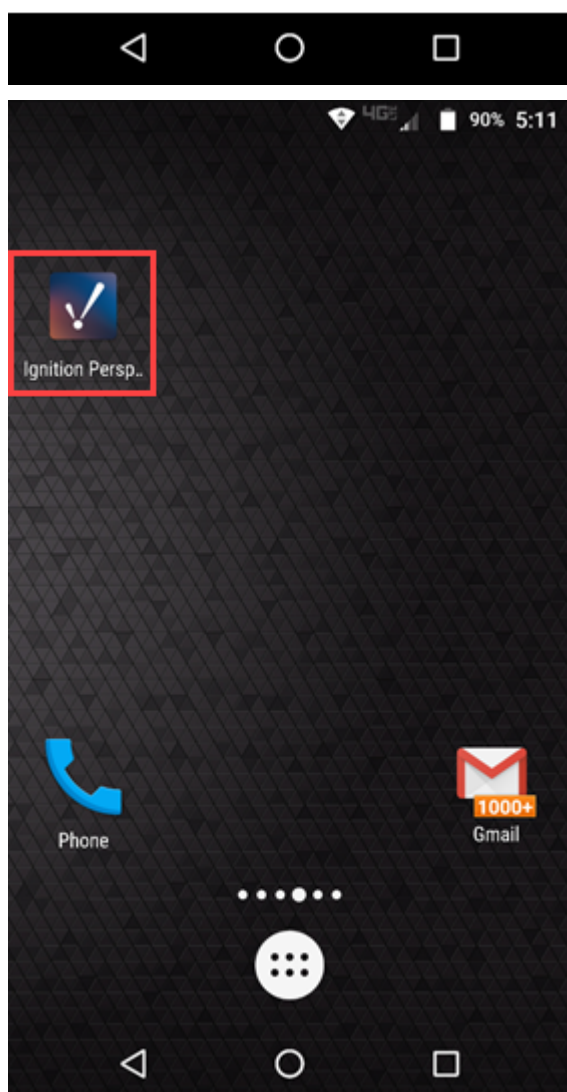
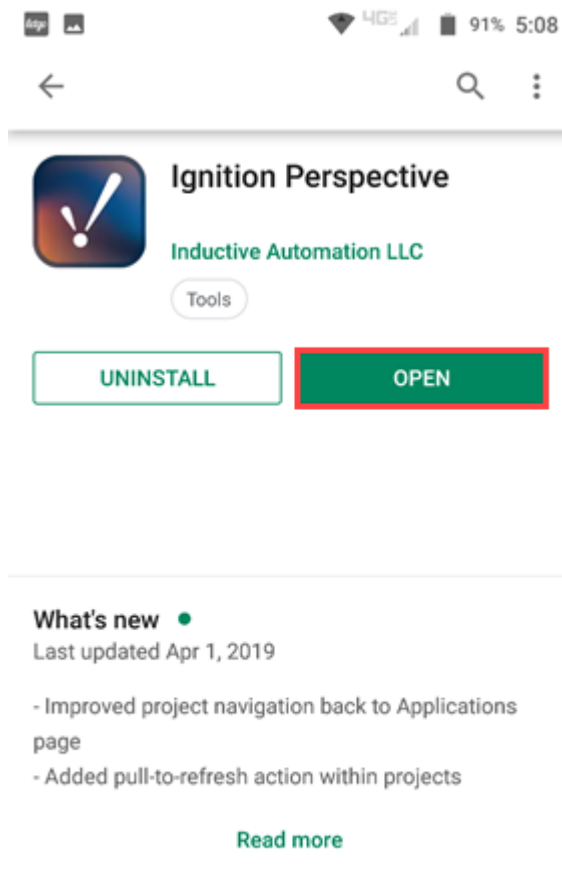
Download the Ignition Perspective Mobile App (iOS or Android)

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

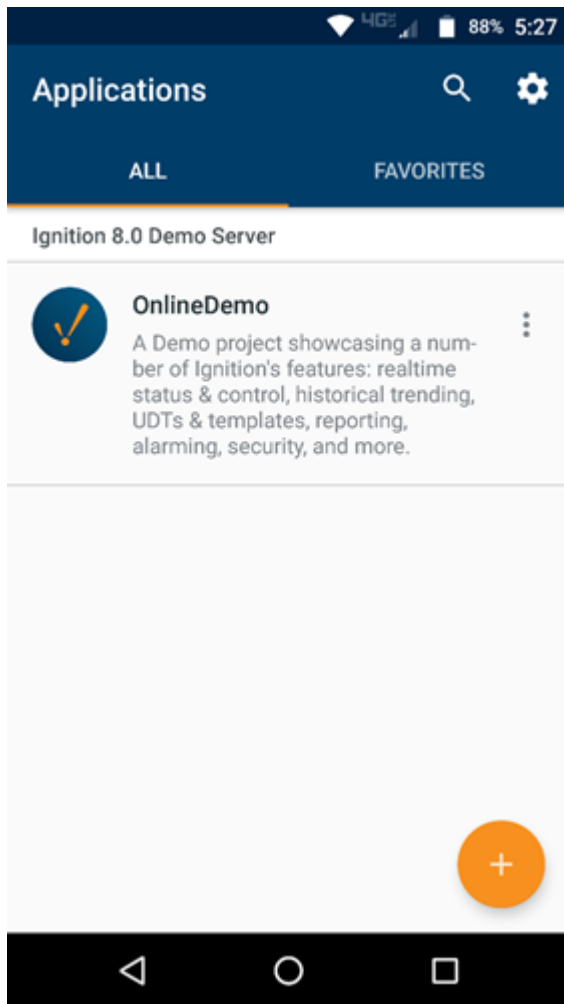
1. In your app store, search for Ignition Perspective.
2. Click the **Install** button.



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



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

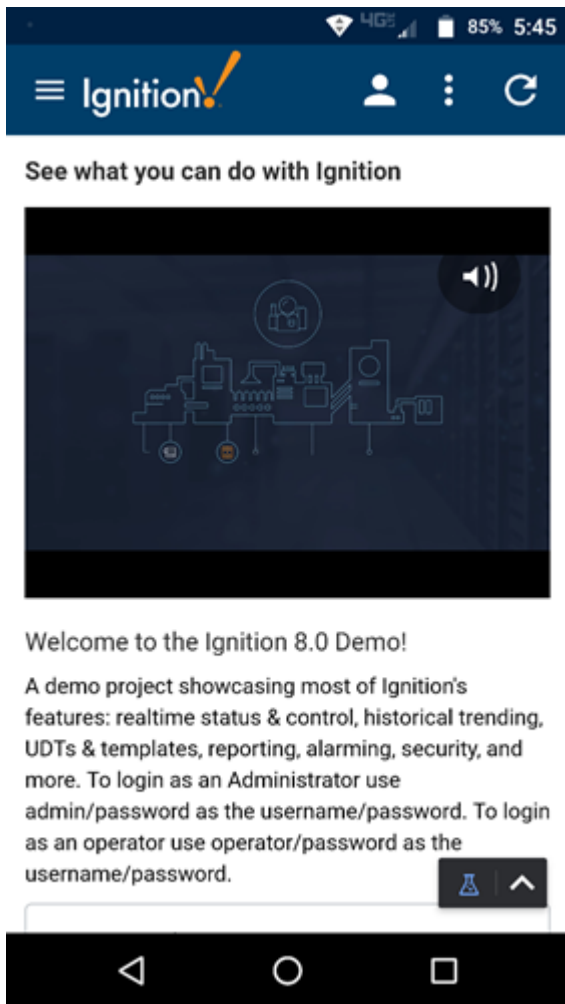


Launch a Perspective Session

On the home screen, you can launch the Public Demo, which is on the Ignition 8.0 Demo Server or launch a project on a different Gateway. For more information on accessing Perspective projects remotely, refer to the [Accessing Projects Remotely](#) section.

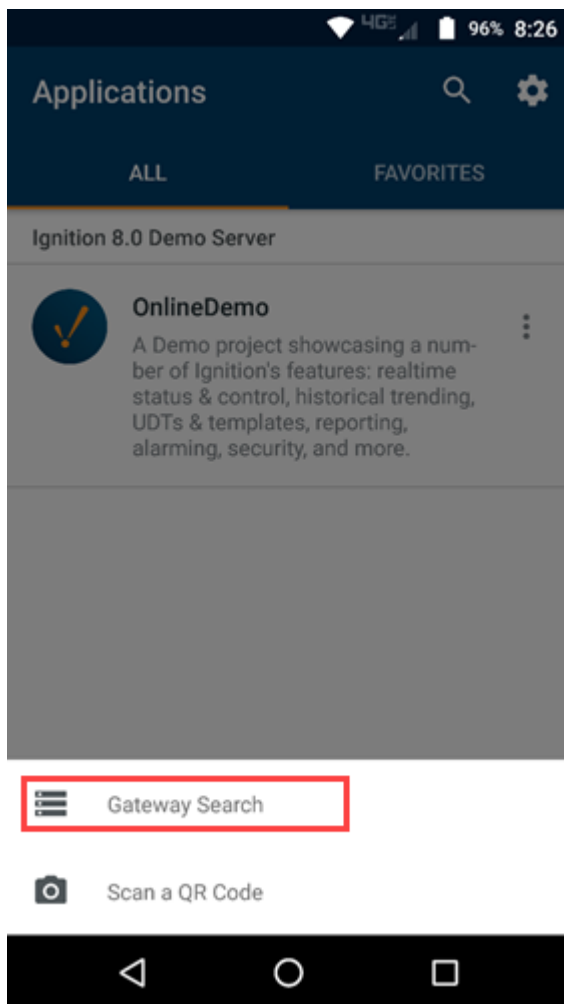
Launch the Public Demo

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



Launch a Project

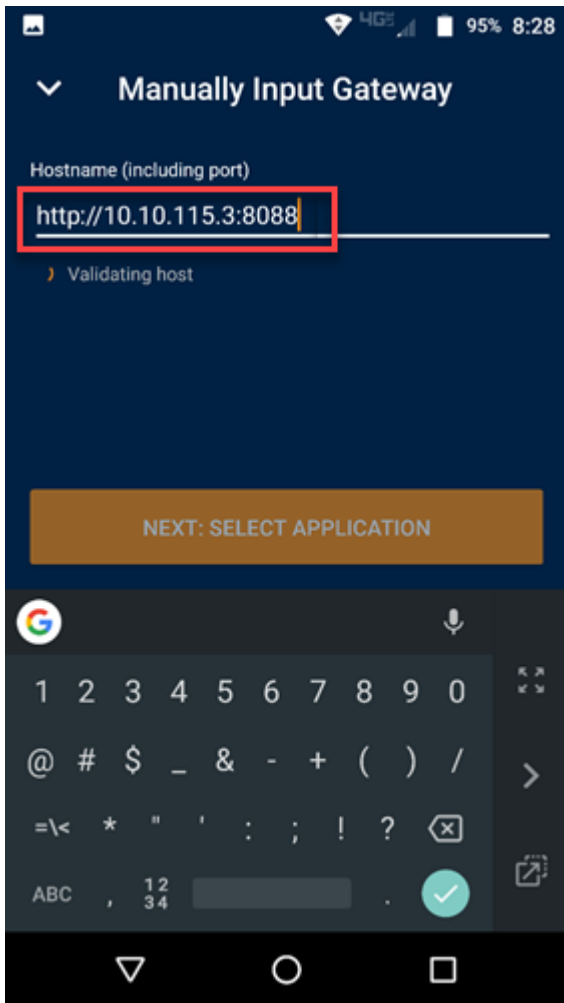
1. To launch a different project, click the **Plus**  icon in the lower right corner of your screen.
2. You then have two options:
 - Scan a QR Code
 - Search for an available Gateway with Perspective projects.
3. For this example, we'll search for available Gateways. Click on **Gateway Search**.



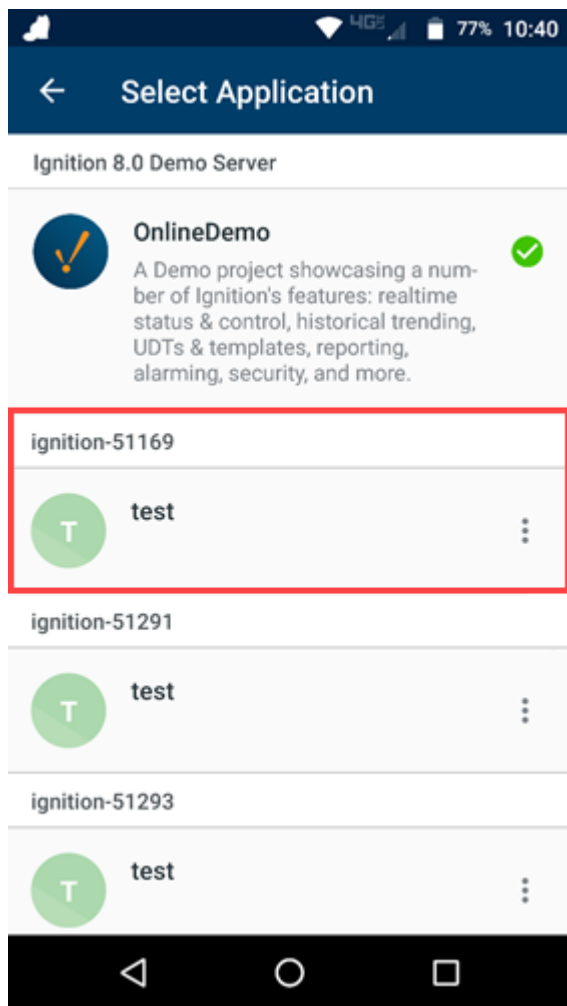
4. Select the Gateway you want to connect to.



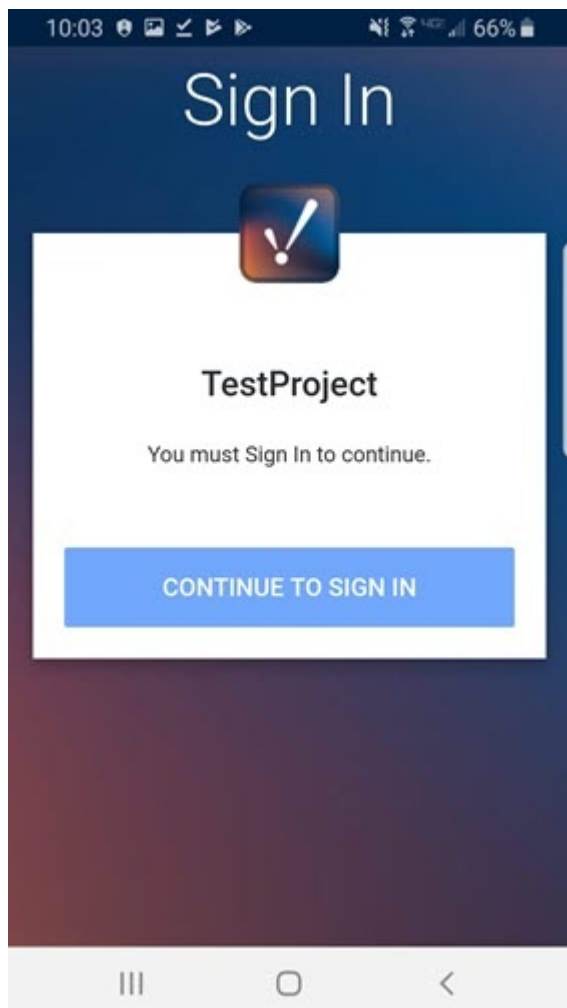
- i. If the Perspective Mobile App doesn't find the Gateway automatically, you can manually input a Gateway host name and port. Click on **Manually Input Gateway**.
- ii. Type in the host name including the port, for example: `http://10.10.115.3:8088` (or `https://10.10.115.3:8043` if using SSL).



5. Click on the project you want to open.



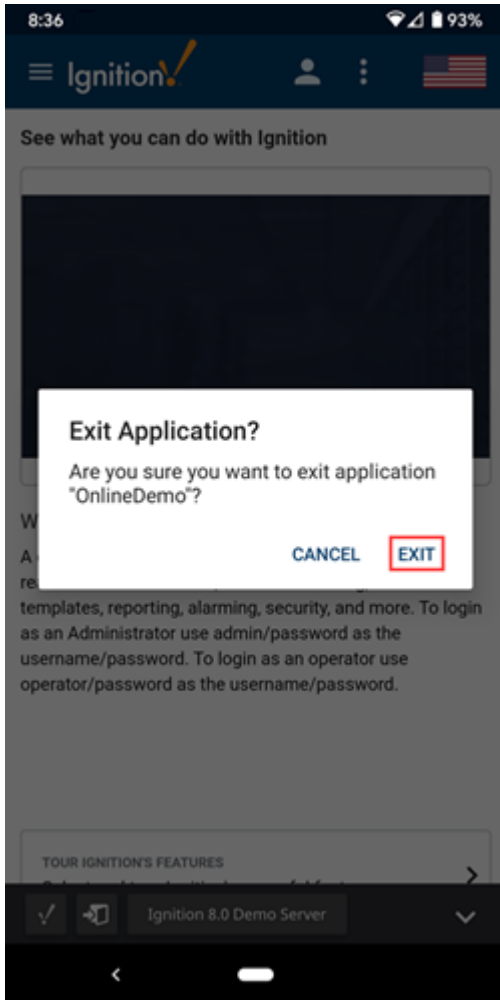
6. If security is set up on the project, you'll be prompted to sign in with your credentials.



Switch to a Different Application

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

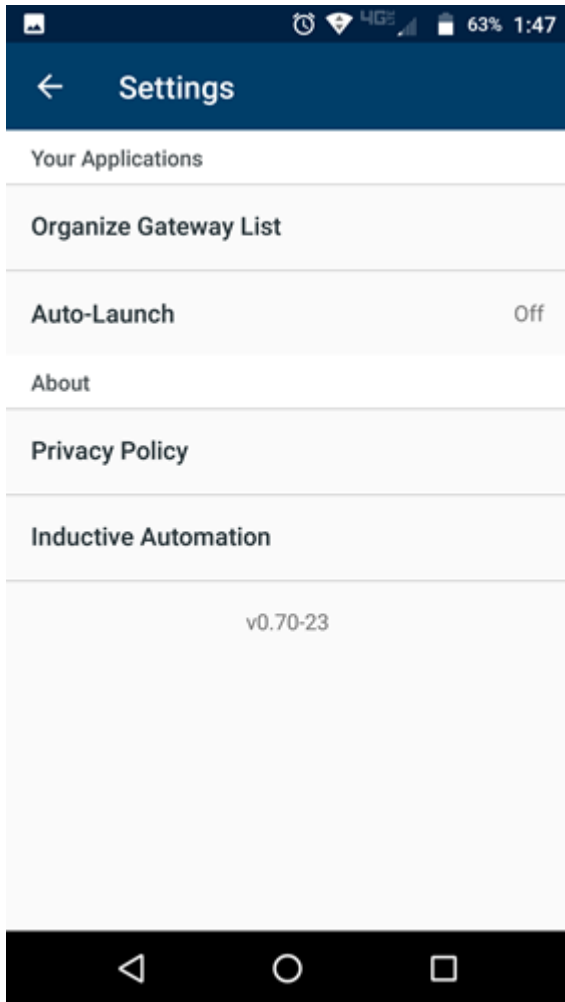
1. Click the **App Bar**  icon in the lower right corner.
2. Click the **Exit**  icon.
3. At the confirmation prompt, click **Exit**.



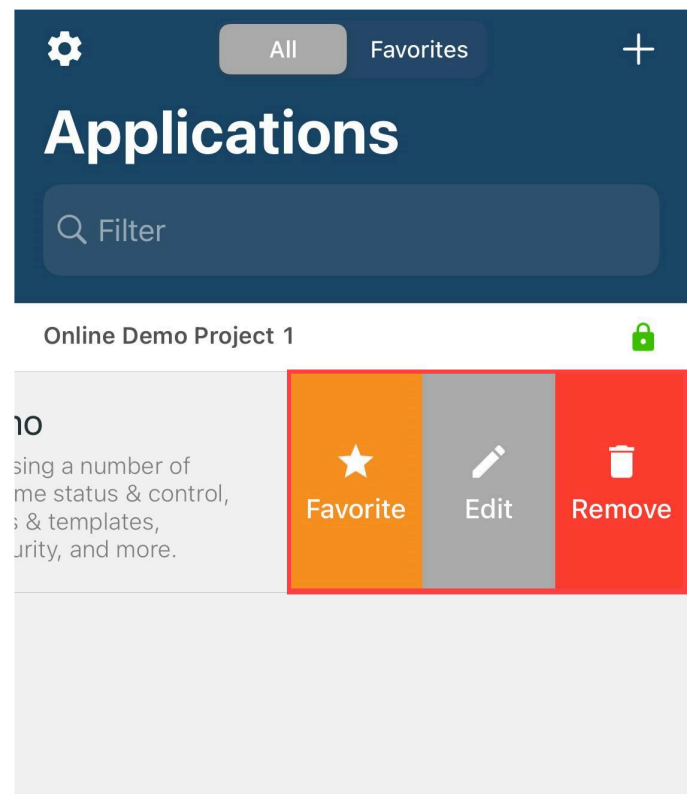
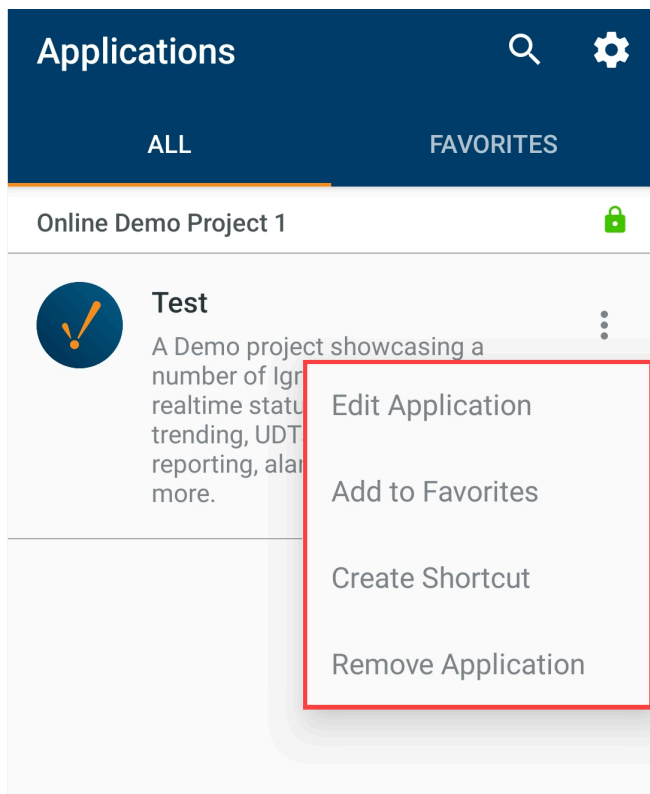
Additional Options

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 Ignition Perspective Mobile 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 Mobile App is opened on this device.



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.
- Enable **Offline Mode** for supported projects.
- 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 ApplicationSAVE



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

CancelManage ApplicationSave



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

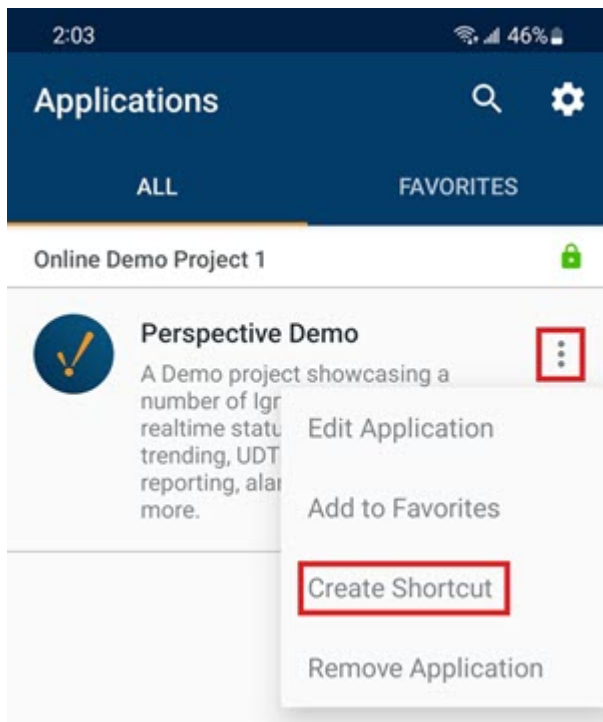
Users can alternatively remove a project by simply tapping the **Remove Application** or **Remove** option, on an Android or iPhone device respectively, from the homepage.

Create Project Shortcuts

If the Ignition Perspective Mobile 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 Mobile App. Long Press instructions are covered below under the second method for creating project shortcuts.

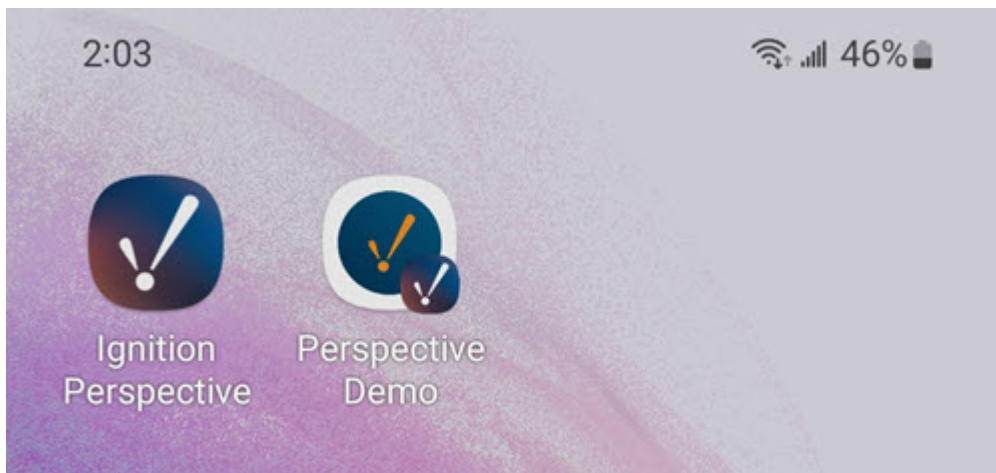
- To create the shortcut, open the Ignition Perspective Mobile App.
- Push the **Three Dots**  icon for your project.

- 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 Mobile App without needing to use the App.

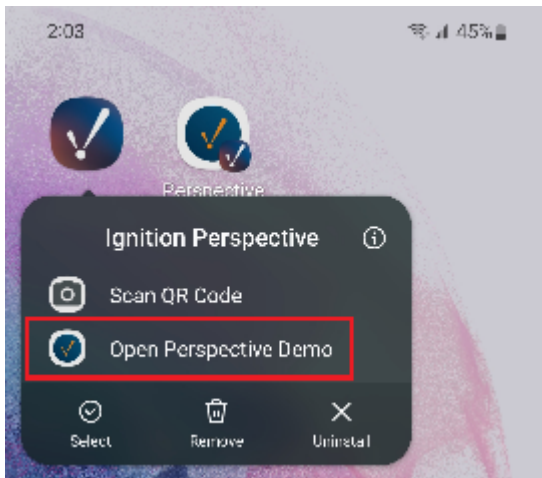


The shortcut icon and App Name will match what is displayed in the Ignition Perspective Mobile App project list.

- 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 Mobile 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 Mobile 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 Mobile 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 Mobile 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.

Key	Description
auto_launch	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. See auto_launch .
hide_demo	If true, the built-in demo project will become hidden.
initial_applications	Allows you to add projects to the main app page. See initial_applications .
initial_gateways	Allows you to add Gateways to the Recent Gateways list. See initial_gateways .

auto_launch

Key	Description
auto_launch_locked	If true, prevents user from disabling auto-launch or changing the auto-launch URL within the app.
prevent_exit	If true, prevents the user from exiting the auto launched project.
auto_launch_url	If set with a project URL, auto-launch is enabled and the app will attempt to launch the specified project on startup. If the Gateway is unavailable but the project is cached and marked Available Offline, the app will launch the offline version of the project instead.

initial_applications

Key	Description
project_url	A URL leading directly to the application, for example: <code>http://www.someurl.com:8088/data/perspective/client/myproj</code> OR <code>perspective://www.someurl.com:8088/myproj</code>
alias	The name for the application, as it will appear in the project list.
is_favorite	Sets this project as a favorite.

initial_gateways

Key	Description
gateway_url	A URL to the Gateway you'd like to add to the list, for example: <code>http://www.someurl.com:8088</code> If a project is marked as Available Offline and cached on the device, the Perspective Mobile App will still launch the project even if the Gateway is unreachable at startup.

Offline Mode

Offline Mode in Perspective allows users to continue interacting with a project even when their device becomes temporarily disconnected from the Gateway. When Offline Mode is enabled, the Perspective module caches necessary project assets locally on the device during initial connection. If a network interruption occurs, users can continue navigating the application and interacting with previously accessed views and their components.

Offline Mode is especially useful for mobile devices, remote installations, and other situations with unreliable connectivity. A project that was opened while connected will continue to run if the device becomes disconnected, but that cached session is limited. You cannot relaunch a project while disconnected, and form submissions will not queue for later delivery. With Offline Mode enabled, the Perspective App stores project resources locally so you can launch projects without a Gateway connection and queue supported interactions such as form submission for automatic delivery once connectivity is restored. This makes it well suited for kiosks, HMIs, and mobile use cases in unstable network conditions.

NOTE

Offline Mode only works in the Perspective App on mobile devices. It is not supported in desktop browsers.

Enabling Offline Mode

Offline Mode is configured in the **Project Properties** panel of the Designer under **Perspective > Offline Mode**. The following configuration options are available.

Enabled

Enables Offline Mode for the project when enabled here and on the **Manage Application** screen in the Ignition Perspective Mobile App. Once enabled, compatible devices running the Perspective App will begin caching project resources.

Security Levels

Defines which users are allowed to use Offline Mode. You can specify one or more required security levels, such as Authenticated or specific roles.

- **All:** The user's security levels must match all selected entries.
- **Any:** The user's security levels must match at least one of the selected entries.

Authentication Token Expiration

Specifies how long a cached authentication token remains valid for offline access. After expiration, the user must re-authenticate online before offline access resumes. Default is 7 days.

- You can also enable **Always ask the IdP to re-authenticate**, which forces the user to log in again every time they connect online, even if the token is still valid.

Themes

Specifies which themes will be cached and available during offline use. Only selected themes will be bundled for offline usage, reducing bundle size.

Available built-in themes include:

- light
- dark
- light-cool
- light-warm
- dark-cool
- dark-warm

If a user selects a theme not listed here, it won't be available while offline.

Languages

Specifies which project translations to include for offline use. This setting helps reduce bundle size by limiting localization assets to selected languages.

Using Offline Mode

When a project with Offline Mode enabled is opened in the Ignition Perspective App, the app caches all necessary project resources locally. This allows users to continue interacting with a project if the network connection drops, and also makes it possible to launch the project without a Gateway connection.

The app provides settings to manage cached projects, delete offline data, and control whether the project should auto-launch offline. For steps on launching and managing projects in the Perspective App, see [Ignition Perspective App](#).

Interaction During Offline Use

Not all functionality is available while offline. The following limitations apply:

- Realtime values, such as tags, alarms, and queries will not update until a network connection is restored.
- Scripts that rely on Gateway events or database access will not run.
- Views that have not been previously loaded will not be available.

Many UI components remain interactive. In Offline Mode, the Form component queues submissions while offline and sends them to the Gateway once a connection is reestablished. In a cached or disconnected session, form submissions do not queue and will fail if attempted without a connection. Other components that support local event scripts, such as **onClick** or **onChange**, will continue to function while offline. Ordering of offline form data submissions may not match the order in which the Gateway receives them, so timestamps should be used when processing or storing form data.

Localization and theme settings should match what's selected in the **Offline Mode** project properties. If a user switches to an unsupported theme or language, fallbacks may not be available while offline.

Session Properties



INDUCTIVE
UNIVERSITY

Session Props

[Watch the video](#)

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.

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.

FileEditViewProjectComponentToolsHelp

Project Browser

Filter

Alarm Notification Pipelines
Sequential Function Charts
Scripting
Perspective
Transaction Groups
Vision
Named Queries
Reports

Tag Browser

Sample_Tags

Tags

UDT Definitions

Tag	Value	Data Type
Ramp		
Random		
ReadOnly		
Realistic		
Sine		
Writeable		

Perspective [samplequickstart]

2.1.20-SNAPSHOT (b2022080515)

Create New View

Page Configuration

Shared settings

/ → home/landing-page
/components → component
/components/:folder/:comp
/ignition-101 → ignitio
/ignition-101/:folder/:fe
/layouts → layouts/land
/layouts/coordinate → 1
/layouts/fluid → layout
/layouts/fluid-max-width

left-

Recently Modified Views

Breakpoint
Column
Coordinate
File
Tab

containers

View

embedded-view

Icon 1
Icon 2
Icon 3
Icon 4

native-app

Perspective Property Editor

Filter

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...

Perspective Components

321 / 1024 mb

Session Properties Table

Name	Description	Property Type
id	Unique Session identifier.	value: string
host	Reflects the connecting system's IP address or hostname. The value of this property is impacted by the gateway's Resolve Client Hostname setting. While Resolve Client Hostname is enabled, the gateway will attempt a reverse DNS hostname lookup. Whatever is returned by this lookup will be set as the value for host .	value: string
theme	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.	value: string
locale	The current locale of this Session, expressed as a BCP-47 language tag , such as <code>en-US</code> or <code>es-ES</code> .	value: string
timeZoneId	Timezone identification code, for example <code>America/Los_Angeles</code>. This sessionProps.timeZoneId is the root property. The other timezone.id properties you may see are gateway.timezone.id and device.timezone.id. These are both read-only representations of what the Gateway and user agent (mobile device, browser, workstation, etc.) report back to the session. You can bind either of these properties to override the sessionProps.timeZoneId by setting the project property Timezone Behavior for Perspective to Gateway or Client. This will 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 react to updates dynamically.	value: string

Name	Description	Property Type
lastActivity	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 property is based off of the gateway's time.	value: string
auth	Represents the user's authentication and authorization for this Session. See auth properties .	object
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 default. See Gateway properties .	object
device	Properties for the device that is running the Session. See device properties .	object
bluetooth	Options and data provided by device Bluetooth services. See bluetooth properties .	object
geolocation	Options and data provided by web or native device geolocation services. See geolocation properties .	object
appBar	Settings relevant to the bottom-docked App Bar, which lists Gateway information. See appBar settings .	object
pipes	An object containing properties that pertain to pipes. See pipes properties .	object
symbols	Settings relevant to the components on the Perspective Symbols Palette . See symbol component properties .	object
googleMapsApiKey	Google Maps API Key. This key is required to use a functional version of the Google Map component.	value: string
address	Represents the IP address of the Session as the gateway sees it.	value: string

Auth Properties

Name	Description	Property Type
authenticated	True if the user is authenticated. False if the user is unauthenticated. Null if the user's authentication status is unknown.	value: boolean
user	Contains information about the user, if they are authenticated. See user .	object
securityLevels	The deepest security levels in the tree granted to the current user, starting with the children of the Public security level. The Public security level is never shown since all Sessions include Public. See securityLevels .	array
idpId	The identity provider's ID. Replaced by the <code>idp</code> property.	value: string
idp	The name of the Identity Provider configuration set on the project.	value: string
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. Ignition supports SAML Response documents. When the project IdP is a SAML IdP, the <code>idpAttributes</code> object will contain an <code>authnResponse</code> property, which holds the SAML Response XML string. See idpAttributes.	object

user

Name	Description	Property Type
id	The IdP's unique identifier for this user. Null if the user is not authenticated.	value: string
userName	The user's username. Null if the user is not authenticated.	value: string
firstName	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.	value: string
lastName	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.	value: string
email	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.	value: string
roles	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.	value: string
timestamp	A timestamp representing the last time the current user authenticated against the Identity Provider.	value: timestamp

securityLevels

Name	Description	Property Type
name	The name for this security level. Must be unique among its siblings.	value: string
children	Security levels which descend from this security level.	array

idpAttributes

Name	Description	Property Type
authnResponse	The SAML Response XML string when the project IdP is a SAML IdP.	value: string

Gateway Properties

Name	Description	Property Type
address	Remote host address of the connected Gateway.	value: string
timezone	Document providing time zone information. See Gateway - timezone .	object
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.	value: boolean

Gateway - 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

Name	Description	Property Type
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.	value: string
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.	value: string
timezone	Document providing time zone information. See device - timezone .	object
userAgent	User agent string of the connected device.	value: string
settings	Array of settings for the device. See device - settings .	object
accelerometer	When continuous read mode is active, represents values retrieved from the accelerometer. See device - accelerometer .	object

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

device - settings

Name	Description	Property Type
pullToRefresh	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

device - 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 Properties

Name	Description	Property Type
enabled	If true, enables bluetooth capability.	value: boolean
options	Bluetooth options. See Bluetooth options .	object
data	Will populate with most recent packets from any detected beacons.	array

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. See filter .	object

filter

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. See altBeacon .	object
eddystone	Eddystone open beacon format. See eddystone .	object
iBeacon	iBeacon format. See iBeacon .	object

altBeacon

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: str

eddystone

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

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: stri

Geolocation Properties

Name	Description	Property Type
enabled	If true, will attempt to populate location data into the 'data' property.	value: boolean
permissionGranted	If geolocation is enabled and a geolocation permission prompt is requested, this field populates true if the user allowed permission. Otherwise, it is false. Read only.	value: boolean
options	See geolocation - options .	object
data	If geolocation is enabled and the device can provide geolocation data, this will hold information about location. See data.	object

geolocation - options

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

geolocation - options - accuracy

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

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

Name	Description	Property Type
togglePosition	The position of the overlaid toggle button that shows the app bar: right, left or hidden.	value: string
about	Settings allowing for more customization of the bottom-docked App Bar. See about .	object

about

Name	Description	Property Type
show	Determine if a custom about page should be shown.	value: boolean
icon	The path of the about button icon.	value: string
path	Path of the view to display in the about modal.	value: string
title	The title of the about modal.	value: string

Pipes Properties

Name	Description	Property Type
autoAppearance	The styling to use when a set of pipes is set to an "auto" style.	value: string
overlapGap	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.	value: numeric

Symbol Component Properties

Name	Description	Property Type
autoAnimationSpeed	Sets the animation speed for any Perspective symbol components that have their animationSpeed property set to auto.	value: string
autoAppearance	Sets the appearance for any Perspective symbol components who have their appearance property set to auto. Options are auto, p&id, mimic, and simple.	value: string

Reverse Proxy Considerations

While a forward proxy proxies on behalf of clients (or requesting hosts), a reverse proxy proxies on behalf of servers. There are many possible use cases for placing Perspective behind a reverse proxy:

- Control what resources the user is able to access
- Direct traffic to a portion of the web server
- Limit which projects are accessible to users via subdomains
- Define an SSL certificate on the reverse proxy

Defining SSL on a Perspective Reverse Proxy

The following example demonstrates how to define SSL on a Perspective Reverse Proxy. For this, we will use **NGINX** to define an SSL certificate and key. NGINX is a popular choice for reverse proxies, though other open source servers such as **Varnish**, **Apache**, and **Traefik** provide similar functionality.

Installation

Install NGINX or NGINX Plus following the relevant [installation instructions](#). Obtain a [Signed Security Certificate](#) (SSL) and place the SSL certificate and key in the directory where you installed NGINX under `etc/nginx/ssl/`.

Configuring NGINX

In this example, we will create an `nginx.conf` file that specifies the configuration of our server. This section details the modules and properties that are required to configure Perspective behind a reverse proxy; for a more general overview of a proxy server configuration, see the [NGINX Admin Guide](#).

Required NGINX Modules

Module	Documentation
<code>ngx_http_core_module</code>	https://nginx.org/en/docs/http/ngx_http_core_module.html
<code>ngx_http_proxy_module</code>	https://nginx.org/en/docs/http/ngx_http_proxy_module.html
<code>ngx_http_ssl_module</code>	https://nginx.org/en/docs/http/ngx_http_ssl_module.html

Websocket Proxying

For compatibility with Perspective's Websocket connections, we change the HTTP version to 1.1. In order to pass **hop-by-hop** headers from the client to the proxied server, the "Upgrade" and "Connection" headers must be passed explicitly in the configuration file:

```
proxy_http_version 1.1;
proxy_set_header Upgrade $http_upgrade;
proxy_set_header Connection "upgrade";
```

Custom Headers

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

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

To allow requests with custom headers, you will need to enable `proxy_pass_request_headers` in your configuration file.

```
location / {
    proxy_pass https://localhost:8043/data/perspective/client/demo/;
    proxy_http_version 1.1;
    ...
    proxy_pass_request_headers on;
    ...
}
```

Additional Location Contexts

Perspective sessions require resources within several folders that must be defined as additional location contexts within the configuration file. The following additional locations are required to enable the Perspective session to run the reverse proxy.

```
server {  
    ...  
    location ^~ /data/ {  
        ...  
    }  
    location ^~ /system/ {  
        ...  
    }  
    location ^~ /res/ {  
        ...  
    }  
    location ^~ /idp/ {  
        ...  
    }  
    location ^~ /.well-known/ {  
        ...  
    }  
}
```

Example Configuration

nginx.conf

```
server {
    listen 443 ssl;
    listen [::]:443 ssl;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    server_name demo.xxx.com;

    underscores_in_headers on;

    # Specify the paths to your SSL certificate and key:
    ssl_certificate /etc/nginx/ssl/reverse_proxy.crt;
    ssl_certificate_key /etc/nginx/ssl/reverse_proxy.key;

    # The next few location contexts define how to process resources required by
    the Perspective session.
    # If the longest matching prefix location has the "^~" modifier then regular
    expressions are not checked.
    location ^~ /data/ {
        proxy_pass https://localhost:8043/data/;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_pass_request_headers on;
        proxy_cache_bypass $http_upgrade;
    }

    location ^~ /system/ {
        proxy_pass https://localhost:8043/system/;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_pass_request_headers on;
        proxy_cache_bypass $http_upgrade;
    }

    location ^~ /res/ {
        proxy_pass https://localhost:8043/res/;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_pass_request_headers on;
        proxy_cache_bypass $http_upgrade;
    }
}
```

```

location ^~ /idp/ {
    proxy_pass https://localhost:8043/idp/;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
    proxy_pass_request_headers on;
    proxy_cache_bypass $http_upgrade;
}

location ^~ /.well-known/ {
    proxy_pass https://localhost:8043/.well-known/;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
    proxy_pass_request_headers on;
    proxy_cache_bypass $http_upgrade;
}

# Here we are setting the protocol type, address, port, and uri (optional)
# that will be the destination of our proxied server:
location / {
    proxy_pass https://localhost:8043/data/perspective/client/demo/;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
    proxy_pass_request_headers on;
    proxy_cache_bypass $http_upgrade;
}
}

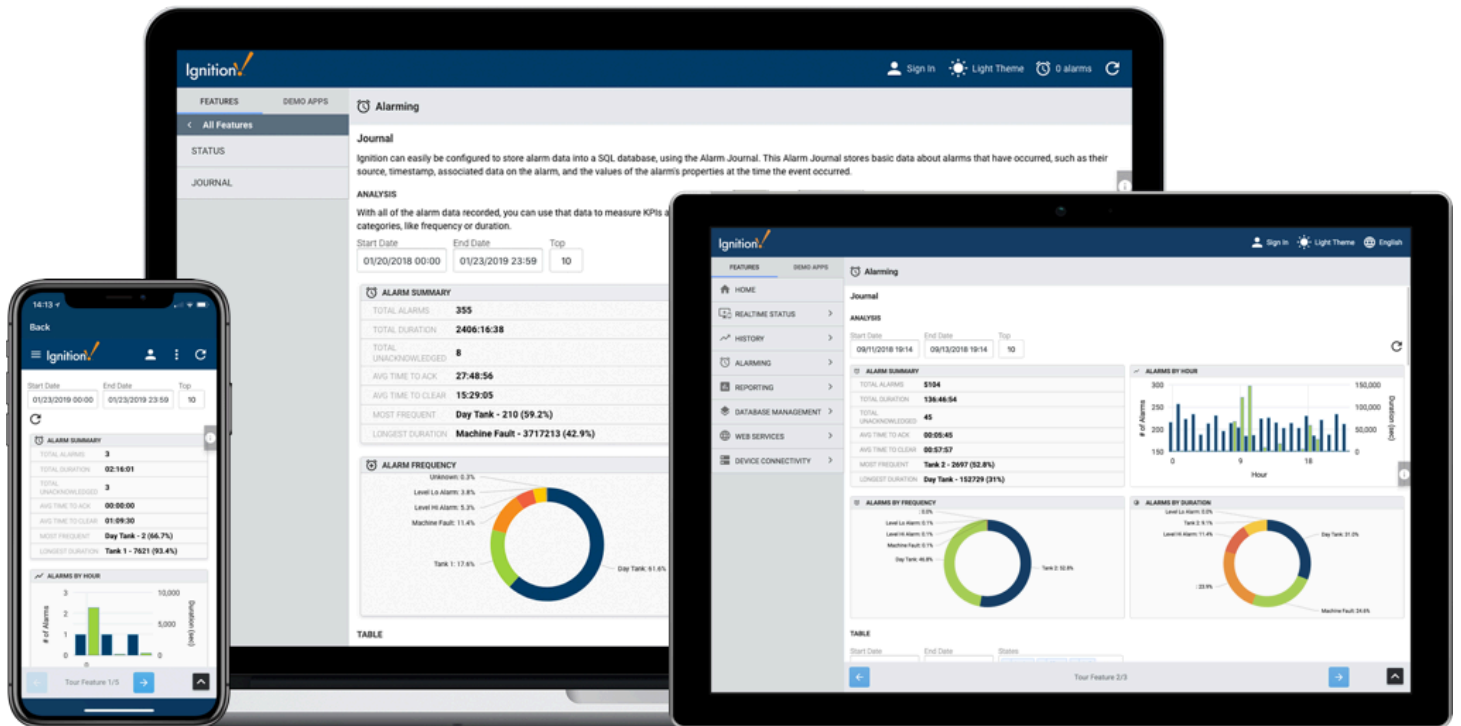
```

Support for Proxy Forwarding of Remote HTTP Client Details

A **Use Proxy Forwarded Headers** setting is available on the Gateway **Network > Network Settings > Web Server** page under the HTTP Settings section. It allows the Gateway to use a reverse proxy's HTTP client details instead of the reverse-proxy itself.

Perspective Design Tips

Designing mobile-responsive applications involves several considerations. Your design should work across a range of devices, from small smartphones to large desktops. While no single strategy fits every project, you can create a layout that adapts to different screen sizes based on your content and use case.



With devices ranging from smartphones to tablets, it's important to understand how your project displays across screen sizes. This page outlines key principles to consider when designing in Perspective. These guidelines can also support your organization's internal design processes.

Before getting started, you may encounter terms like **mobile optimized**, **responsive design**, and **user experience** during project planning.

- **Responsive design** uses flexible layouts that adjust to any screen size.
- **Mobile optimized** means the layout is adapted for smartphones and tablets.
- **User experience (UX)** affects how users interact with your project, including performance, navigation, and visual clarity.

Start by defining clear goals. Identify the problem you're solving and understand your users' needs. Ask:

- What devices will users access the project on?
- Are they viewing data, entering data, or controlling equipment?

- What do components need to do?
- Should the project work offline?

Capture requirements early and observe workflows where possible. Requirements may evolve during the design process.

Designing for Mobile-Responsive Applications

Perspective supports both desktop and mobile design. Whether users are seated at a desk, walking the floor with a tablet, or responding to alerts remotely, your project should adapt to their environment.

Designing for multiple screen sizes early in the project is more efficient than adding support later. A responsive design adapts to different screen sizes without requiring multiple versions of the same project.

Responsive design is a layout strategy that dynamically adjusts the position and size of components. You can resize your browser window to see how a Perspective project rearranges its layout. On mobile devices, this happens automatically based on available space.

Understanding Views and Containers

Views and containers are the foundation of Perspective projects. They define how components are arranged and how your project behaves across devices.

- A **view** contains components and represents a logical unit of design.
- A **container** holds other components and defines a layout strategy.

Each container type responds differently to changes in screen size. Choose the container that best supports your layout goals.

Experiment with different container types to see how they respond when resized.

For more details, see the [Perspective Containers](#) section.

Design Principles

The following principles help you build effective designs across screen sizes.

Touch and Ergonomics

Mobile devices rely on touch input, so interactive elements must be easy to tap.

Use these general guidelines:

- **Minimum touch target size:** 25 px
- **Optimal touch target size:** 40 px
- **Minimum spacing between elements:** 10 px

Also consider how users hold their devices. The **thumb zone**, or the lower center area of the screen, is typically easiest to reach. Place primary actions in this area. Place less-used or destructive actions where accidental taps are less likely.



Test your design on different devices and screen sizes. When possible, involve users in testing to get direct feedback.

Mobile-First Design Approach

Mobile-first design starts with smaller screens and builds upward. This approach helps prioritize essential features and space efficiency.

Benefits of mobile-first design:

- Better performance on smaller devices
- A single, consistent data model across platforms
- Lower long-term maintenance

Fluid Content

Fluid layouts provide a consistent experience across screen sizes. These layouts can be achieved with Perspective containers, which provide a way to organize and position components within a view. Using containers effectively helps avoid clutter on small screens and empty space on larger ones. Perspective containers help manage how components are arranged by:

- Building responsive layouts
- Reusing views across projects
- Adjusting content dynamically based on container size

Design for flexibility. Layouts should adapt without losing clarity or functionality.

Content as UI

Content and design work together to create an intuitive user interface.

To build a content-driven UI:

- Keep it simple
- Use consistent layout and spacing
- Apply common UI elements to build familiarity
- Use visual cues to indicate behavior

Think of content as part of the interface. Users should be able to understand what they see and how to interact with it.

Declutter

Avoid overwhelming users with unnecessary information. Too many elements reduce clarity and increase cognitive load.

Keep the interface focused:

- Display only essential content
- Use minimal UI elements
- Prioritize clarity and usability

A clean interface is easier to navigate, especially on mobile devices.

Pages in Perspective



INDUCTIVE
UNIVERSITY

Creating Pages

[Watch the video](#)

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.

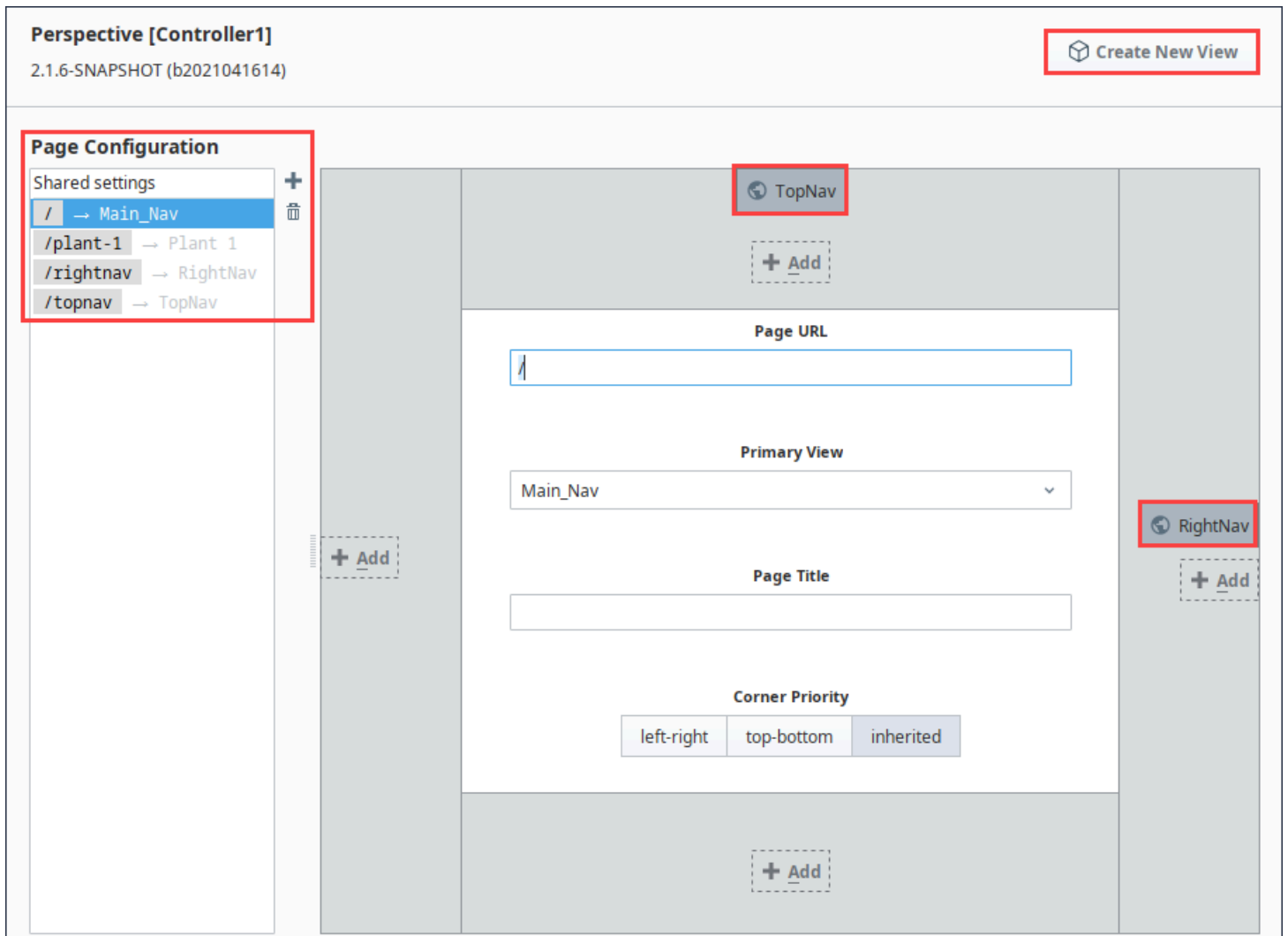
Page Configuration

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.

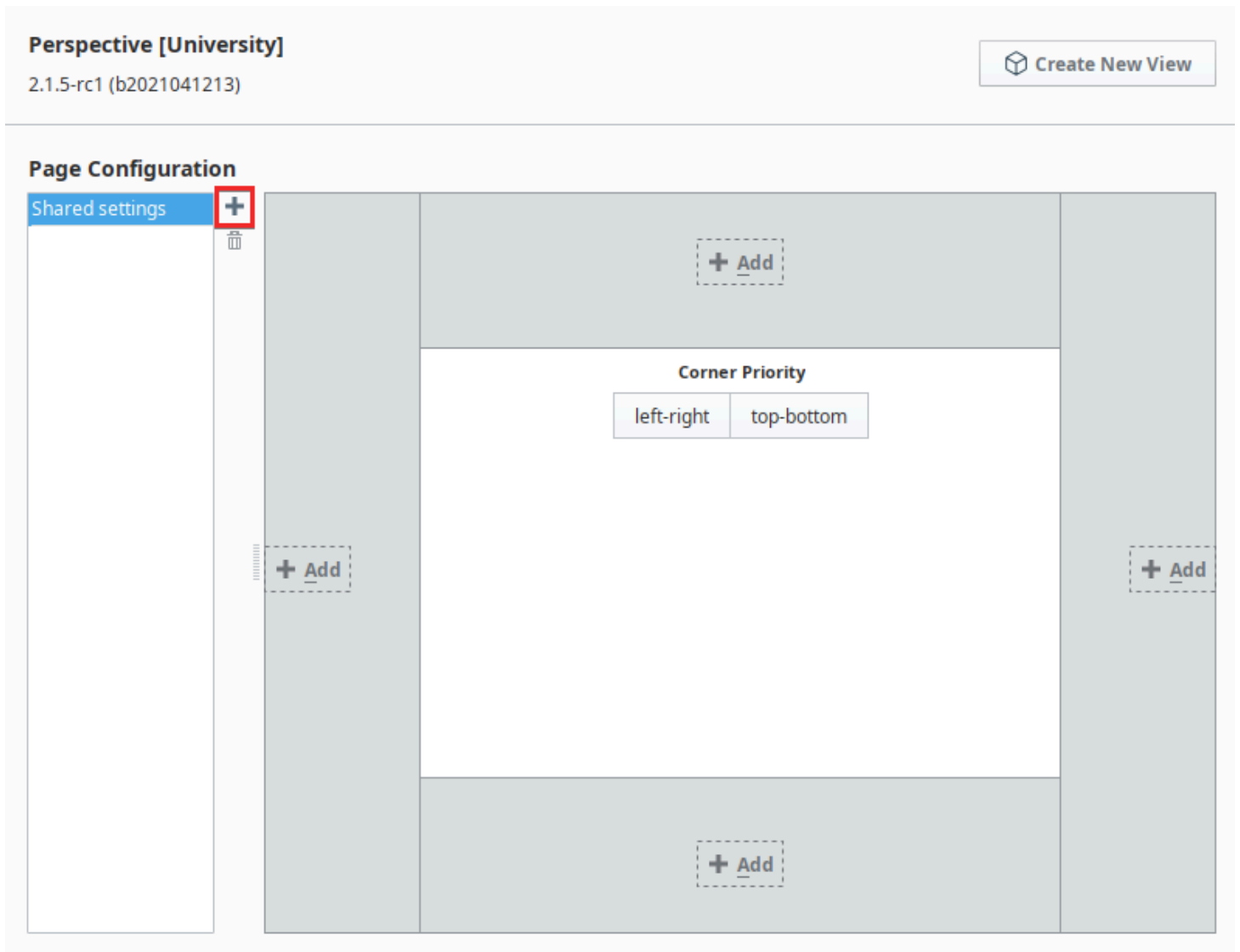
Below is an image of a Page Configuration screen. The Page Configuration column on the left side shows all the Page URLs assigned to views in your project. You can click on the **Shared Settings** to see all docked views that are shared across all pages of your project. To see docked views that are specific 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 Page Configuration page includes a 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.

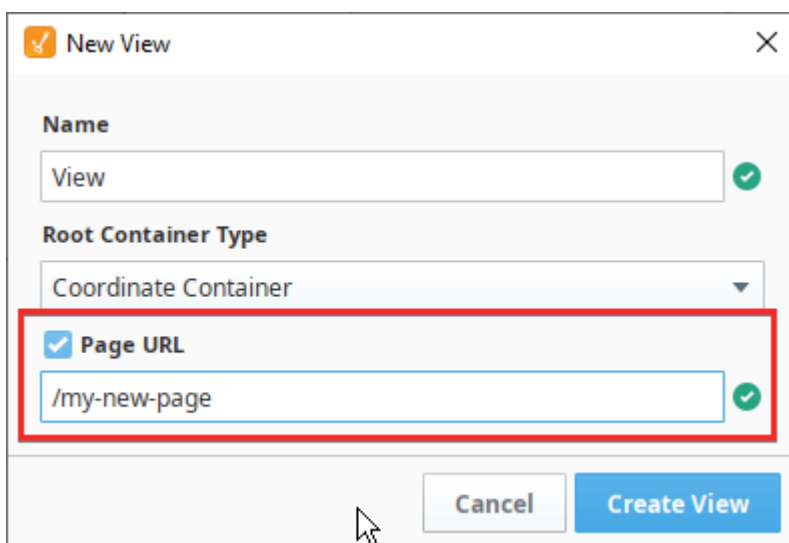


Creating a Page Configuration

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.

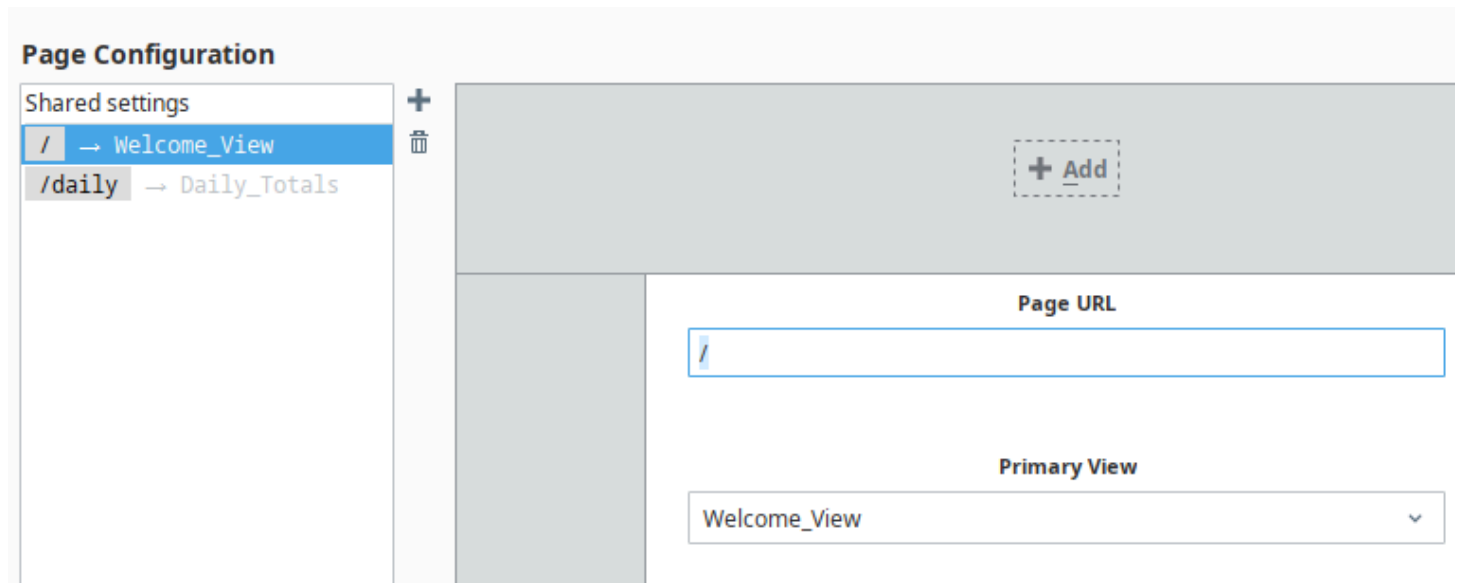


Alternatively, whenever a new view is added to a project, a page can be created automatically by opting into the Page URL setting and typing value for the **Page URL**. Note that all Page URLs should start with a forward slash character ("/"). Creating a page through this method will automatically set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



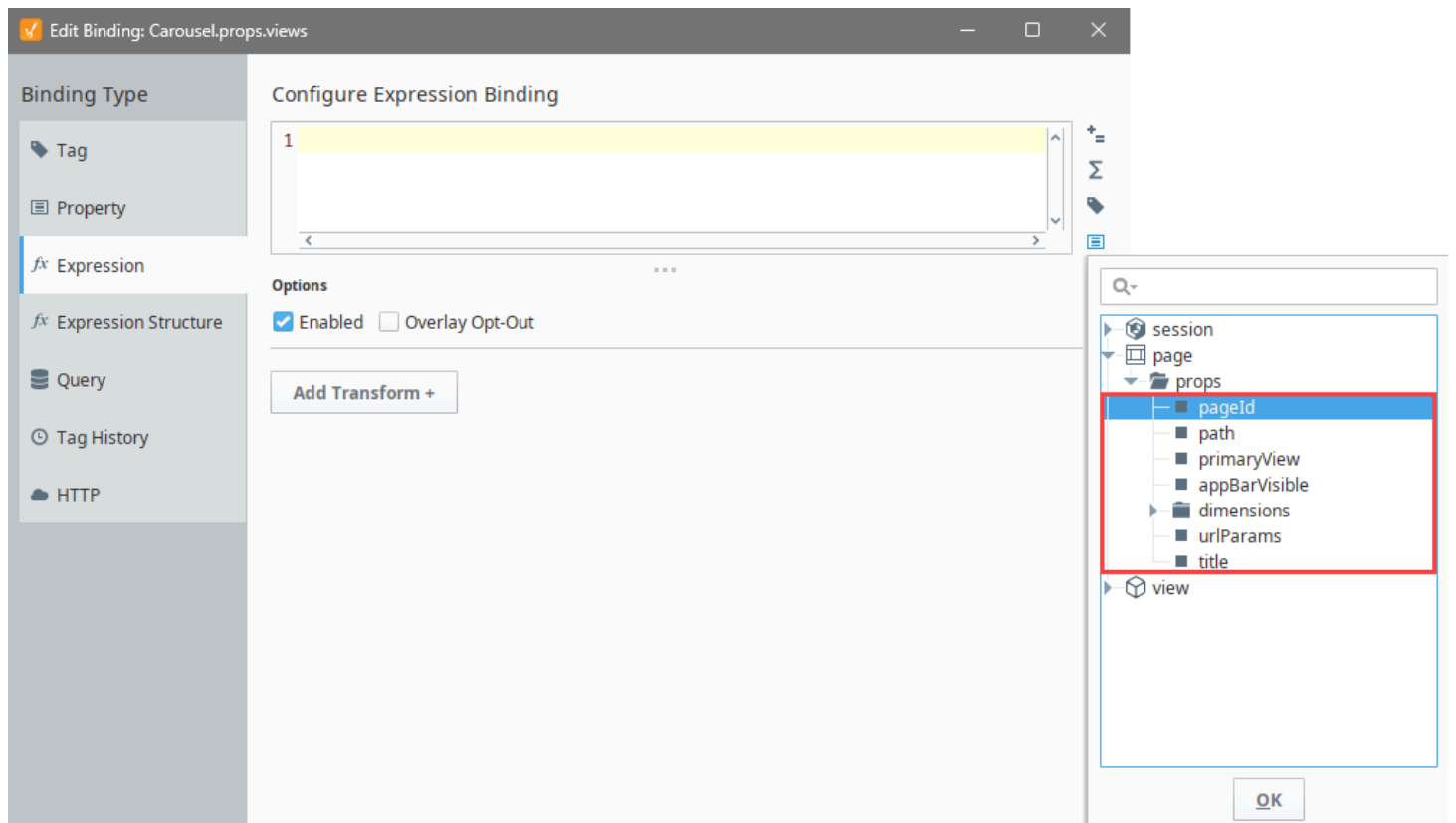
Initial Page URL

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 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.



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
path	Represents a string "path" to the page. Effectively the same as the Page URL.	value: string
primaryView	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.	value: string
appBarVisible	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.	value: boolean
dimensions	Settings for the dimensions on a page. See dimensions .	object
urlParams	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. Note: All values passed in a URL are passed as Strings.	object
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

dimensions

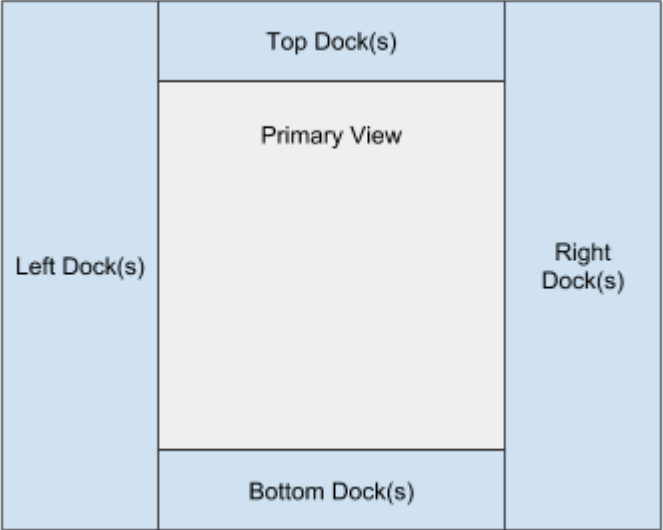
Property	Description	Property Type
enabled	The page dimensions in this object will update accordingly if true. While false, the dimension values will not update. The remaining dimension properties in this section are read-only.	value: boolean
viewport	Current settings for the "inner" boundaries of the browser, i.e., the dimensions of the page (read-only). See • width: Width of the inner boundary. Value is numeric. • height: Height of the inner boundary. Value is numeric.	object
screen	Settings for the dimensions of the screen the page is on. Note that this only updates on browser resize and page startup (read-only). • width: Width of the screen the page is on. Value is numeric. • height: Height of the screen the page is on. Value is numeric.	object
primaryView	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). • width: Width of the primary view. Value is numeric. • height: Height of the primary view. Value is numeric. • scrollLeft: Distance in pixels that the view scrolled to the left. Value is numeric. • scrollTop: Distance in pixels the view is scrolled from the top. Value is numeric.	object

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.

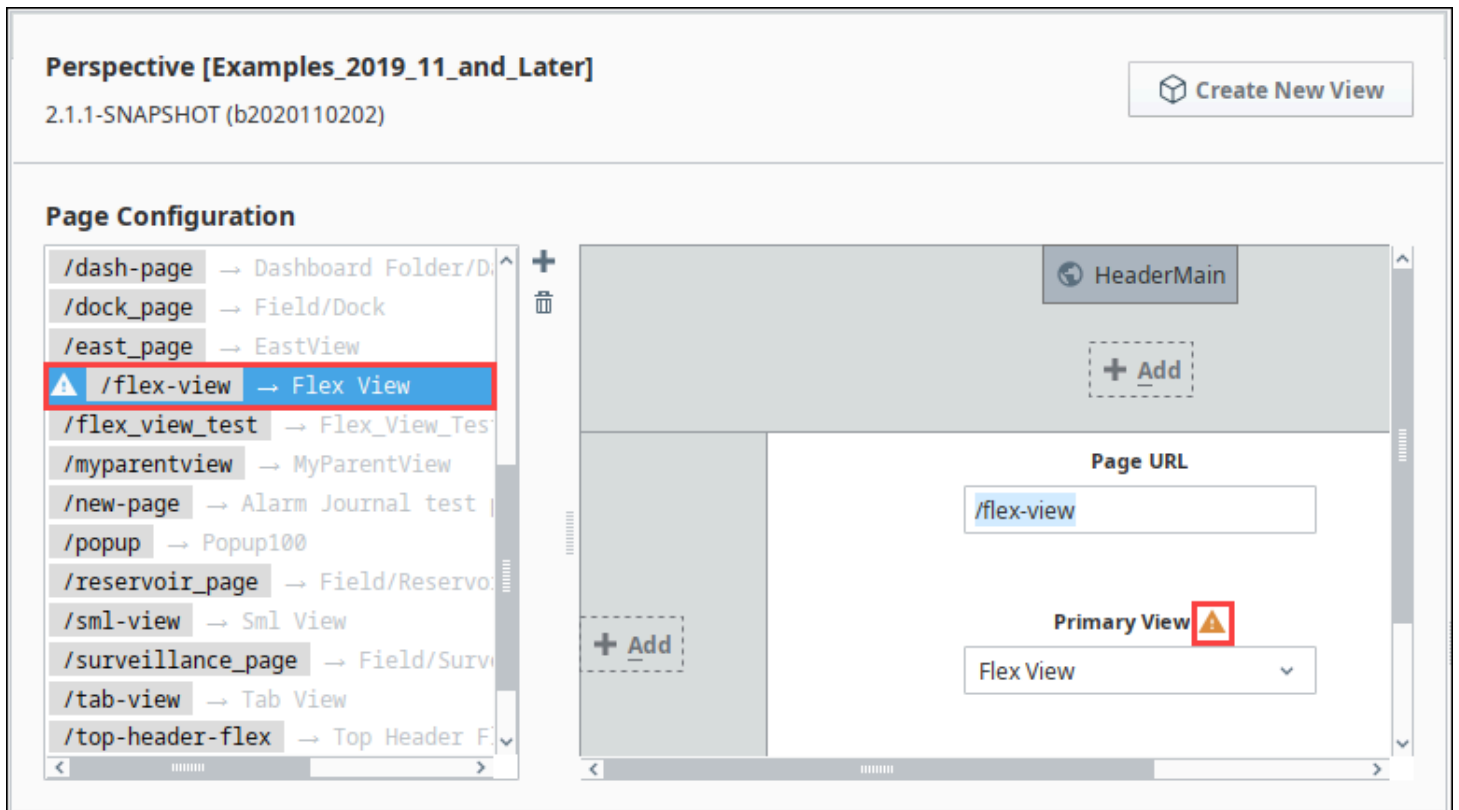
The Primary View is in the center of each page taking all available space in the browser window. Each page must have a single view configured to be its primary view.

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 and dismissed by the user.



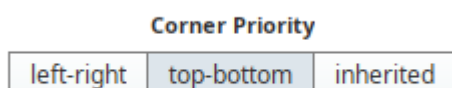
Page URLs

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.



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.



Passing Parameters to Pages



INDUCTIVE
UNIVERSITY

Passing Parameters to Pages

[Watch the video](#)

Pages can be mounted at URLs that include parameters, which are used make the URL dynamic. Perspective interprets the use of parameter replacement syntax in a Page URL as the input parameter

to a 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. If each Tower has an ID number that uniquely identifies it, we can use a parameter to configure the page's URL so that a user can navigate to any Tower using their browser's URL bar.

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

NOTE

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: `/` `?` `#` `%`

Edit Binding: Dropdown.props.value

Binding Type

- Tag
- Property**
- Expression
- Expression Structure
- Query
- Tag History
- HTTP
- Remove Binding

Configure Property Binding

view.params.towerNumber

Options

☒ Enabled ☐ Overlay Opt-Out ☐ Bidirectional

Add Transform +

Binding Preview

Property value

OK Cancel Apply

We configured our project's Tower site view, called Towers, as the page's primary view. We now need to configure the Page URL to enable navigating to any Tower by using the parameter replacement syntax, which is a colon followed by the input parameter. In this example it looks like this: **/:towerNumber**.

Page Configuration

Shared settings

/:towerNumber

→ Towers

+ Add

+ Add

+ Add

+ Add

Page URL

/:towerNumber

Primary View

Towers

Page Title

Corner Priority

left-right

top-bottom

inherited

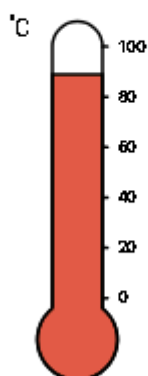
+ Add

The colon signifies the portion of the URL that is meant to be dynamic and map to an input parameter on the page's primary view. Now, if the user were to navigate to the `<URL>/2`, Perspective would render `/` as the primary view with an input parameter mapping of "towerNumber = Tower2", displaying the Tower2 information.

Tower2

Fan Speed

1482.8657



Temp

88.89

Passing 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 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?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?key=8

Views in Perspective



INDUCTIVE
UNIVERSITY

Anatomy of a View

[Watch the video](#)

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](#).

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, so they can be referenced by all child components and containers in that view.

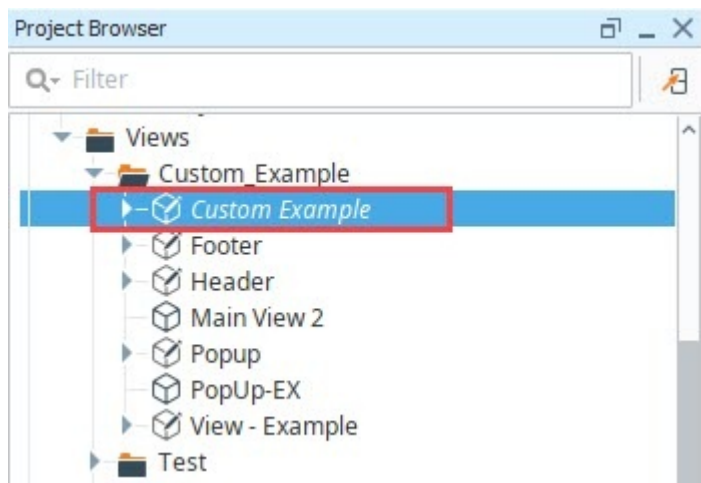
Each view contains exactly one root level container, which may be any of the available **container types**. Therefore, the design experience of a view is simply the design of the selected container type. This makes it important to understand the needs of your view when creating it since once the view is created, the root container type cannot be changed.

The view properties have three categories:

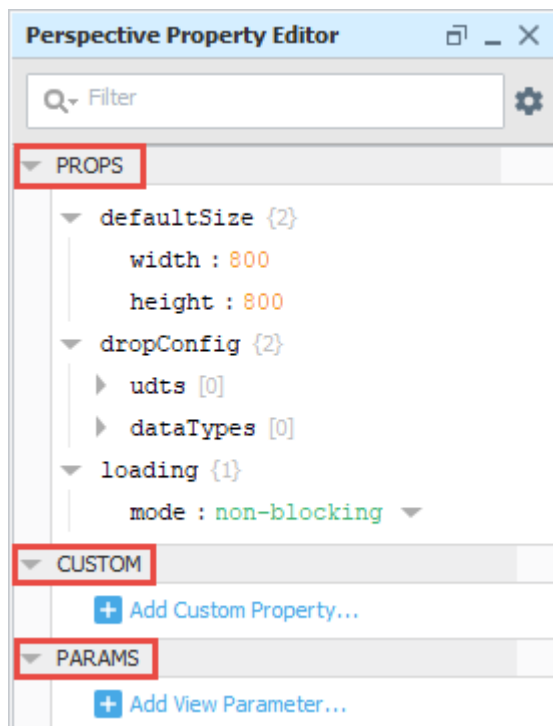
- **Props** - Properties used to configure the component's visual appearance, behavior, and data.
- **Custom** - Properties defined by the user. They have no direct effect on the component, but are used as variables for the application designer's convenience.

- **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.



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.

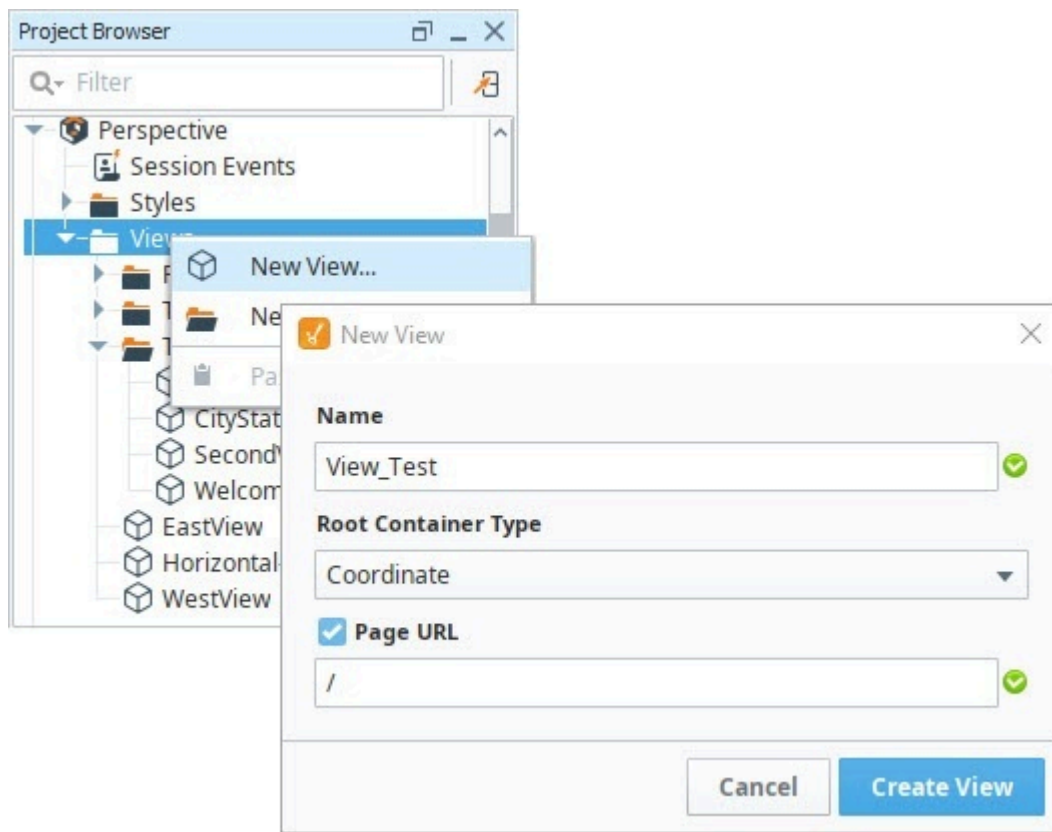
- An **output parameter** is the opposite. These parameters are bindable from within the configuration of the view. If an instance of that view is then placed inside another container, the output values will appear as properties, but not be bindable; they will be read-only from the outside.
- An **in/out 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

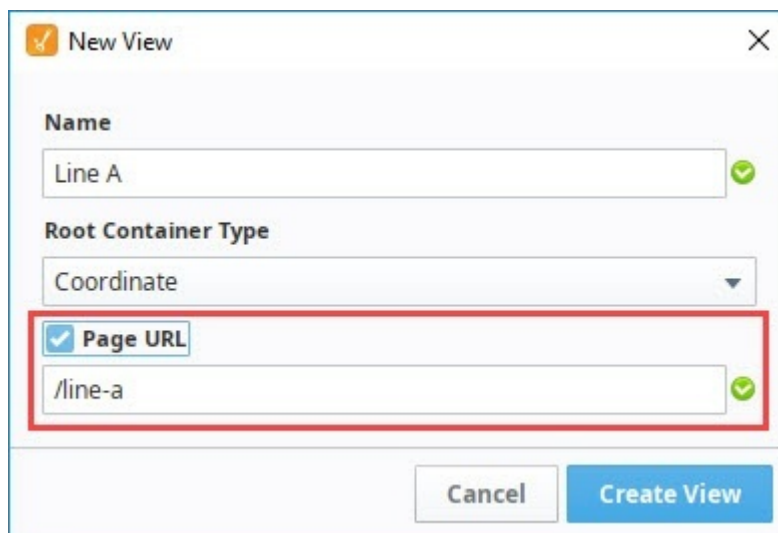
Views can be configured in many different ways. They can be displayed as the entire browser window if it is configured to be the primary view of a page. Views can also be displayed across the top, bottom, or sides of the browser window if it is configured to be a docked view. A view can also be displayed 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.

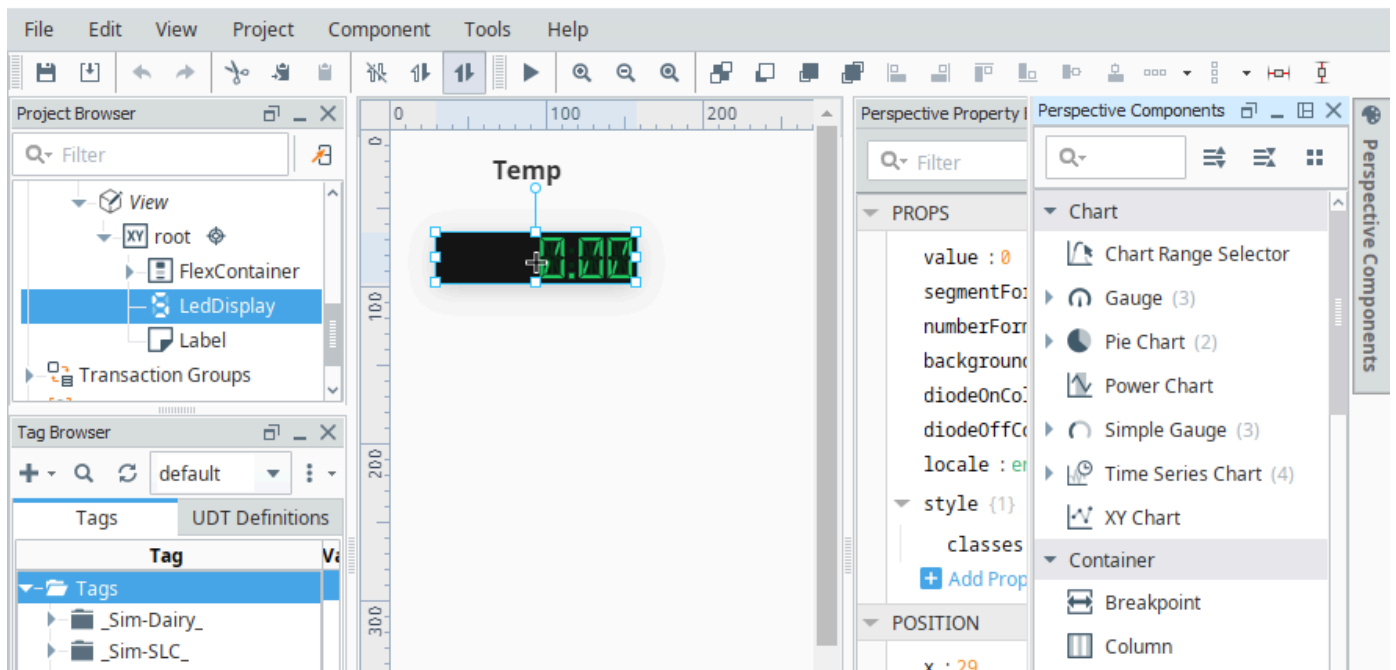
1. In the Project Browser, right click on the Views folder and click **New View**.
 - i. In the New View window, enter a name for your view.
 - ii. Select a root container type.
 - iii. Check the **Page URL** checkbox if you want your view attached to the Page URL. Perspective will match the Page URL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



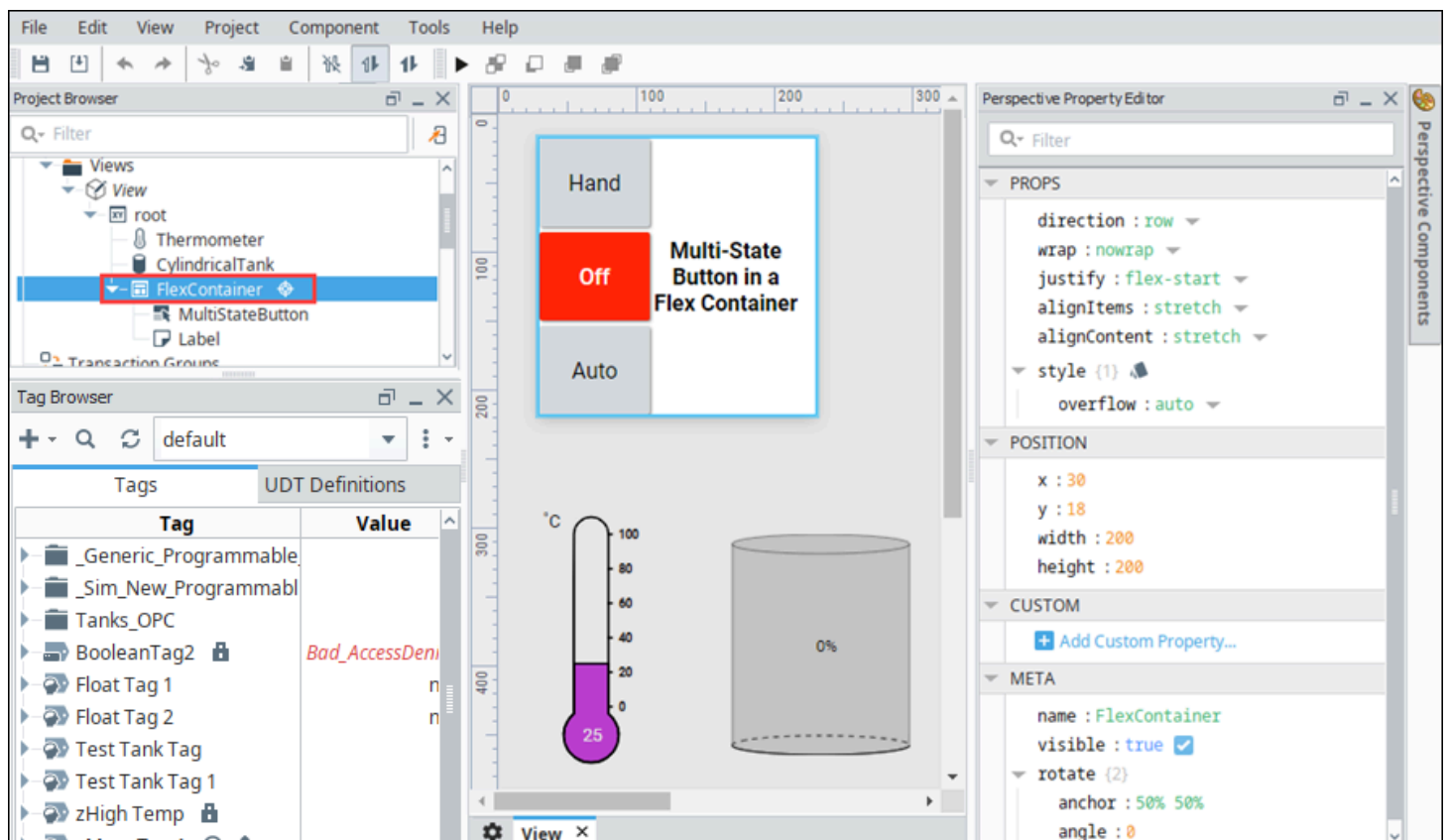
2. Once you create your view, the view will open in the Designer workspace, and you can begin adding and configuring components.



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



INDUCTIVE
UNIVERSITY

Docked Views in Perspective

[Watch the video](#)

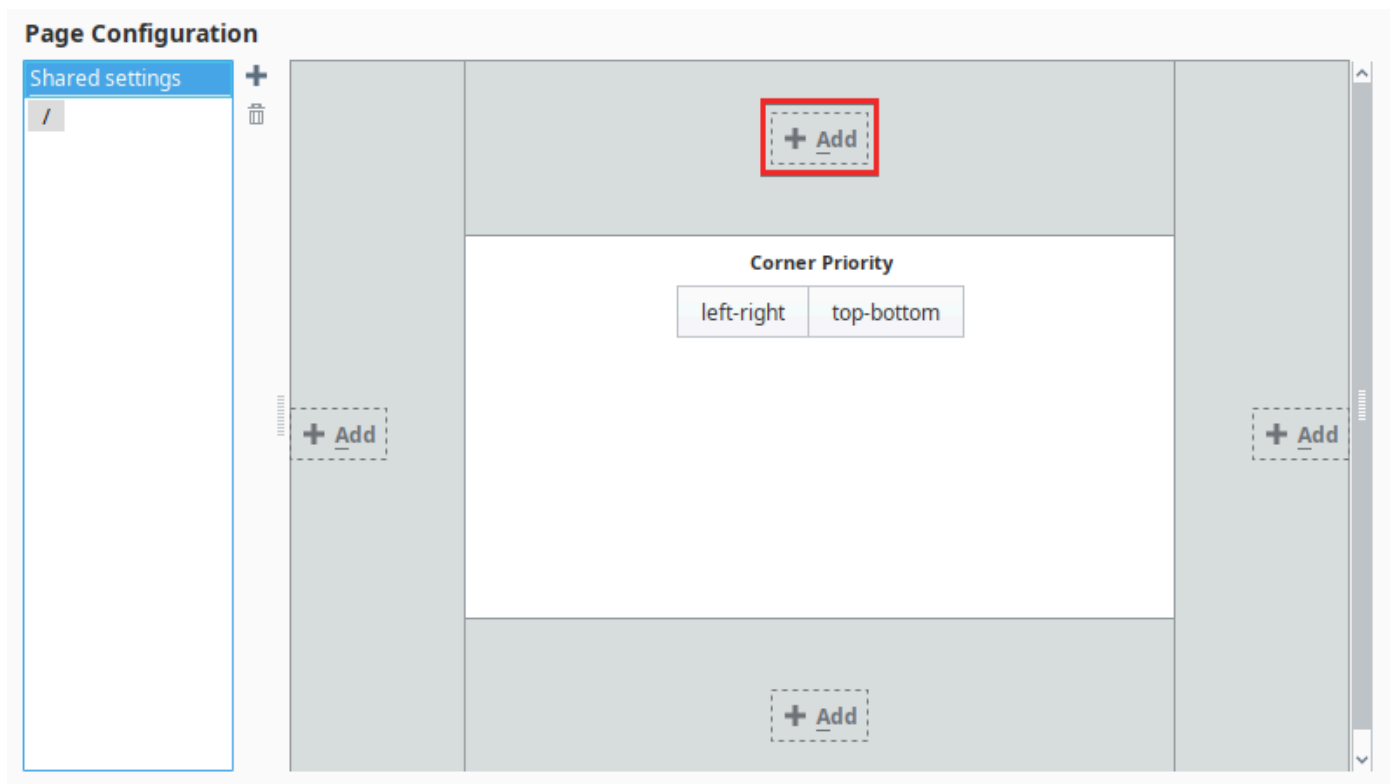
Views can be anchored to an edge of a session. This configuration is called a docked view, as the view is "docked" to an edge in the session.

Docks are created in the [Page Configuration](#) interface. Note that only one docked view can be added to each edge position.

Configuring a Docked View

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

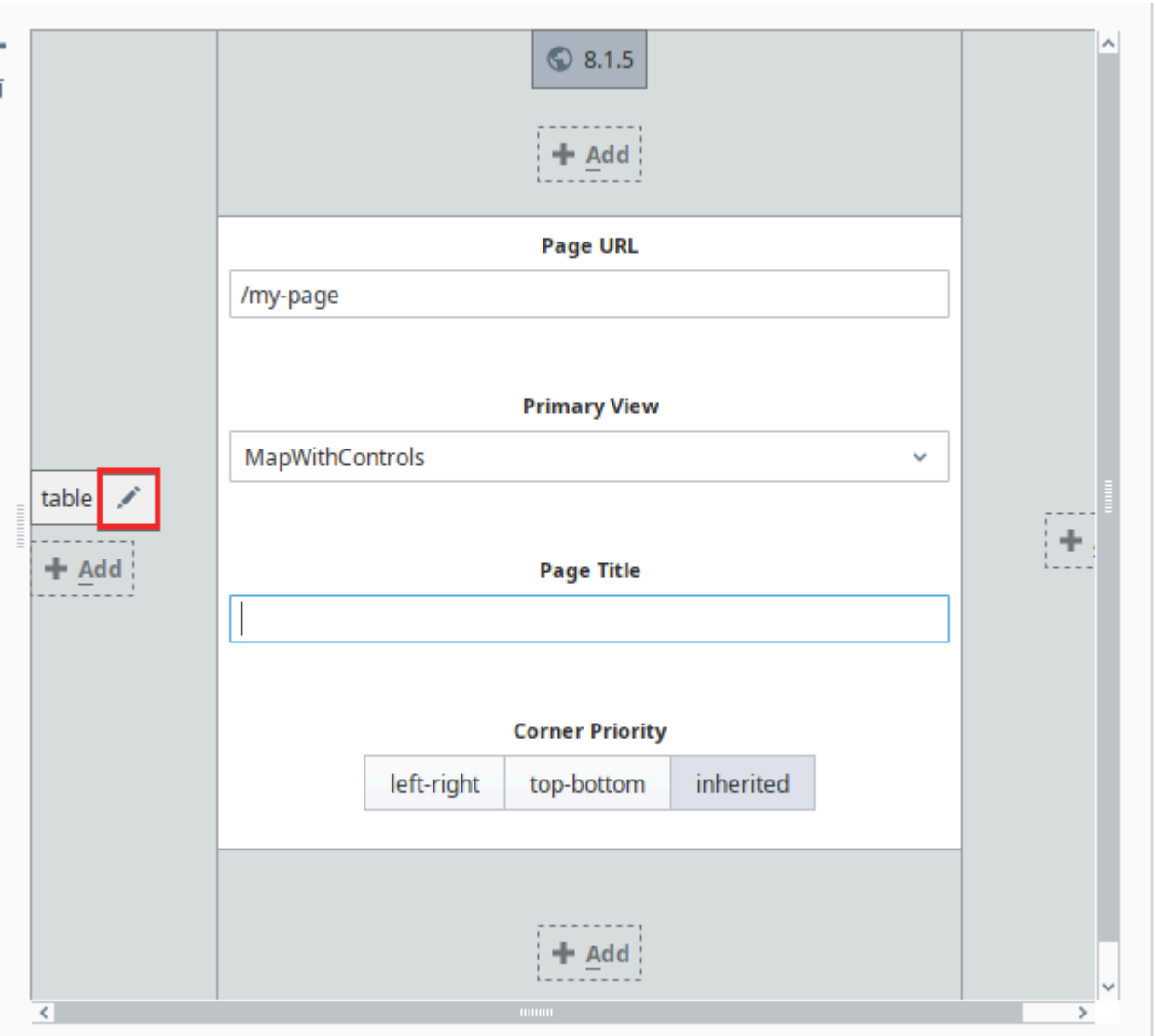
1. Select the configuration page you wish to add the dock to.
2. Click the **Add** button on the desired edge. In this example, we'll add a north docked view by clicking at **Add** button at the top of the diagram.



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.



Docked View Properties

Property Name	Description
View	The currently selected view. Changing this will change which view is mounted to this position.
Display	This property allows you to show or hide the docked view. See display .
Resizable?	Determines whether the docked view may be resized or not.
Modal?	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 Display property is set to onDemand .
Content	Determines how the docked view interacts with other views on the page. See content .
Anchor	Allows you to make a view always visible while scrolling. Only available on North docked view configurations. • fixed: 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. • scrollable: 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.
Size	Determines the size, in pixels, of the view. • If the view is docked to the North or South edge, then size determines the height. • If the view is docked to the East or West edge, then size determines the width.

Property Name	Description
Auto Breakpoint	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 Display property is set to auto .
Dock ID	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.
Handle	Allows you to show or hide a handle for users to expand/collapse the view. • Show: Show handle at all times. • Hide: Hide handle at all times. • AutoHide: Hide handle when page is not active.
Handle Icon	Path to an icon used to identify the view when the view is hidden.
View Parameters	Allows specific parameter values to be passed to the docked view when navigating to the page.
Remove	Deletes the view from the page.

Display

Option	Description
visible	The docked view is always expanded/displayed.
onDemand	The docked view is collapsed, but allows the user to display the view by clicking on the docked view's handle.
auto	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).

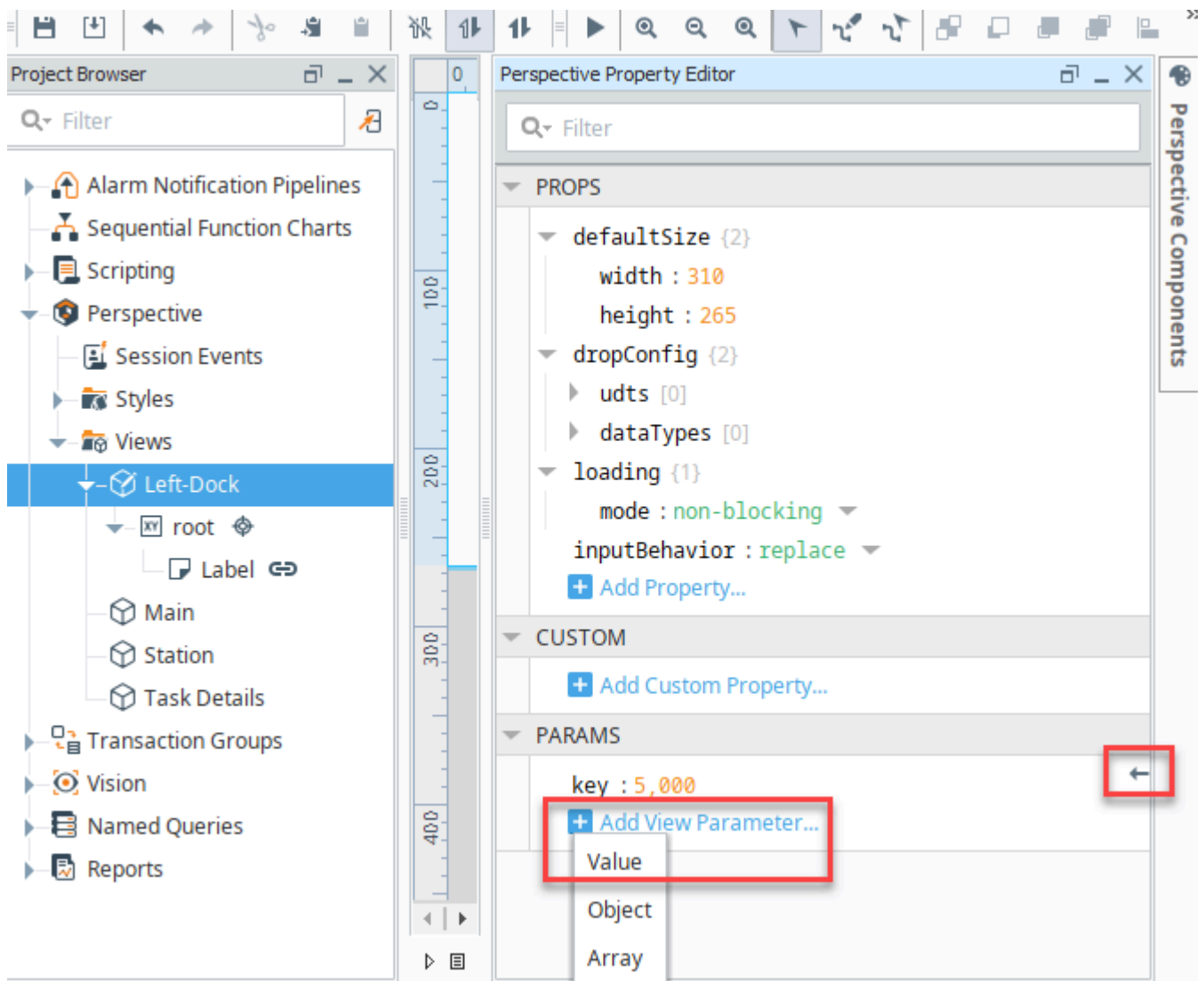
Content

Option	Description
push	Opening or closing the docked view causes the content in the center to resize: the center view will be 'pushed' out of the way.
cover	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.
auto	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.

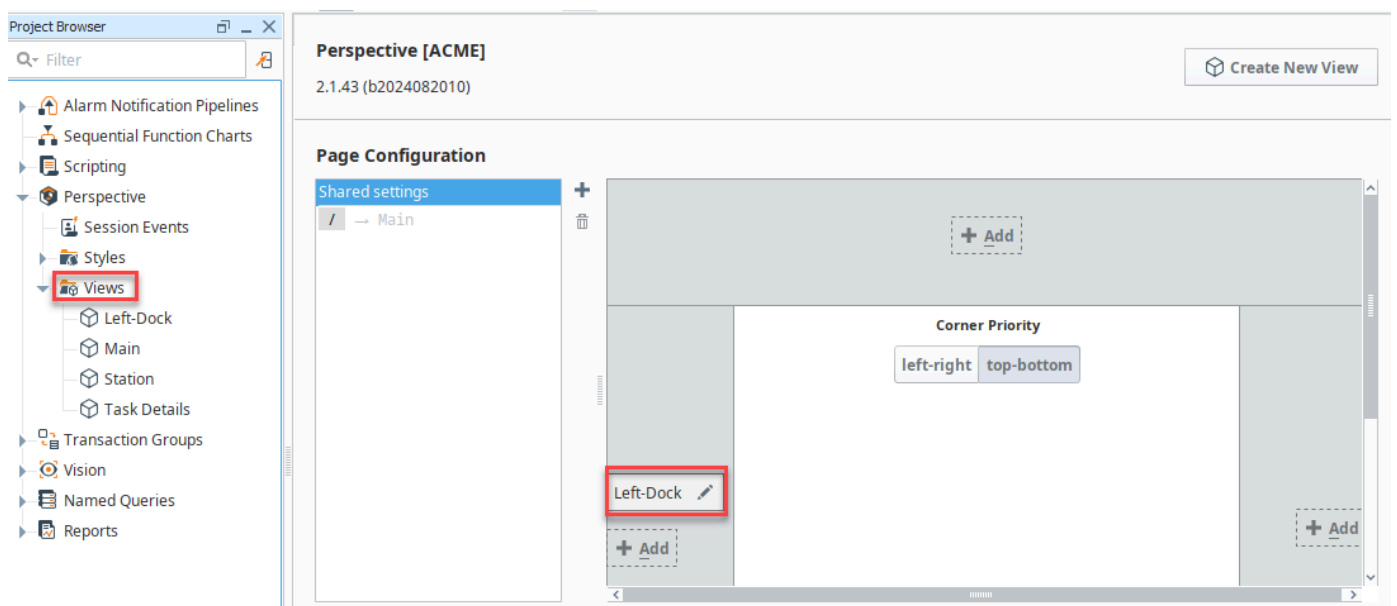
Example

Creating a View Parameter in a Docked View

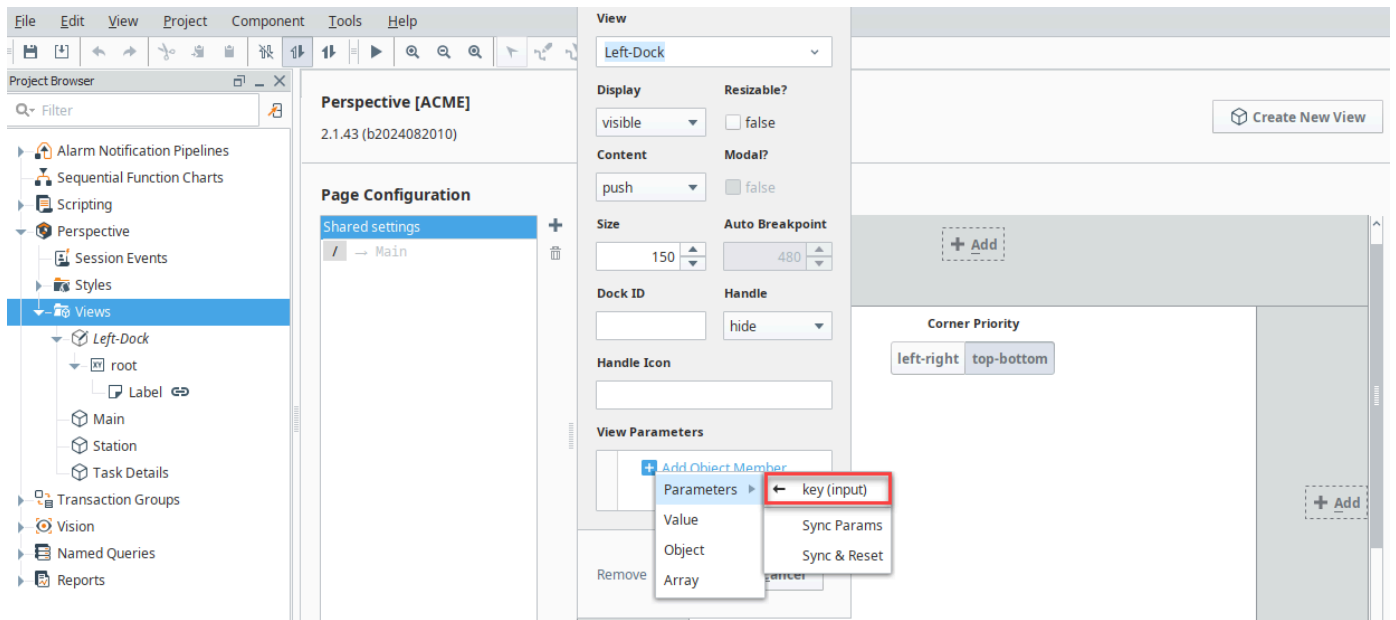
1. Select your docked view from the Project Browser.
2. Click the Add  icon for **Add View Parameter** in the Perspective Property Editor.
3. Click on **Value** and assign a name and a value. For example, Key:5000
4. Verify the parameter input/output arrow  is kept in the input direction.



5. Click on the **Views** folder in the Project Browser to go into Perspective's Page Configuration.
6. Click on the Edit  icon for your docked view.



7. Click on the Add  icon for **Add Object Member** and select the parameter you just created.



Now that you have a parameter created for your docked view, you can use it to dynamically pass data and display different data based on user input.

Embedded Views



INDUCTIVE
UNIVERSITY

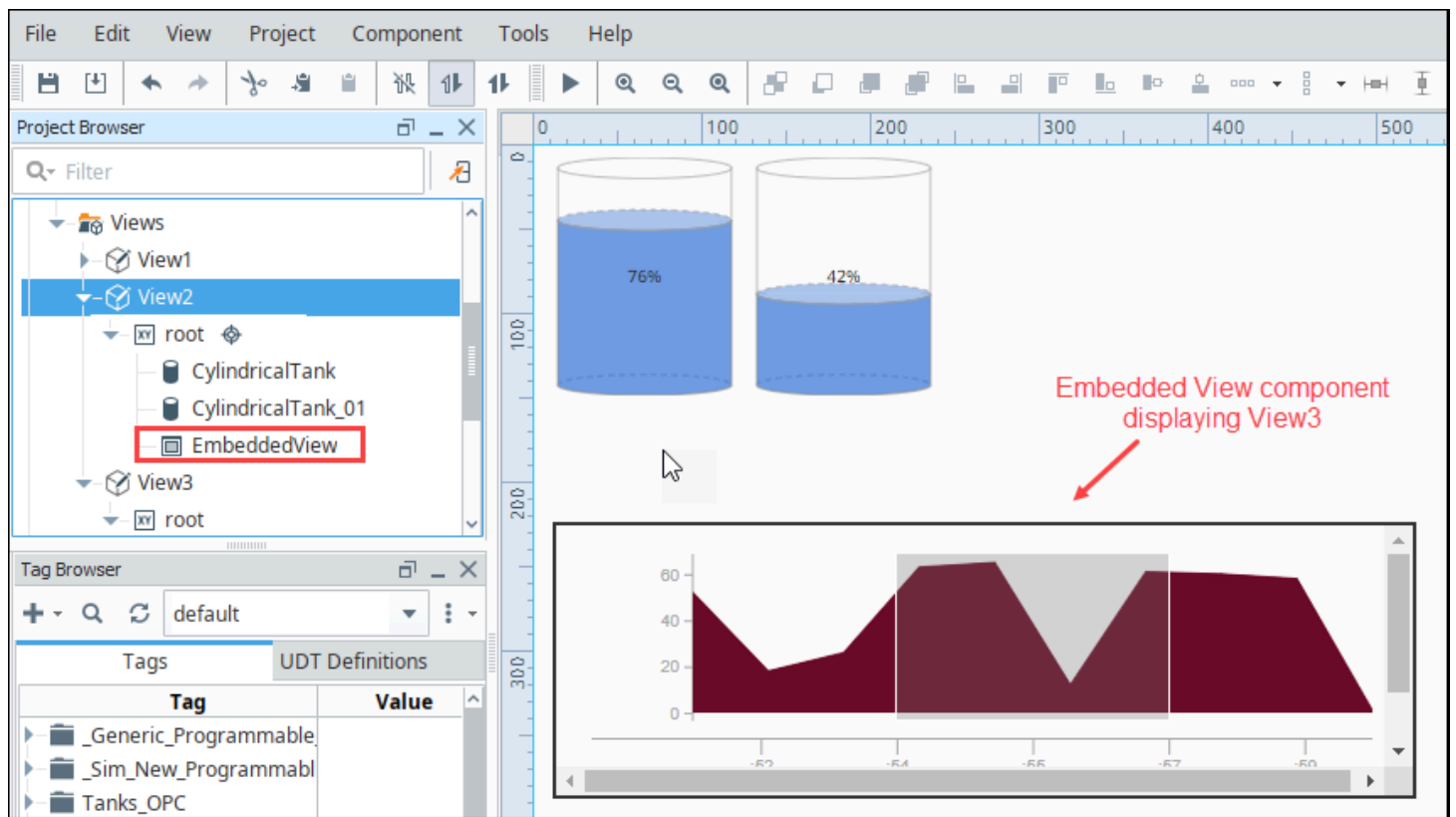
Embedded View

[Watch the video](#)

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.

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.

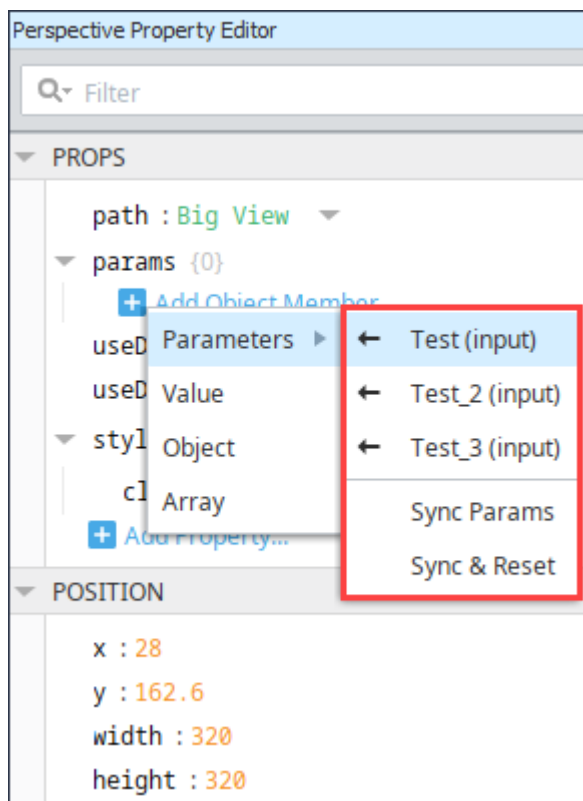


The view parameters available from the embedded view will appear on a dropdown list of parameters in the Property Editor. You can select any or all of the parameters to add. Two additional operations are available:

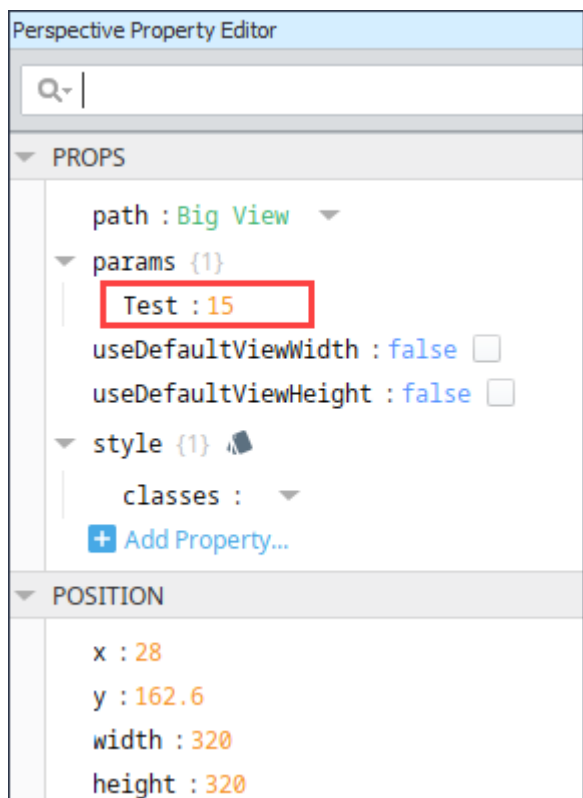
- **Sync Params** - Adds all the embedded view's parameters and removes any parameters that don't exist on the embedded view.
- **Sync & Reset** - Resets the parameter values to their default.

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



INDUCTIVE
UNIVERSITY

Popup Views

[Watch the video](#)

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.

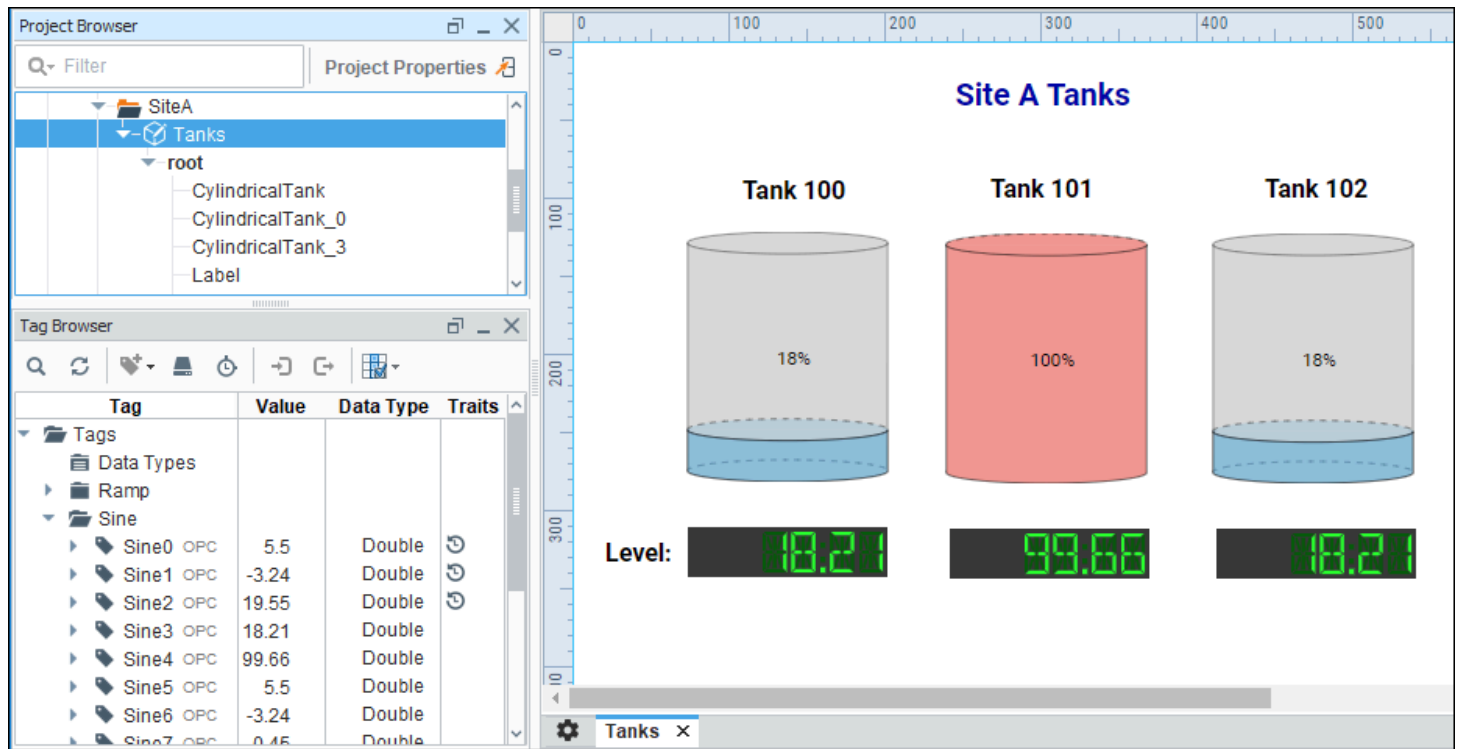
Title Bars on Popups

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).

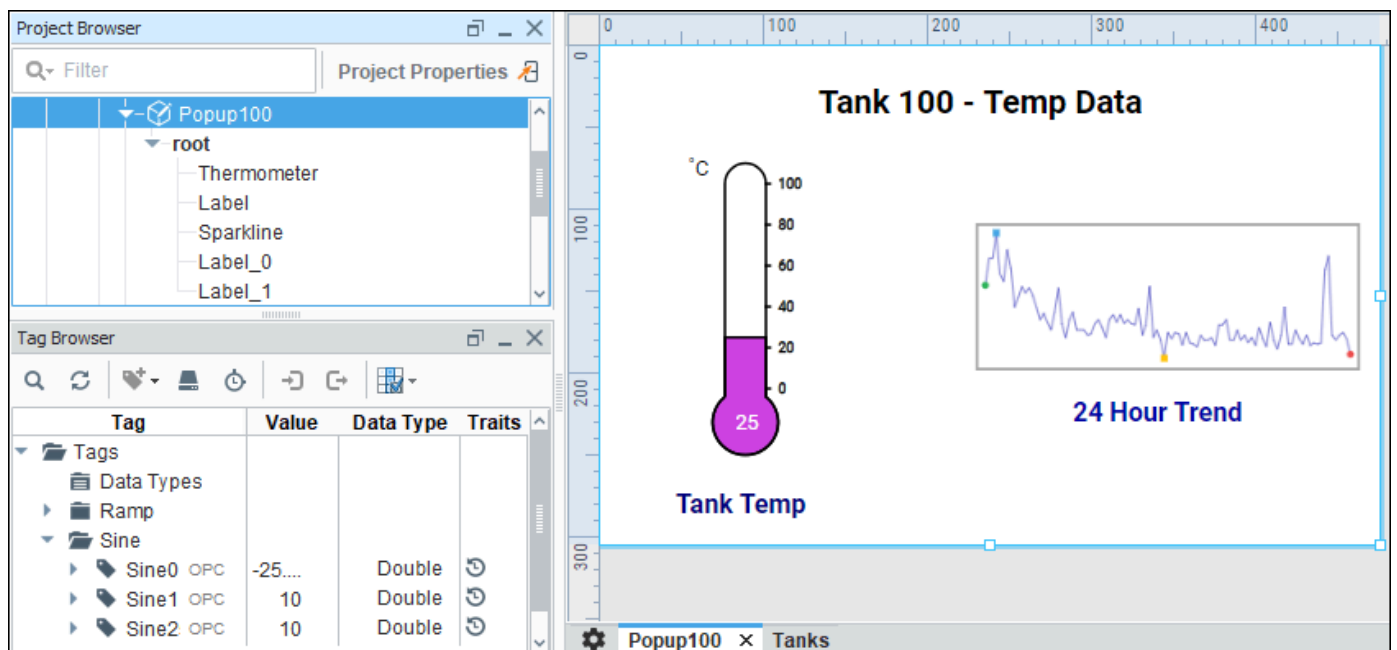
Configuring a Popup View

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.



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 may 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.

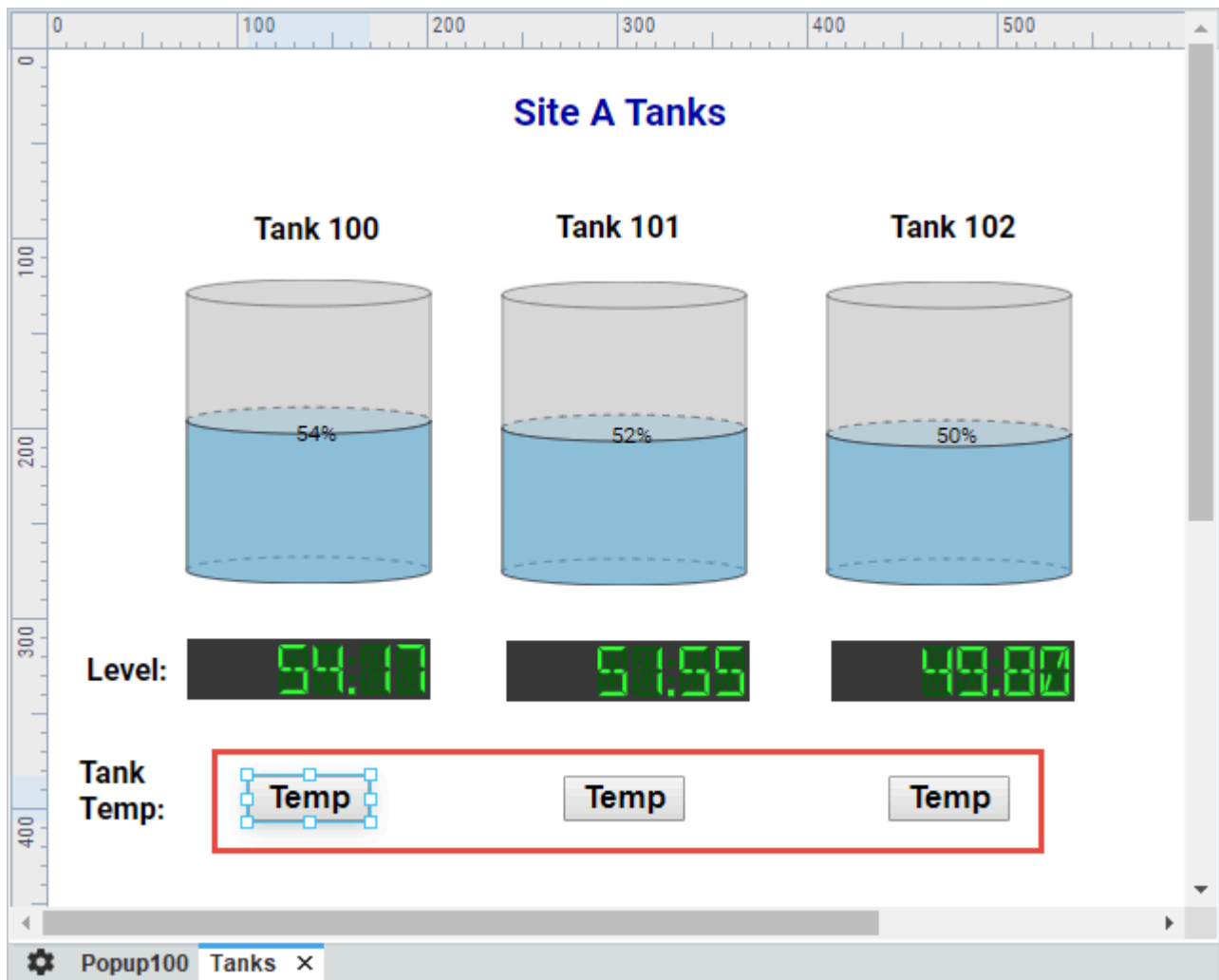


NOTE

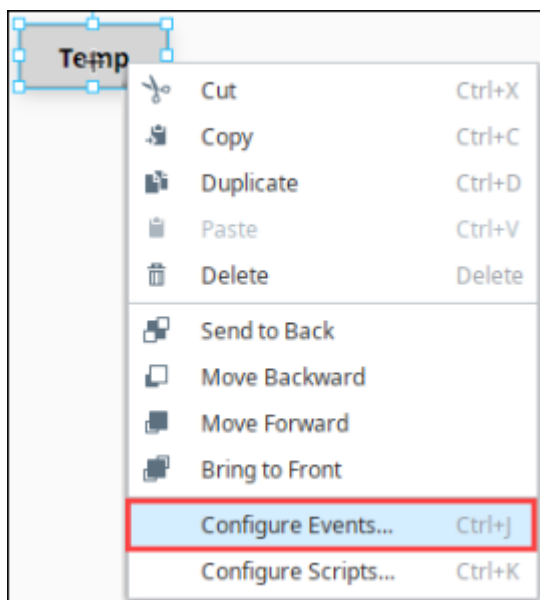
To display the additional data for all Tanks, you will need to create a Popup View for each Tank. You can use the [Flex Repeater](#) component to easily create multiple instances of components for

display in another view each having the same look, feel, and functionality of the original components.

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

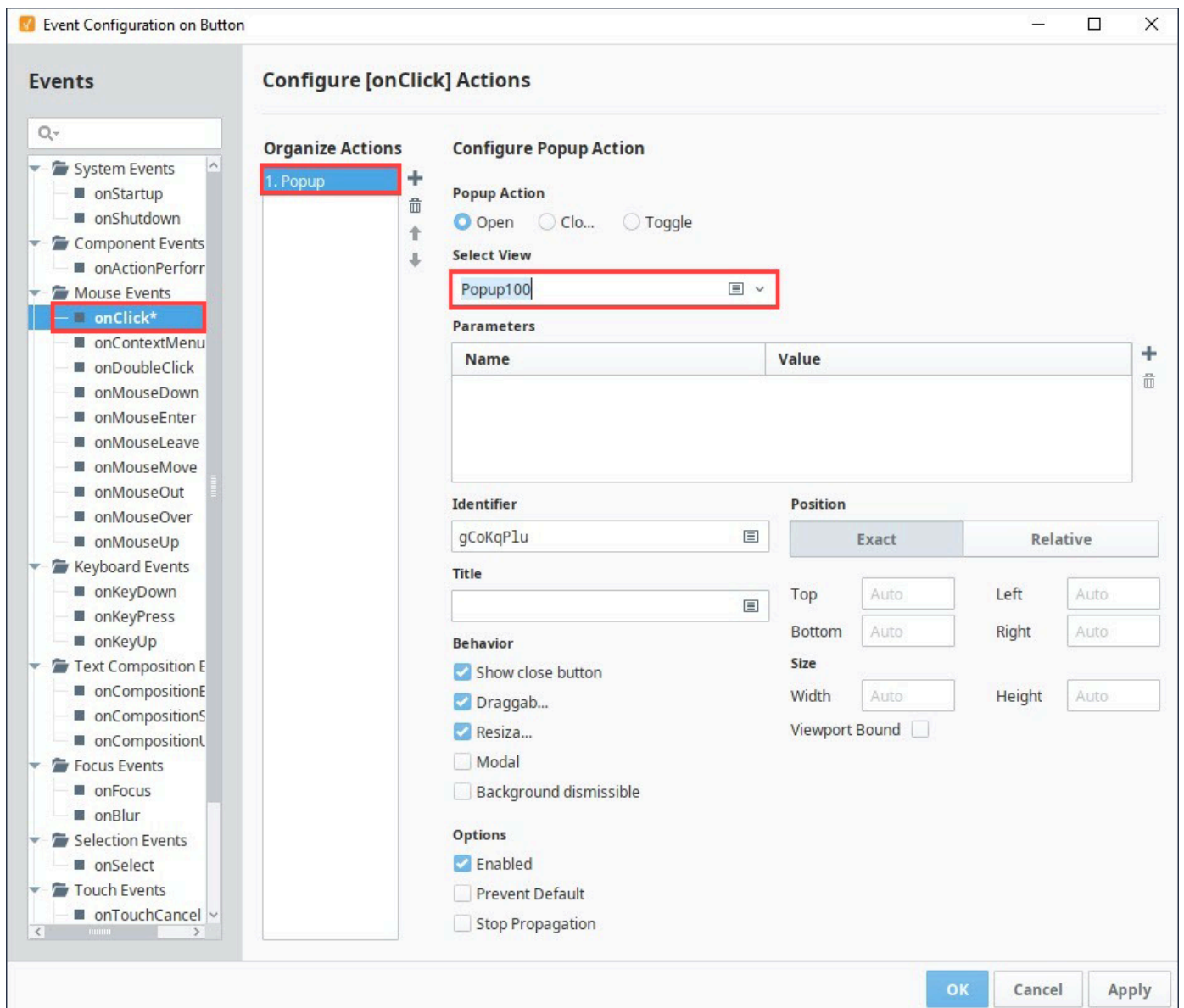


2. Right-click on the Temp button for Tank 100, and select **Configure Events**.

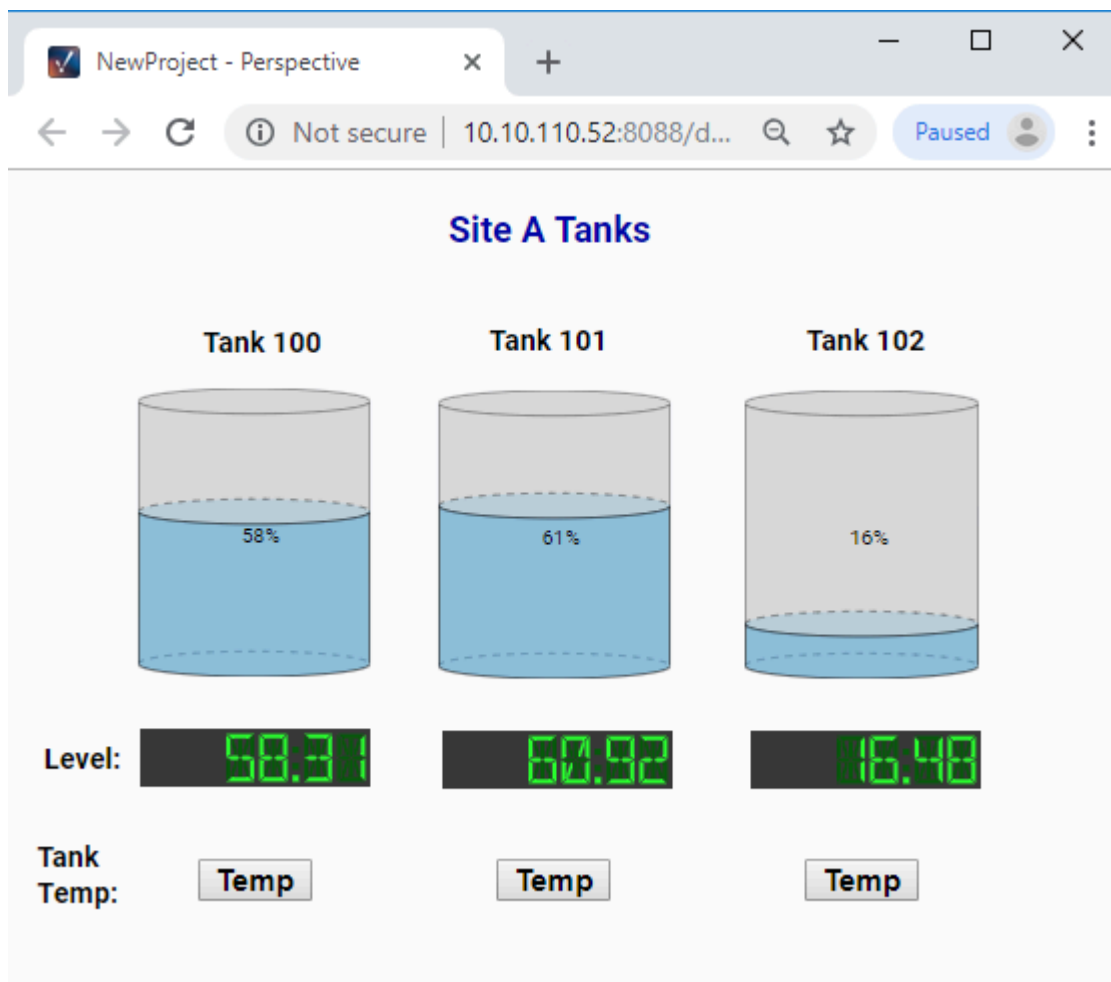


3. The Event Configuration window opens. Under Events, select **onClick**.

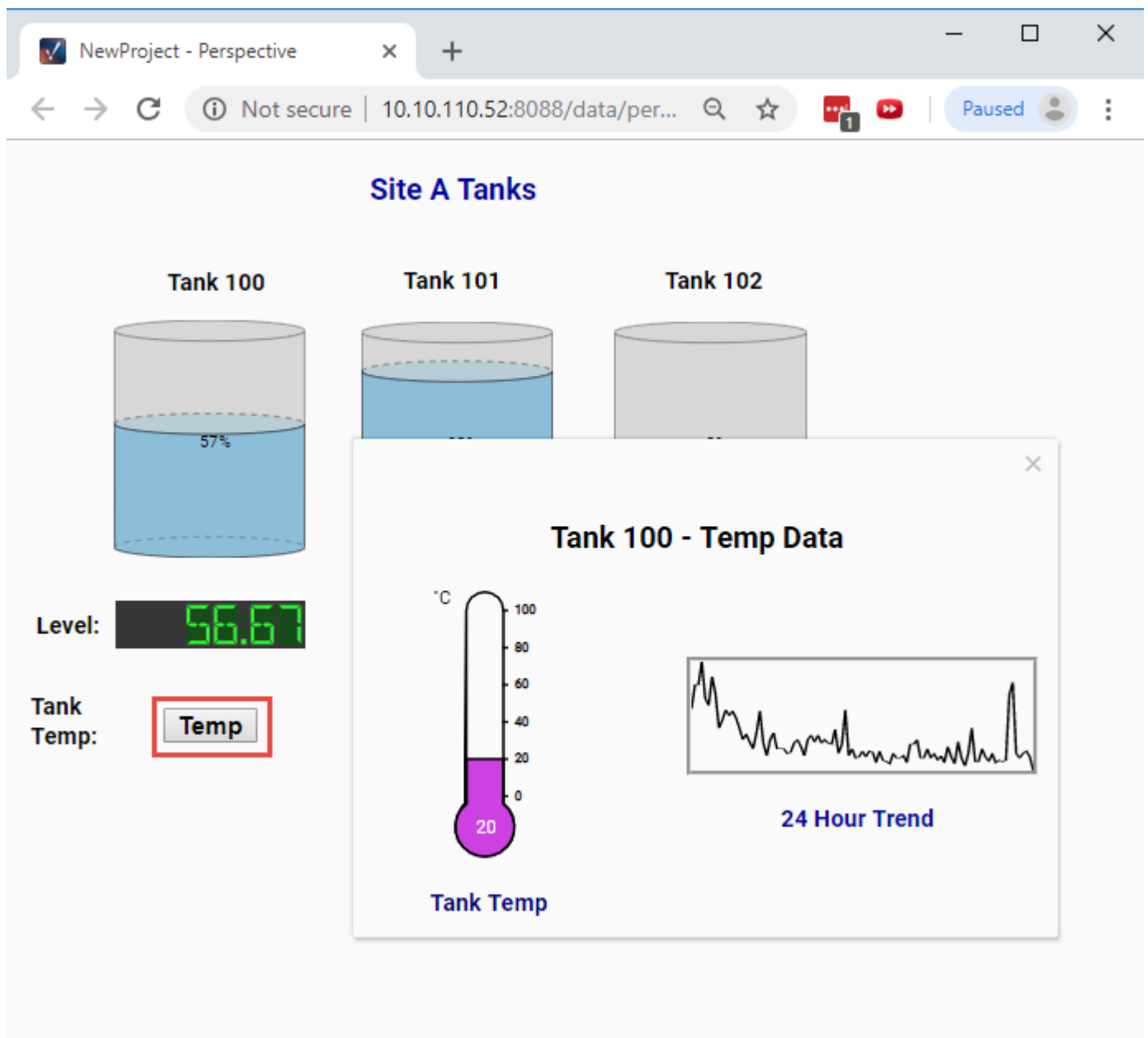
4. Under Organize Actions, click the **Add +** icon to choose the appropriate action for the Temp button. Since we are opening a Popup View, select **Popup** from the dropdown list.
5. You have several Popup Actions to choose from.
 - i. **Open** - Opens the Popup View.
 - ii. **Close** - Closes the Popup View.
 - iii. **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.
6. 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**'.
7. 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**.
8. Repeat Steps 4 through 9 for Tank 101 and Tank 102.



9. **Save** your project.
10. 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**.



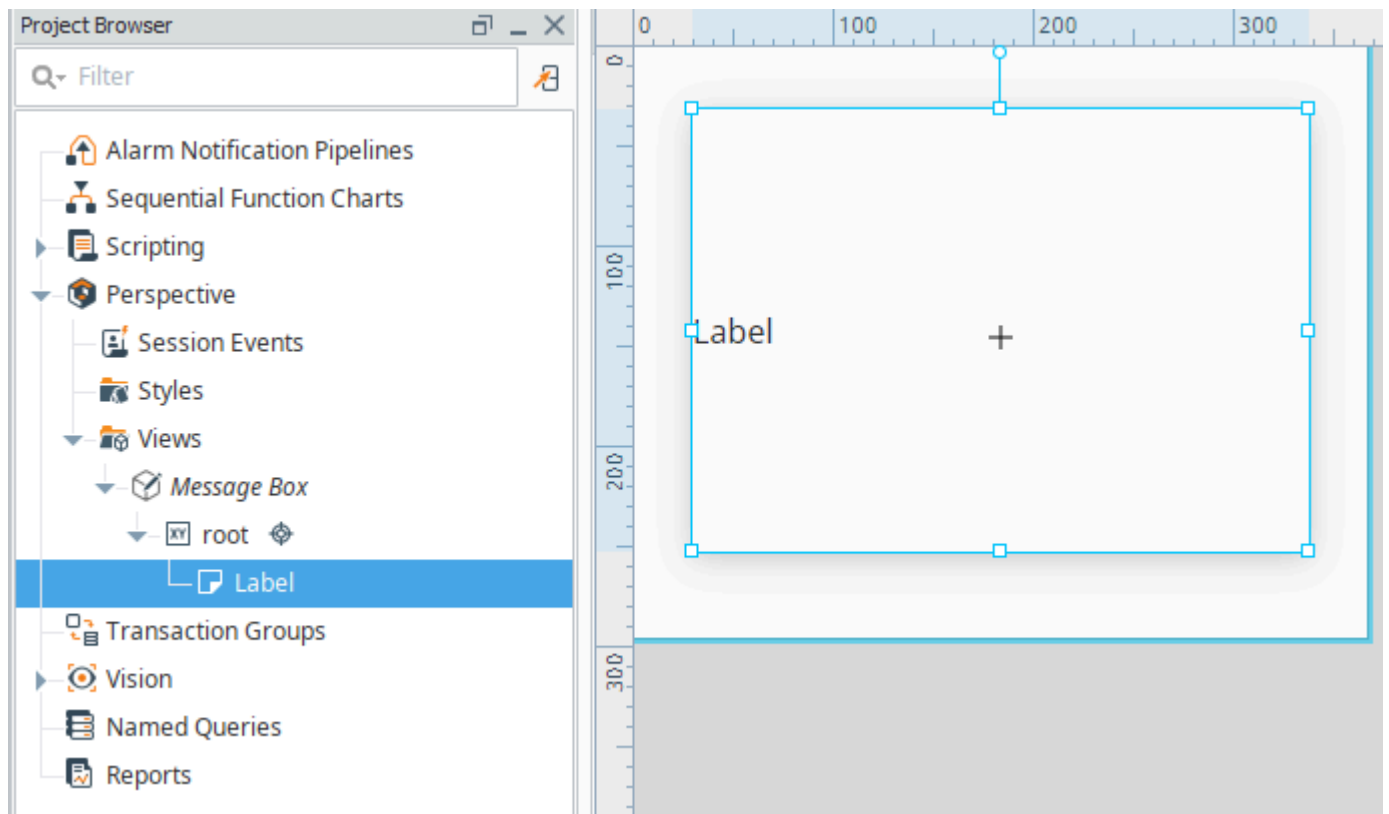
11. Click on the **Temp** button for **Tank 100**. You'll notice Tank 100's Popup View will open displaying its Temp Data. You can drag the Popup View around the screen or even change its size.
12. To close the Popup View, click either the Temp button, or click the **Close**  icon in the upper right corner of the Popup view.



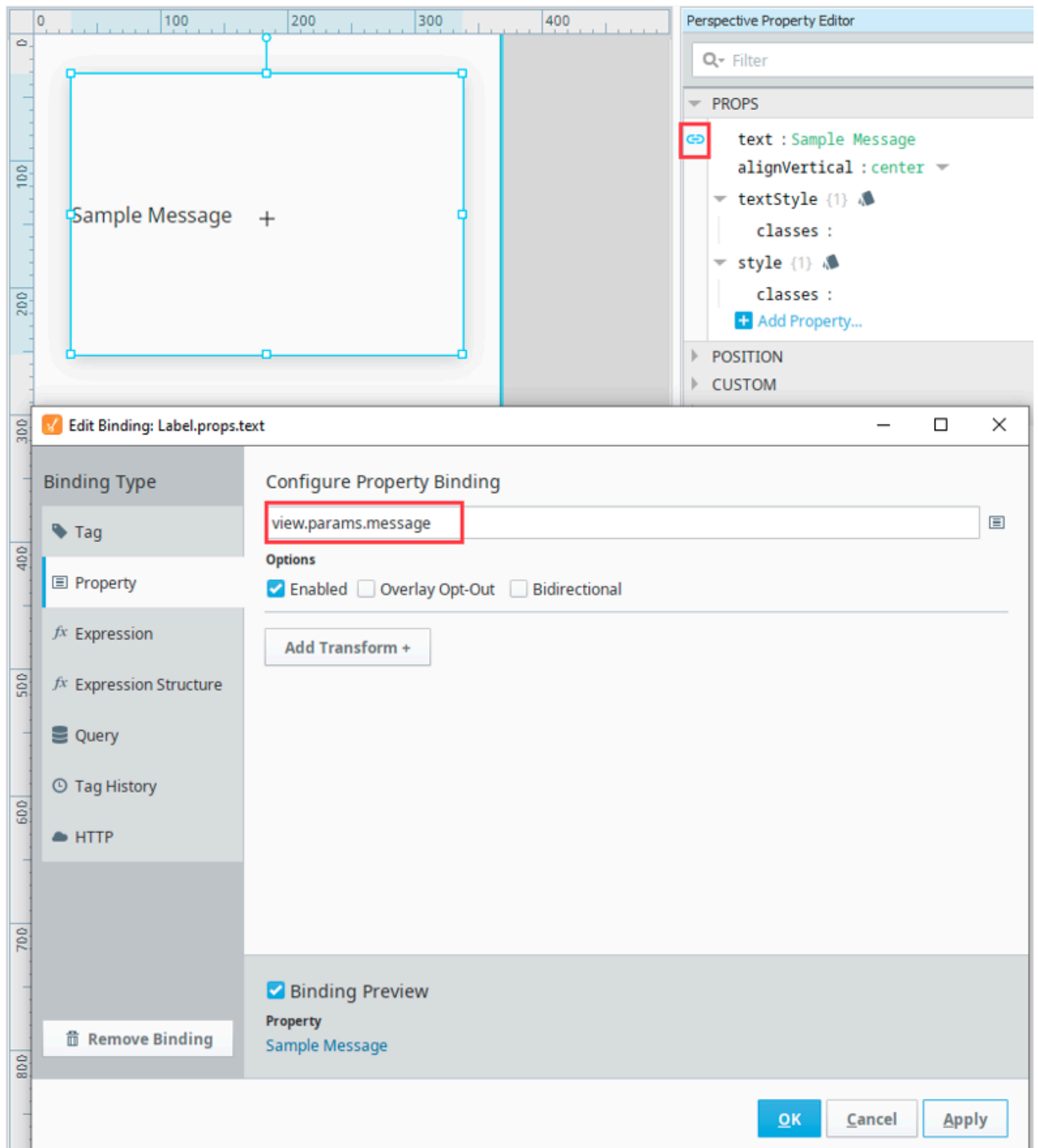
Configuring a Parameterized Popup View

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.

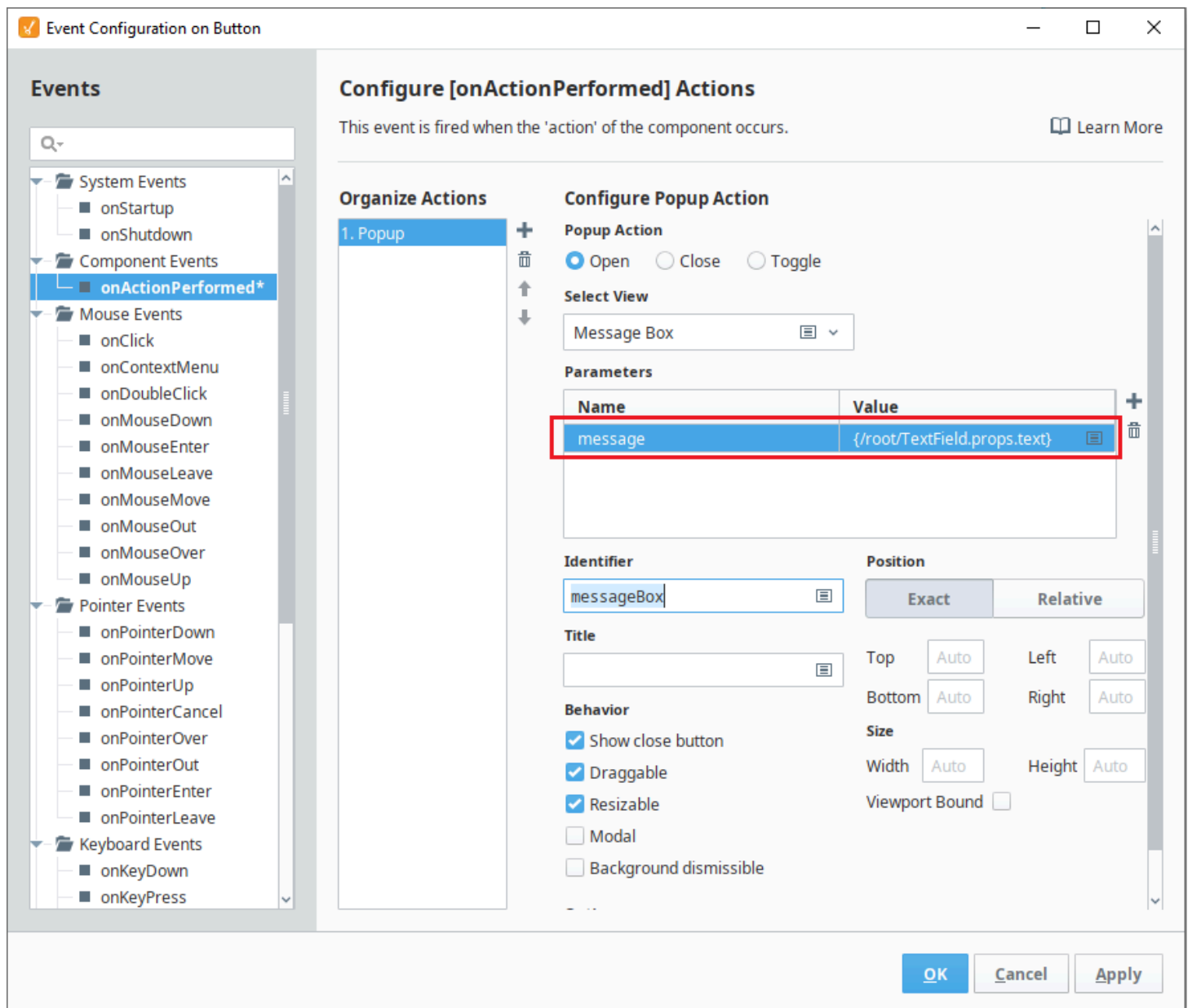


3. Select the Message Box view from your Project Browser, and from the Property Editor, define a **new** value parameter named "message" with a default value of "Sample Message."
4. Bind the Label component to our newly created view parameter.



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.



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()`

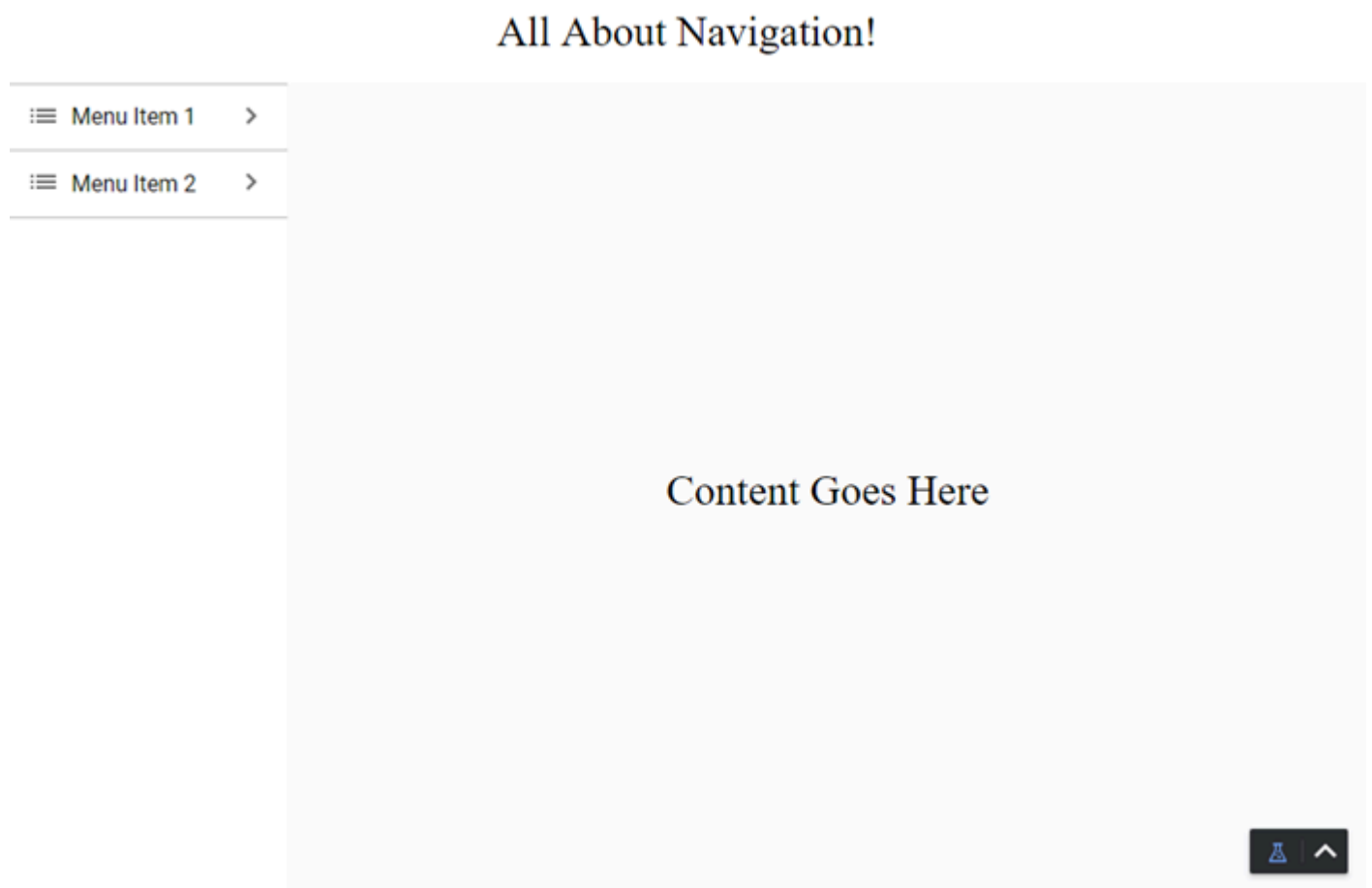
```
system.perspective.openPopup("messageBox", 'Popups/MessageBox', params =  
{'message':self.getSibling("TextField").props.text})
```

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, Menu Item 1 and 2 tabs are visible on the left-hand side of the screen:



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



All About Navigation!

Content Goes Here



Only the title and main content for the screen are visible. A menu icon is available in the top left-hand corner to expand menu options as desired.

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.

Configuring a Navigation Drawer

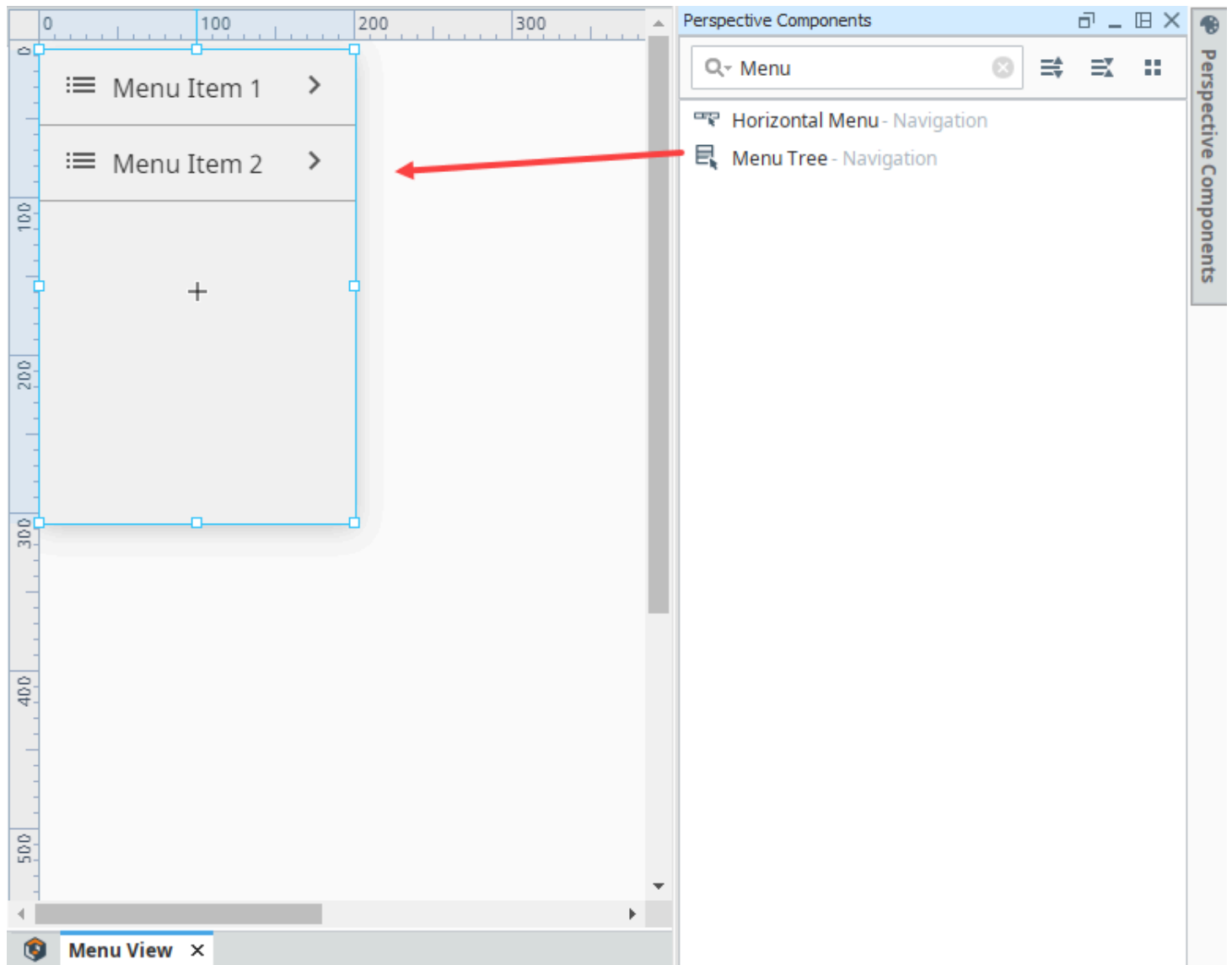
The following example walks through how to configure this self-hiding navigation drawer. There is no mention of setting up the Menu Tree or any content pages, it is strictly a guide to show you the layout type.

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.
 - i. In the Project Browser, right click on **Views** and click **New View**.
 - ii. Name the new view **HeaderSmall**, and set the Root container Type to **Coordinate**.
 - iii. Click **Create View**.

New View

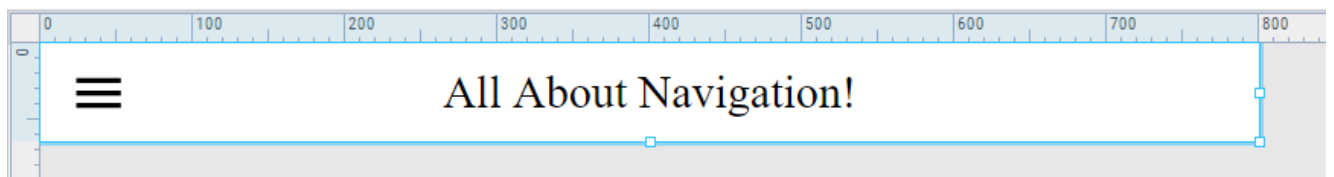
Name
HeaderSmall ✓

Root Container Type
Coordinate ▼

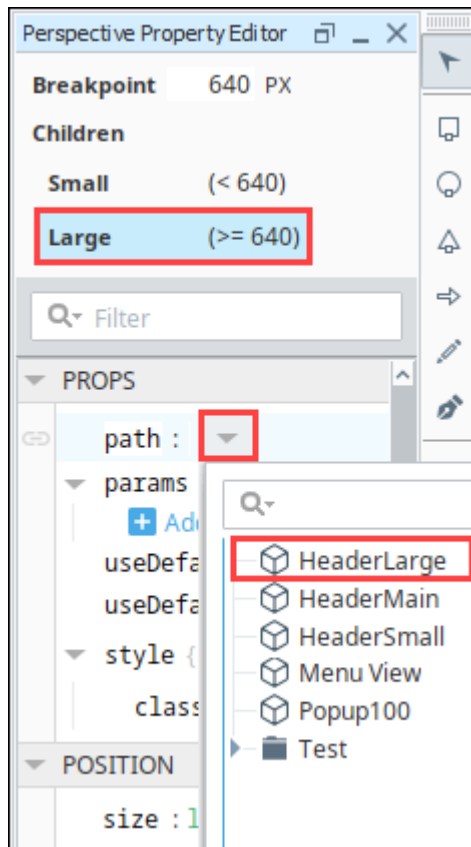
☐ **Page URL**
/headersmall ✓

Cancel Create View

- iv. Set the height of the view to **75** pixels.
- v. Drag an Icon component onto the upper left side of the view, then click on the component to select it.
- vi. 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](#).
- vii. 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
 - i. Create another view called **HeaderLarge**, again with a layout of **Coordinate**.
 - ii. Set the height to **75** pixels.
 - iii. 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.
 - i. Create a view called **HeaderMain** with a layout of **Breakpoint**.
 - ii. Set the height to **75** pixels.
 - iii. In the Property Editor, click on **Large** (under Children).
 - iv. Drag an Embedded View component on the HeaderMaster view.
 - v. Under PROPS, click on the **Expand** ▼ icon next to the path property then select **HeaderLarge**.

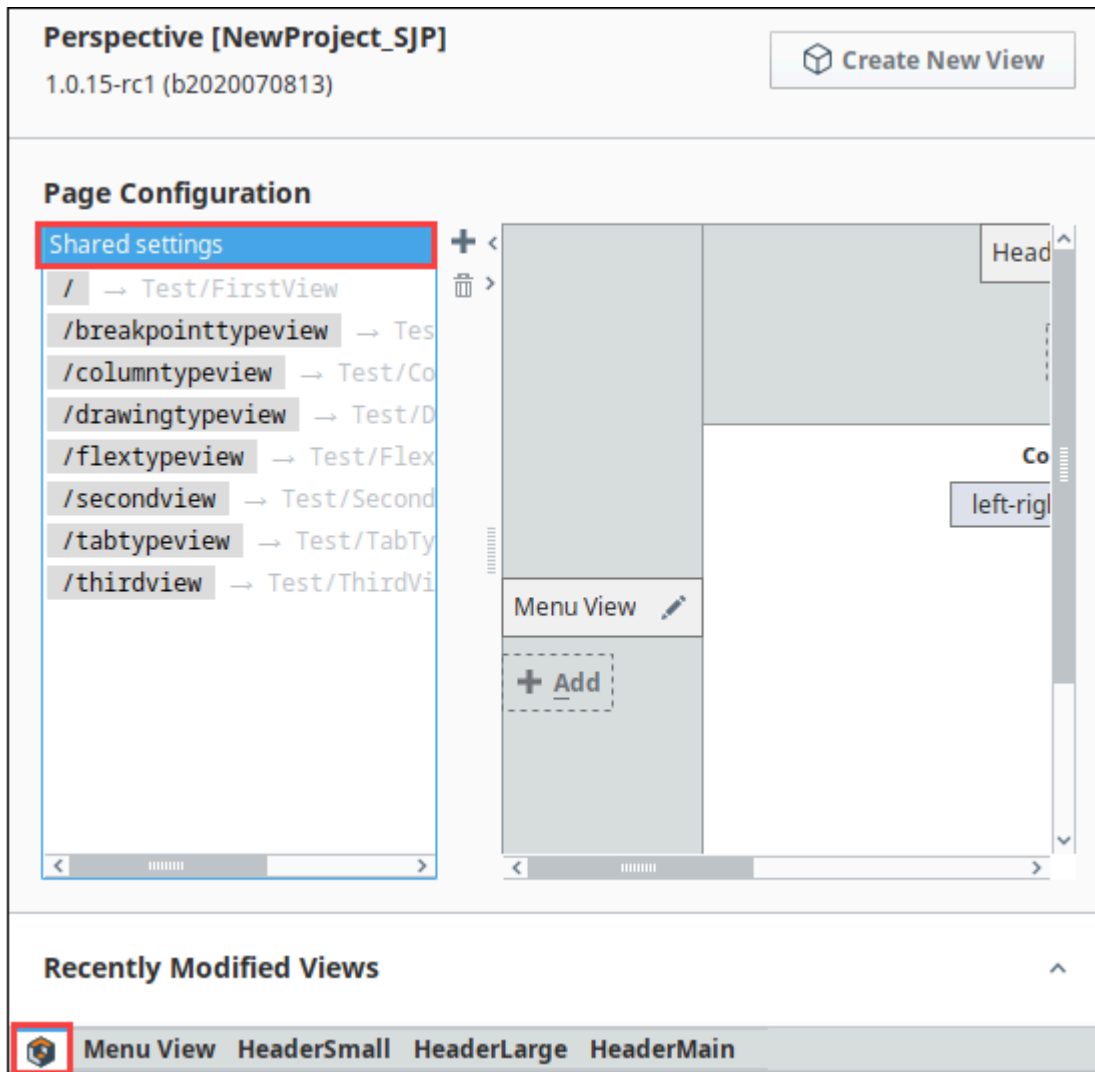


- vi. Now we'll do the same for the small child of this Breakpoint container. In the Property Editor, click on **Small** (under Children).
- vii. Drag an Embedded View component on the HeaderMain view.
- viii. In the Property Editor, click on the **Expand ▼** icon next to the **path** property then select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

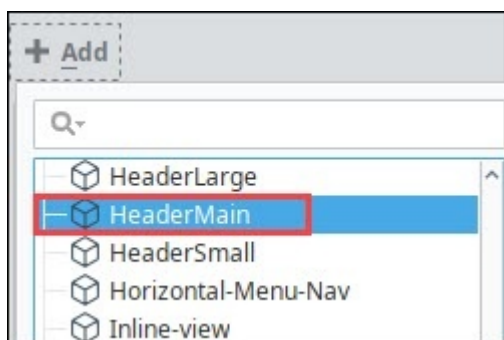
Page Setup

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.



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? Modal?

☐ false ☐ false

Content

push 

Anchor

fixed 

Size

75  

Auto Breakpoint

480  

Dock ID

Handle

hide 

Handle Icon

View Parameters

 Add Object Member...

Remove

OK

Cancel

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

+ Add Object Member...

OK

Cancel

- 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]

Create New View

Page Configuration

Shared settings

/ → Test/FirstView

/breakpointtypeview → Test/Co

/columnview → Test/Co

/drawingtypeview → Test/D

/flexview → Test/Flex

/secondview → Test/Second

/tabtypeview → Test/TabTy

/thirdview → Test/ThirdVi

HeaderMain

+ Add

Corner Priority

left-right

top-bottom

Menu View

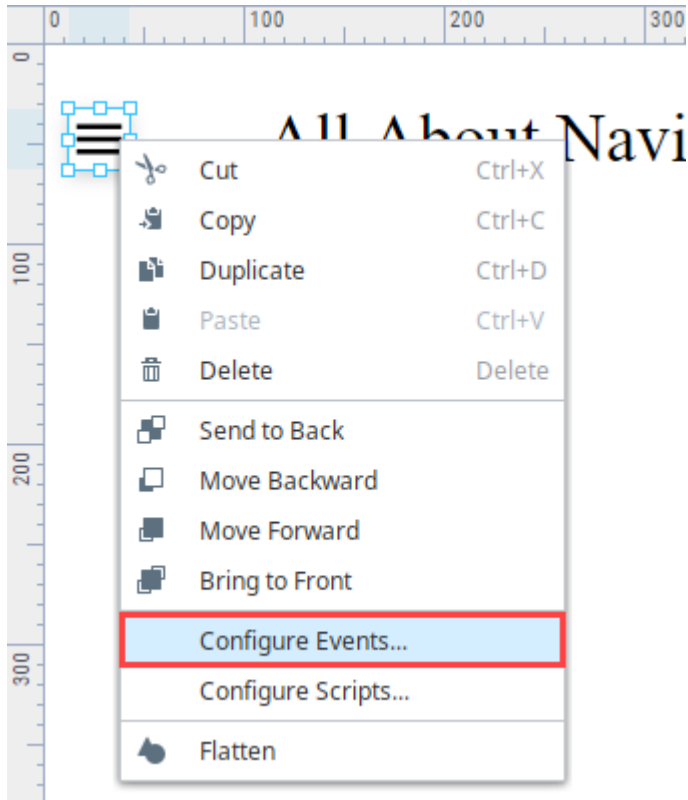
+ Add

+ Add

Configure Menu Icon

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

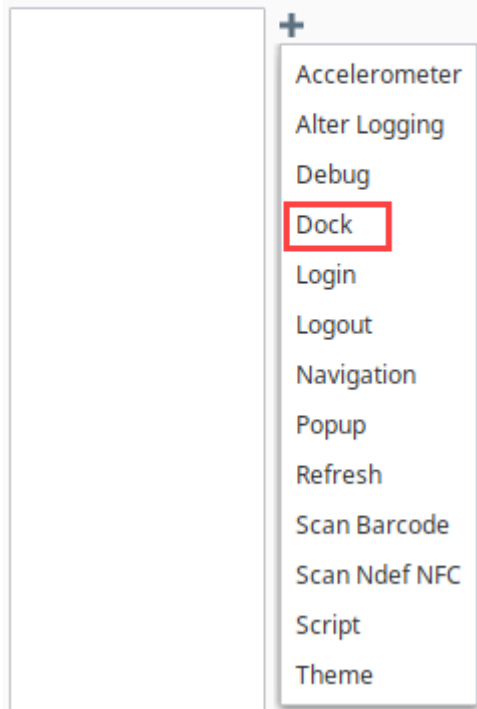
1. Open the HeaderSmall View.
2. Right-click on the Icon component, and select **Configure Events....**



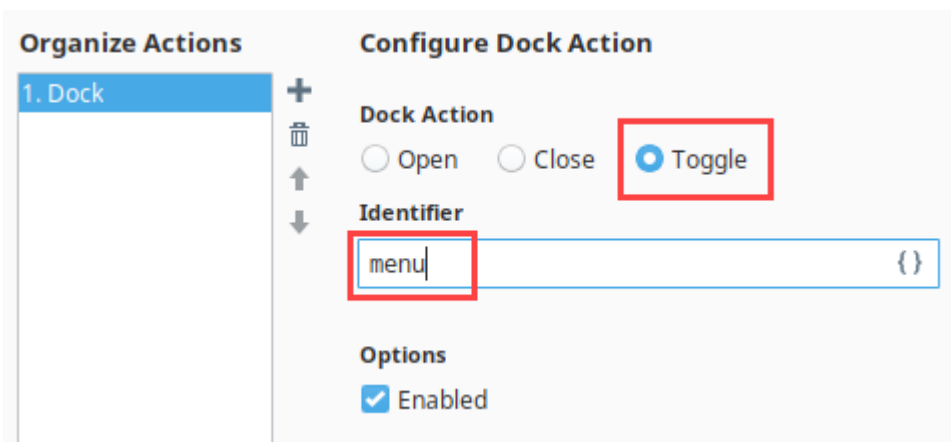
3. Under Mouse Events, select **onClick**.
4. Next click the **Add +** icon to add an action. Select the **Dock** action.

Configure [onClick] Actions

Organize Actions



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



Now you can 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



INDUCTIVE
UNIVERSITY

Component Overview

[Watch the video](#)

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

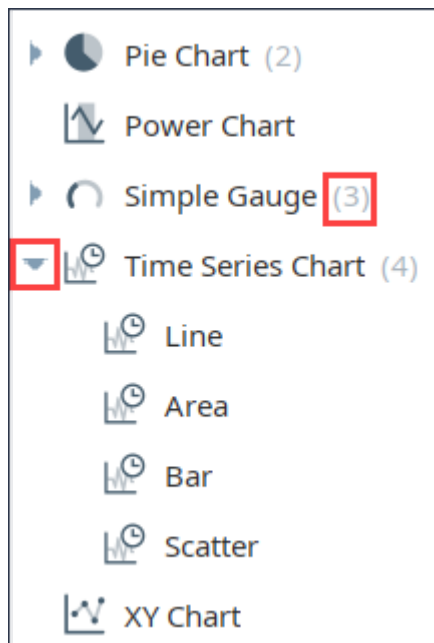
Properties on Perspective components are separated into categories.

- **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](#).

Variants

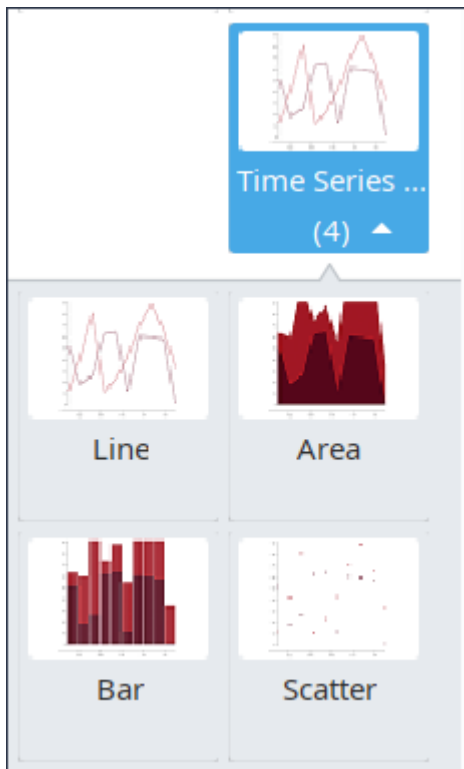
Some Perspective components have multiple pre-configurations, called variants, based on options a user might commonly use. For example, the Time Series Chart components has variants that set the chart up as a Line chart, Area chart, Bar chart, or Scatter chart. Using a variant can save you time. Variants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



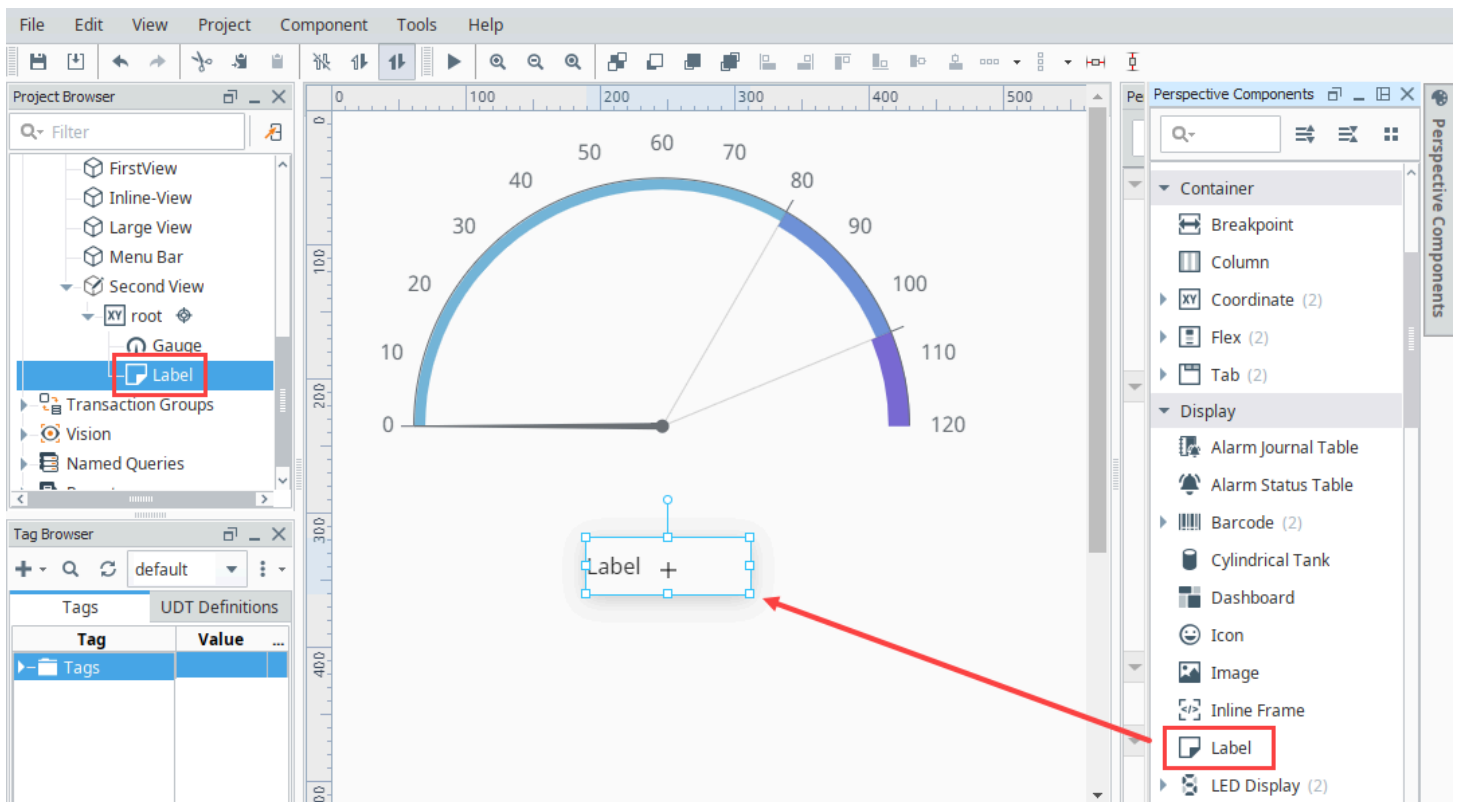
List View vs Tile View

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.

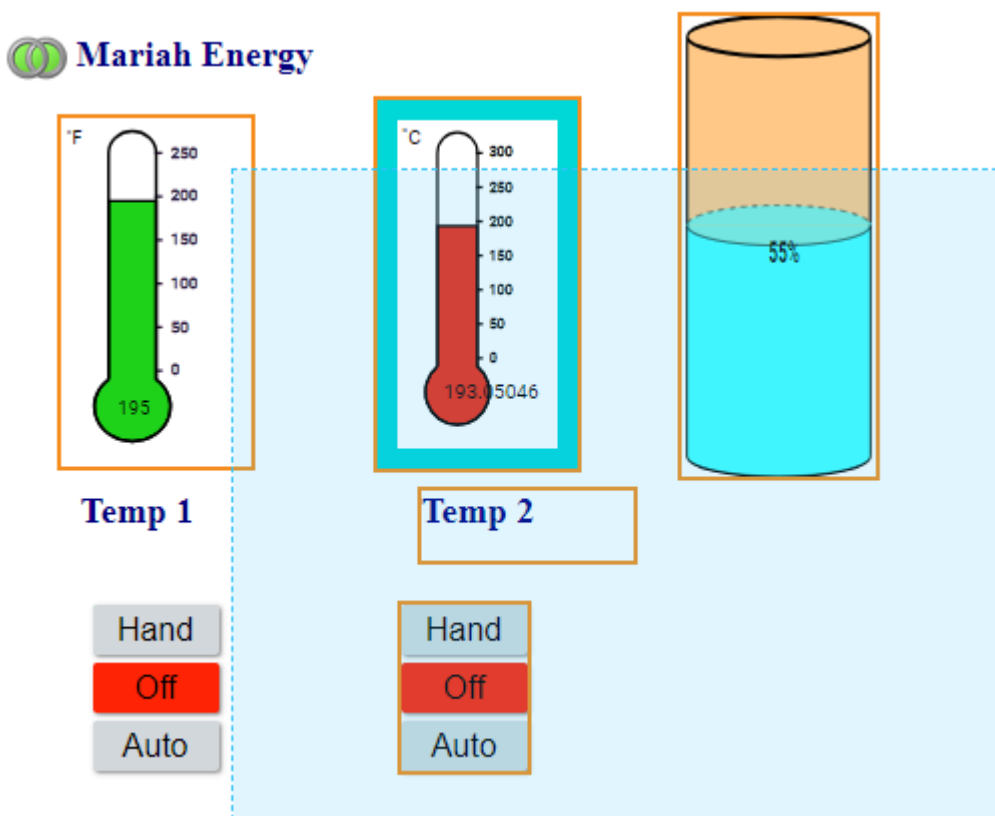


Selecting Components

Mouse Selection

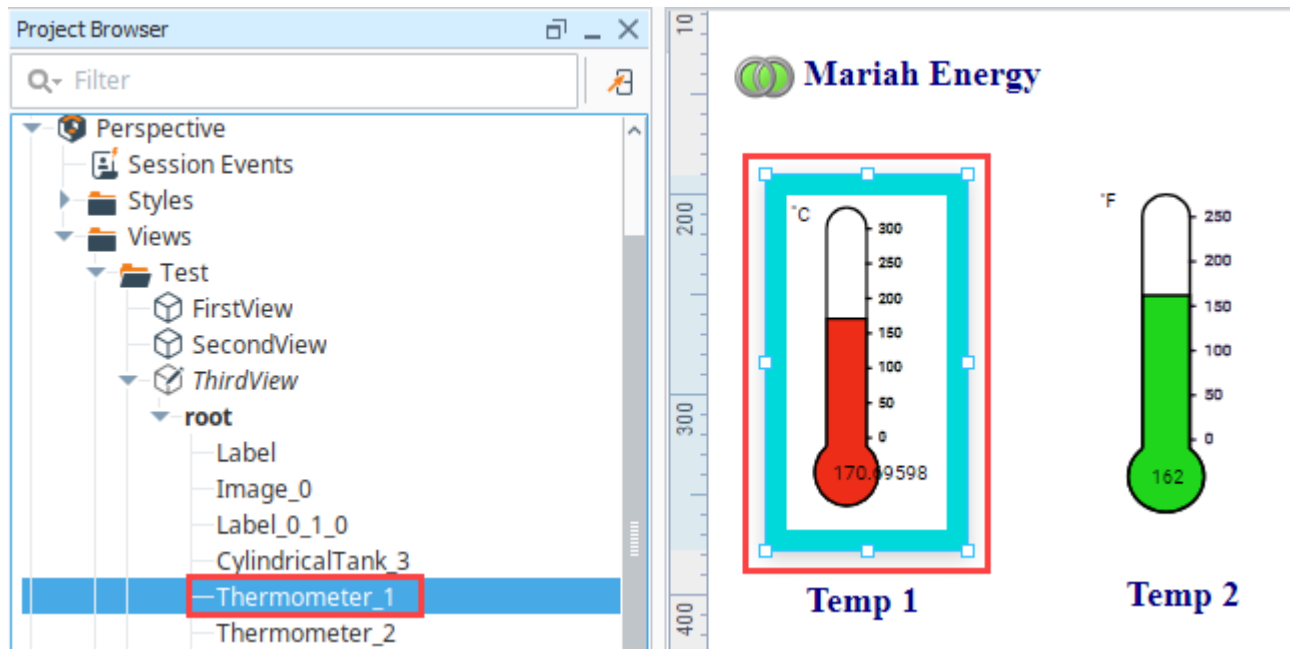
Using the mouse is the most common way to select components, where a single click on a component selects it and double clicking **Deep Selects** it.

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.

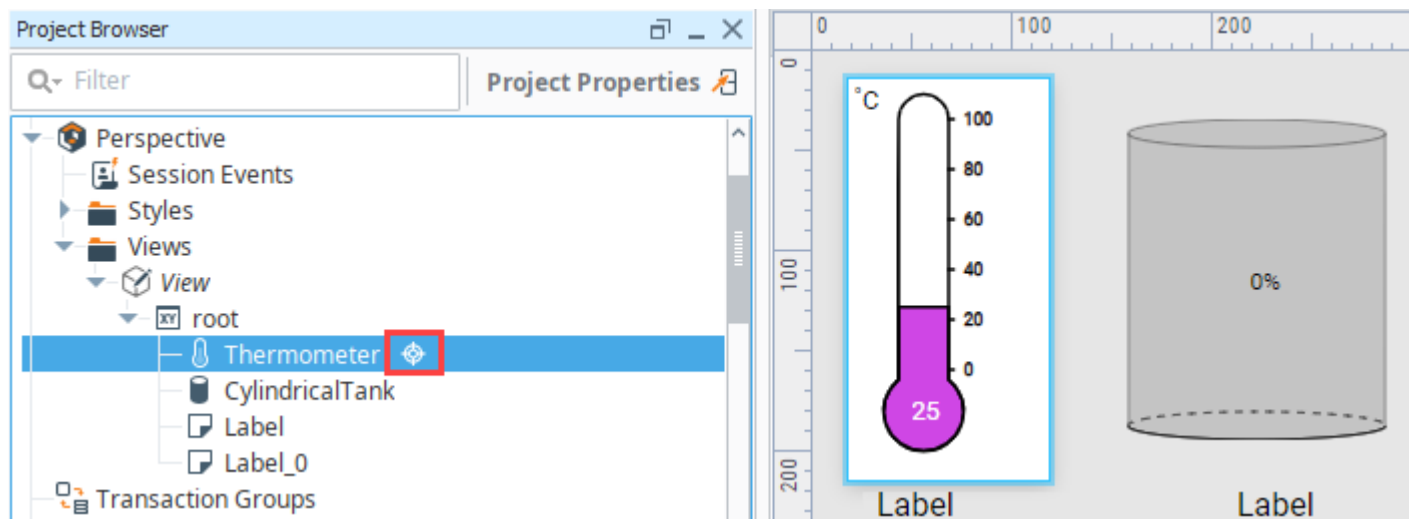
To deep select a component, **double-click** on it. Three things happen to indicate you have deeply selected into the component and not just selected the component itself:

- 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.

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

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.

	Cut	Ctrl+X
	Copy	Ctrl+C
	Duplicate	Ctrl+D
	Paste	Ctrl+V
	Paste & Replace	Ctrl+Shift+V
	Delete	Delete
	Send to Back	
	Move Backward	
	Move Forward	
	Bring to Front	
	Wrap in Container	
	Configure Events...	Ctrl+J
	Configure Scripts...	Ctrl+K

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

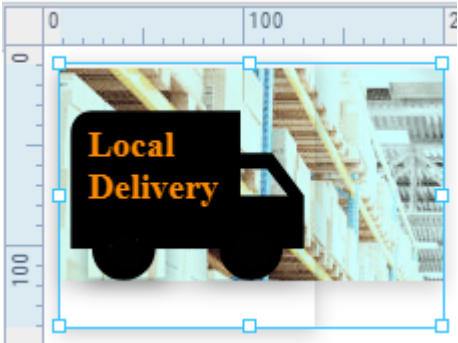
Function	Icon	Description
Cut		Removes the selected component but keeps a copy on the clipboard. You can also use the Ctrl-X shortcut to quickly cut a component.
Copy		Copies the selected component to the clipboard. You can also use the Ctrl-C shortcut to quickly copy a component.
Paste		Pastes the current 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.

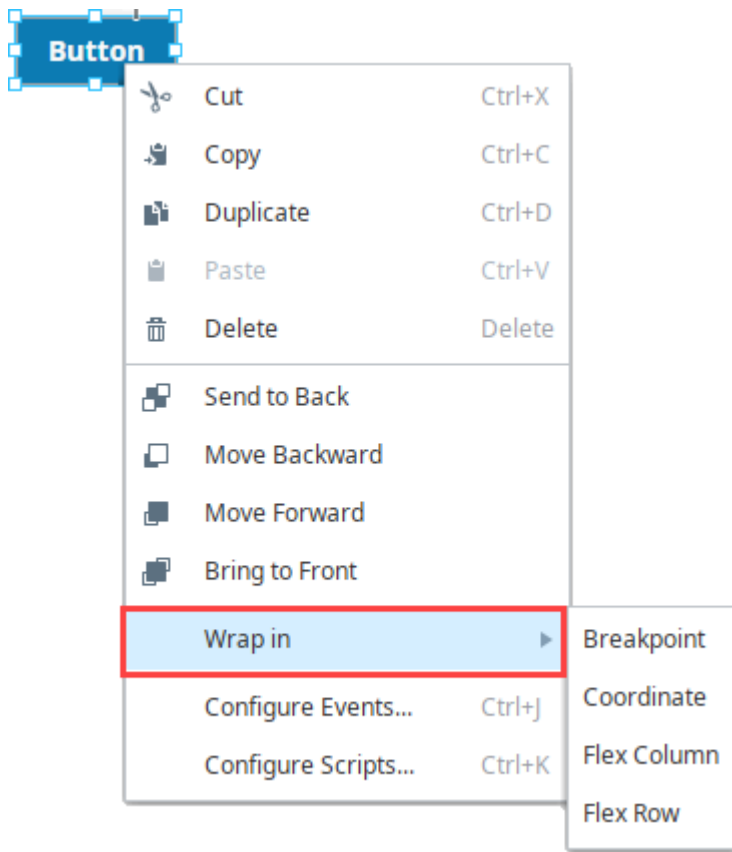
Function	Icon	Action
Send to Back		Move the selected components to the back of the z-order.
Move Backward		Move the selected components backward in the z-order relative to any overlapping components.
Move Forward		Move the selected components forward in the z-order relative to any overlapping components.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

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

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.

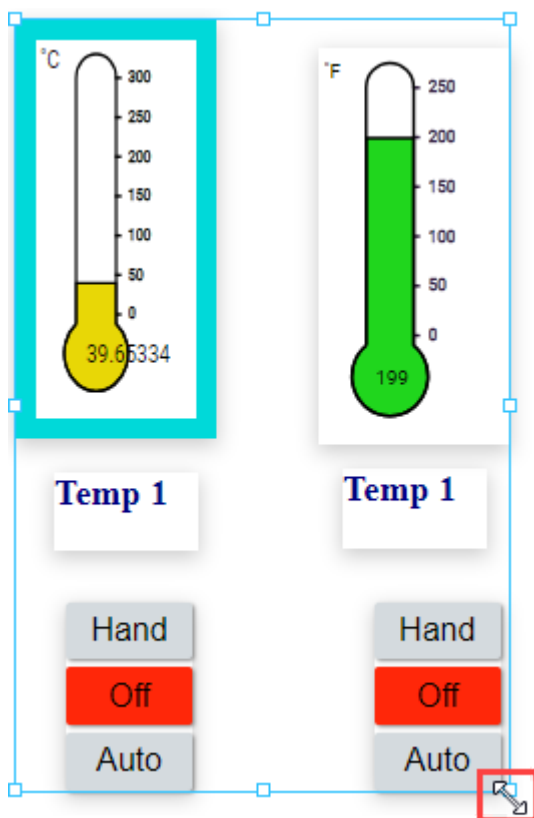
Moving

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

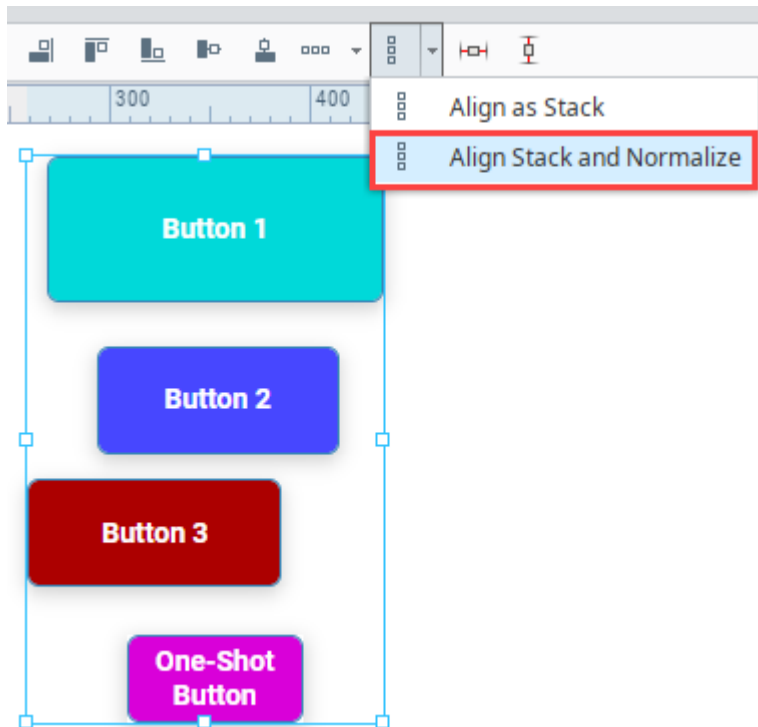
These tools can only be used with components in a Coordinate container.

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.

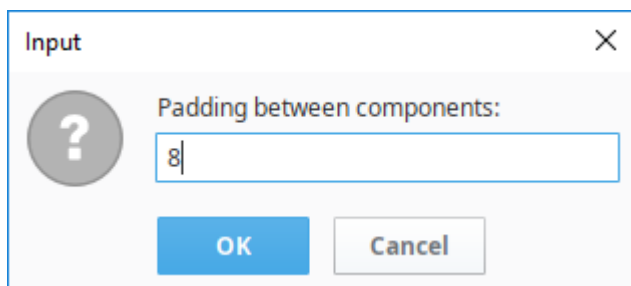
Align and Normalize Example

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.

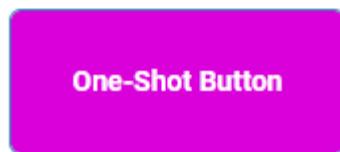
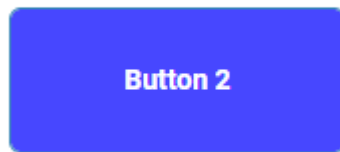
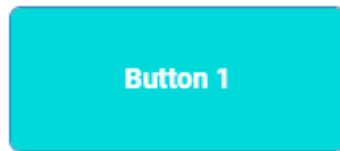
1. Select the buttons, then choose Align as Stack and Normalize.



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



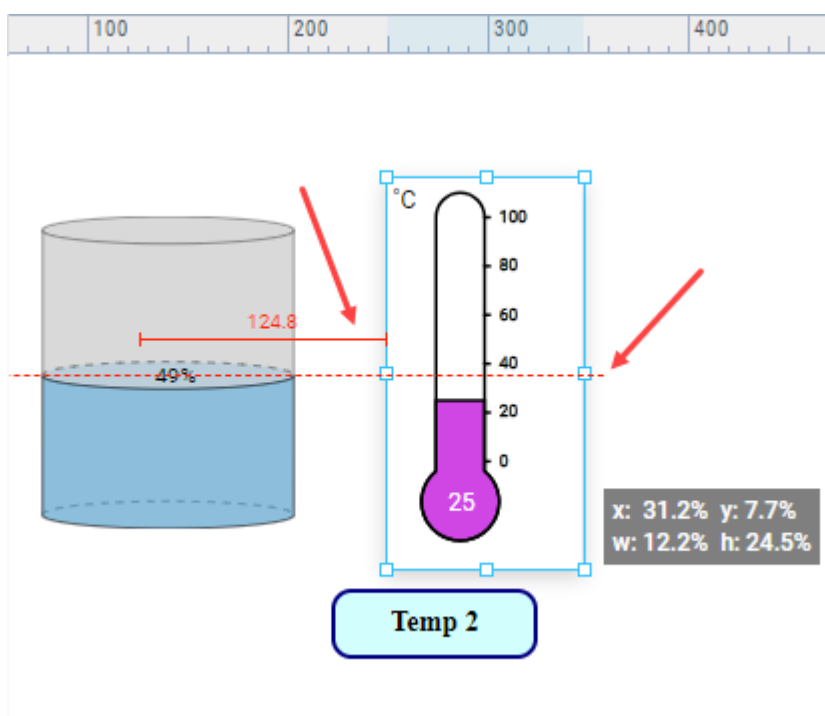
3. Click **OK**. The buttons will be stacked and normalized.

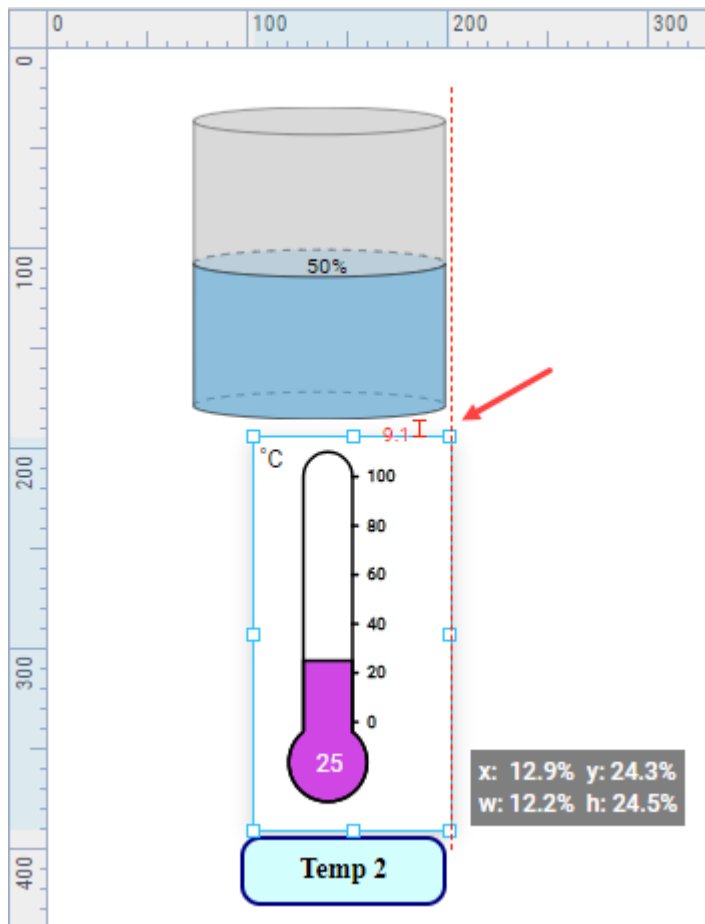


Alignment Guides

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.



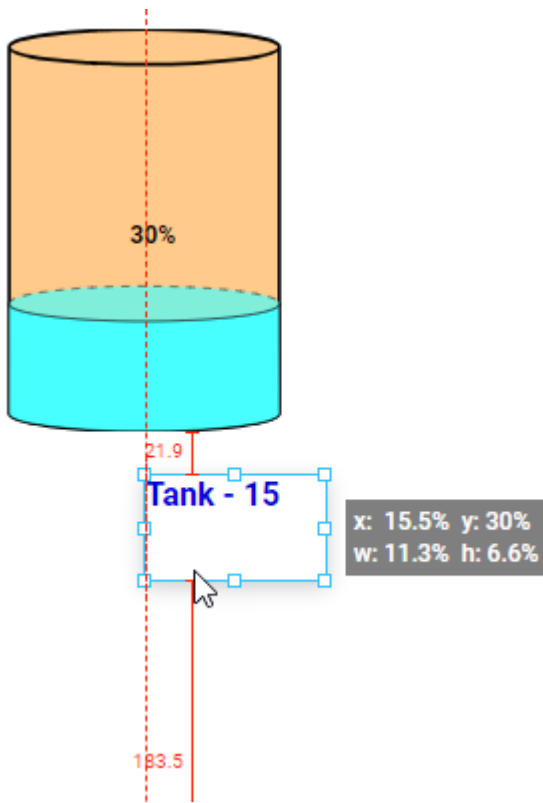


The Designer also has Vertical and Horizontal guide that can be set up to help you align components. For more information, see [Vertical and Horizontal Guides](#).

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

Text

Font family

Size

Color

#0000D9

Weight

bold

☐ Italic

Line height

Letter spacing

Word spacing

Alignment

Text align

center

Indent

White space

Text Options

Transform

Tt TT tt

Decoration

S ~~S~~ S

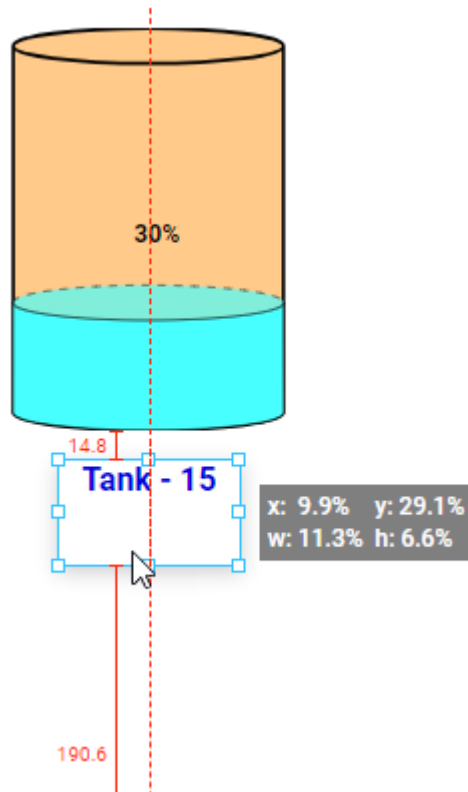
☐ Break-word

Shadow ☐

OK

Cancel

- 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.
- Stop dragging when the Label is centered under the Cylindrical Tank.



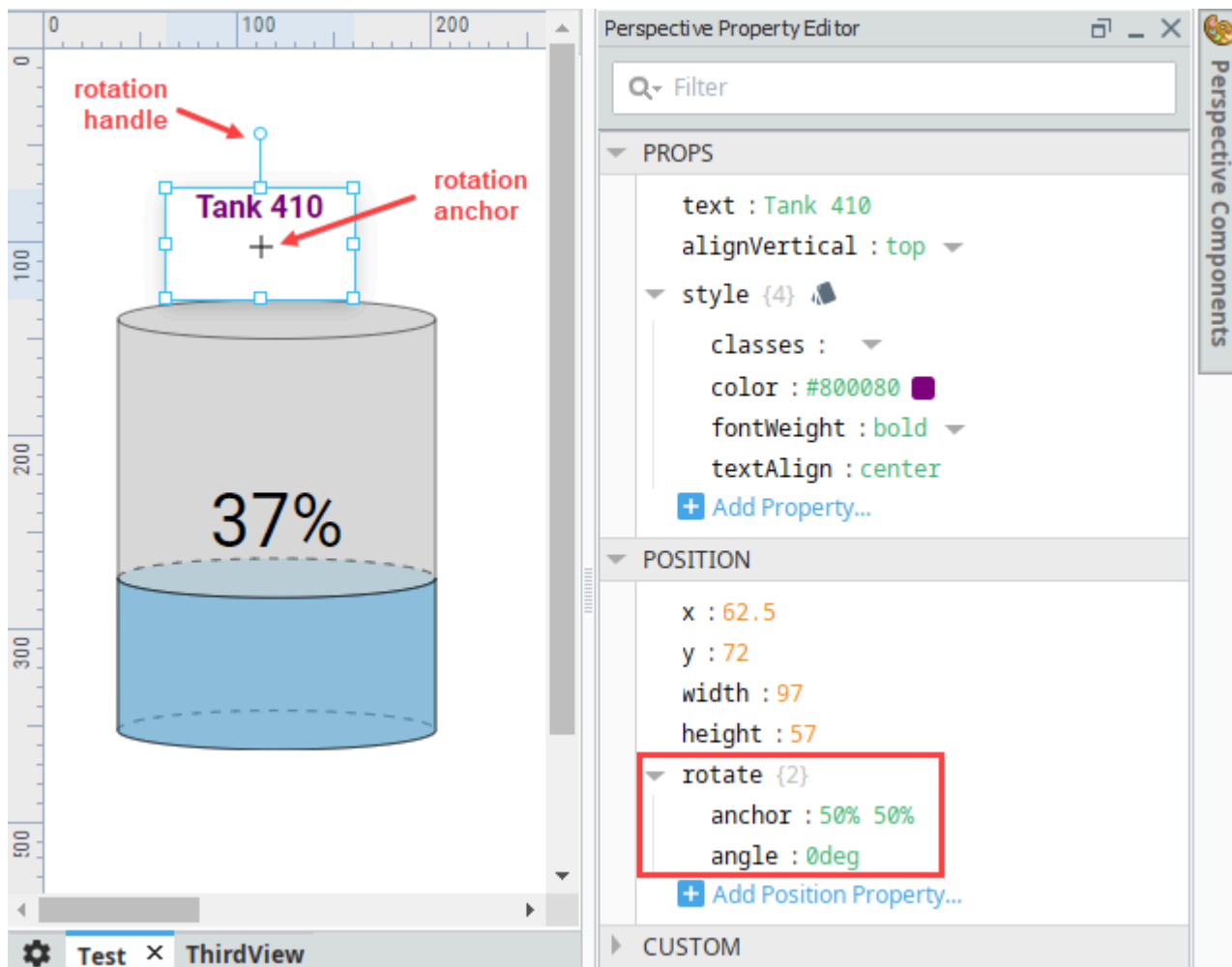
Rotating

Components placed on coordinate containers can be rotated. The Rotate property is located on the 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 rotated as you'd like.

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



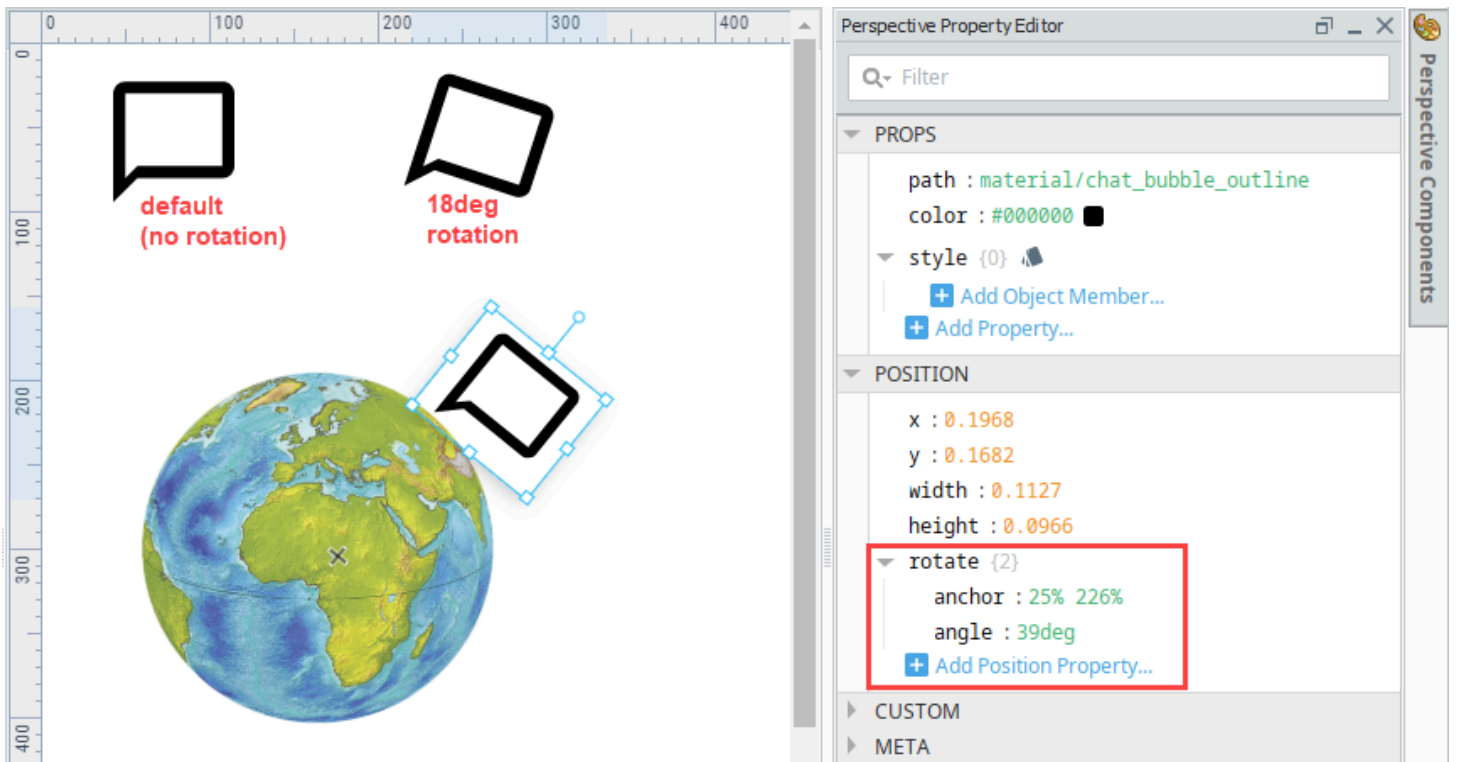
NOTE

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.



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.

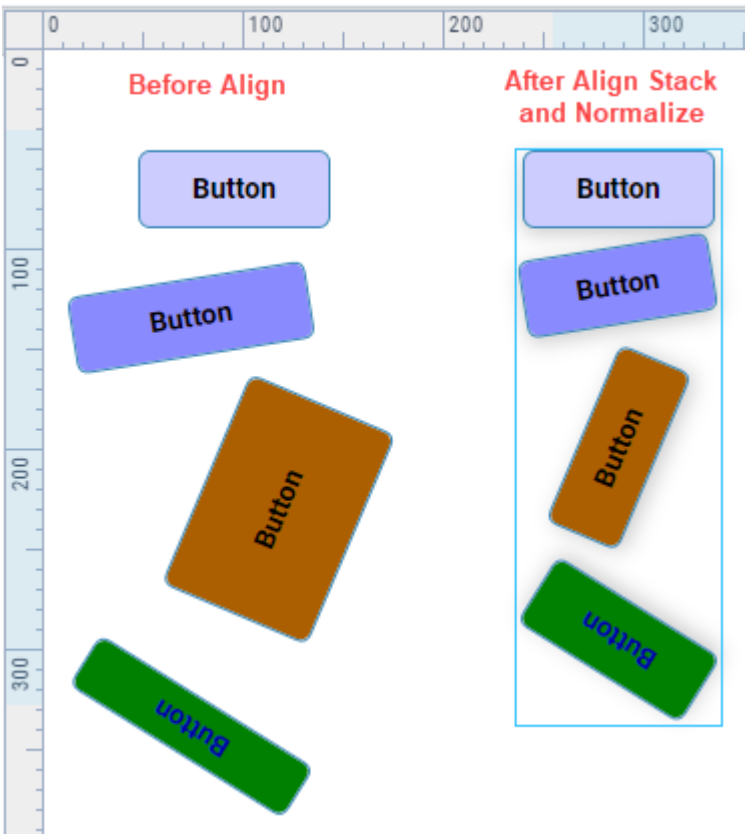


Image Source

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 web-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](#).

Bypassing Browser Cache

Some Perspective components serve as wrappers for instances of a Chromium browser, which comes with powerful features like being able to make HTTP GET requests to any domain on the internet. Since this occurs inside of your web browser, it's important to pay attention to the browser cache and how it might affect interactions with other webserver.

For example the **Inline Frame** component will render the result returned from HTTP GET requests inside of an instance of Chromium embedded in your View. If you wanted to use an iFrame to render a PDF file you had hosted on a remote webserver, you would plug the URL to the PDF into the iFrame `src` property and the iFrame would render the file.

However, if you later realize you uploaded the wrong version of the file to the webserver, you may notice that uploading a new version with the same file name will not change the rendered file in your iFrame component. This is a result of the **HTTP Cache** in action. You can fix this issue by clearing the

cache or renaming the file and updating the `src` property in the Designer to point to the new URL. In cases where neither of these fixes are an option, you can alternatively bypass the cached resource and pull in the new file using an Expression binding on the `src` property. This is accomplished in the following expression:

```
'http://YOURURL?timestamp=$' + floor(toMillis(now())*0.00001)
```

Make sure to replace `YOURURL` with the relevant webserver address. This example will update once every 100,000 milliseconds. Adjust the 0.00001 value in the expression to an update rate that works for you.

Perspective Component Properties



INDUCTIVE
UNIVERSITY

Component Properties

[Watch the video](#)

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.

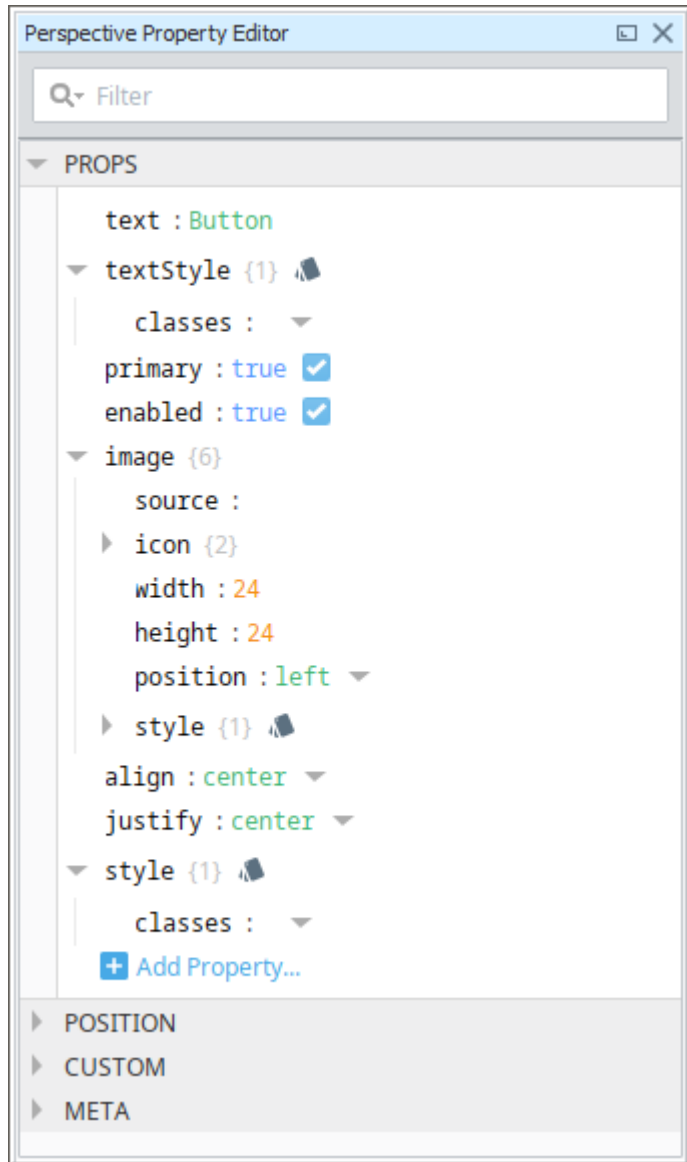
Property Categories

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.



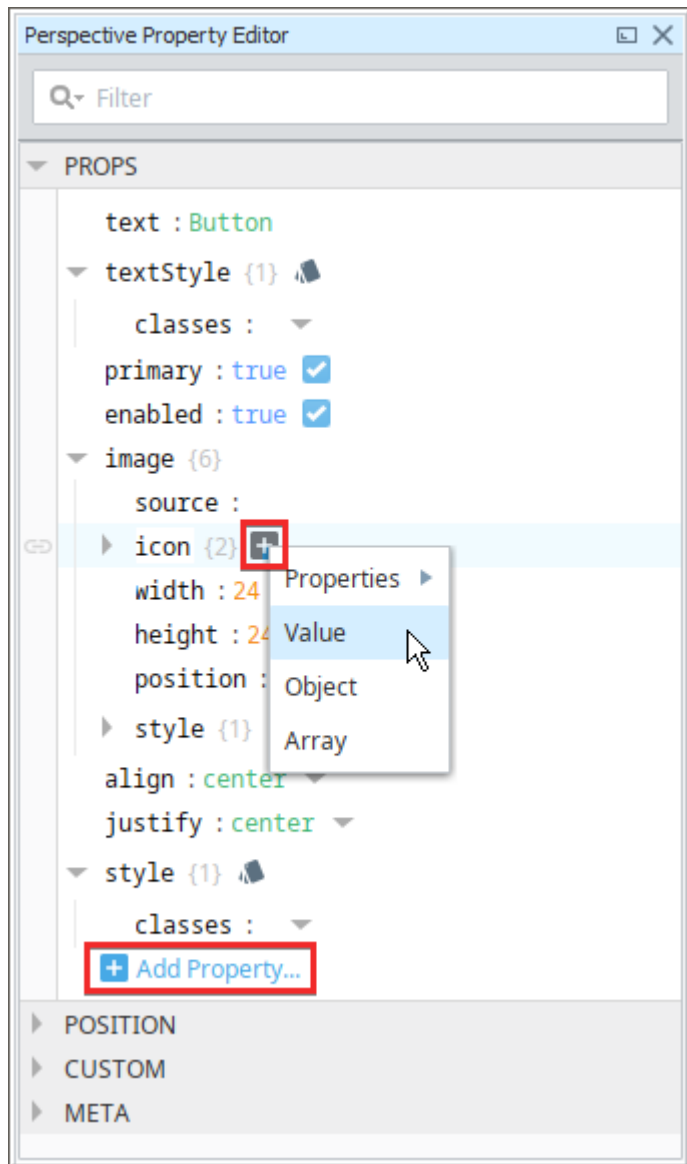
User Created Properties

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. Components 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 unintended behavior from the component. The Custom and Params categories don't have such properties, so they're a safe location for user created properties.

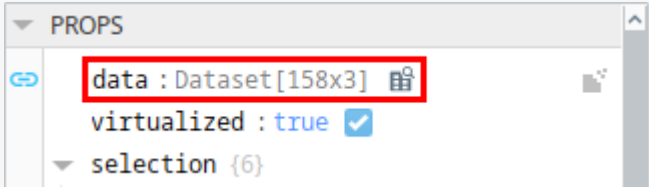
Alternatively, new elements can be added to objects and arrays by hovering over the object/array in the property editor, and clicking the **Add +** icon.

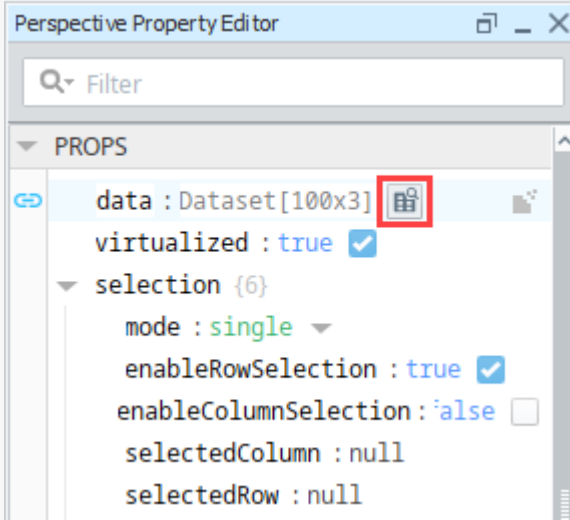
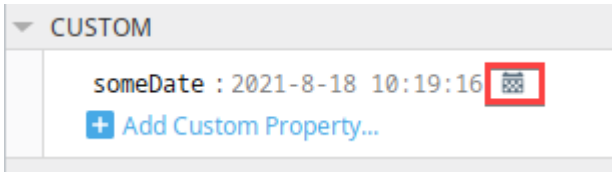


Property Data Types

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, or 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 Type	Description
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. Boolean values are displayed in blue. • Numeric: An integer up to the maximum value for a long integer. Numeric values are displayed in orange. • String: A string of characters can be numeric, alpha, or a combination. String values are displayed in green.
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. 
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).

Property Type	Description
	
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 [rowsxcolumns] 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. 
	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.

Property Type	Description
	 With the Dataset Editor you can add and delete columns and rows, delete all rows, and copy information to or from the clipboard. For Editor icons, see Dataset.
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.  Clicking the icon will cause a popup calendar to appear, making it easy to select a different date.

Dataset

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

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.

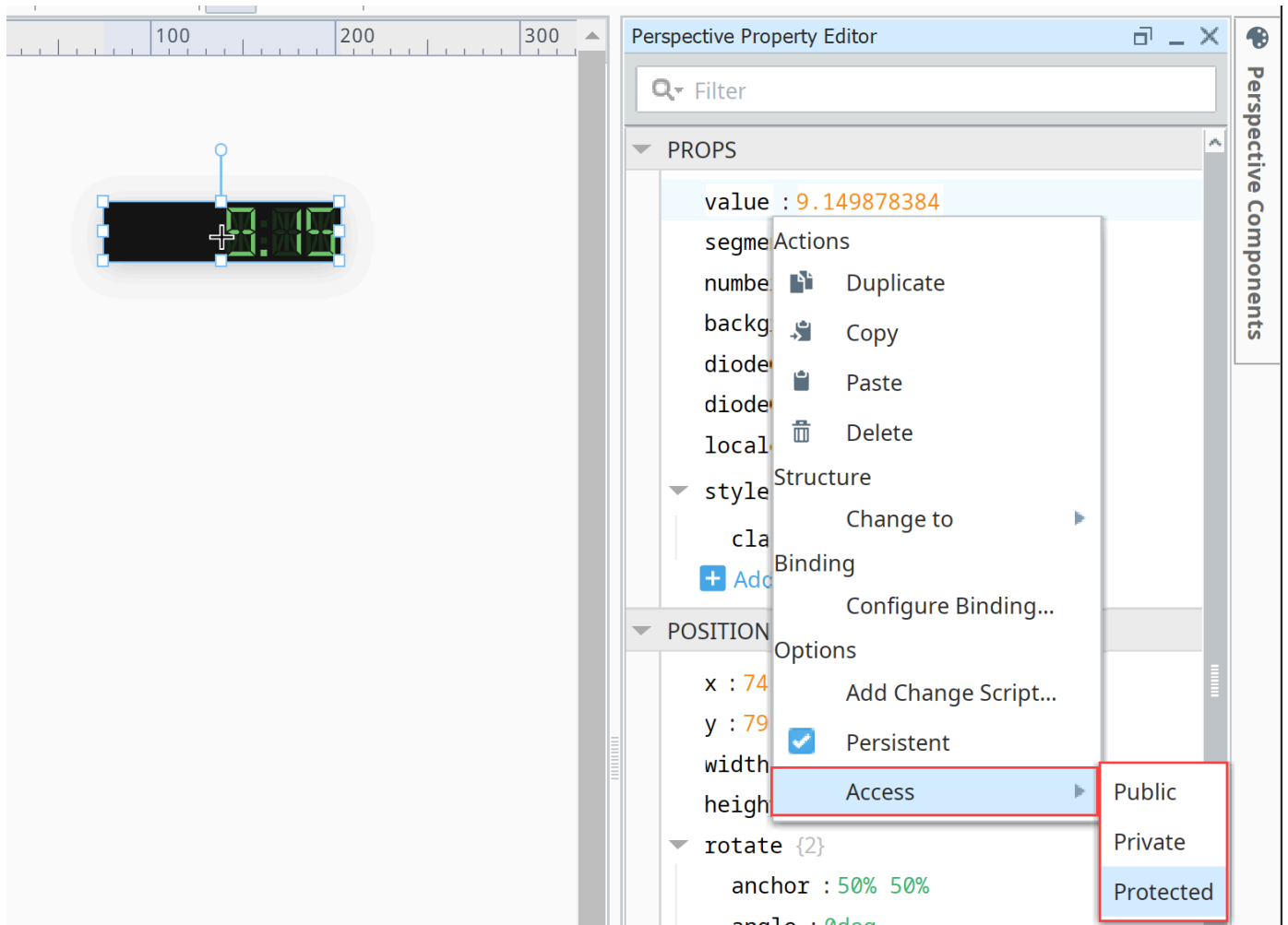
One caveat to property access is that the names of style classes are always visible and interactable from the browser, so style class names should not include sensitive information.

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.

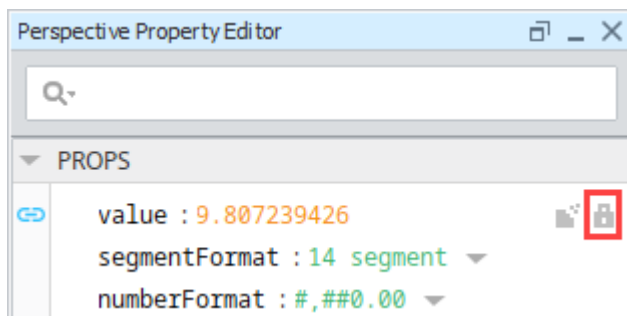
Restricting Access to Component Properties

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.

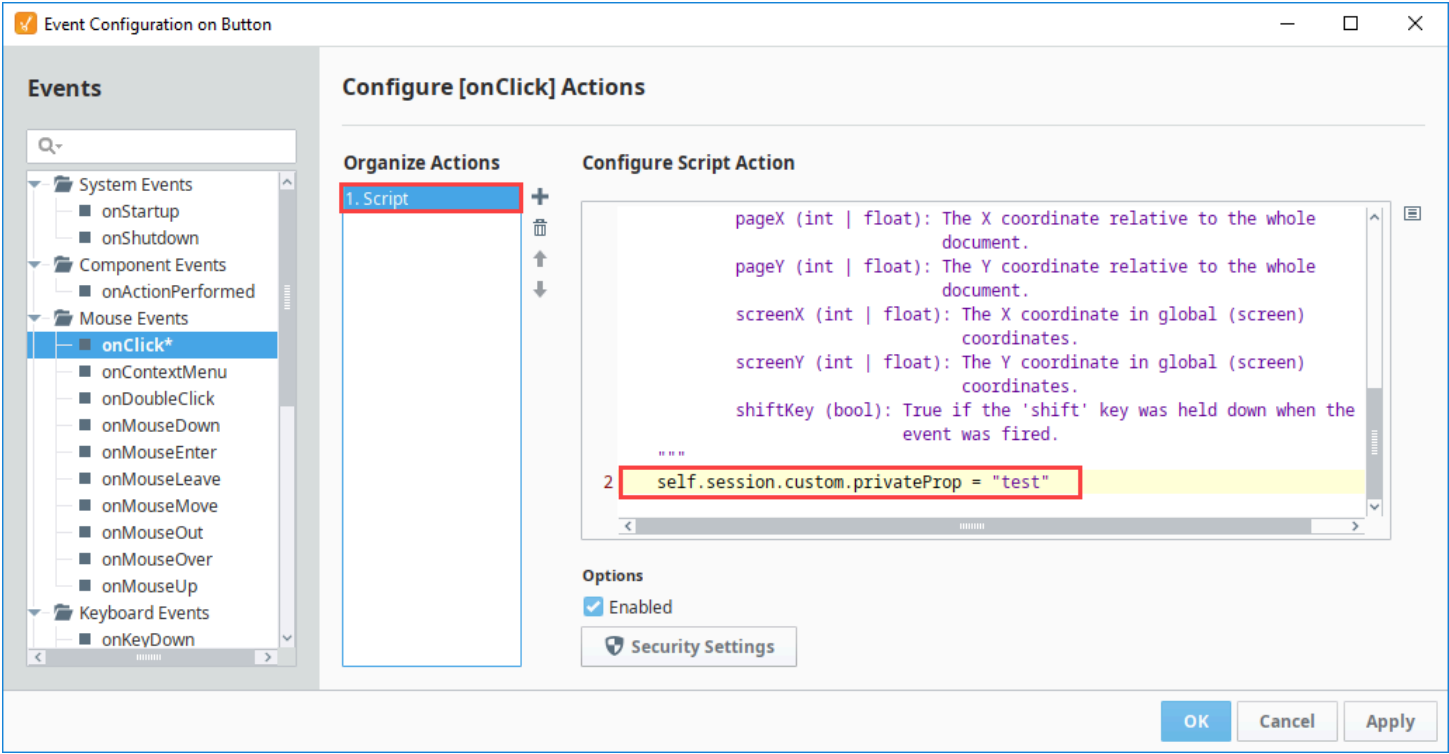


3. Once the restriction access is configured, the **Property Editor** will place one of three badges on any non-public property: Private , Protected , and System .

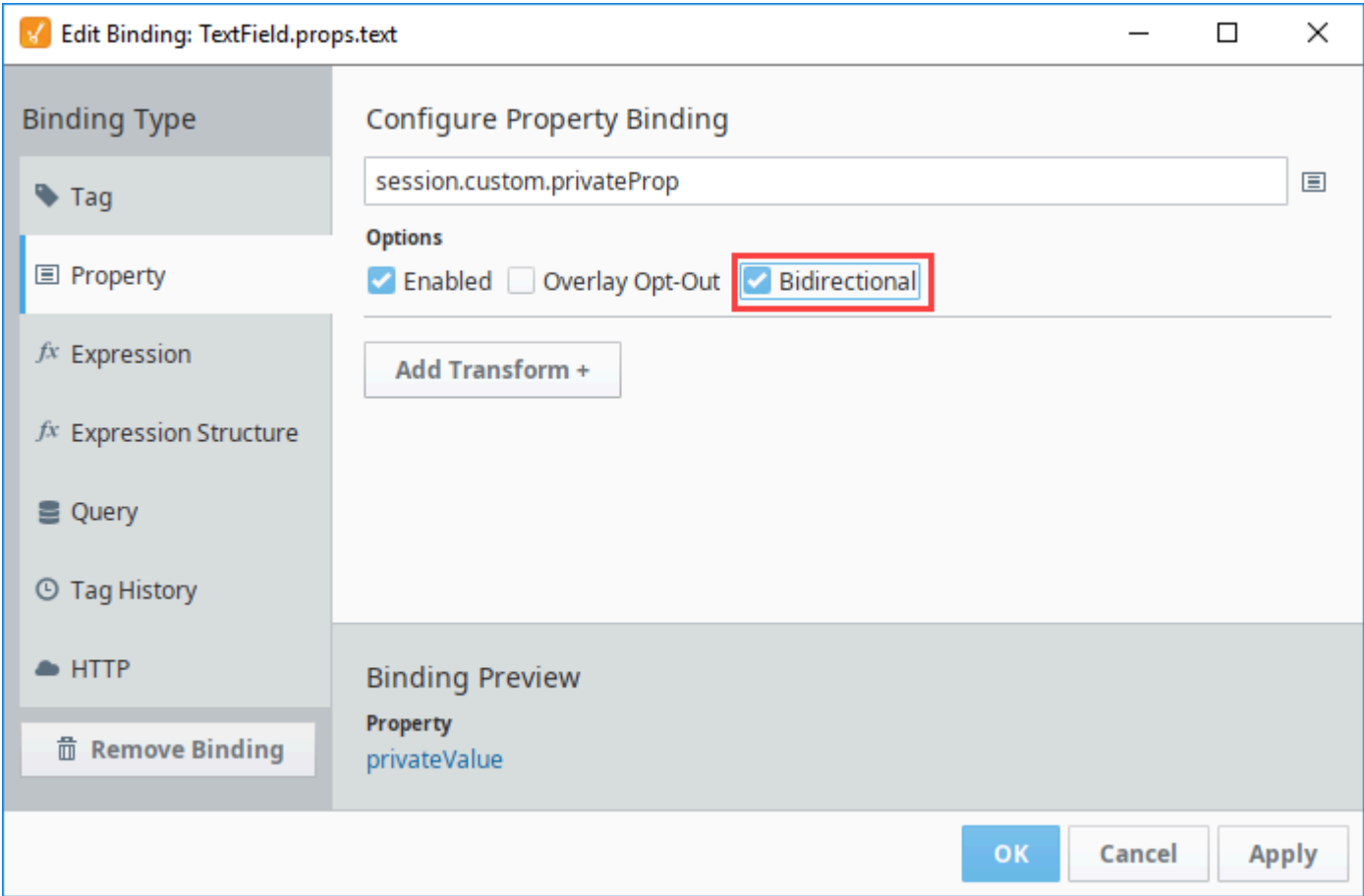


Writing to Private or Protected Session Property

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



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



Persistent Properties



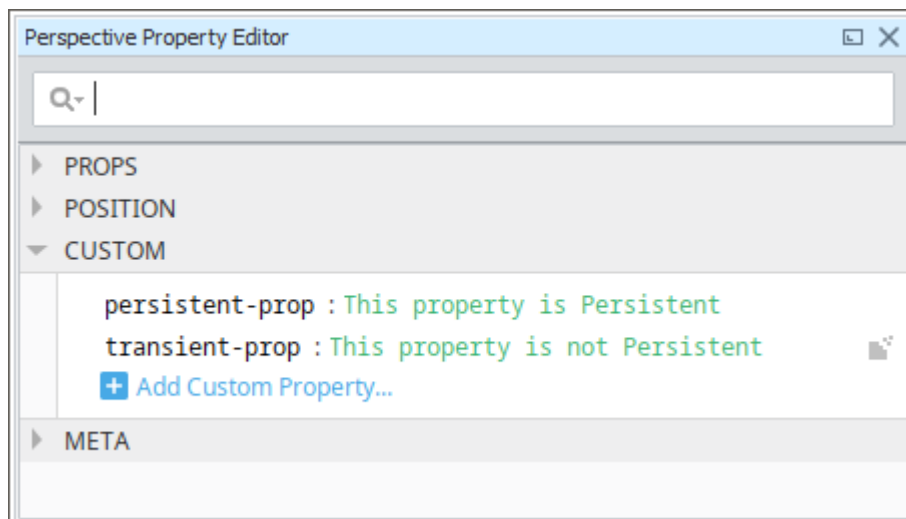
INDUCTIVE
UNIVERSITY

Persistent Properties

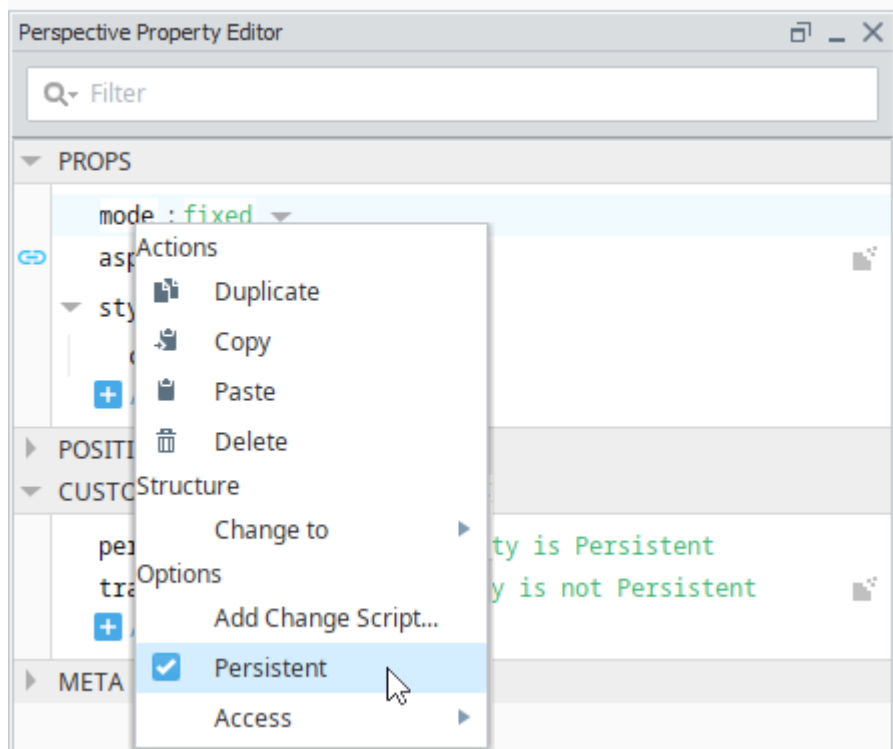
[Watch the video](#)

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.



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. The existence of a binding on a property will result in the property being 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 up-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.

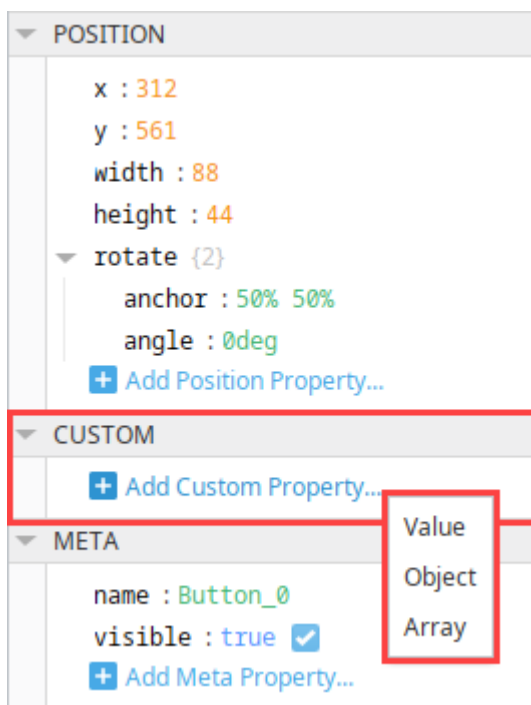
Custom Properties

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.

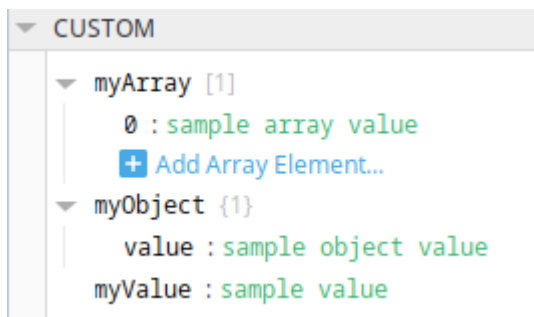
Creating Custom Properties

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

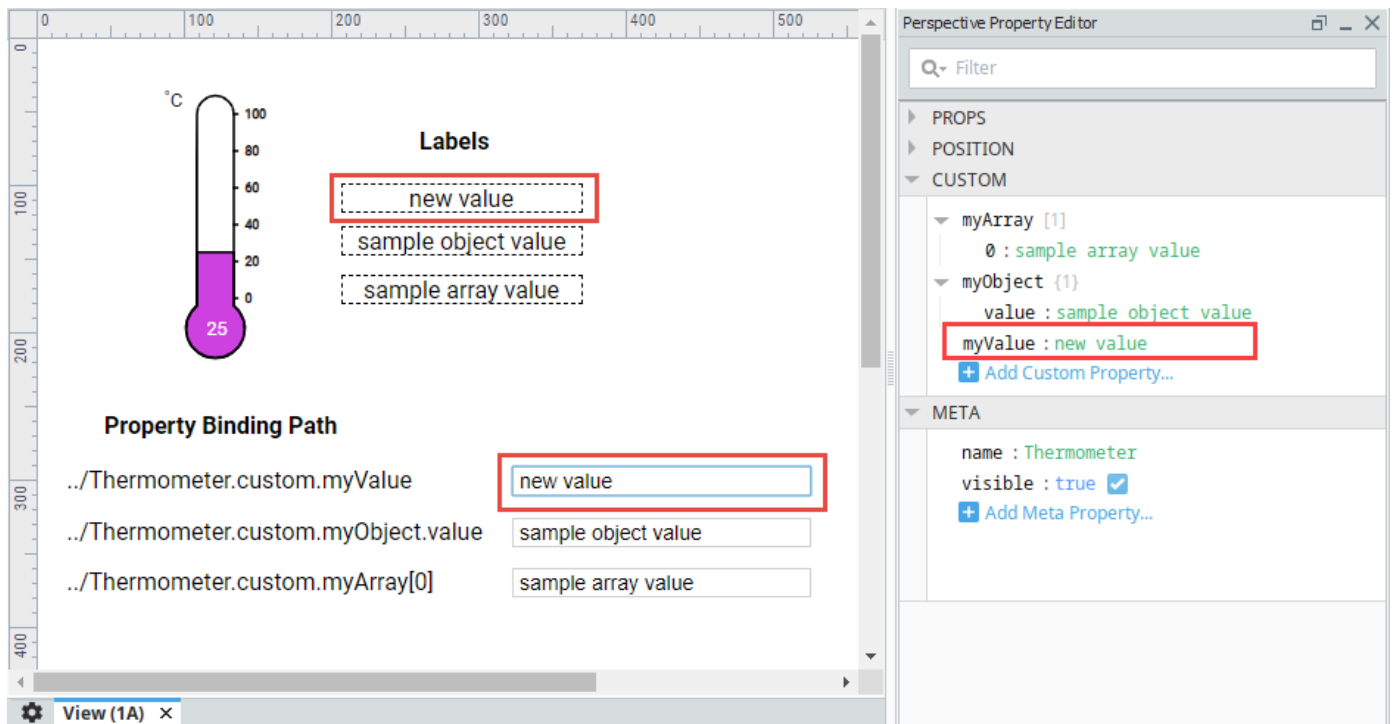
1. Custom Properties are created by clicking on the "Add Custom Property..." link, or by right clicking on a property in the Property Editor to bring up the **Action Menu**. There are three property types to choose from: Value, Object, and Array.



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).



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.



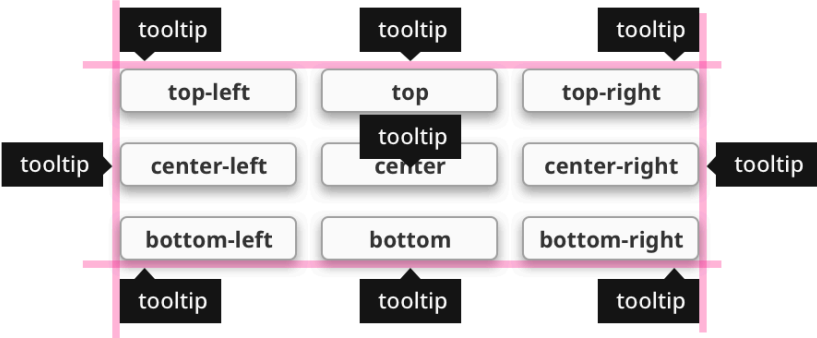
Meta Properties

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.

Name	Description	Data Type
name	Name of the component used when navigating tree paths by name.	value: string
visible	Whether or not the component should be visible.	value: boolean
tooltip	When enabled, the component will display a customizable tooltip. See tooltip .	object
contextMenu	When enabled, the component displays a context menu when right-clicked. See contextMenu .	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
tabIndex	Hidden by default. When added to the Meta category on a component, it allows you to set an integer value to establish the sequence in which each component will receive focus by tabbing through the contents of a page.	value: integer

tooltip

Name	Description	Data Type
enabled	If true, a tooltip for the component will display on hover or when requested by a script.	value: boolean
width	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.	value: string
text	The message to display in the component tooltip. See Multi-Line Tooltips for more information.	value: 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: • The tooltip must remain in the viewport at all times. • The tooltip will expand away from the component relative to the pink guidelines shown below.	value: string

Name	Description	Data Type
	 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.	
tail	Enables a decorative triangle on the tooltip that points to the component. Ignored when location is mouse.	value: boolean

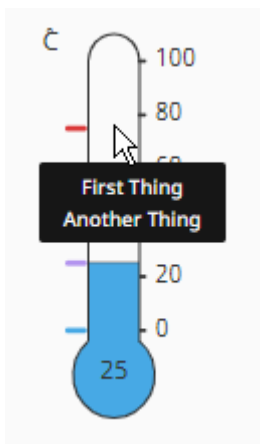
contextMenu

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. See items .	object
style	Standard style object for the context menu.	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. • path: Shorthand path to icon source, using format: library/iconName. Value is 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 is 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

Multi-Line Tooltips

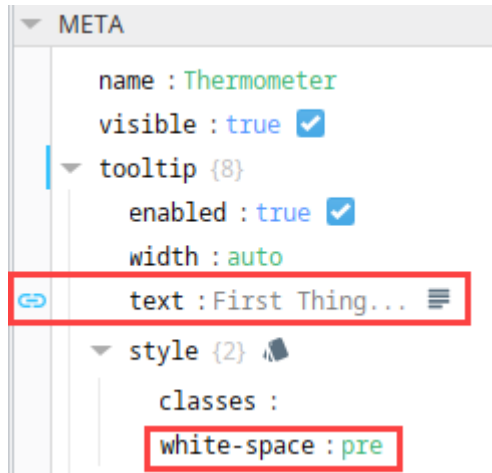
By default, the content in the `meta.tooltip.text` property is all rendered as a single line, only wrapping when text length exceeds the width. However it is possible to render text that spans multiple lines.



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"
```



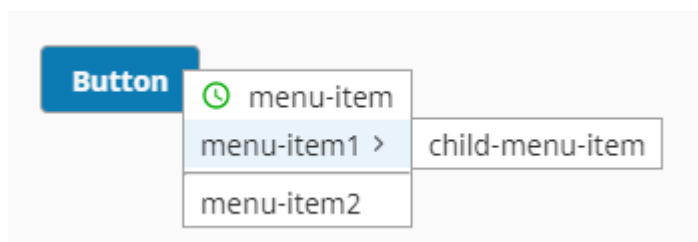
Context Menu Item Types

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.

Item Type	Description
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 specify 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 the [Docked Views](#) page.

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

Remove OK Cancel

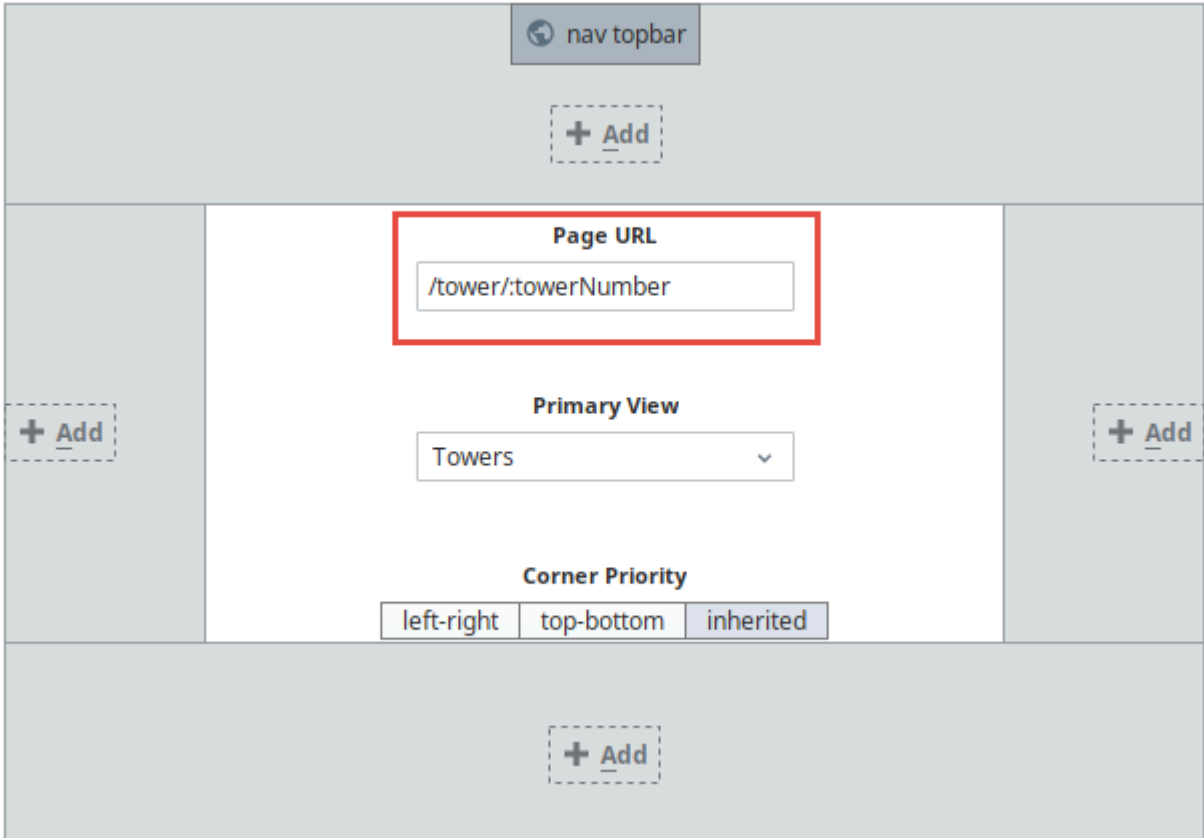
Embedded View

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/output**. 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

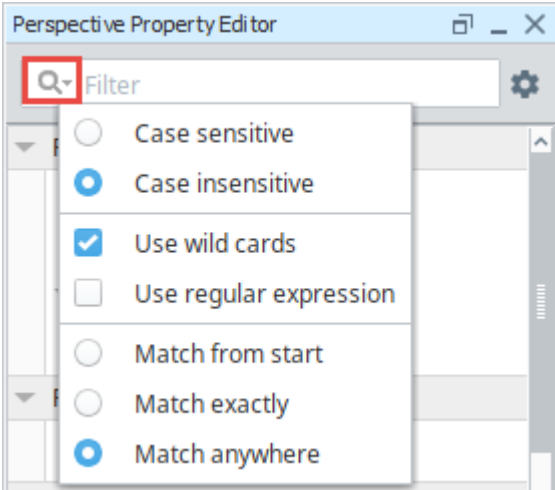
Passing parameters to a Page essentially means passing a URL parameter to a page. Parameters are used to allow a page to be mounted at a dynamic URL, allowing information in the URL to be interpreted as input parameters to the page's primary view. A primary view can see whatever parameters are passed 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 be dynamic and map to an input parameter on the page's primary view. To learn more, go to [Passing parameters \(URL Parameters\)](#).



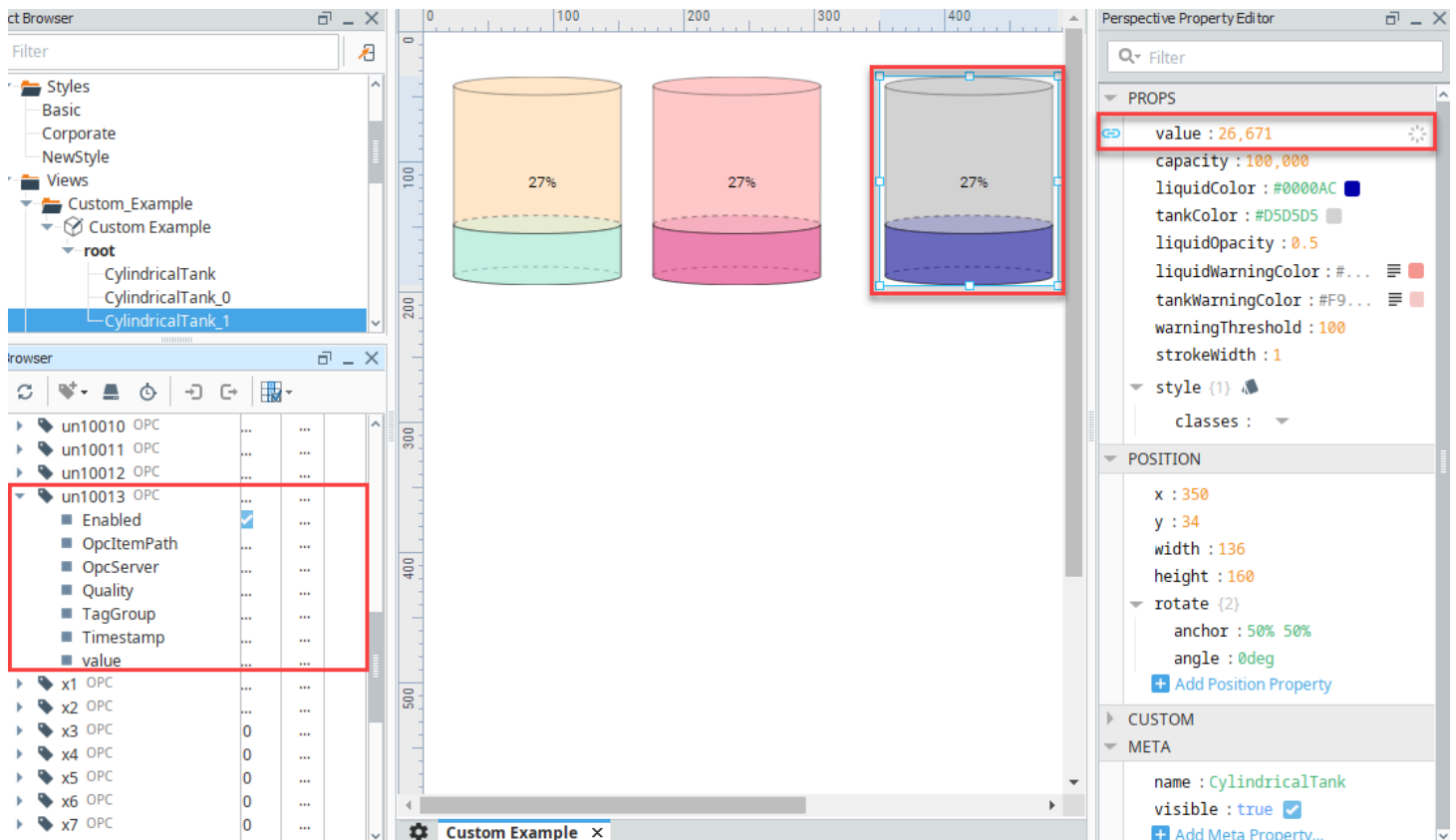
Search Filter

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.



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.



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.

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

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 clipboard 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



INDUCTIVE
UNIVERSITY

Images in Perspective

[Watch the video](#)

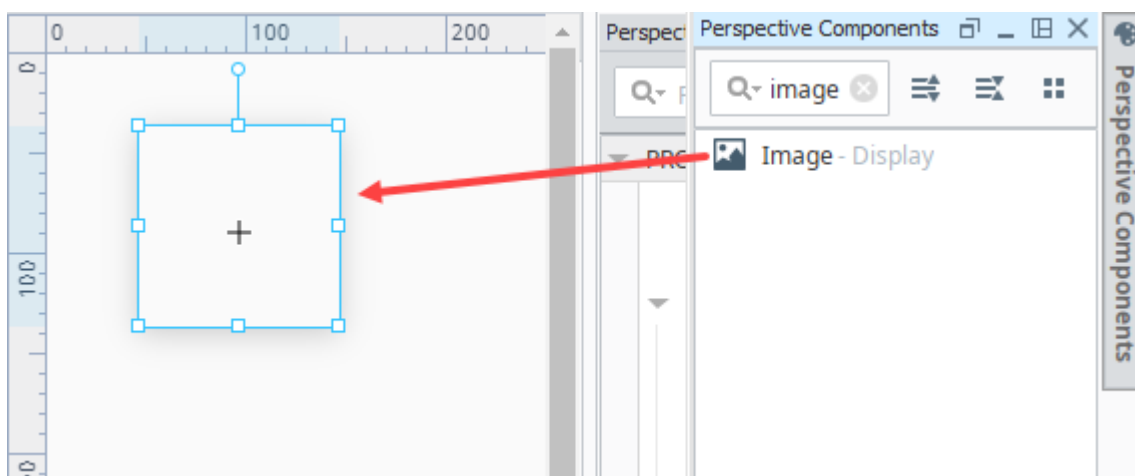
Perspective supports the import and display of PNG, JPG, JPEG, GIF, and SVG images. Images can be loaded into Perspective using the **Image Management** tool, dragged and dropped directly into a view, or derived from an external web address. All images are displayed using the **Image** component.

Similarly, icons can be added to view using the **Icon** component, or the `icon` property within other select components. Icon paths are selected from a pre-configured library within the Designer, which can be further customized with additional icons as desired.

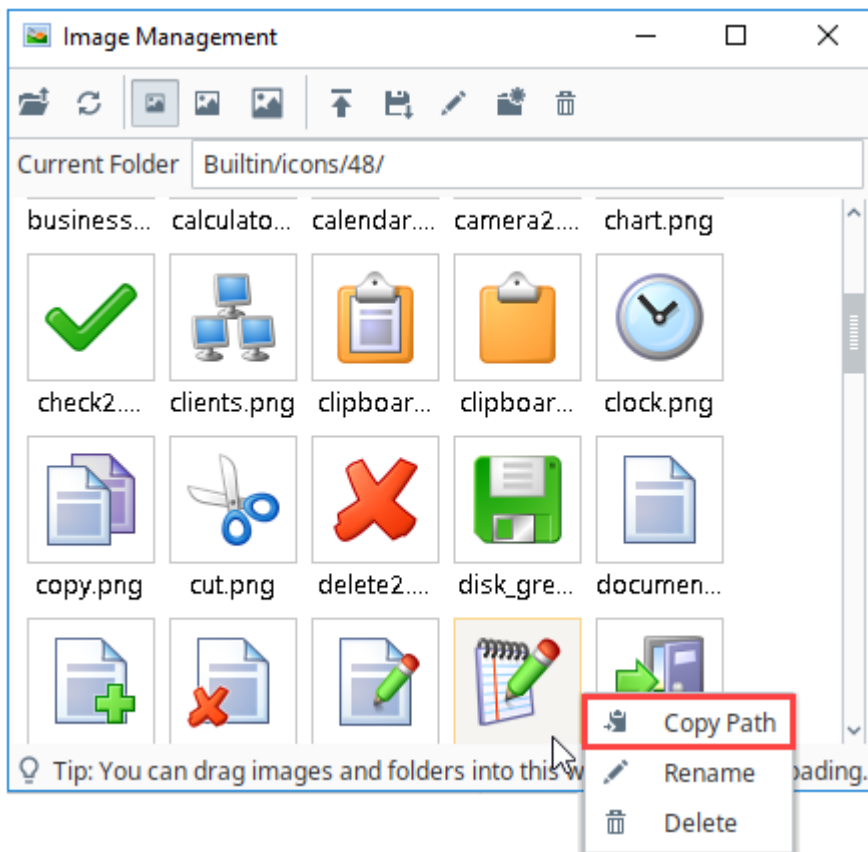
Use the Image Management Tool

The following example walks through bringing an image from the Image Management tool using an Image component. Note that referencing images from the Image Management tool requires adding `"/system/images"` to the beginning of the path.

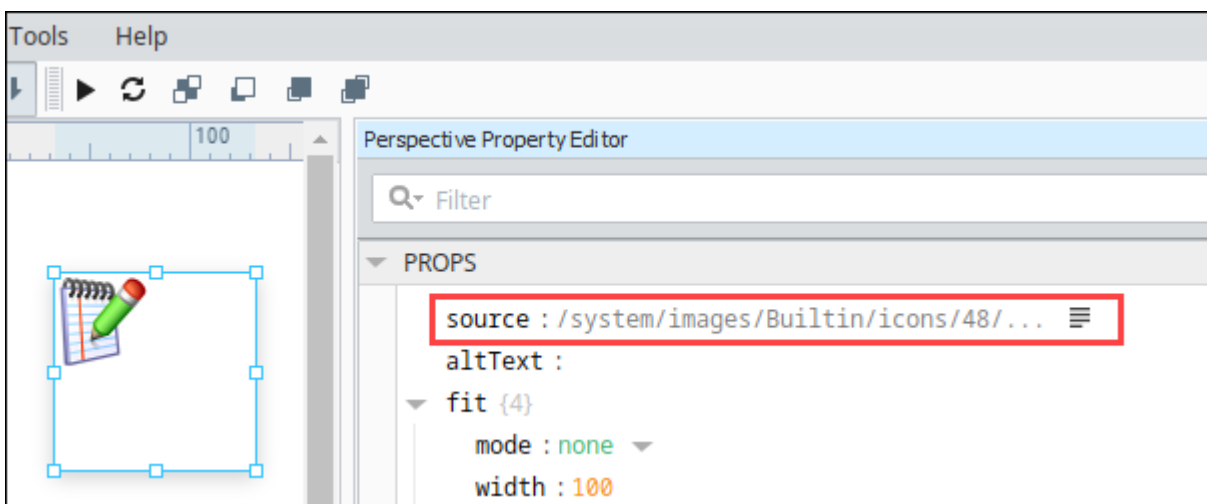
1. Drag an **Image** component onto the view.



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



5. In the Property Editor, paste the image path into the `source` property for the Image component, making sure to add `/system/images` to the beginning of the path.



The image now appears inside the Image component.

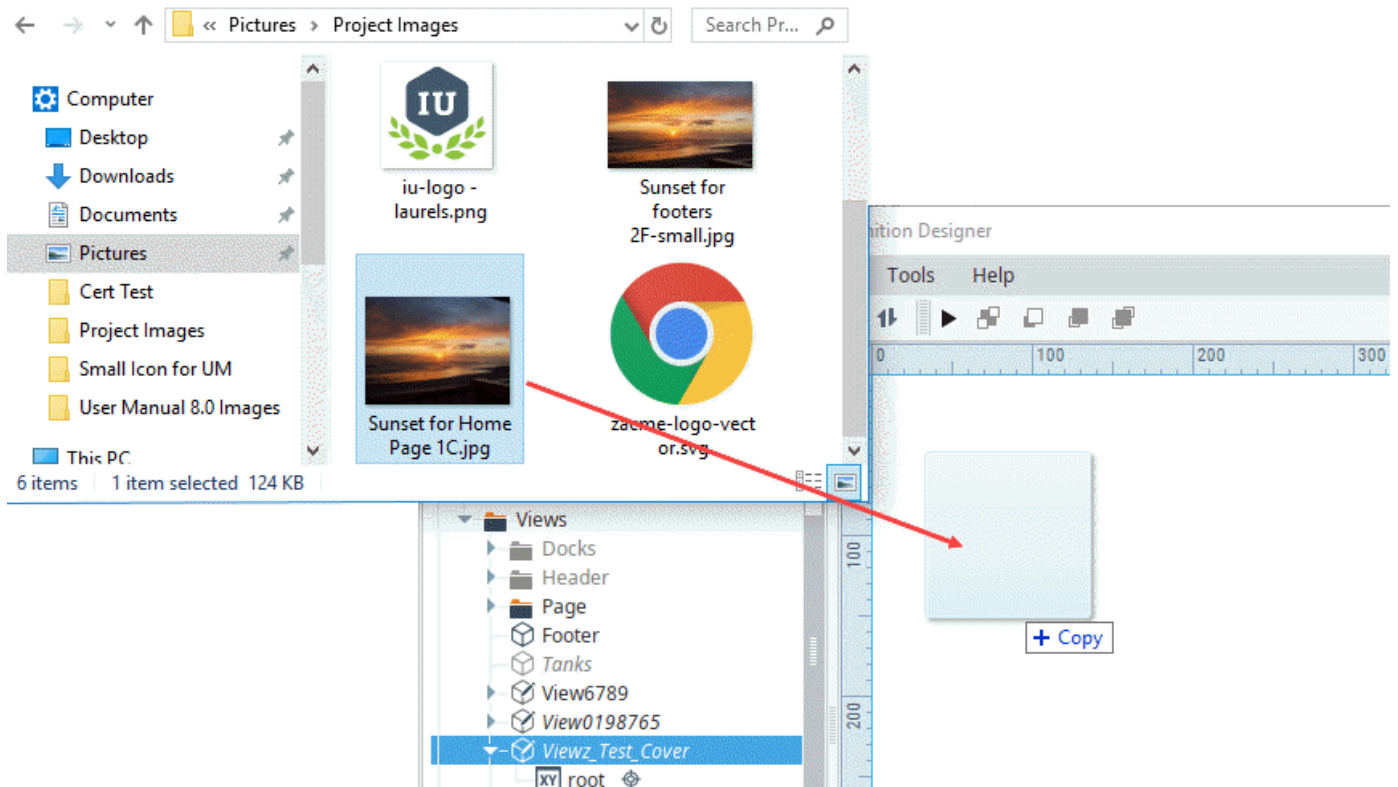
Drag and Drop an Image onto a View

When using the drag and drop method to add an image, you will have the option to upload it into the Image Management tool to be then referenced, or to embed it 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. Images saved in the Image Management tool can be referenced and used repeatedly as described in the [Using the Image Management Tool](#) section above.

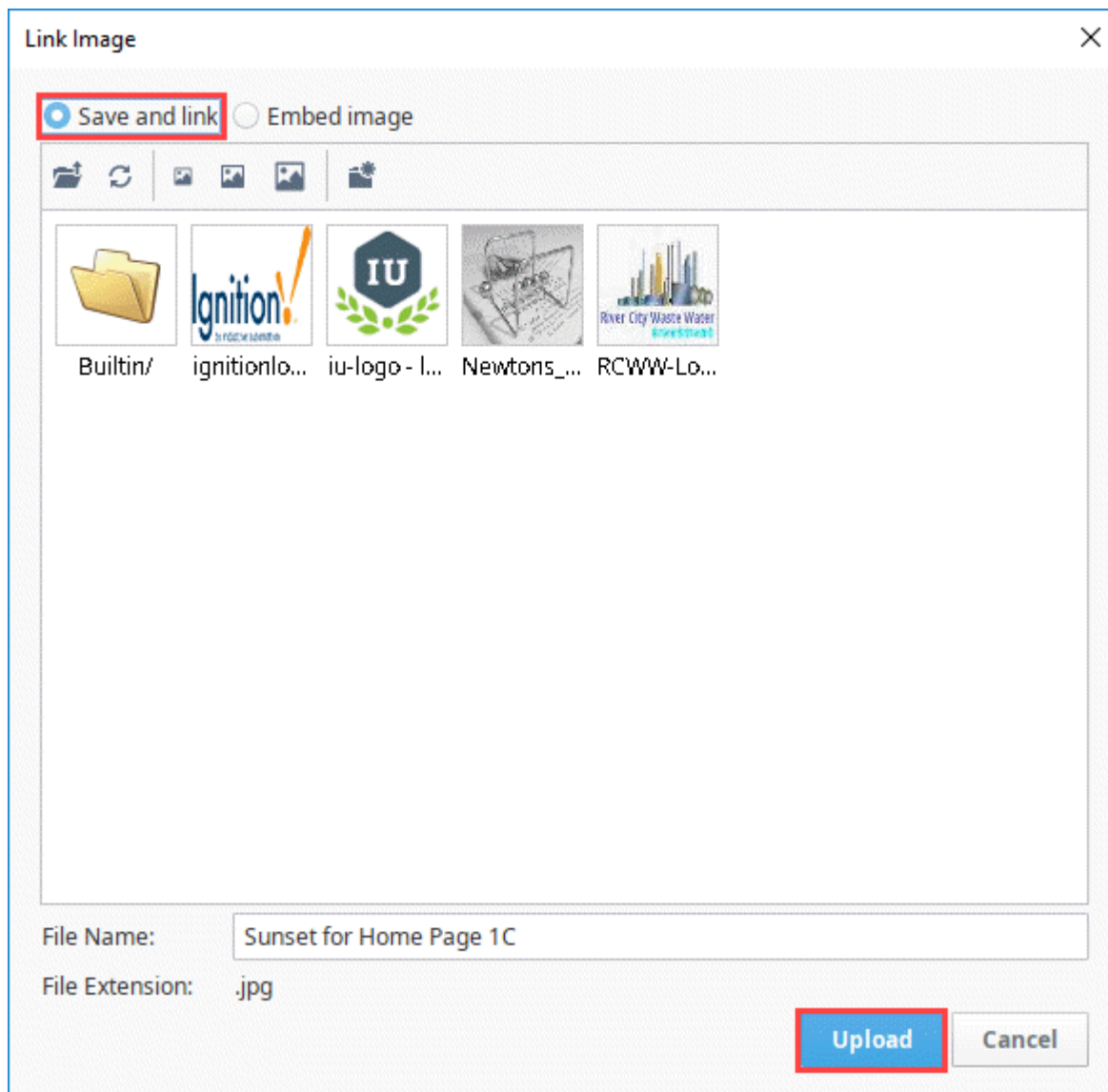
1. Open a view in the Designer.
2. Locate an image on your computer of an acceptable file type and drag it onto the view.

i NOTE

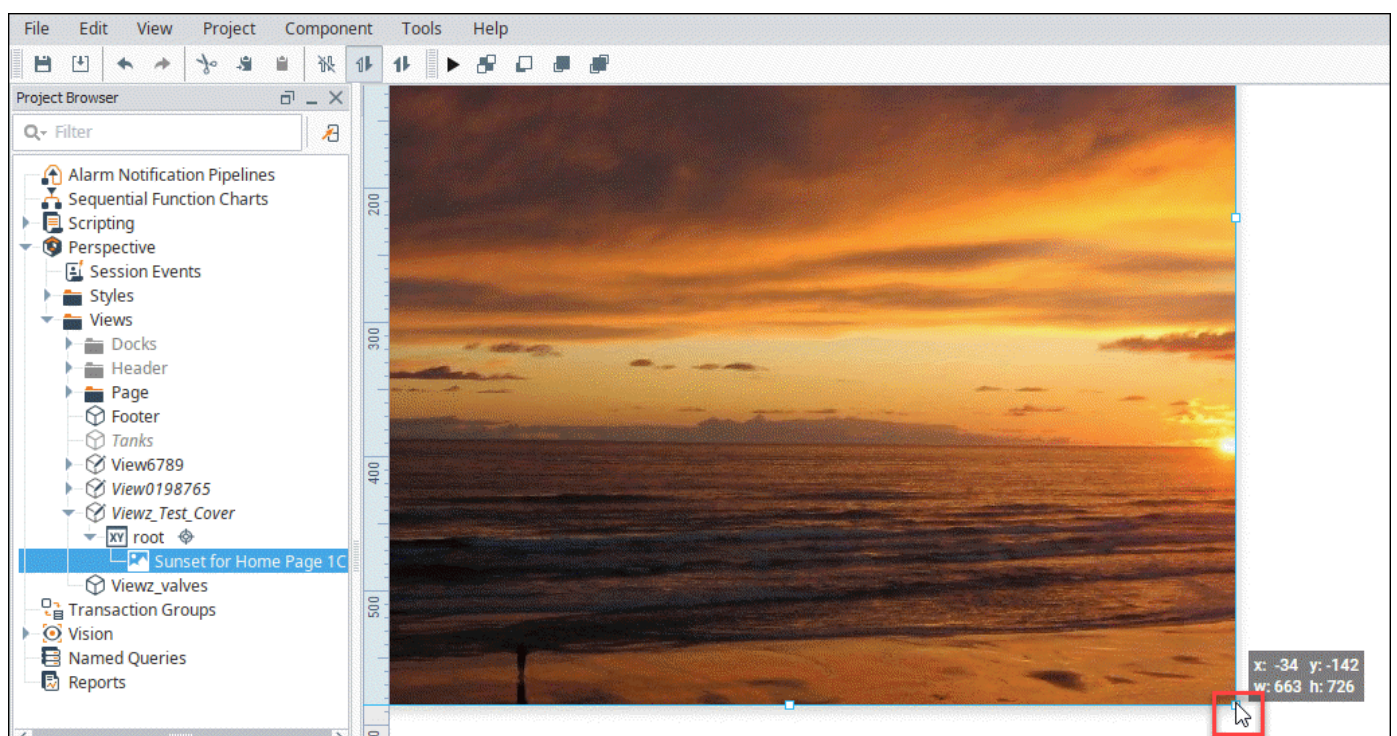
The image in this example has been resized in a graphics editing program to be the size we want.



3. A Link Image popup will appear for you to select whether you want to save the image as a link (in the Image Management tool) or embed the image. For this example, we will save the image as a link, making sure the **Save and link** option is selected and then clicking **Upload**. Refer to the [Embedding Images](#) steps below for selecting **Embed image**.



- The image will now appear in the view in an Image component, but the component shape still needs to be adjusted to fit the image size. Drag one of the corner nodes to match the image component border with the full image.



5. Add any further components and graphics to your view as desired and save your project. For this example, we added a company logo SVG file and navigation buttons.



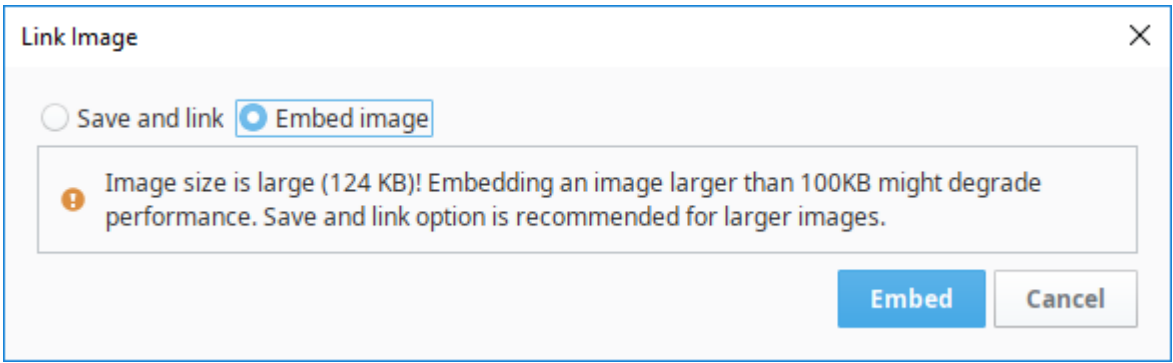
Embedding Images

Embedded images can degrade performance, but 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. Click the **Embed image** radio button.

i **LARGE IMAGE FILE**

If the image is larger than 100KB, you will see a warning message in the confirmation popup recommending the Save and link option instead.



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

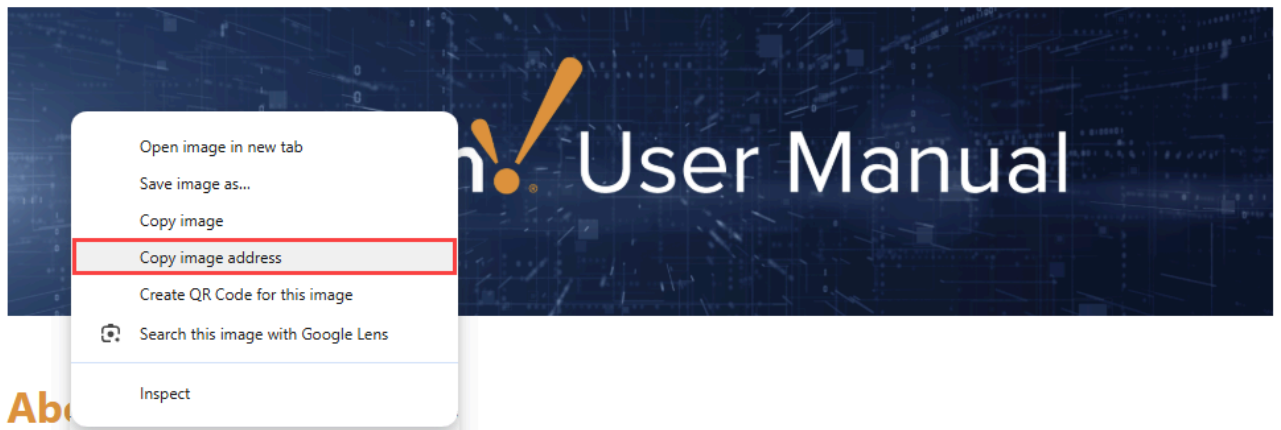
4. Adjust the corner nodes of the Image component border to match the image size.

Use an External Web Address

Images can also be added to the Image component by pasting a copied image address from a web browser into the `source` property.

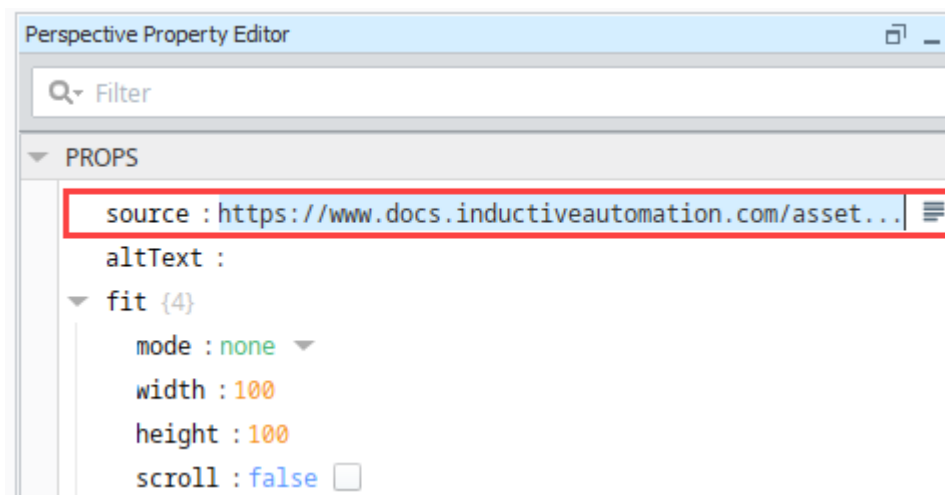
1. Right-click an image and select **Copy image path**.

Welcome

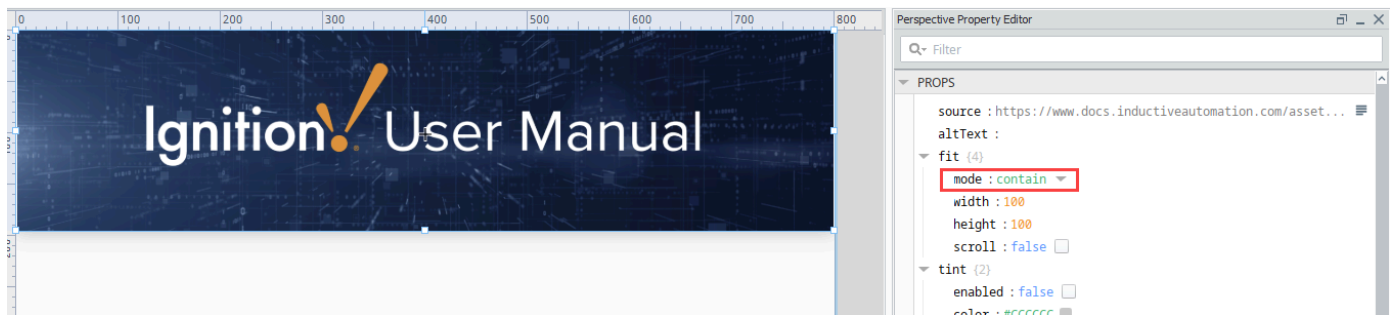


2. In the Designer, drag an **Image** component onto a view.

3. Paste the image address into the `source` field in the Property Editor.



4. Adjust the corner nodes to display the full image or select a desired fit from the `mode` dropdown.



Use Icons



INDUCTIVE
UNIVERSITY

Icons

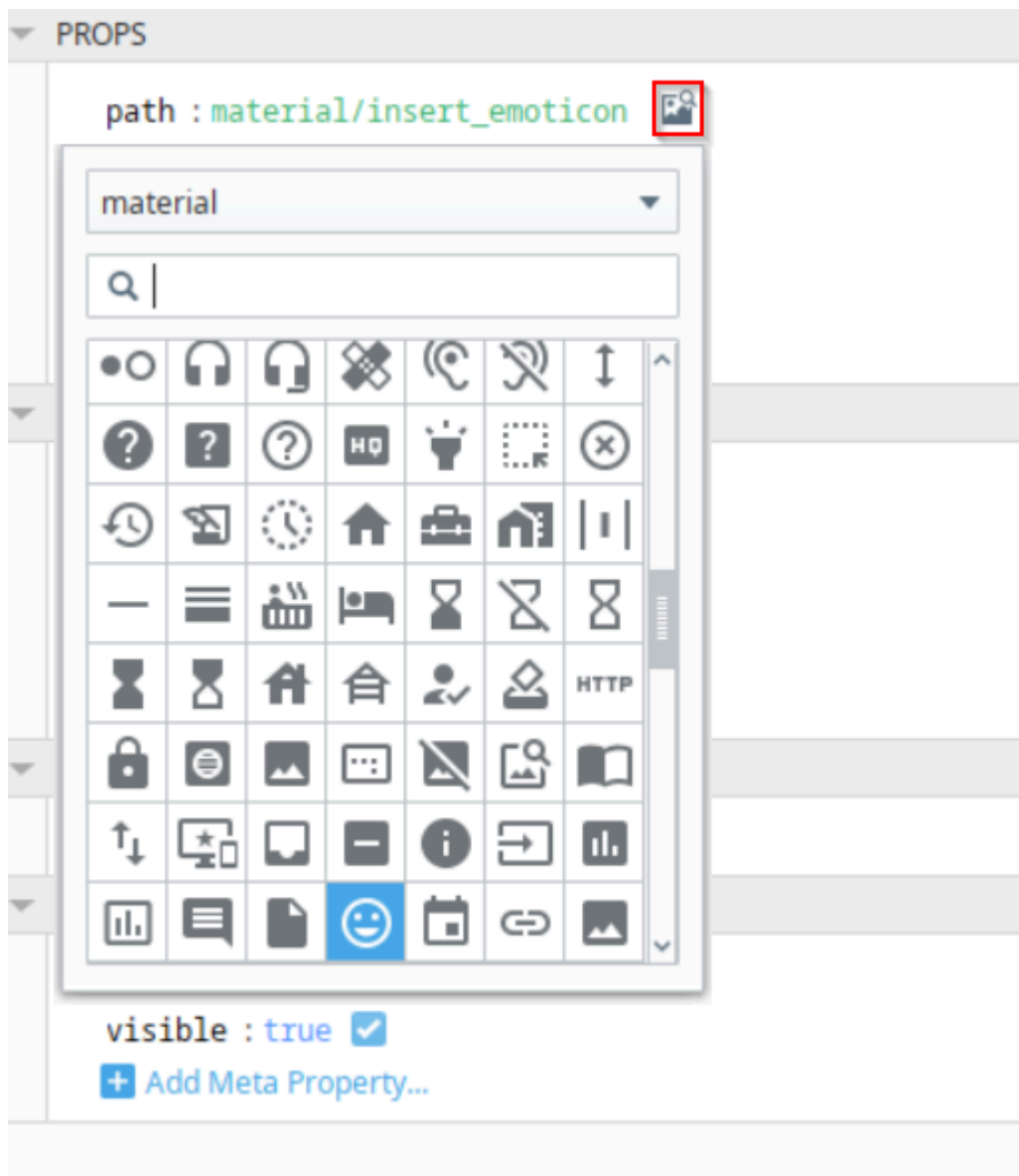
[Watch the video](#)

Icons in Perspective include a collection of specifically curated Scalable Vector Graphic (SVG) images. The bulk of the icons included with Perspective are derived from the **materials** icon library, which is fully detailed on [Google Fonts - Material Symbols & Icons](#), but an **ignition** and **sample-component** folders are also available by default. Icons are referenced using icon paths within Perspective. For example, the path `material/location_on` will reference the `location_on` material design icon. This starting collection of icons can be expanded with custom icon repositories.

Icons can be displayed using the following icons:

- [Icon](#)
- [Horizontal Menu](#)
- [Map](#)
- [Accordion](#)

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



Add a Custom Icon Repository

Although the built-in icon repository is a great source for icons, you can set up a custom library of icons 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 MacOS Linux

```
C:\Program Files\Inductive
Automation\Ignition\data\config\resources\core\com.inductiveautomation.perspective\ic
<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 both 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>
```

A Designer restart is required once an icon file has been added to pull in the new icons for development.

Custom Icon Repository Example

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.

 Proposal  Project Documents  Appendix

1. Begin by creating 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. Save the file in the Ignition directory:

```

C:\Program Files\Inductive
Automation\Ignition\data\config\resources\core\com.inductiveautomation.perspectiv

```

3. You will also need to create both a `config.json` and `resource.json` file within the same directory as `example.svg`. The `config.json` file should have a single entry that contains the same name as your `.svg` file:

```

{
  "svgFileName": "example.svg"
}

```

4. The `resource.json` file should reference both `config.json` and `example.svg`, and contain several other items:

```

{
  "scope": "A",
  "version": 1,
  "restricted": false,
  "overridable": true,
  "files": [
    "config.json",
    "example.svg"
  ],
  "attributes": {
  }
}

```

5. After creating these files, we will need to restart our Designer and Gateway for the changes to be recognized. If you want to avoid restarting your Gateway, you can use the **Scan File System** button found on the top right of the Gateway's **Platform Overview** page to pull in your file changes.
6. Now we can use the icons in this icon repository for our components. For this example, we put a Horizontal Menu component in a Coordinate view.

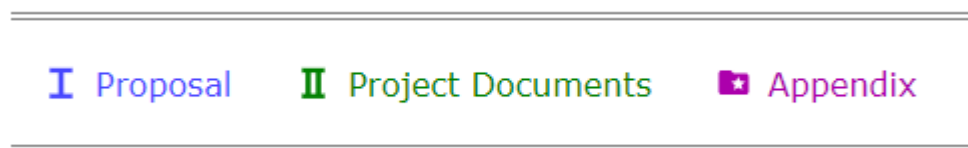
7. We will use two icons from the new example repository and one from the materials repository. We will set the properties as follows for the Horizontal Menu component:

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

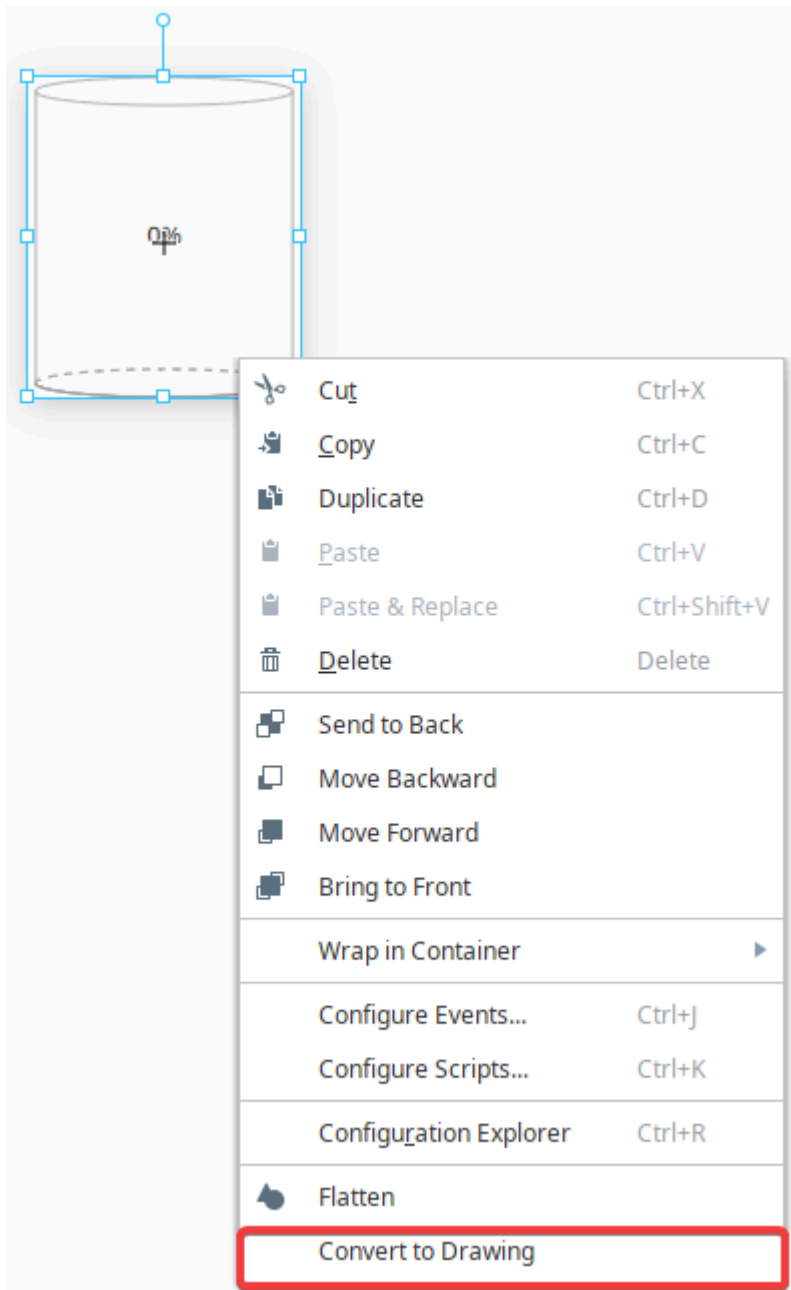
Property	Value
props.style.borderRightStyle	none
props.style.borderTopStyle	double
props.style.borderTopWidth	4
props.style.borderBottomWidth	4

8. Enable the Designer Preview ► mode to see the Horizontal Menu appearance.



Convert SVG-based Components to Drawings

Right-clicking an SVG-based Perspective component will show a Convert to Drawing option. Converting the component to a drawing transforms the component's properties. For example, the Cylindrical Tank component props list changes to include properties like viewBox, preserveAspectRatio, and elements.



Component Example	Properties	Property Editor Image
Cylindrical Tank as a Component	• value • capacity • liquidColor • tankColor • liquidOpacity • liquidWarningColor • tankWarningColor • warningThreshold • strokeWidth • valueDisplay ◦ enabled ◦ style • style	
Cylindrical Tank as a Drawing	• viewBox • preserveAspectRatio • elements ◦ 1 ◦ 2 ◦ 3 ◦ 4 ◦ 5 • style	

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](#).

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

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 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.

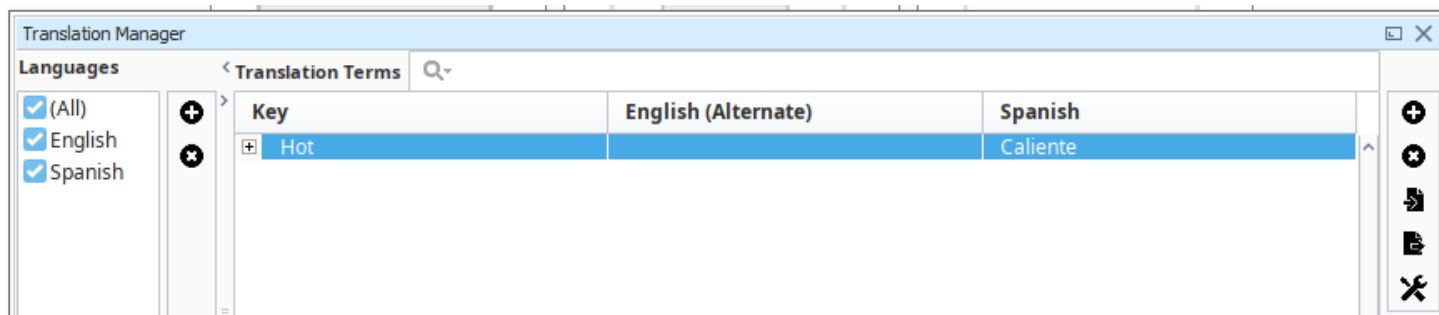
NOTE

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

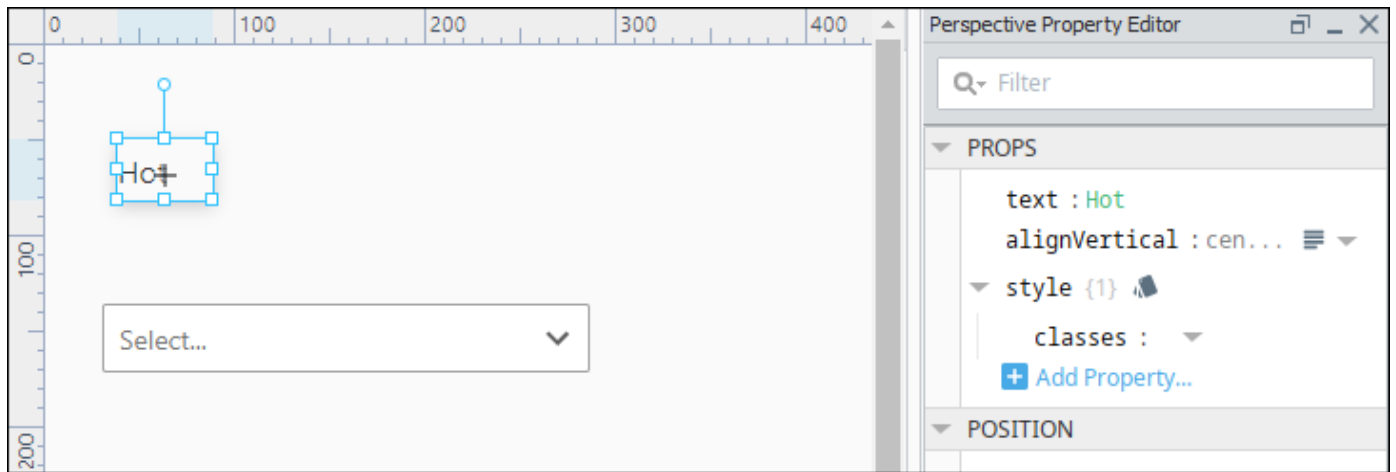
In this example, we'll set up a dropdown list from which the user can choose a language. For this example, we've already set up a Spanish [Translation List](#). We've added a key "Hot" and its Spanish translation, "Caliente".



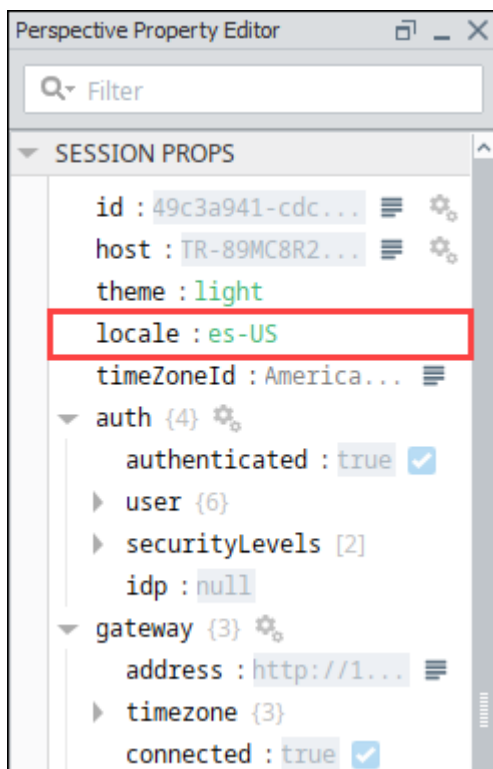
Now we'll set up a view where the user can choose between English and Spanish.

1. Create a new view and put a Label component and a Dropdown component on the view.

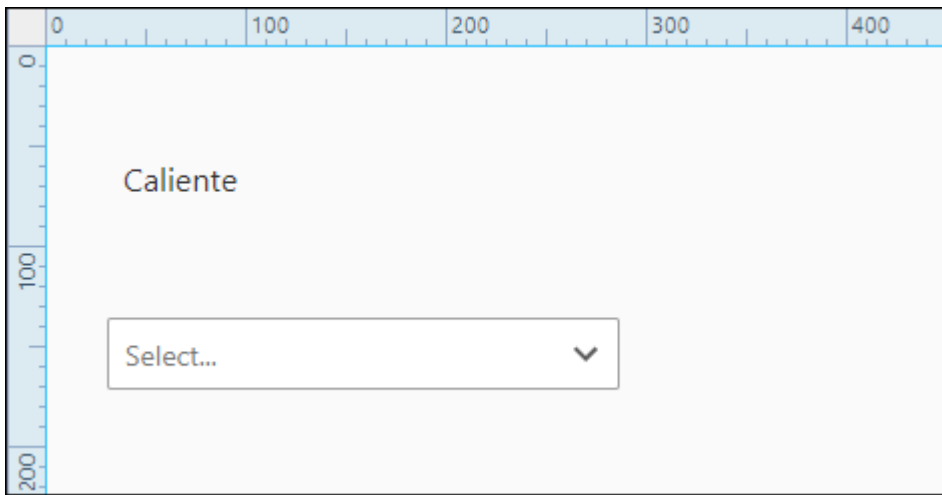
2. Set the text property of the label to "Hot."



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).



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**

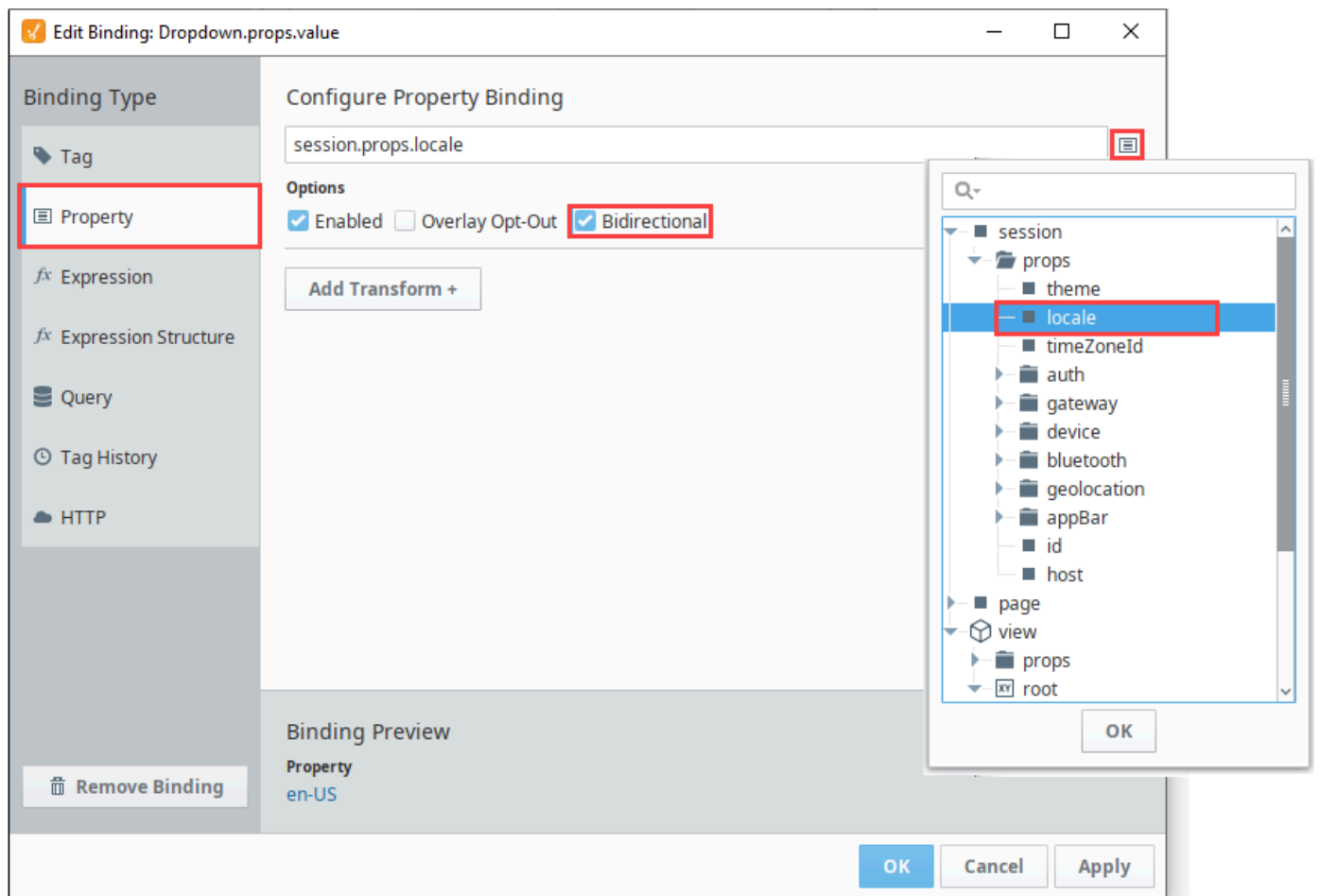


5. Next we'll bi-directionally bind the Dropdown's Value property to the Locale session property. In the Property editor, click the **Binding**  icon for the value property.

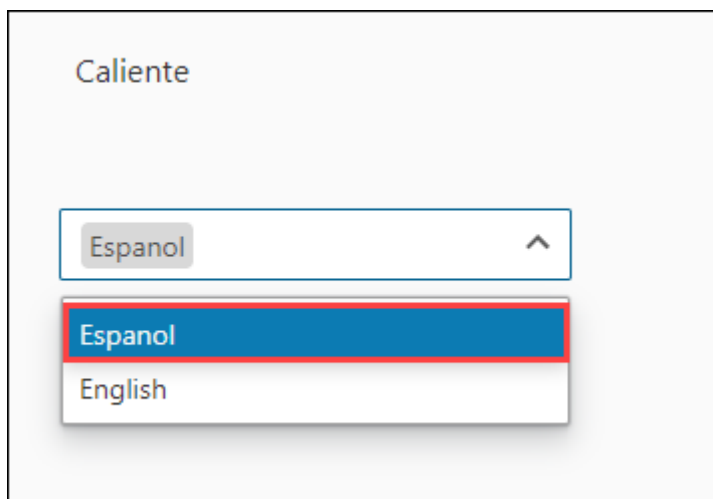
6. On the Edit Binding screen, select the **Property Binding** type.

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.

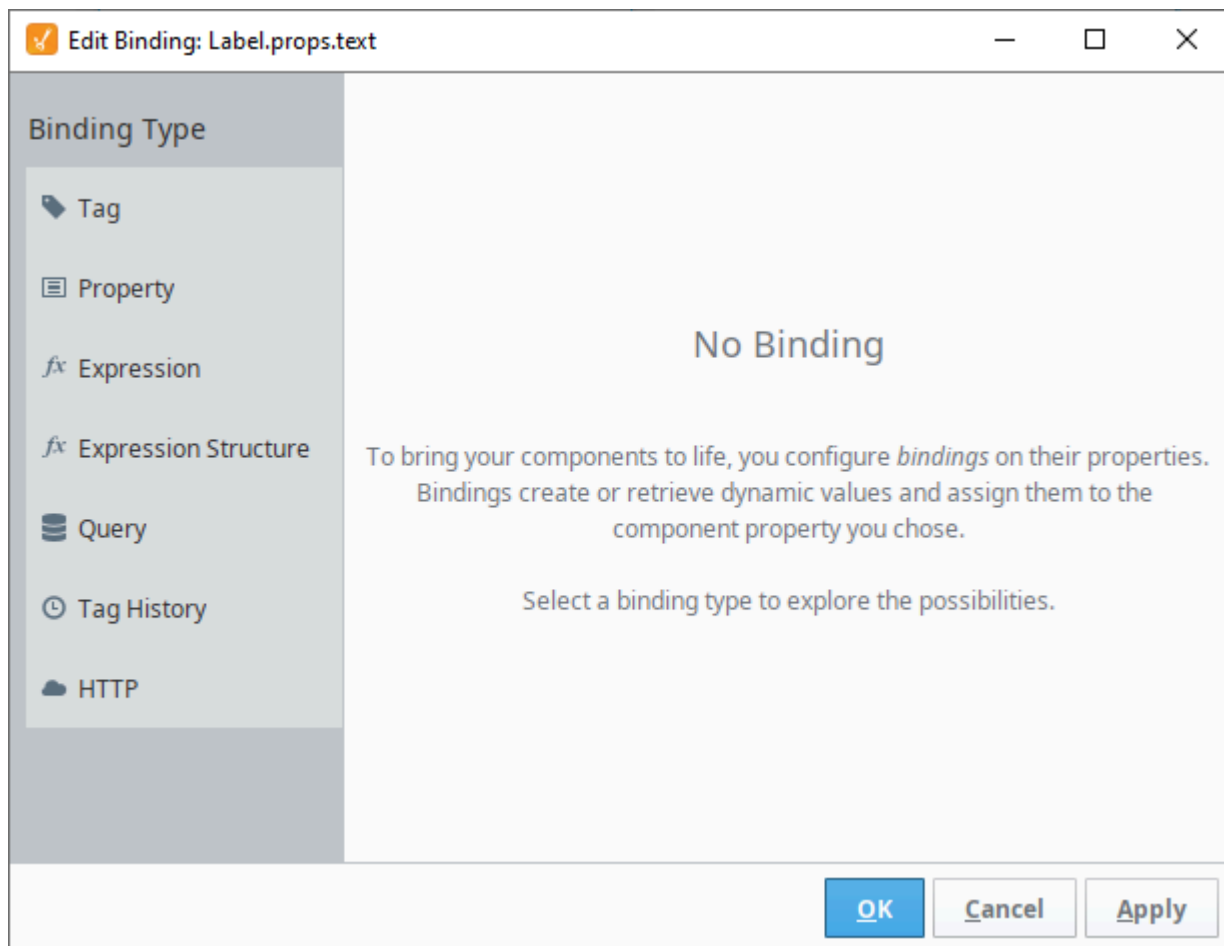


9. Put the Designer into **Preview** ► mode and test the Dropdown component. Since your Dropdown component now drives your Locale session property, you will be able to see your Label value go from its Spanish to English translation as you change the language selection.



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.



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.

Coalesce

When the Bidirectional checkbox is set to true, a Coalesce binding config property becomes available. When Coalesce is selected, changes to the target property's child properties will be combined into a single writeback to the binding source. The Coalesce property is false by default.

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.

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.

Edit Binding: Label.props.text

Binding Type

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

Remove Binding

Configure Tag Binding

☒ Direct ☐ Indirect ☐ Expression **Mode Selection**

Tag Path

Options

- ☒ Enabled ☐ Overlay Opt-Out
- ☐ Publish Initial Uncertain Value
- ☐ Bidirectional Fallback Delay 2.5 **Options**

Add Transform +

☒ **Binding Preview**

Tag

- BrowseValue** [value=[null, B...

OK Cancel Apply

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	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 <code>Uncertain_InitialValue</code>. 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 Delay	When a value is manually written to the bound property, a write request will be sent to the target (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



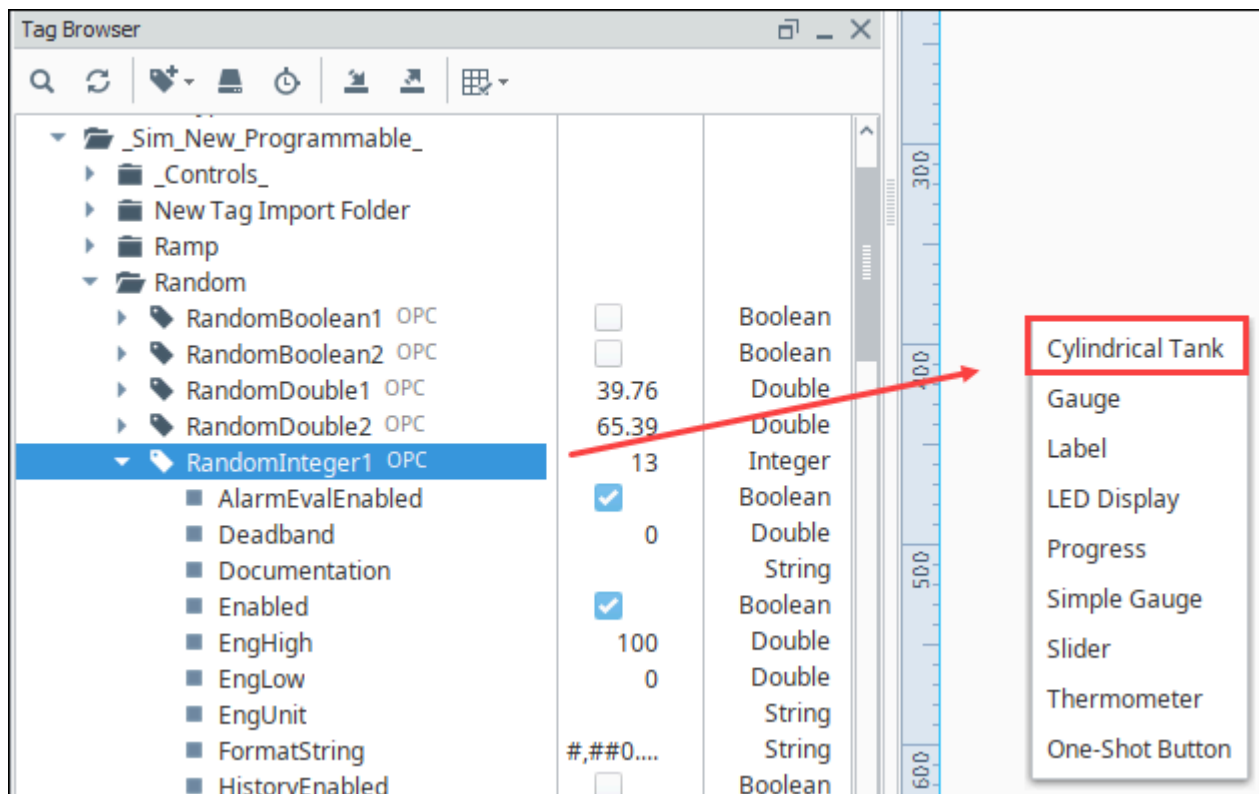
INDUCTIVE
UNIVERSITY

Tag Drop

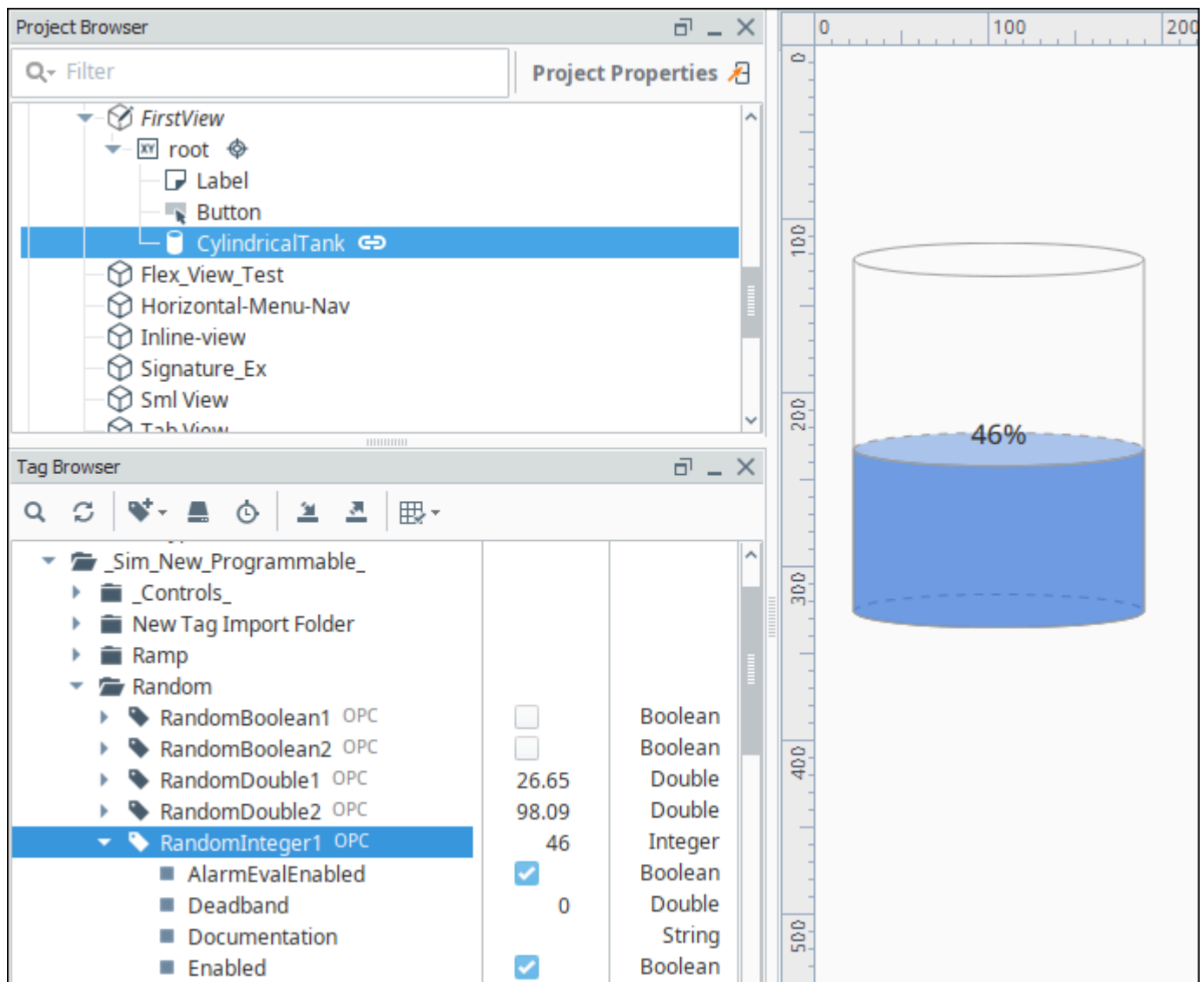
[Watch the video](#)

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 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.



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.



Direct Tag Binding Mode

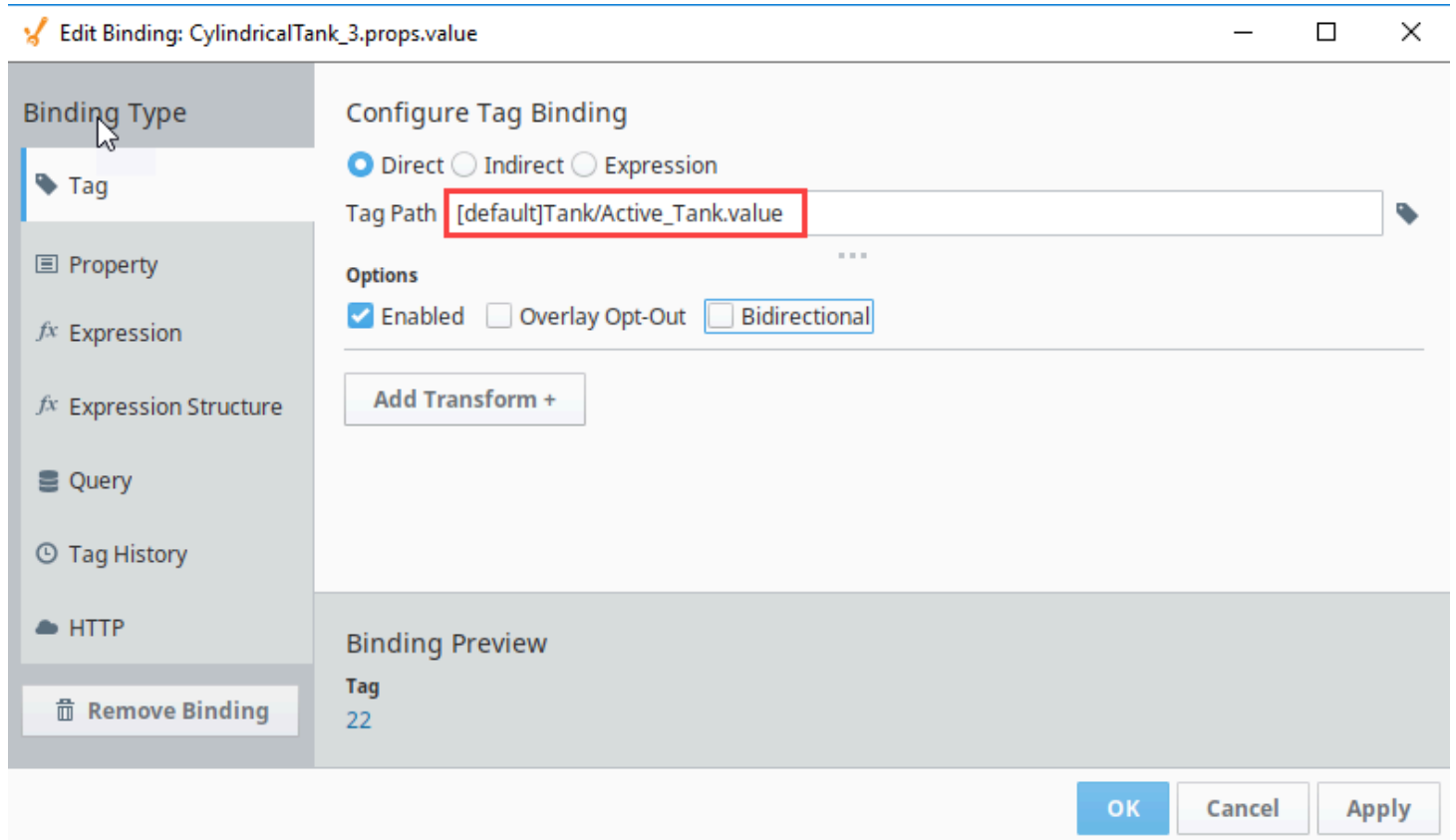
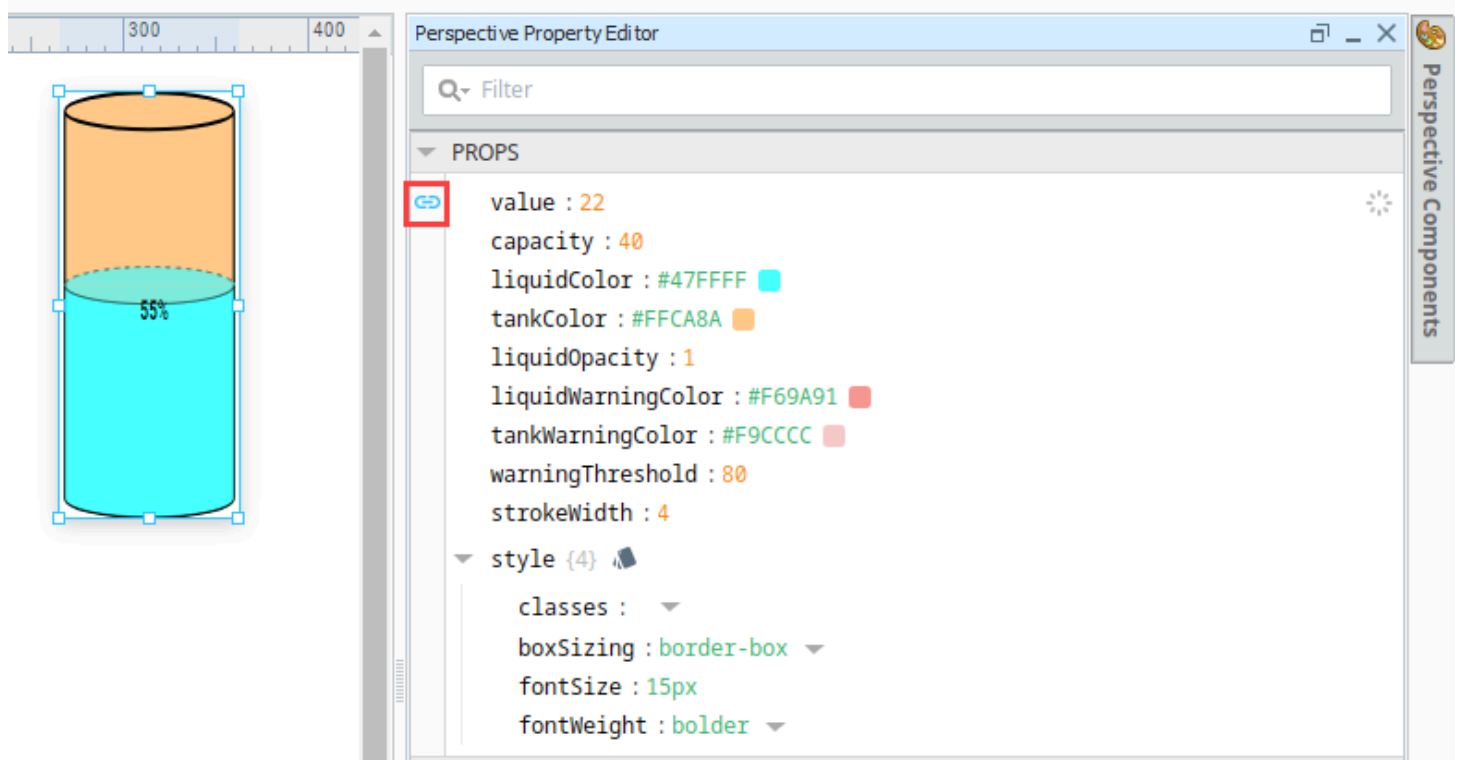


INDUCTIVE
UNIVERSITY

Tag Binding

[Watch the video](#)

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.



Indirect Tag Binding Mode



INDUCTIVE
UNIVERSITY

Indirect Tag Binding

[Watch the video](#)

An Indirect Tag binding is very much like a Direct 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

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

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/Valve{intParameter}/FlowRate

Reference	Property
intParameter	{view.custom.intParameter}

Options

Enabled

Overlay Opt-Out

Bidirectional

Publish Initial Uncertain Value

Fallback Delay

2.5

Add Transform +

Binding Preview

Tag

29

OK

Cancel

Apply

The `{intParameter}` dynamic **Reference** must be pointed to a **Property** or **View Parameter**. Your property can be 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

Binding Type

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

Configure Tag Binding

☐ Direct ☒ Indirect ☐ Expression

Tag Path

Reference	Property
intParameter	{view.custom.intParameter} fx

Options

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

Fallback Delay

Add Transform +

☒ Binding Preview

Tag
29

Remove Binding

OK **Cancel** **Apply**

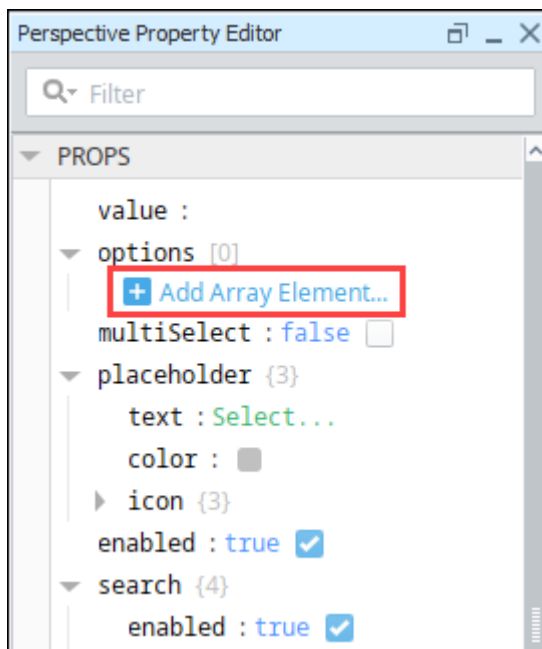
Indirect Tag Binding Example

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 /a mps
Motors/Motor 2 /a mps
Motors/Motor 3 /a mps
Motors/Motor 4 /a mps
```

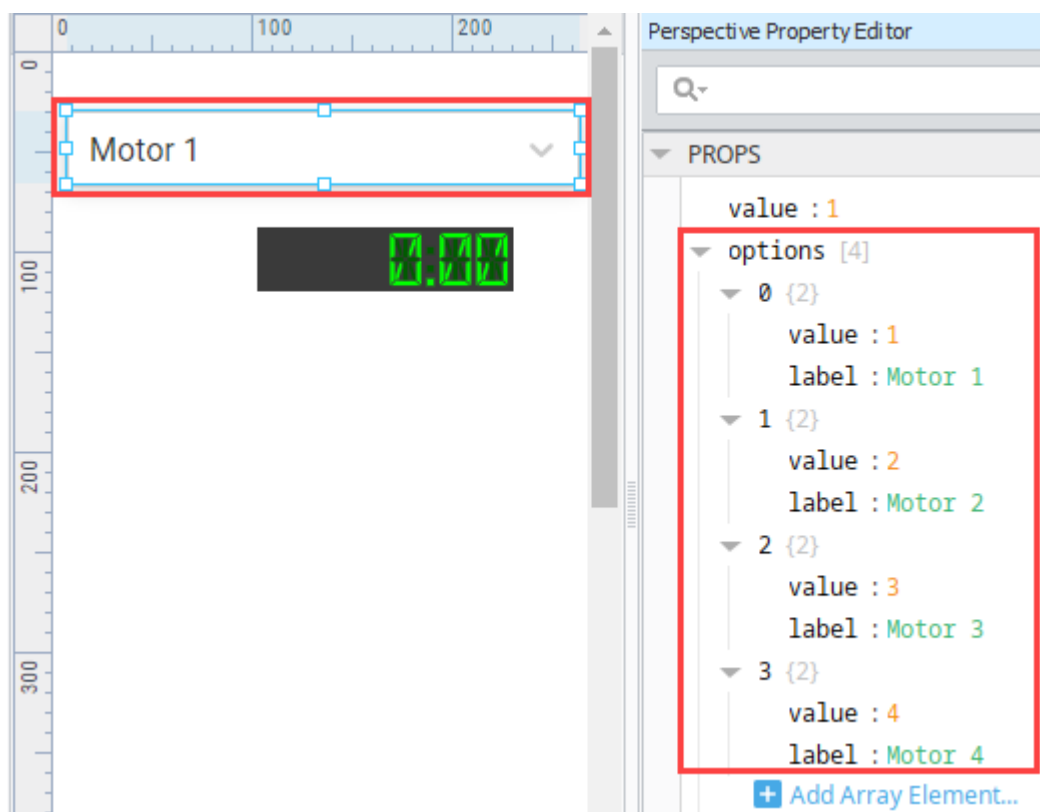
Instead of creating four different displays for these four different tags, we can create a single display and make it indirect. We need two things for this example: A component to display the value in, and a component which allows the user to select which motor they are looking at.

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....**



4. Set the value to 1 and the label to Motor 1.

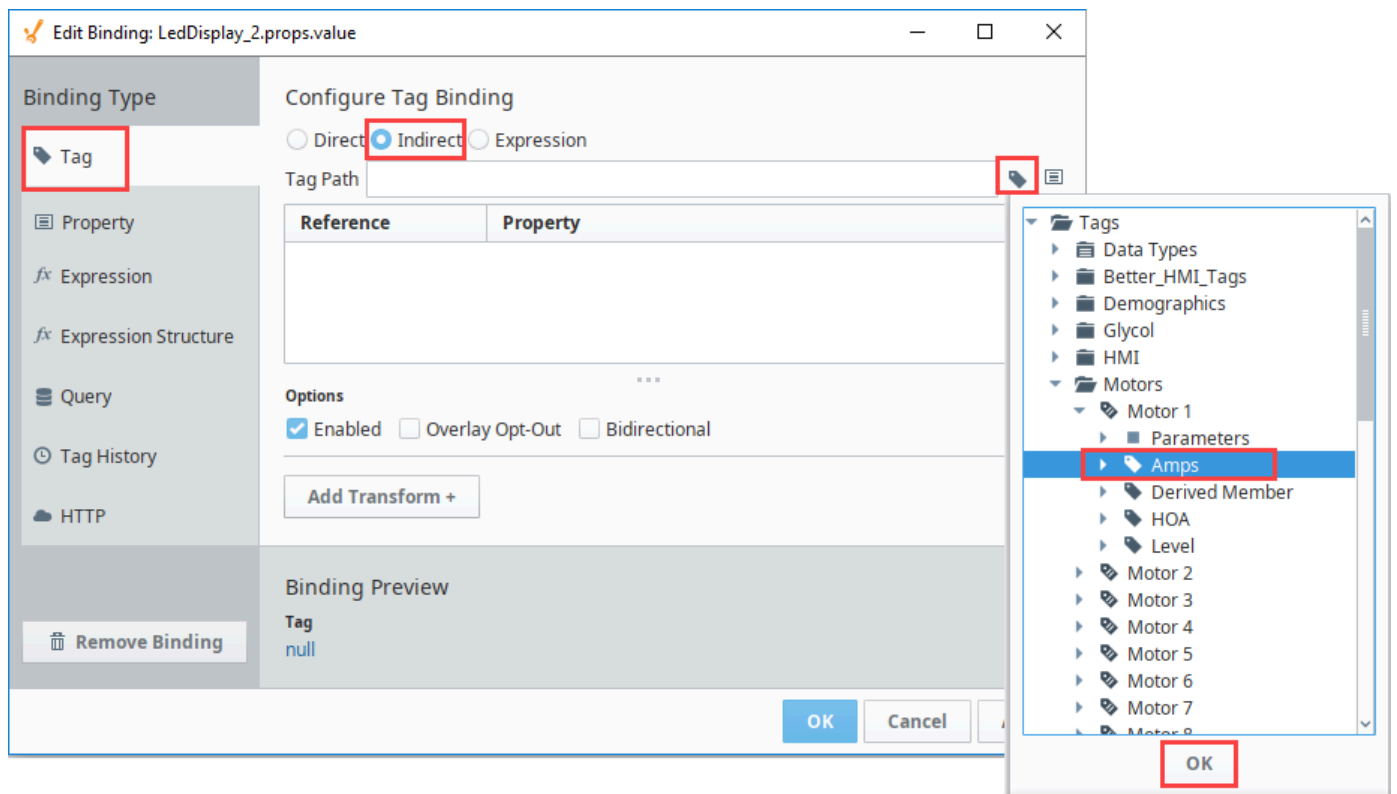
5. Repeat steps 3 and 4 to add array elements for Motors 2, 3, and 4.



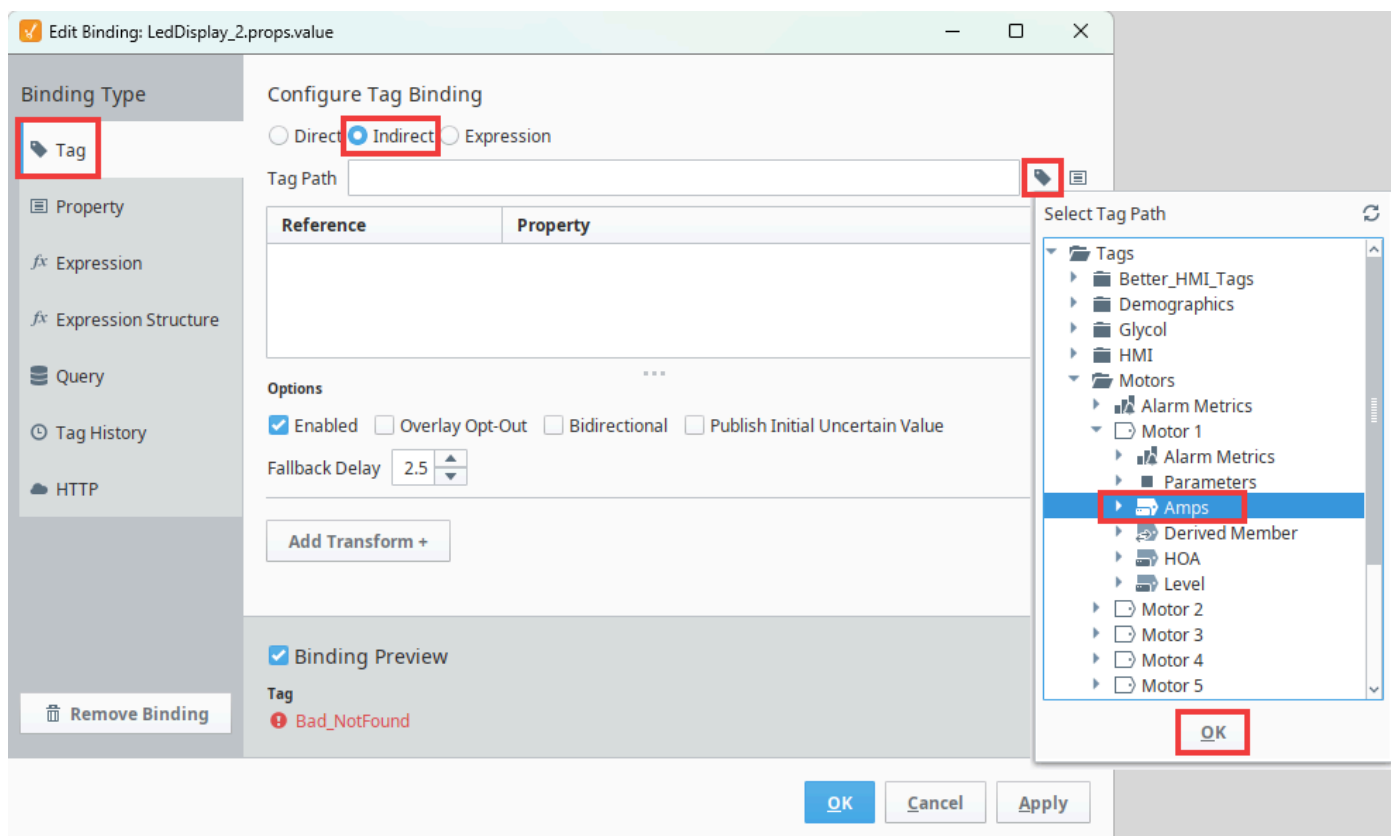
6. Click on the LED Display component. Select the **Binding**  icon in the value property.

7. Click on the Tag binding type then click the **Indirect** button.

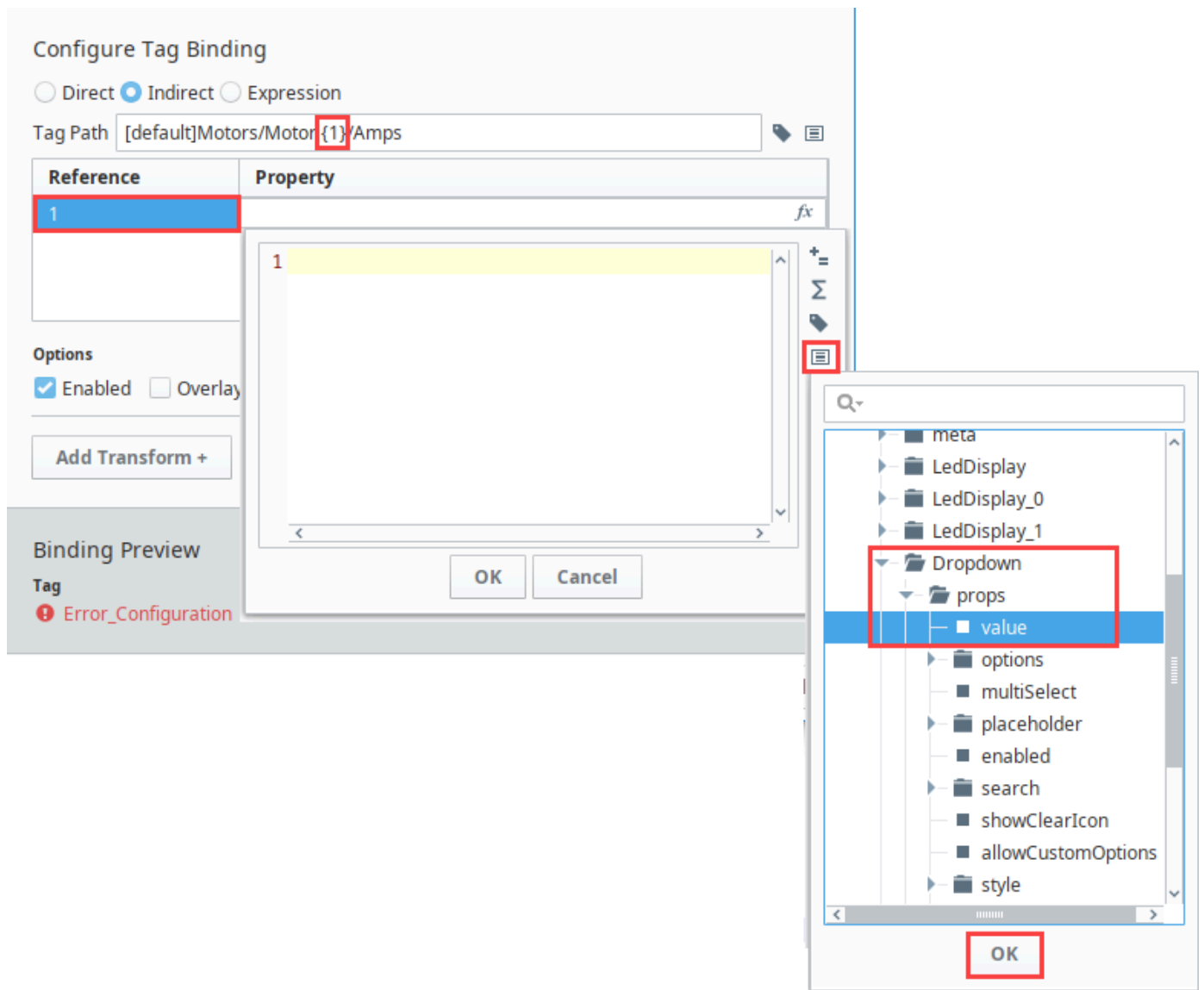
8. Click the **Tag**  icon and scroll down to the Motors/Motor 1/Amps tag. Click **OK**.



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.

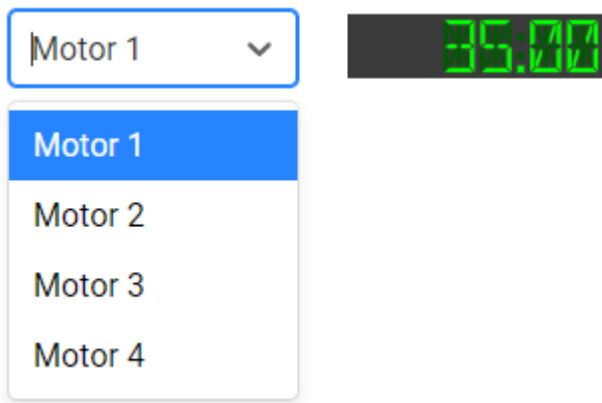


10. In the References list, select the row. Click the **Properties**  icon.
11. Scroll to the Dropdown component and select the **value** property of the Dropdown. Click **OK**.



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



INDUCTIVE
UNIVERSITY

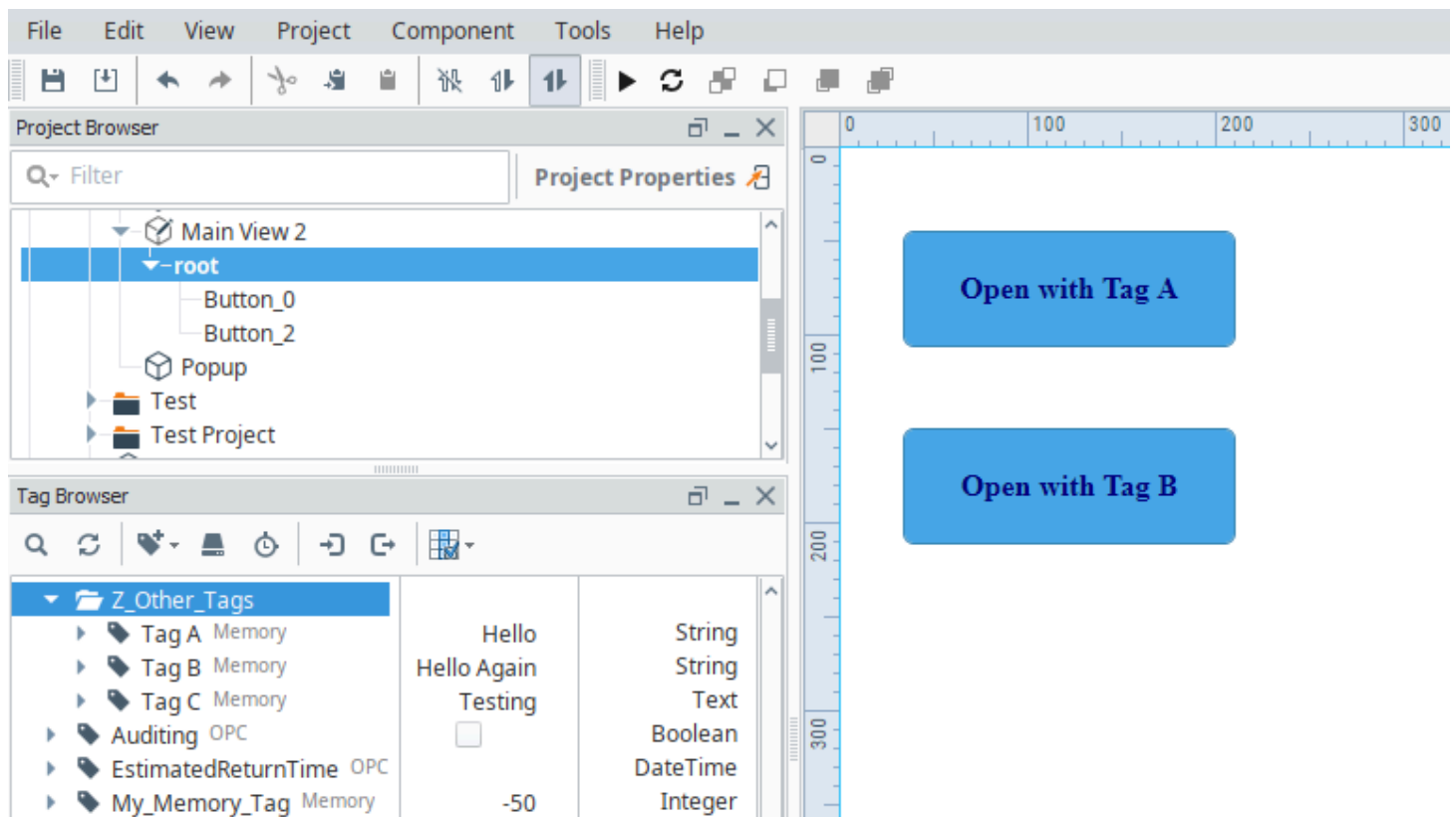
Tag Binding - Expression

[Watch the video](#)

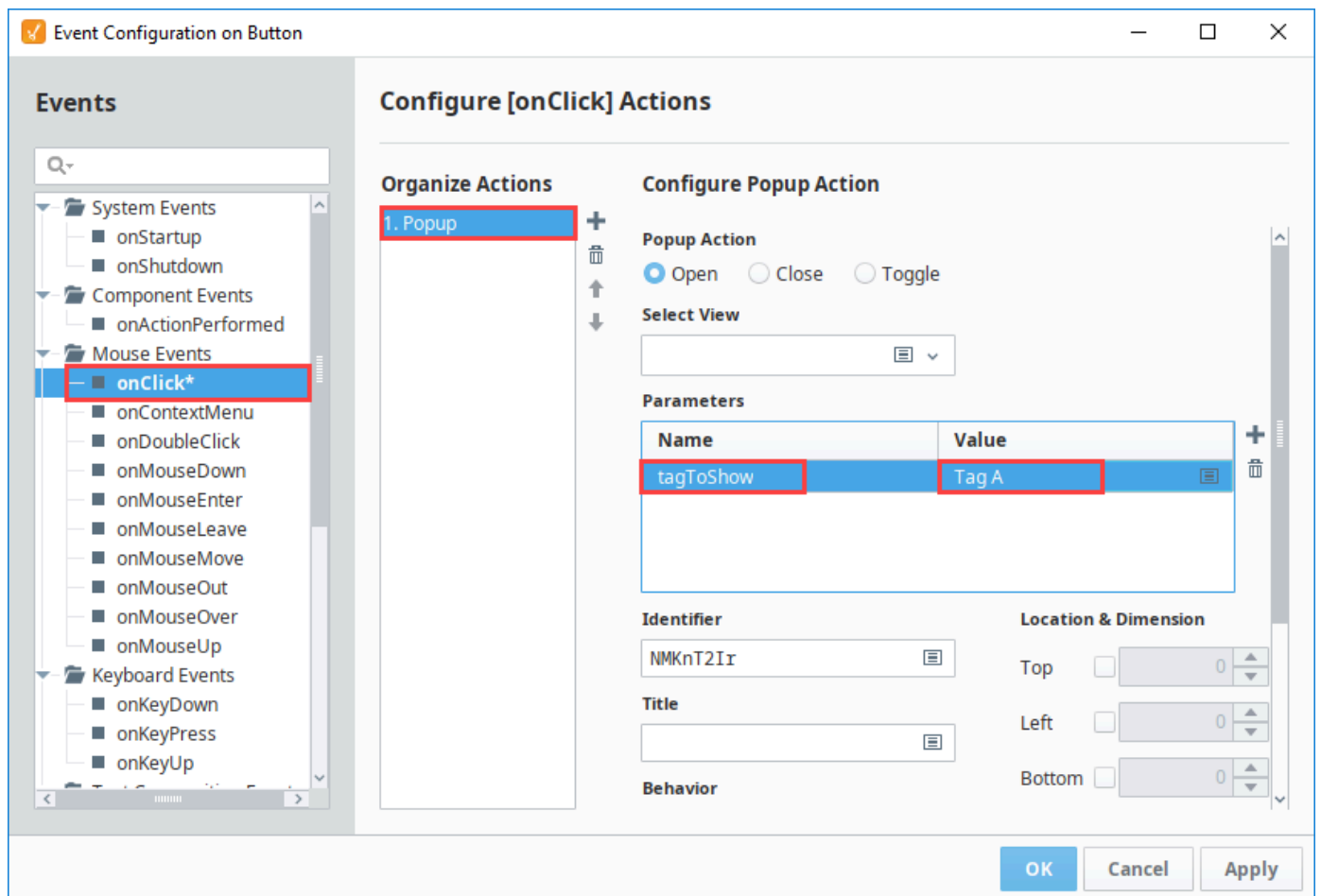
The tag Expression binding uses the Expression language to specify an entire tag path. This mode allows the bound property (tag) to be bidirectional. The tag path in the Expression is expected to be a string. Note that is different and not to be confused with an [Expression Binding](#).

Tag Expression Binding - Example

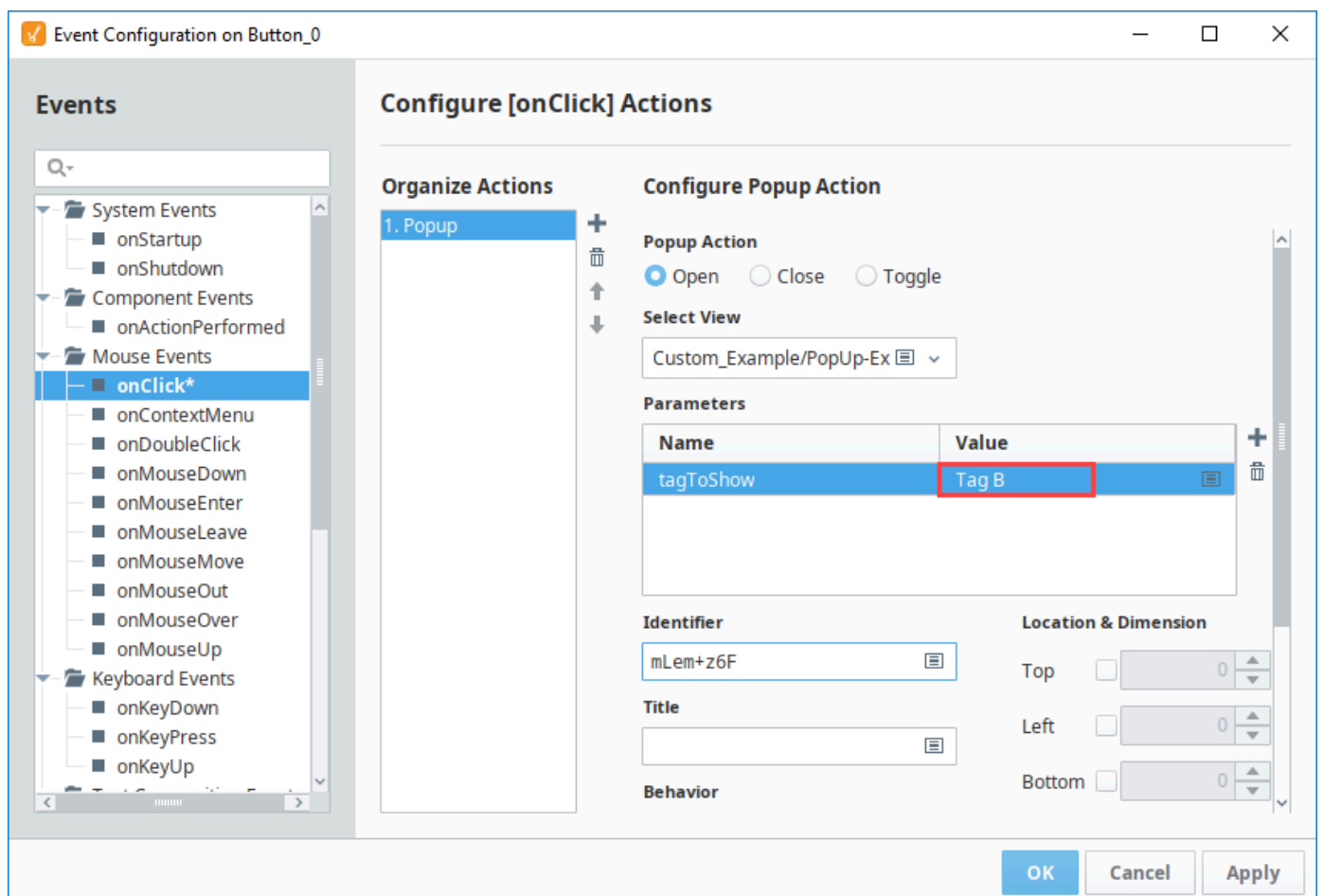
For this example, we start with two tags: Tag A and Tag B and two buttons on a view.



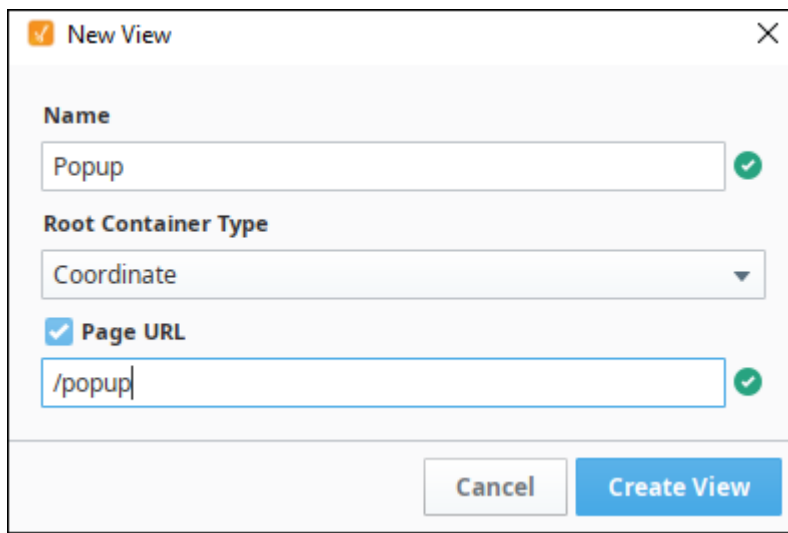
1. Right-click on the first button and select **Configure Events**.
2. Select **onClick**.
3. Under Organize Actions, click the **Add +** icon and select **Popup**.
4. Under Parameters, click the **Add +** icon to add a new parameter.
5. Enter "tagToShow" in the Name field and "Tag A" in the Value field.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



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



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



New View

Name
 ✓

Root Container Type

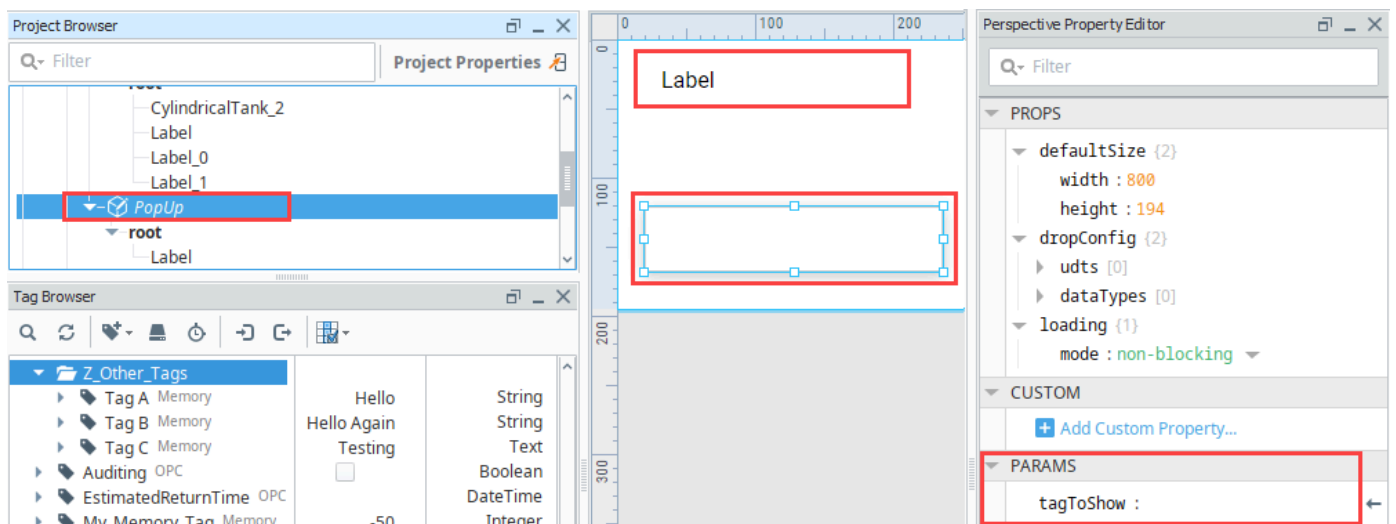
☒ **Page URL**
 ✓

Buttons: Cancel, Create View

8. Drag a Label component and a Text Field component onto the Popup view.

9. In the Popup view, click on **Add View ParameterA**.

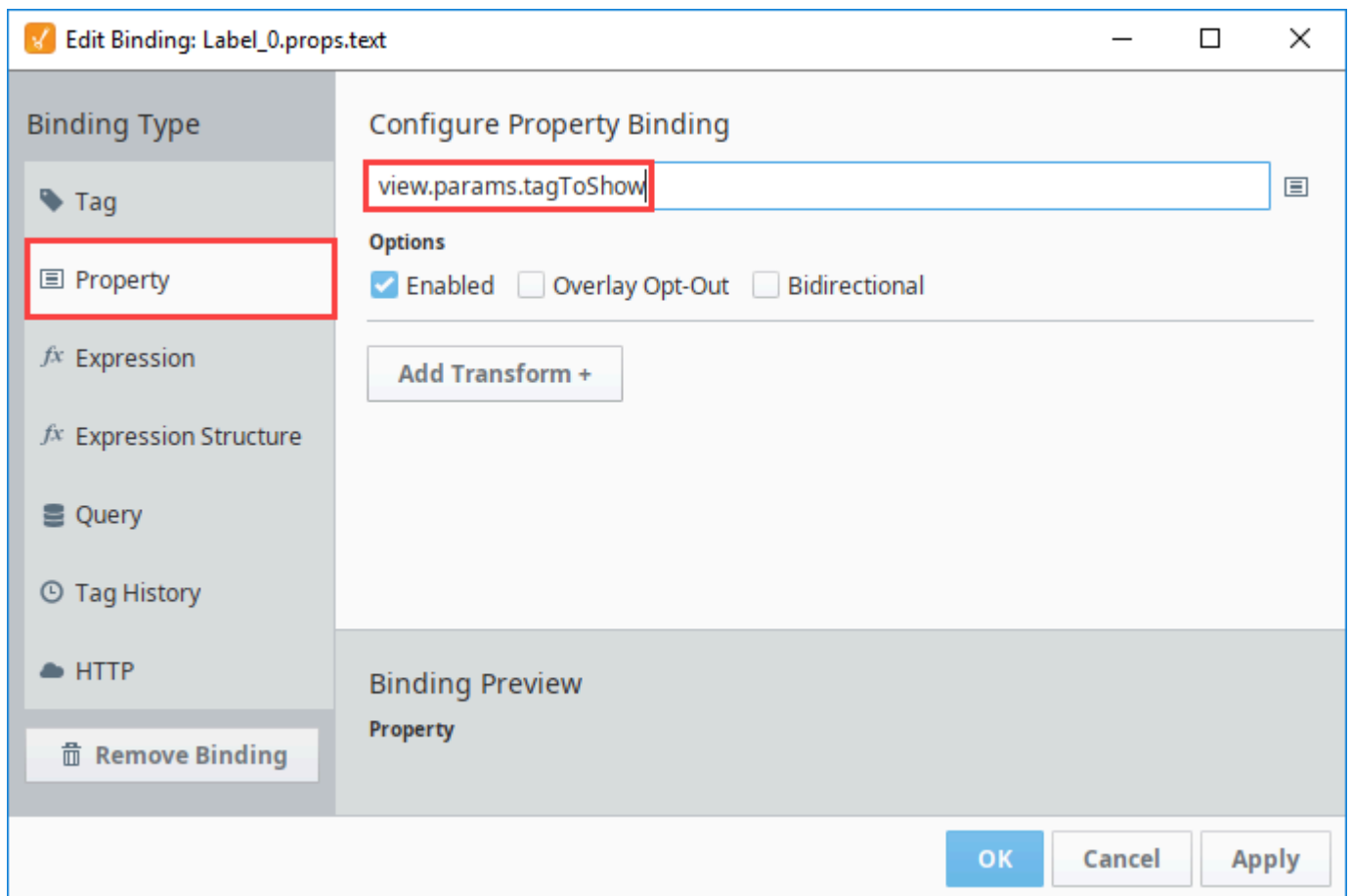
10. Add a new parameter, **tagToShow**.



11. Select the Label component. Click on the **Binding**  icon next to the text property.

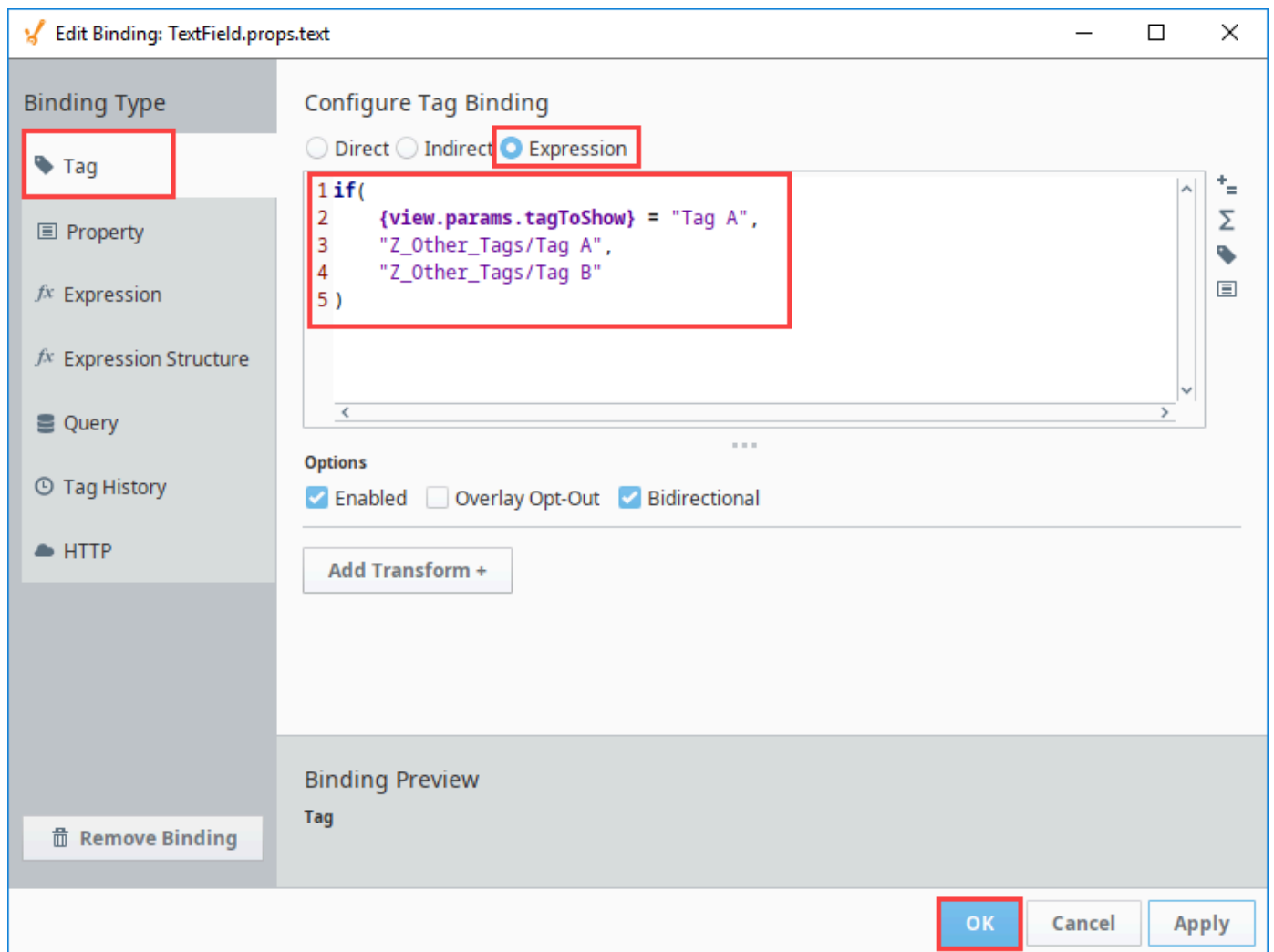
12. On the Edit Binding screen, select the Property binding type. Enter **view.params.tagToShow** in the Configure Property Binding field.

13. Click **OK**.



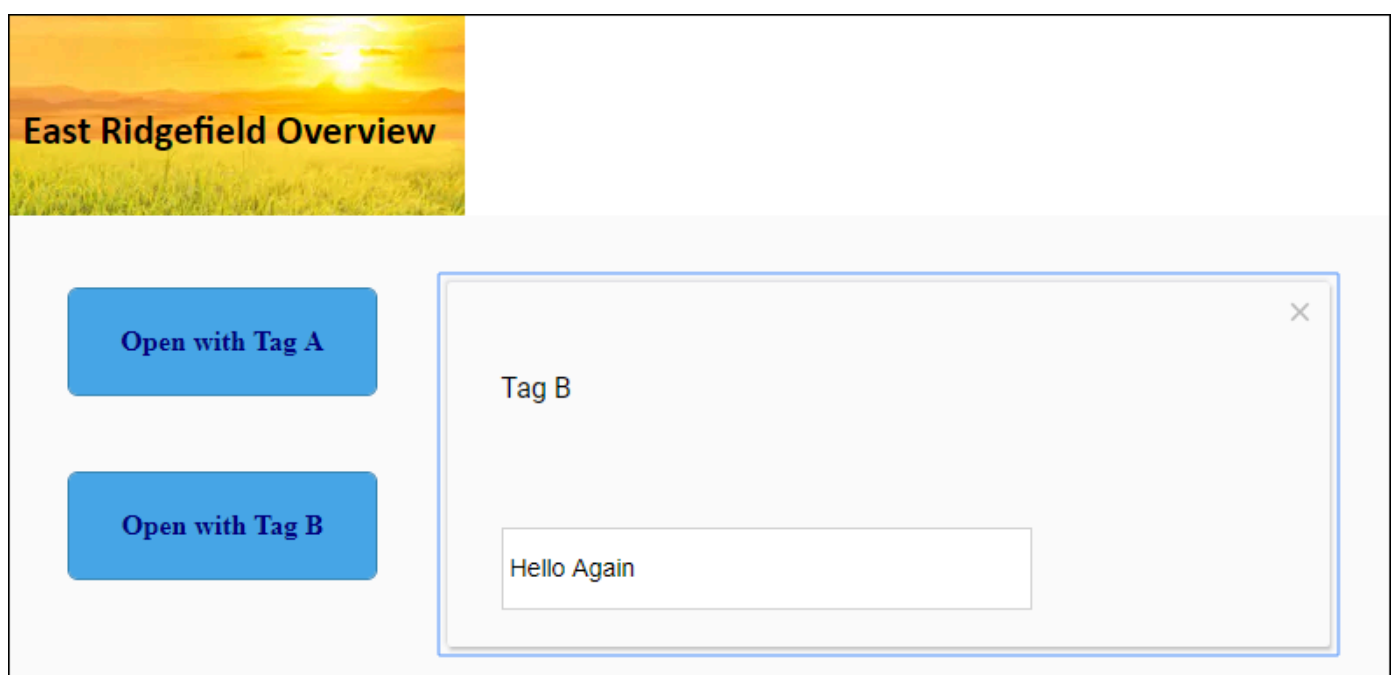
14. Next set a binding on the Text Field component. Select the component then click on the **Binding**  icon next to the text property.
15. On the Edit Binding screen, select the Tag binding type and select the **Expression** button.
16. Enter the following expression. This expression in the example 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")
```



17. Click **OK** to save.

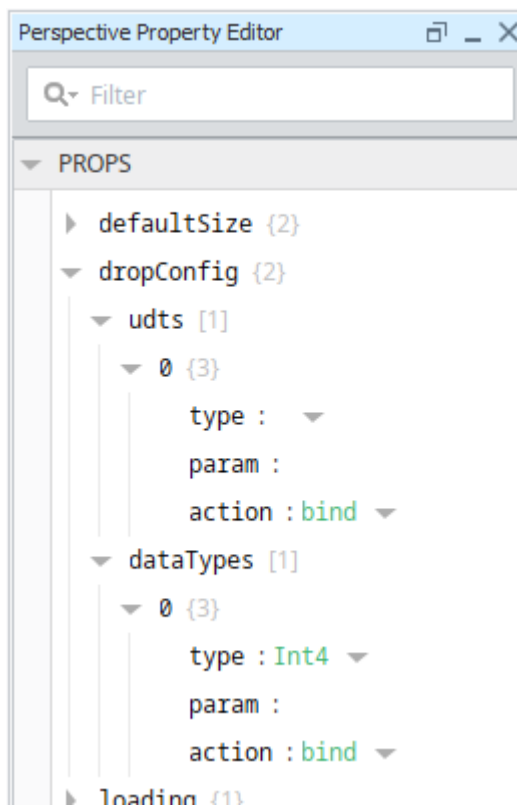
18. Put the Designer into **Preview mode** ►. When you click on the Open with Tag B button, the popup appears to show the Tag B label.



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.



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

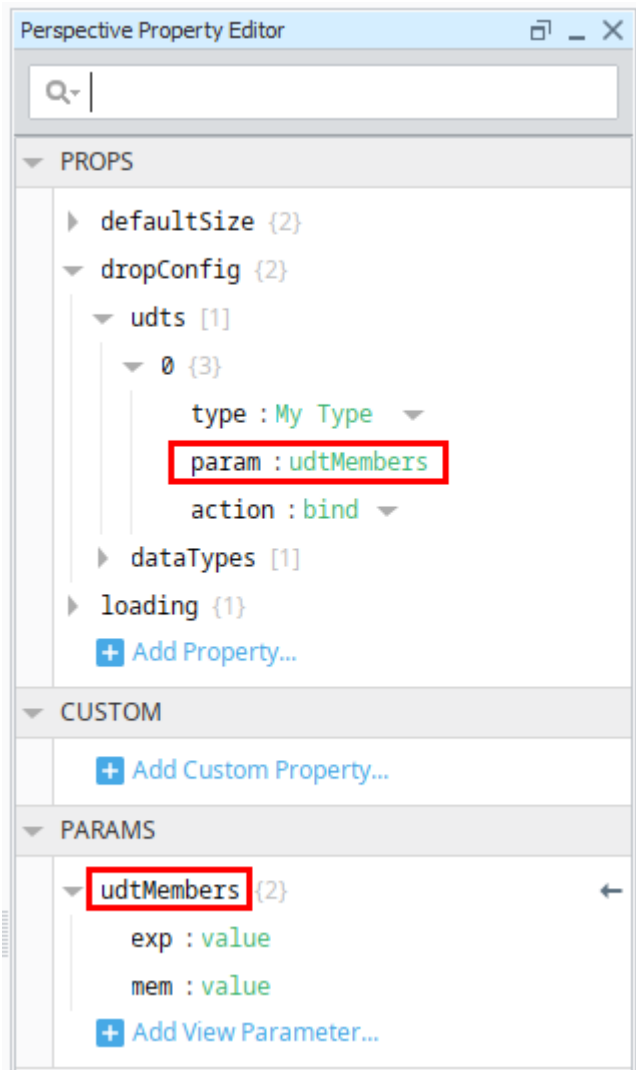
The **param** and **action** Properties

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.



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



INDUCTIVE
UNIVERSITY

Tag Drop

[Watch the video](#)

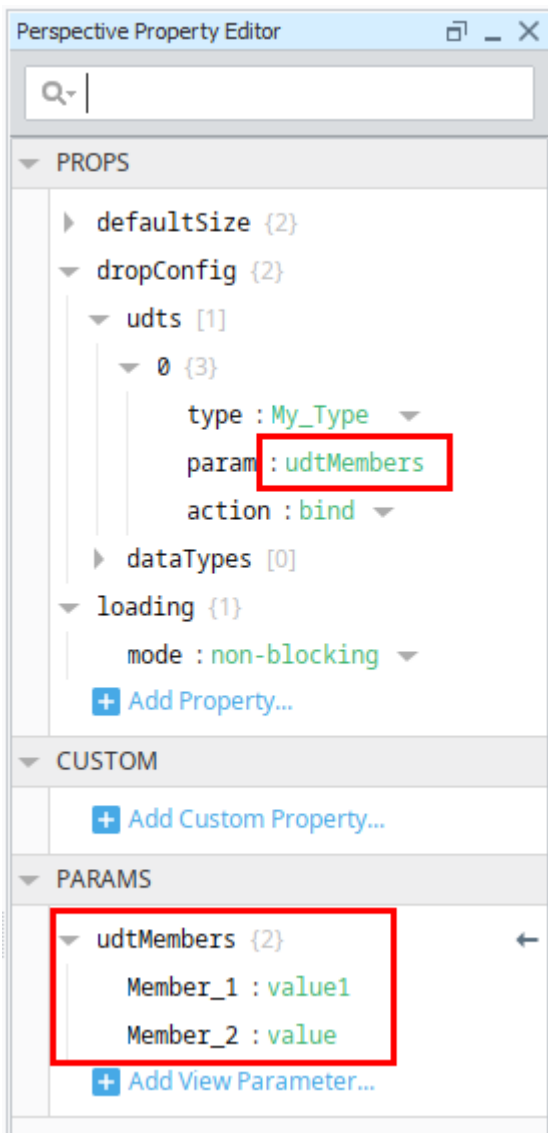
The **udt** type drop configurations can associate an entire UDT instance, along with its members, to a view. Using a UDT drop configuration is exciting because if the associated view parameter is set to a JSON object, then the configurations will also expose values on the members of the UDT to elements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

The screenshot shows a 'Tag Browser' window with a table of tags. A red rectangle highlights the 'My_Type' UDT and its members.

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.

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.



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"
              }
            ]
          }
        ]
      }
    ]
  }
]

```

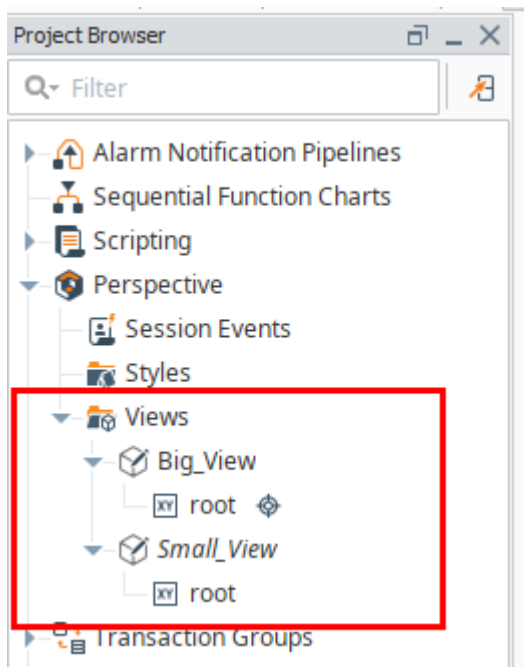
```
    },  
    {  
      "valueSource": "expr",  
      "expression": "\"Hello!\"",  
      "dataType": "String",  
      "name": "Member_1",  
      "tagType": "AtomicTag"  
    }  
  ]  
}  
  
]  
}  
]  
}  
]  
}
```

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

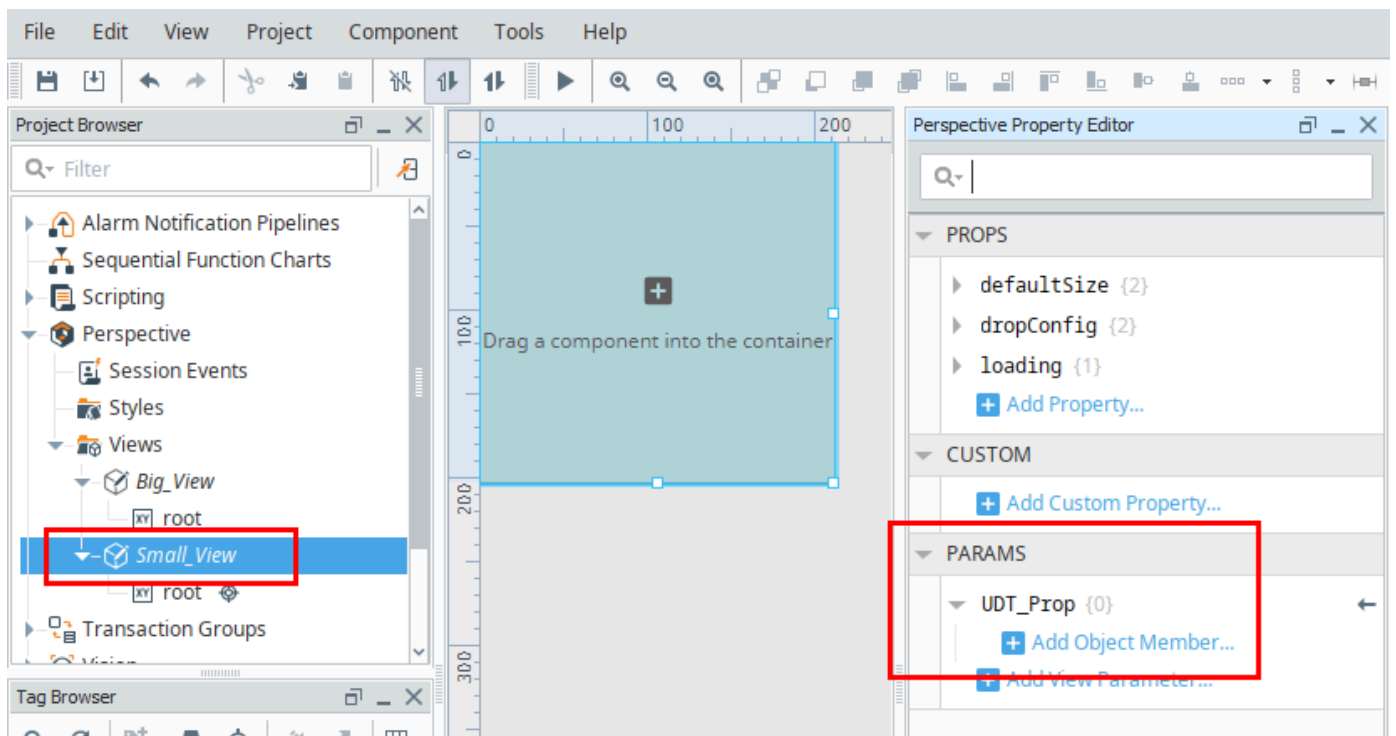
The screenshot shows the 'Tag Browser' window. The left pane displays a tree structure of tags. The right pane displays a table with columns 'Value', 'Data Type', and 'Traits'.

Tag	Value	Data Type	Traits
Tags			
Data Types			
UDT_Drop_Target			
My_Type	Hello!	String	Integer

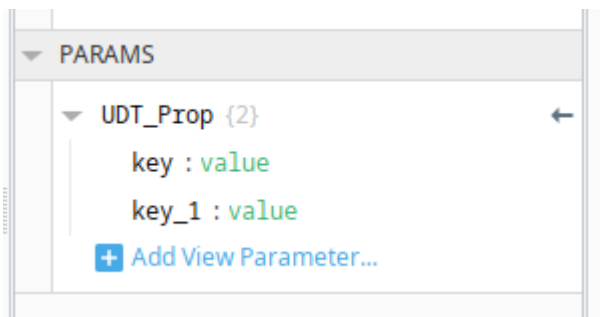
- Create two views. The first will be `Big_View` that we'll be dragging tags onto, and the second will be `Small-View` that we'll associated to the UDT above. For the purposes of this example, we'll use `Coordinate Containers` as the root container for both views.



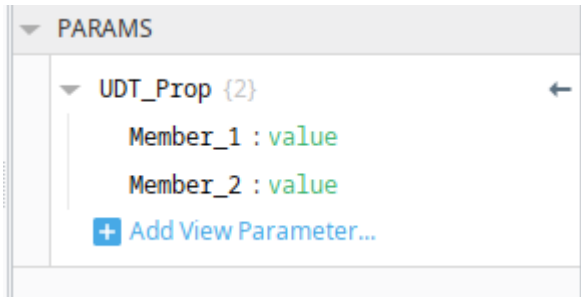
3. In the **Small_View**, we'll make a minor cosmetic change so it is easier to see when we're embedding it later. We'll change the background color to something other than the default.
4. Then, with the view selected, add a view parameter. When selecting the parameter type, choose **Object**. We'll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



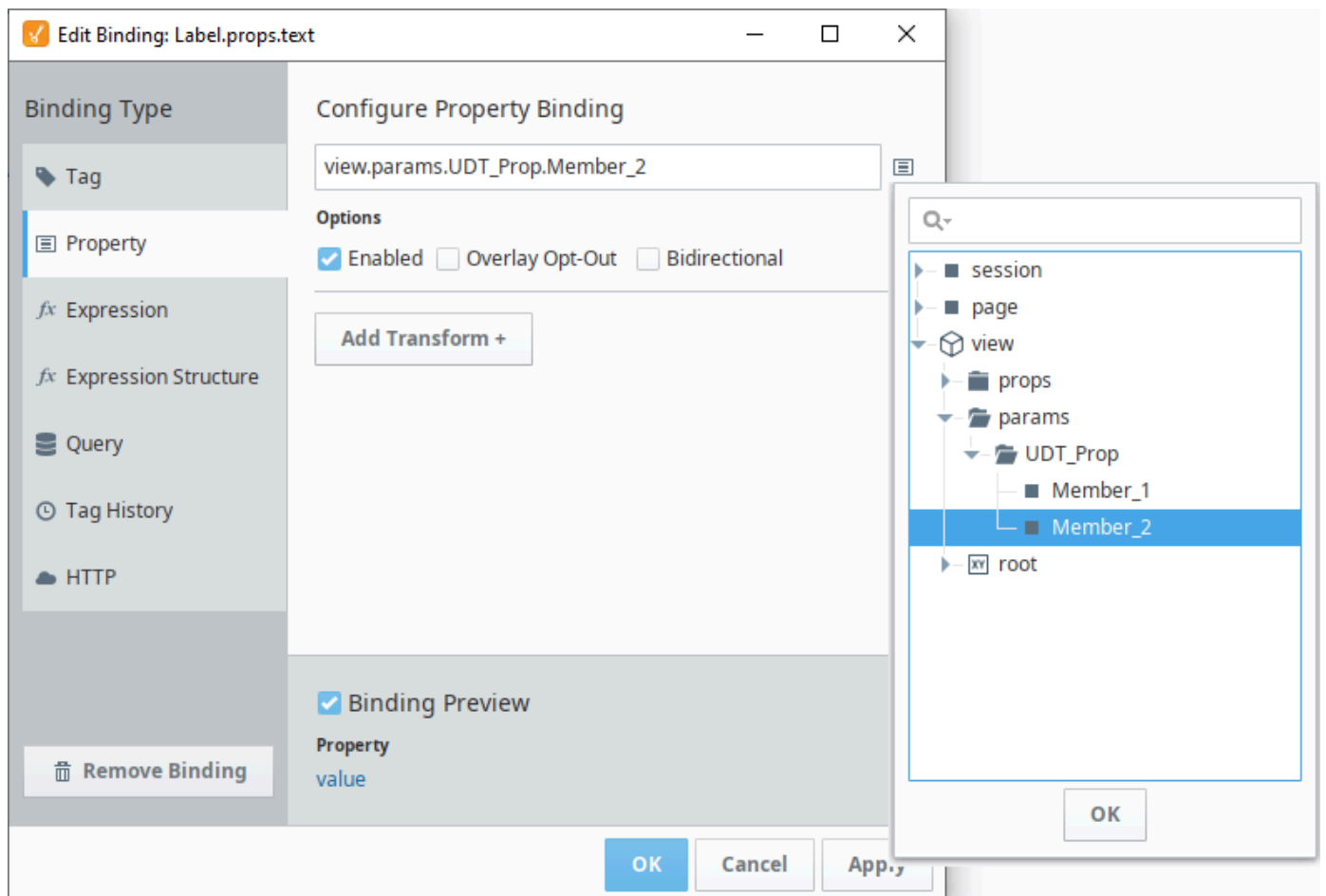
5. Add two object members to **UDT_Prop**. These members will represent the values on the two UDT members in our UDT. The values on our UDT members are integer and string, so both only have a single value. Thus, the object members we need to add here only need to show a single value each. Thus we'll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



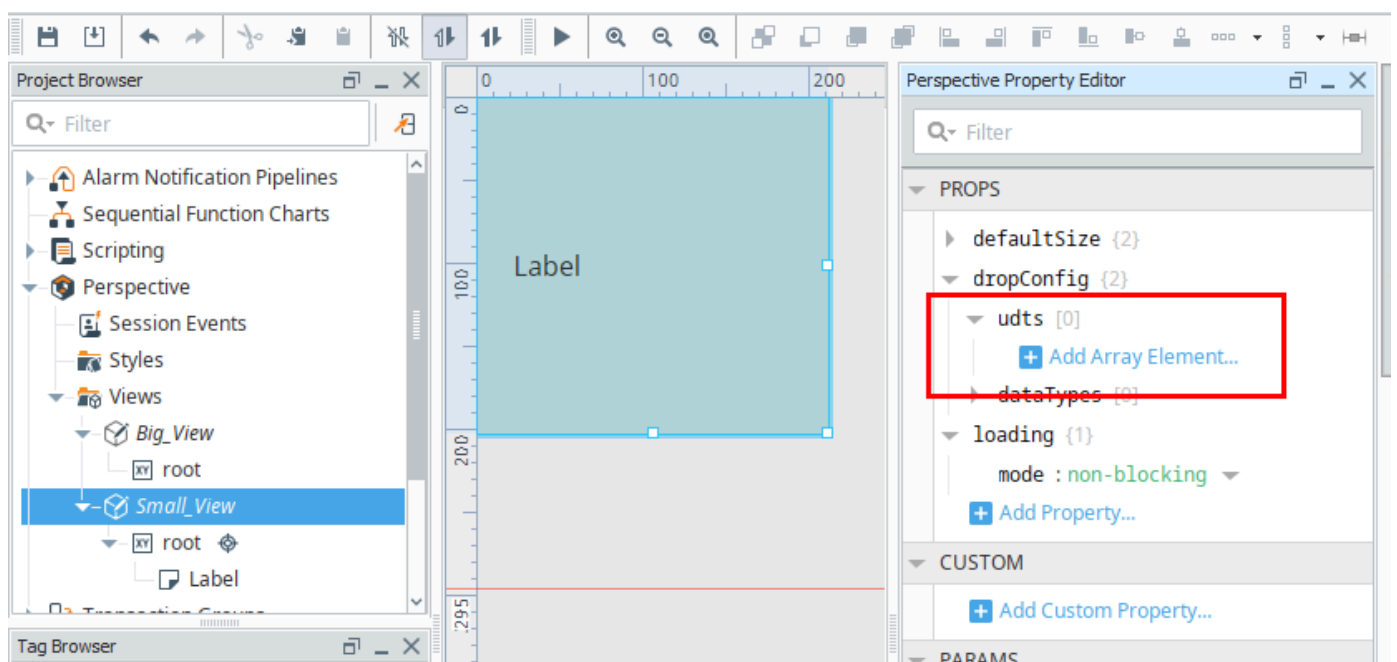
6. For the view parameters to know which members in the UDT to associate with, we need to rename them so they match the name of a UDT member. Change the name of the view parameters to **Member_1** and **Member_2**, to match our UDT members.



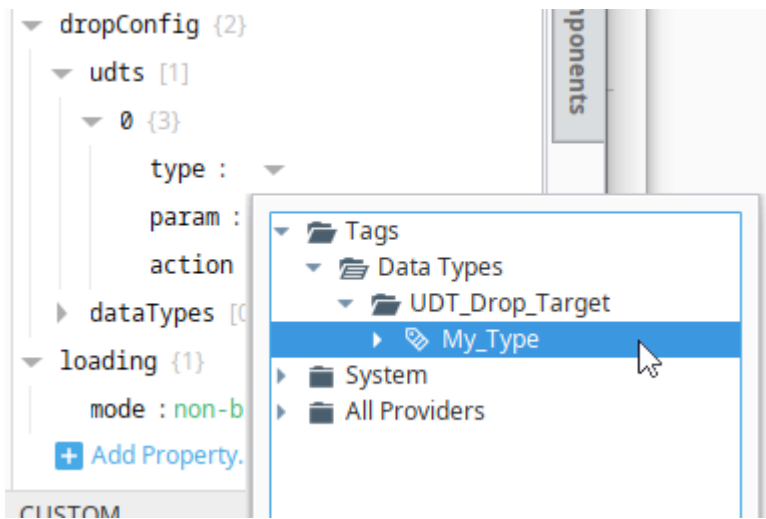
7. Now that we have some parameters that will eventually house values from our UDT, we can create components that use these parameters. Add a **Label** component to **Small_View**.
8. On the Label, configure a **Property Binding** on the **Text** property, leading to the **Member_2**. We're using Member_2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



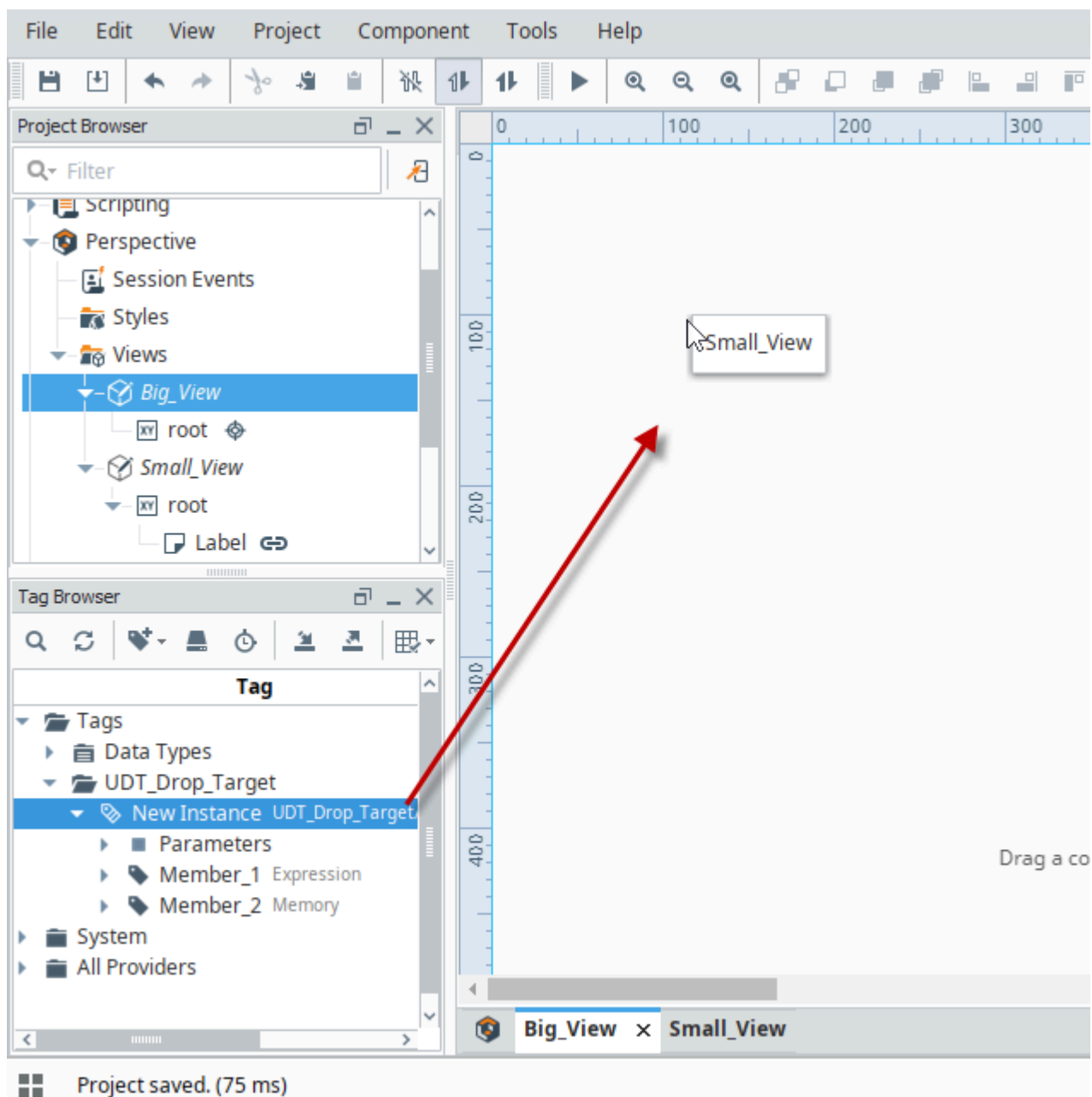
9. Next, we need to configure the drop configuration settings on **Small_View**. Select the view, and locate the **udts** property.



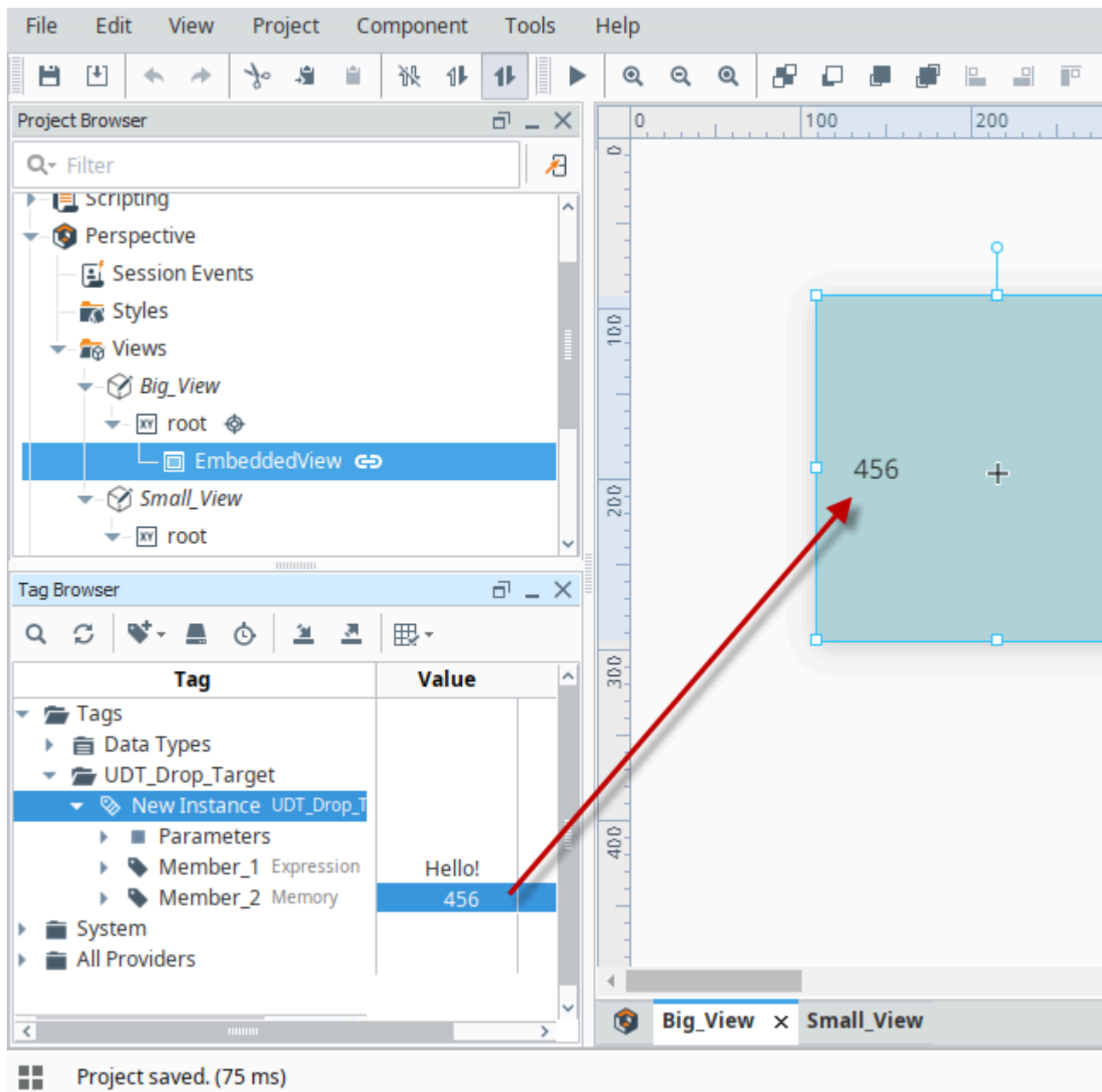
10. Add an array element to the **udts** array. For **type**, click the dropdown and select our UDT definition.



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. Furthermore, 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

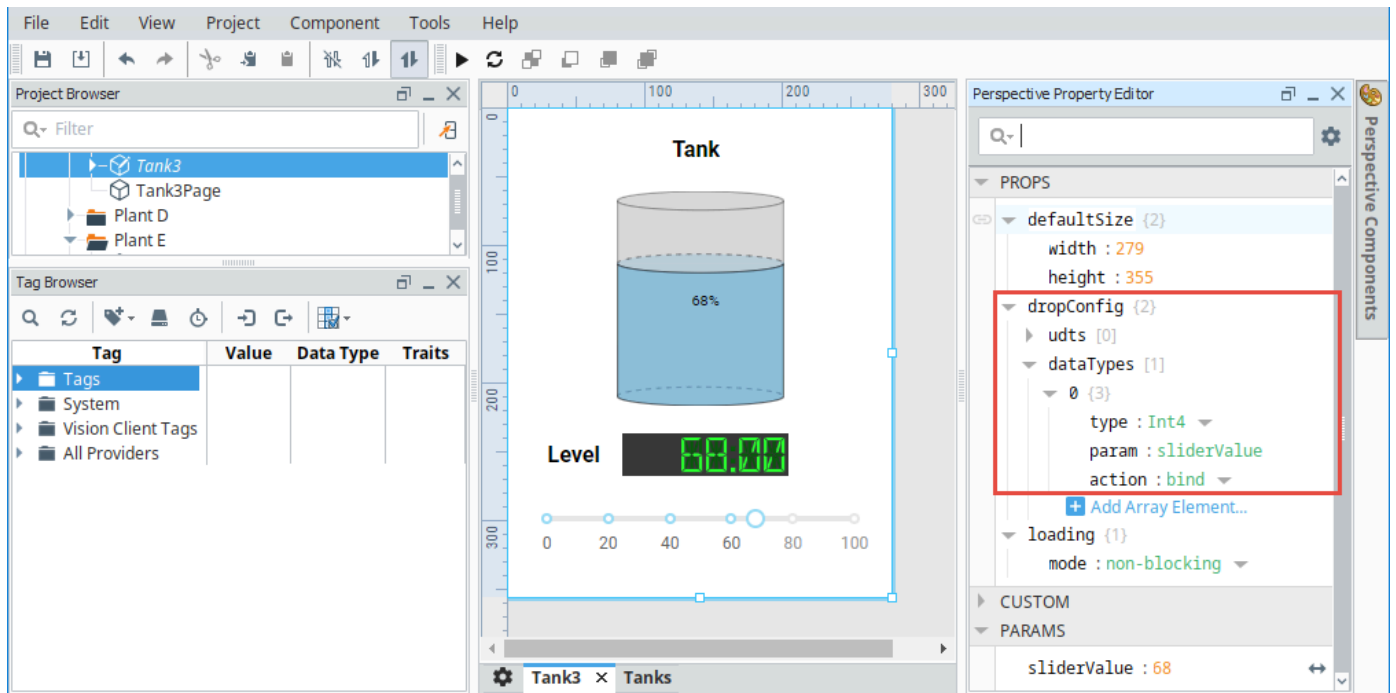
Data type drop configurations allow you to associate a view with a certain tag data type, as well as specify what bindings occur when dropping the view.

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:
 - i. Select the data type - **Int4**

ii. Set the param - **sliderValue**

iii. Identify the action - **bind**

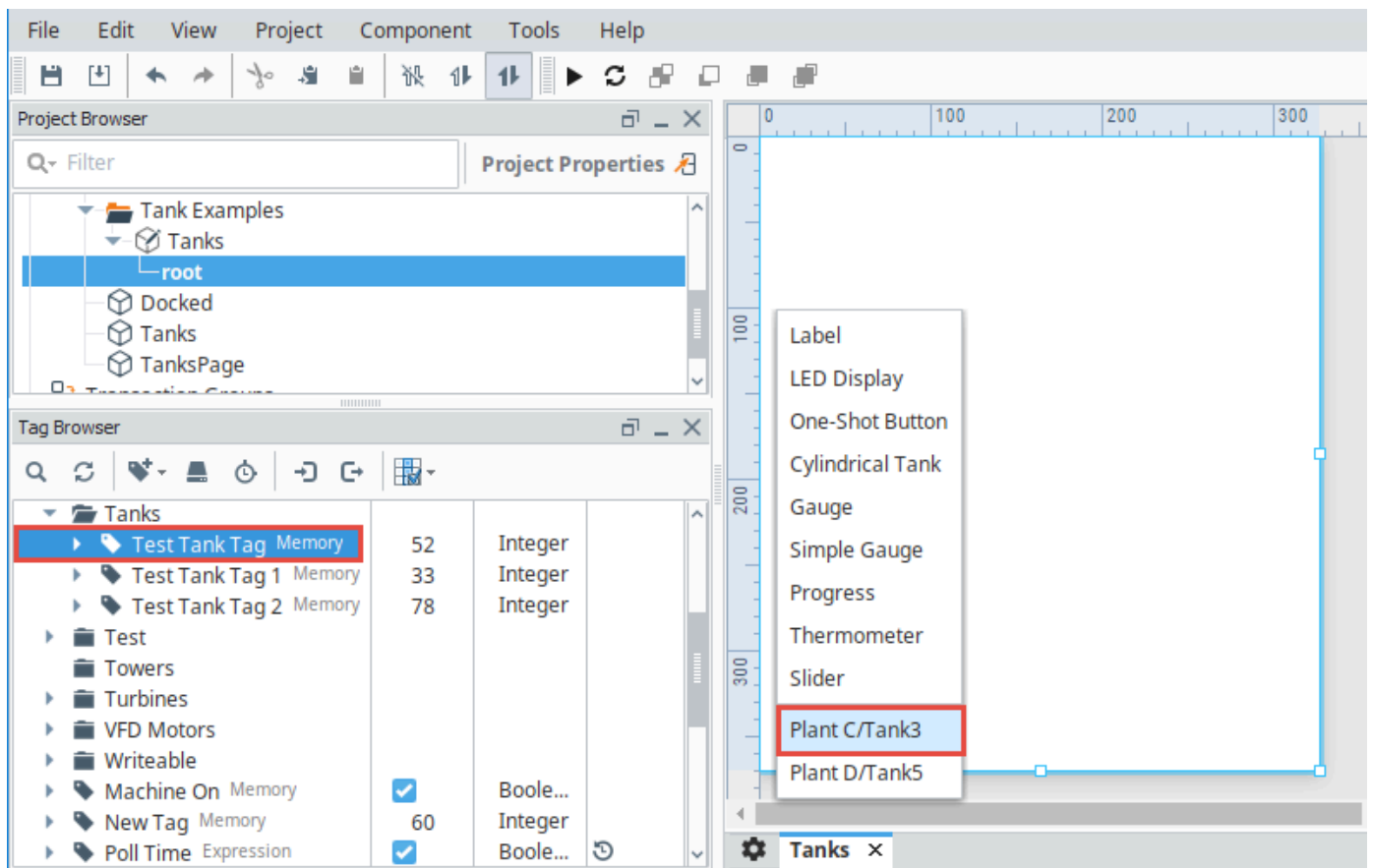


3. Let's create a couple of tags so we can use the '**dropConfig**' feature to create multiple embedded views. Copy the tag you created in the last example (i.e., Test Tank tag) to make two more tags, and change their values. This example uses 52, 33, and 78 for the values.

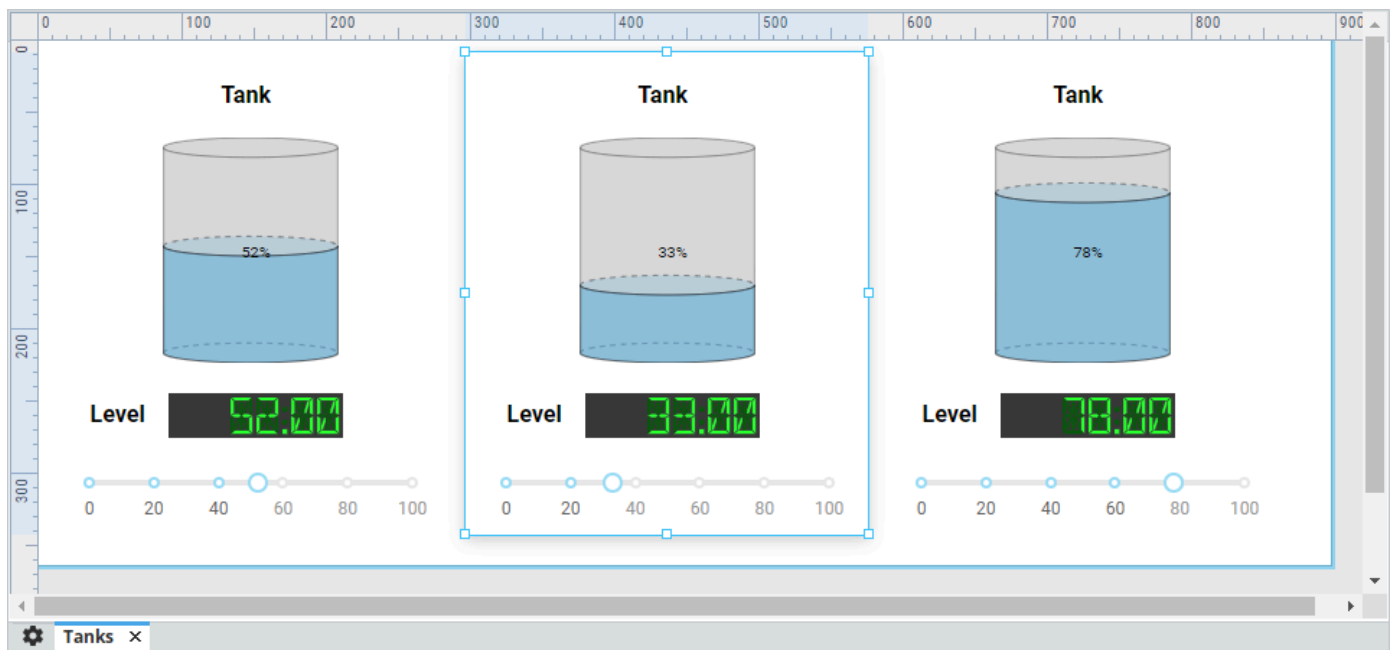
Tag Browser				
Tanks				
Test Tank Tag	Memory	52	Integer	
Test Tank Tag 1	Memory	33	Integer	
Test Tank Tag 2	Memory	78	Integer	

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).



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



INDUCTIVE
UNIVERSITY

Property Binding

[Watch the video](#)

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.

i USING PROPERTY BINDINGS ACROSS VIEWS

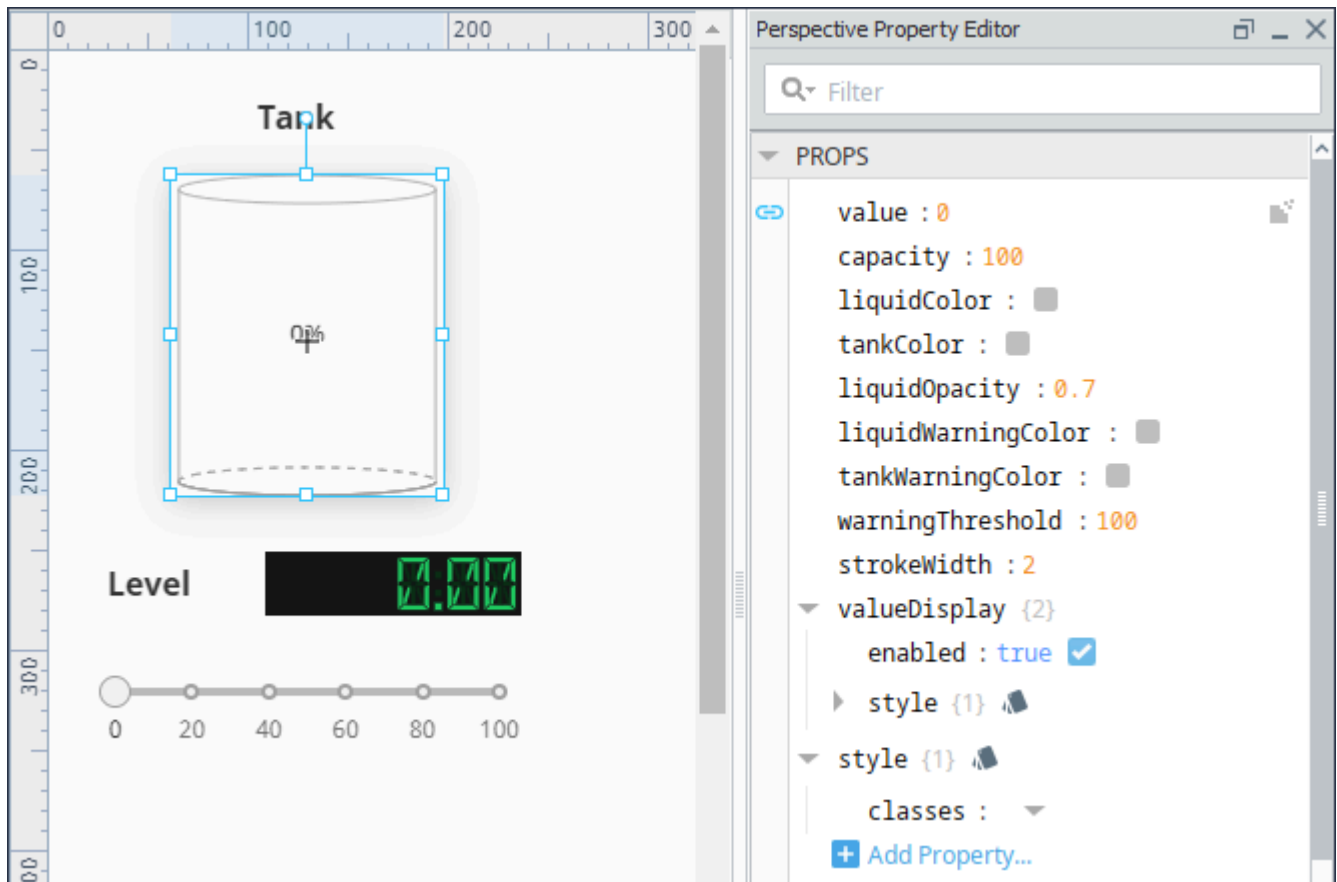
You cannot have a binding refer to a property in another view instance even view instances that may be embedded in a view. You can pass a property into an embedded view through a view parameter.

Property to Property Binding

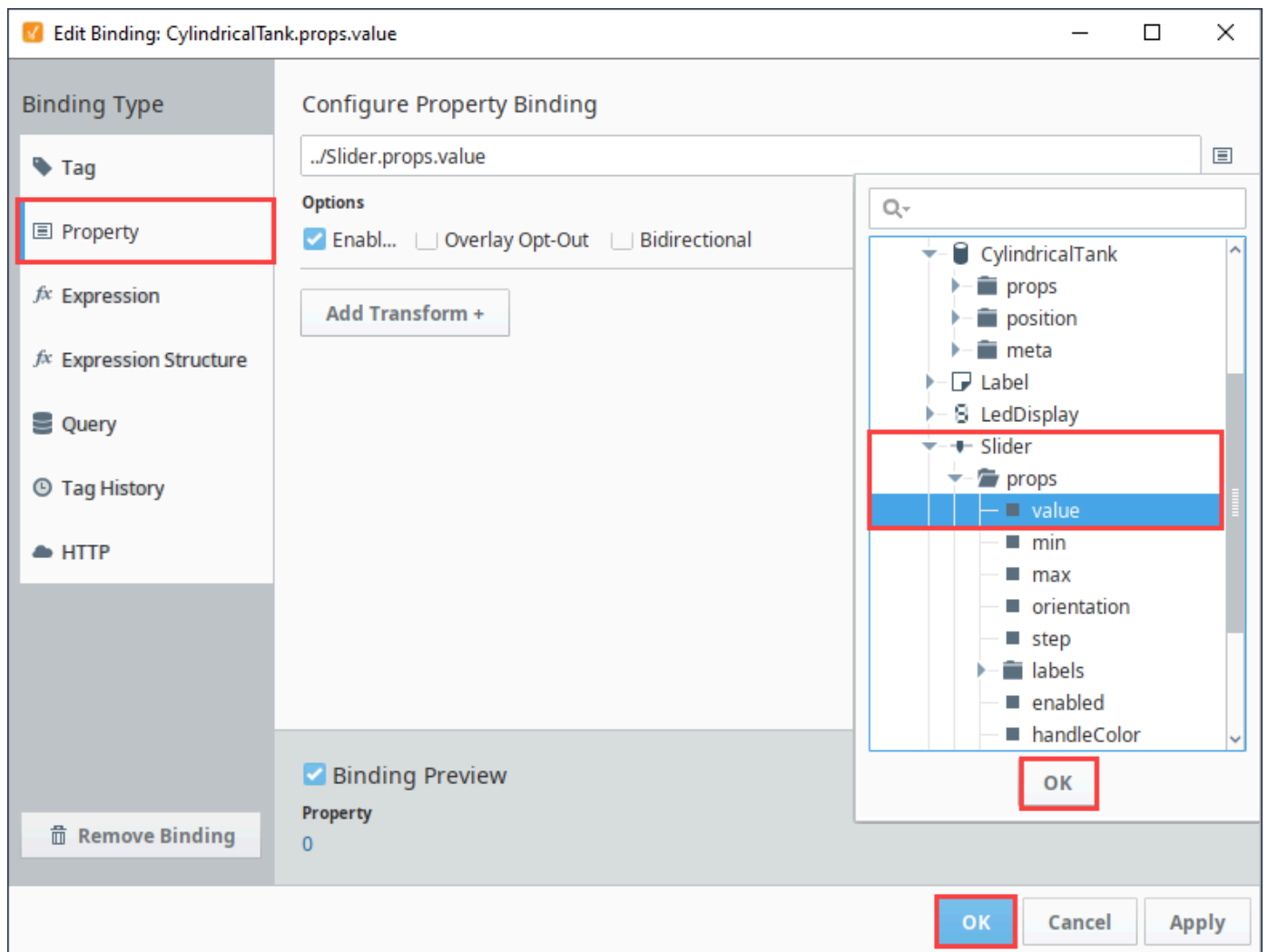
A property binding simply binds one component's property to another. When that property changes, the new value is pushed into the property that the binding is set up on. In this example, we'll use a Tank, LED Display, and Slider components. We'll bind the 'value' properties of the Tank and LED Display 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.

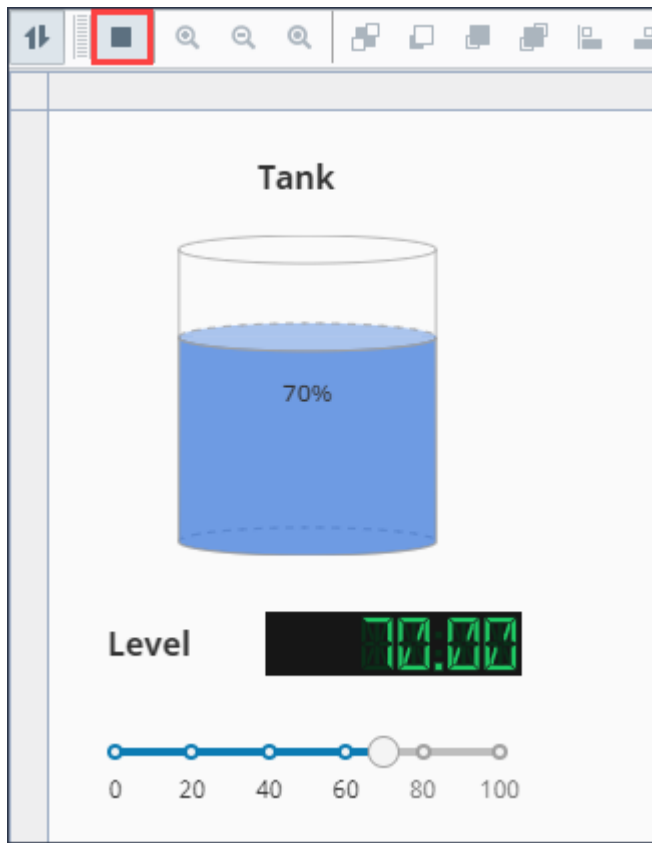
2. From the Component Palette, drag **Cylindrical Tank**, **LED Display**, and **Slider** components into your Designer workspace.
3. Select the **Tank**. In the **Property Editor**, click on the **Binding**  icon for the **value** property.



4. This opens the Edit Binding window. Configure the following settings:
 - i. Select the **Property** binding type.
 - ii. 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**.
 - iii. Click **OK** to save the binding settings.



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.



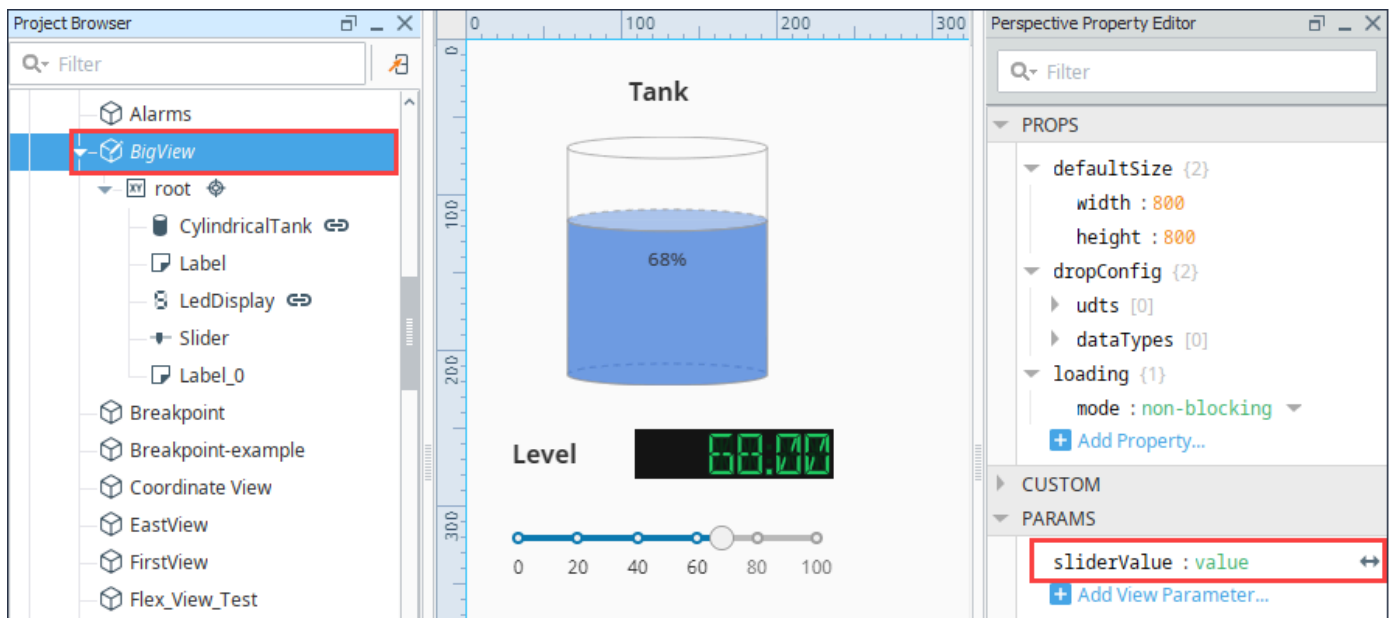
Pass a Property into an Embedded View Using a View Parameter

The only way to 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 an 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.

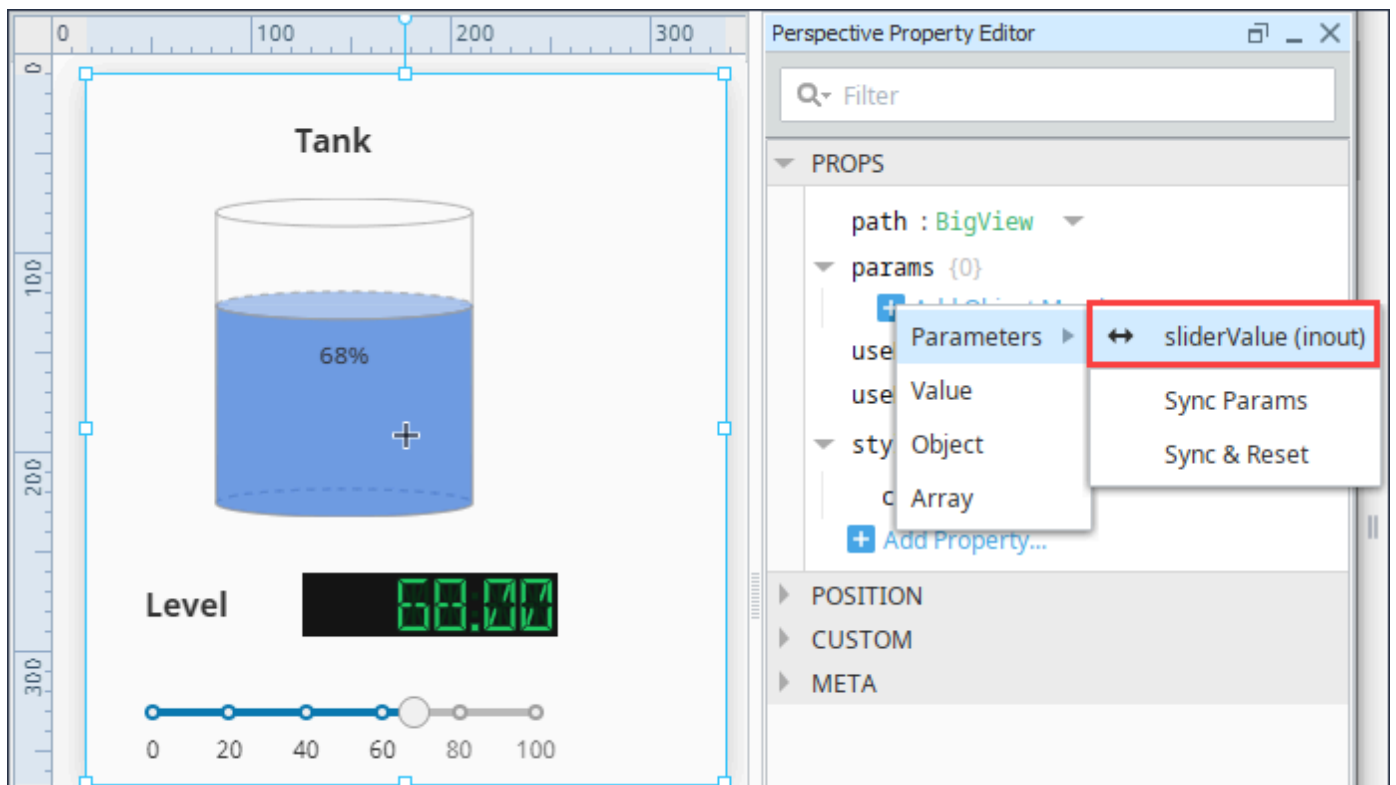
Use a Property Binding

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 an **Embedded View** from the Component Palette to the Designer workspace.
 - i. With the Embedded View selected, set the 'path' property to your original view (i.e., BigView) from the dropdown list.
 - ii. In the **Property Editor**, click the **Add Object Member**  icon under props.params.
 - iii. In the Parameters list, select the sliderValue parameter.



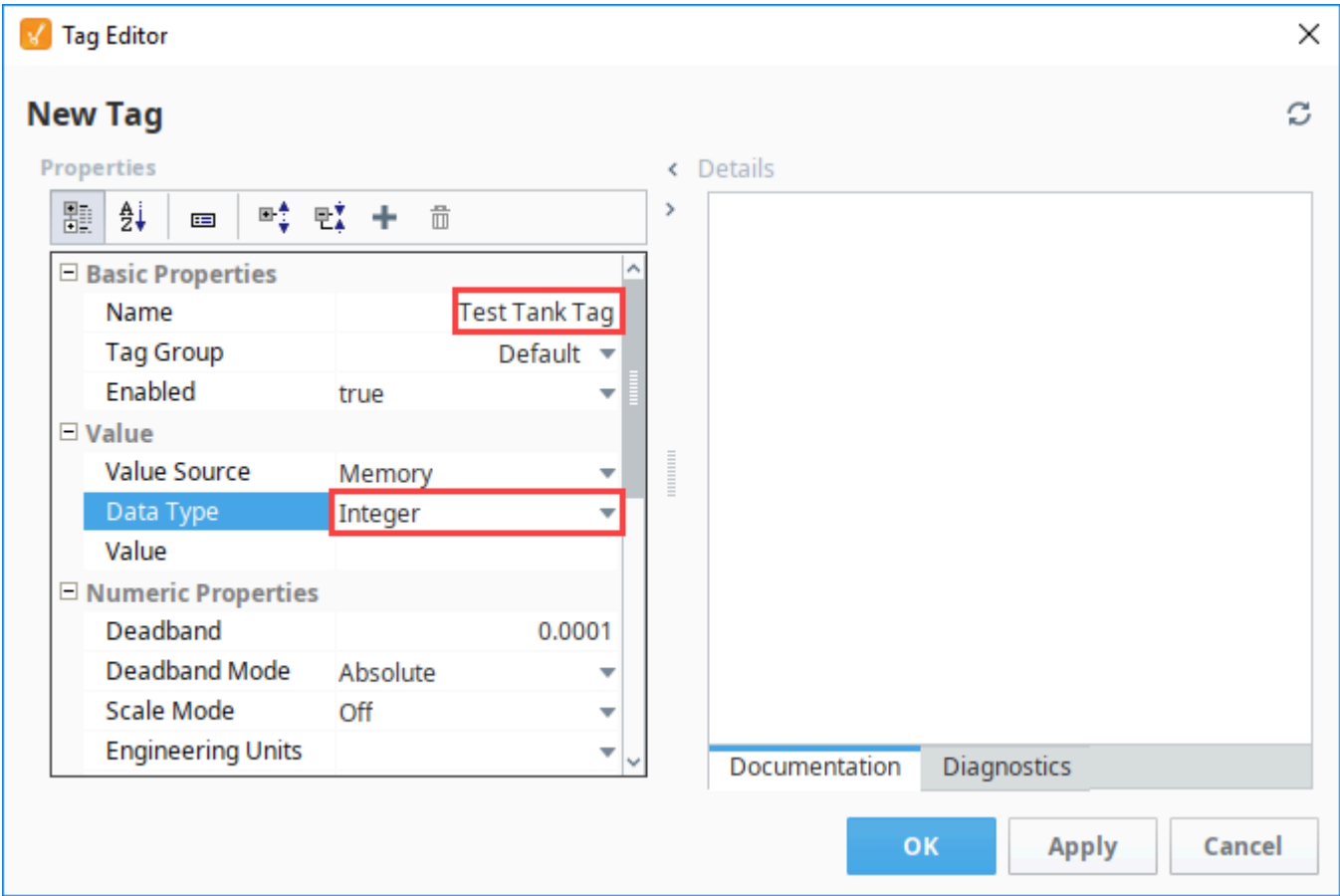
5. **Save** your project.
6. From this new view, put the Designer in **Preview Mode** . Drag the slider to a value to change the value on the Tank and LED Display.

7. You can see from the Embedded View (i.e., Tank3Page), that your sliderValue reflects the same values as your slider.

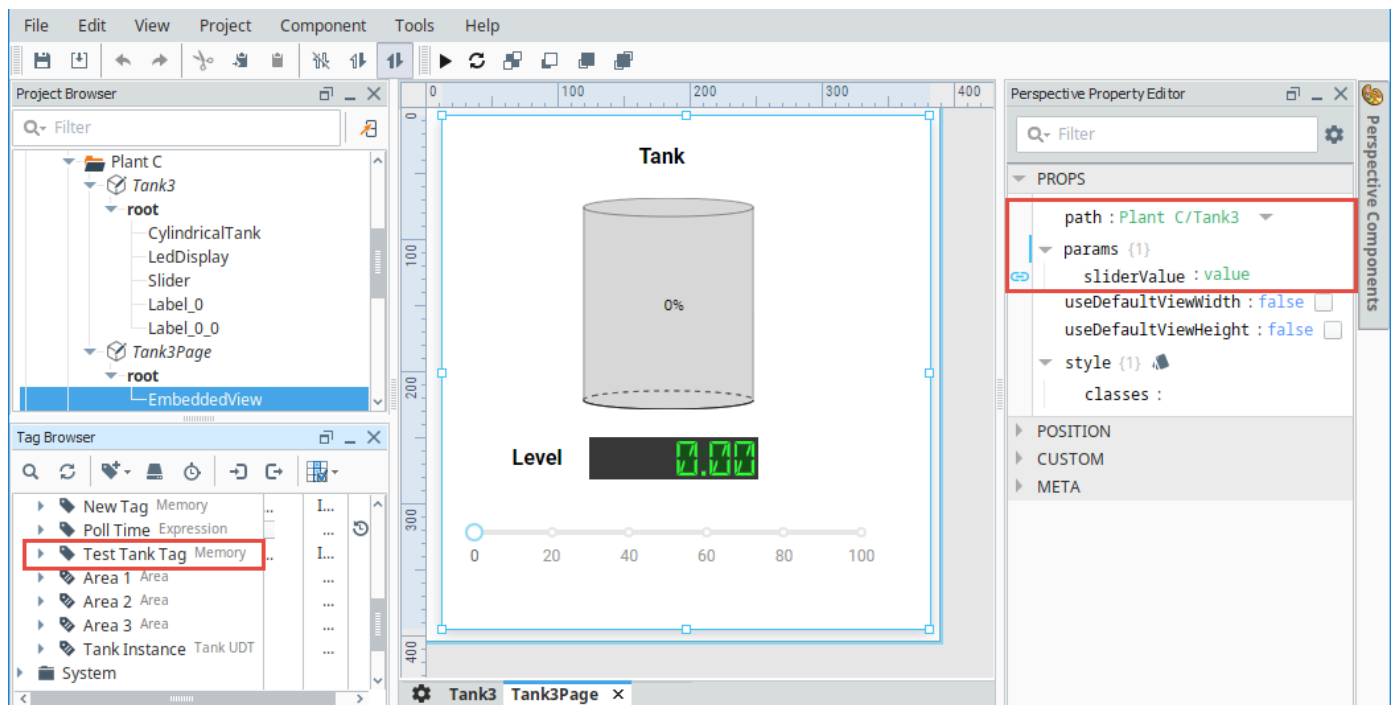
Using a Tag Binding

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).

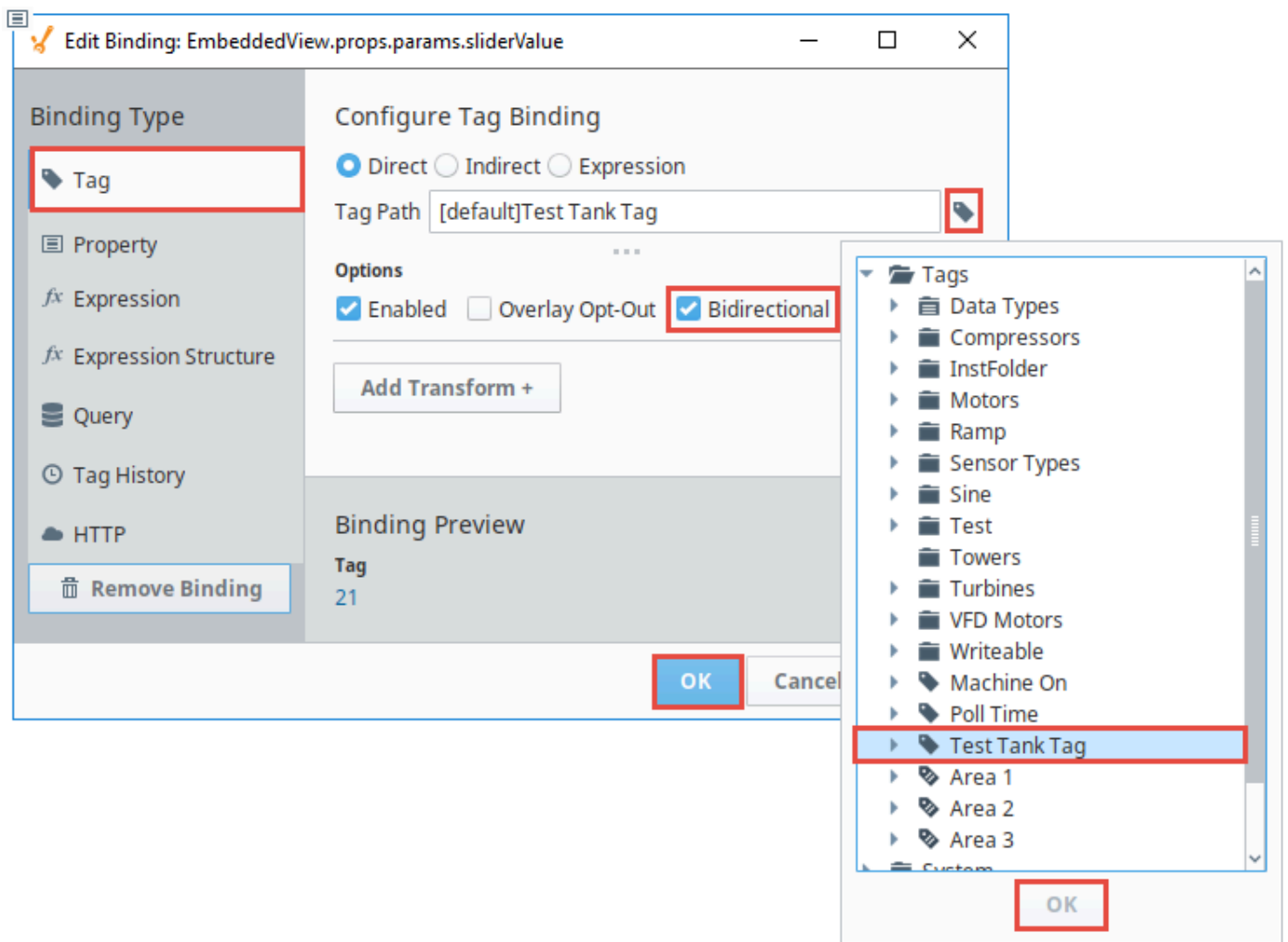


2. Select the **Embedded View**. Click on the **Binding**  icon next to the '**sliderValue**' and bind it to the Memory tag (i.e., Test Tank Tag).



3. This opens the Edit Binding window. Configure the following settings:

- i. Select the **Tag** binding type.
- ii. 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**.
- iii. Click the **Bidirectional** checkbox. allowing the tag to be updated by the embedded view.
- iv. Click **OK** to save the binding settings.



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



INDUCTIVE
UNIVERSITY

Expression Binding

[Watch the video](#)

Binding Properties to the Outcome of an Expression

An expression binding is one of the most powerful kinds of property bindings. It uses a simple **expression language** to calculate a value. This expression can involve lots of dynamic data, such as other properties, tag values, results of Python scripts, queries, and so on. 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.

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.

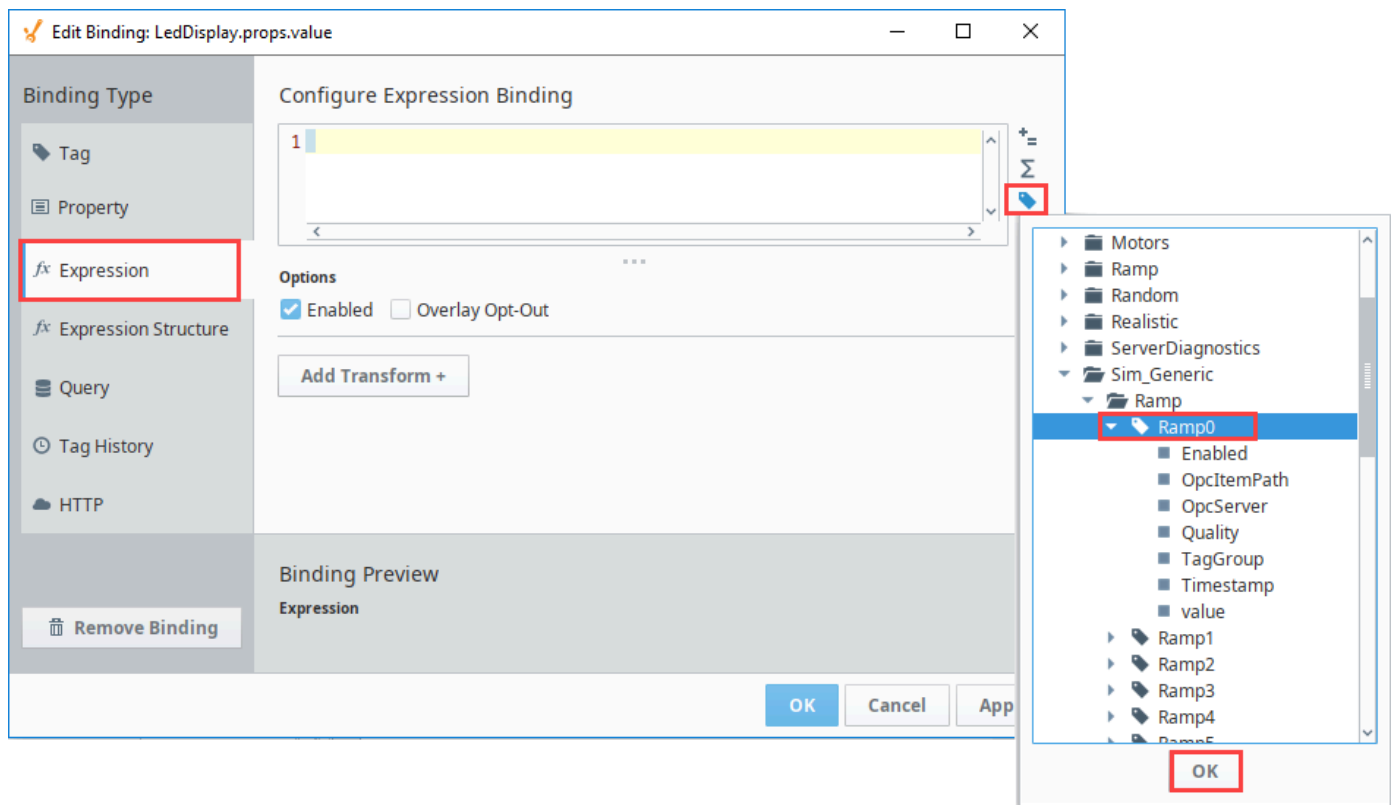
To the right of the Expression Binding window, there are four buttons which can be used to reference specific objects or functions easily.

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.
	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.

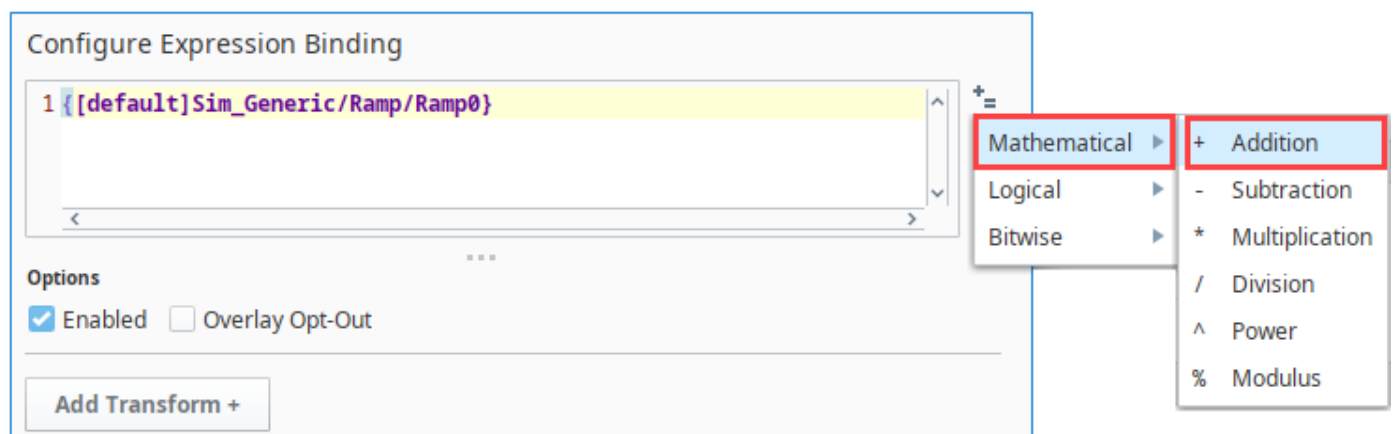
Example 1

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**.

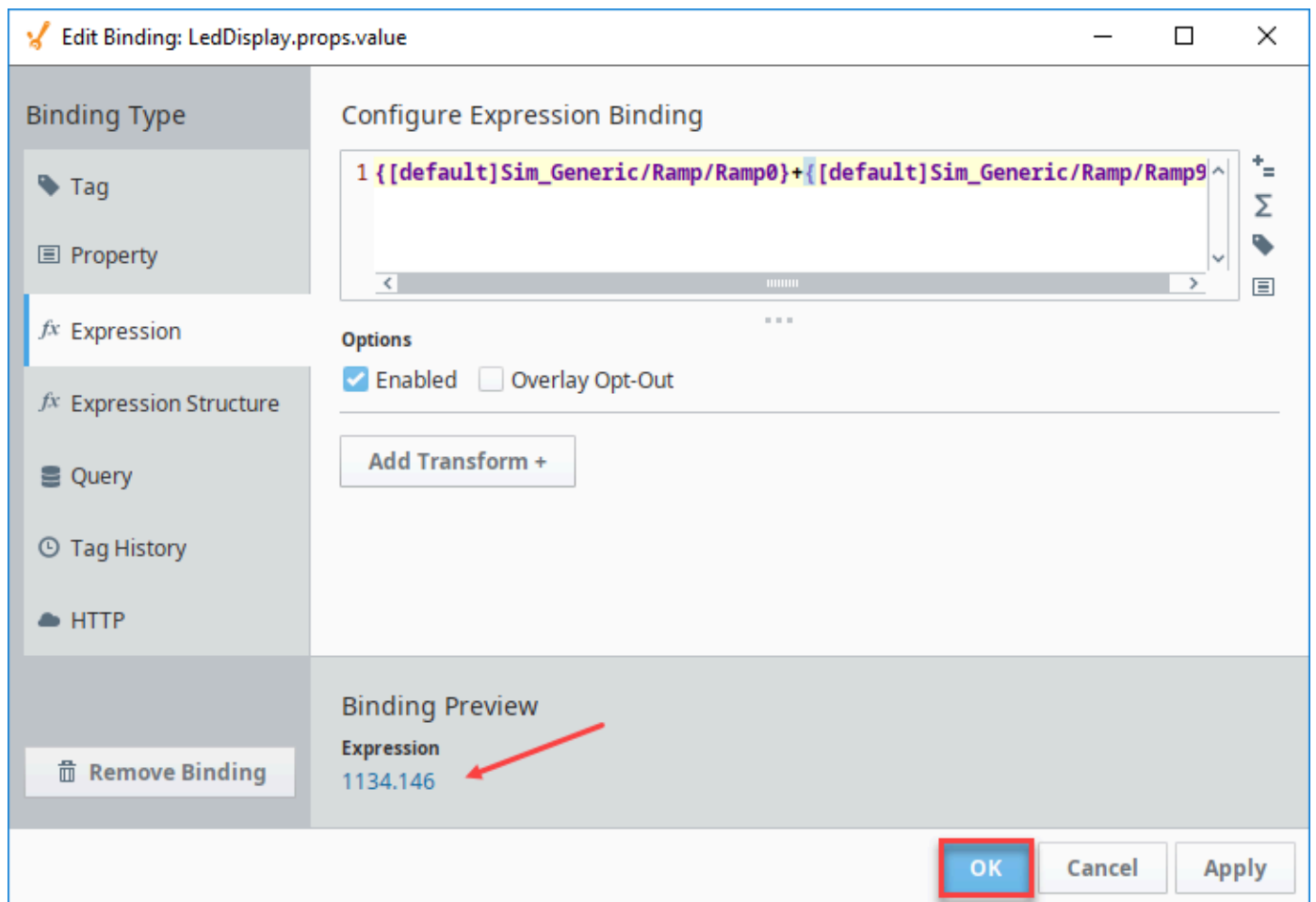


6. Next click the **Operators** $+=$ icon. Select **Mathematical**, and then choose the **Addition** option.



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

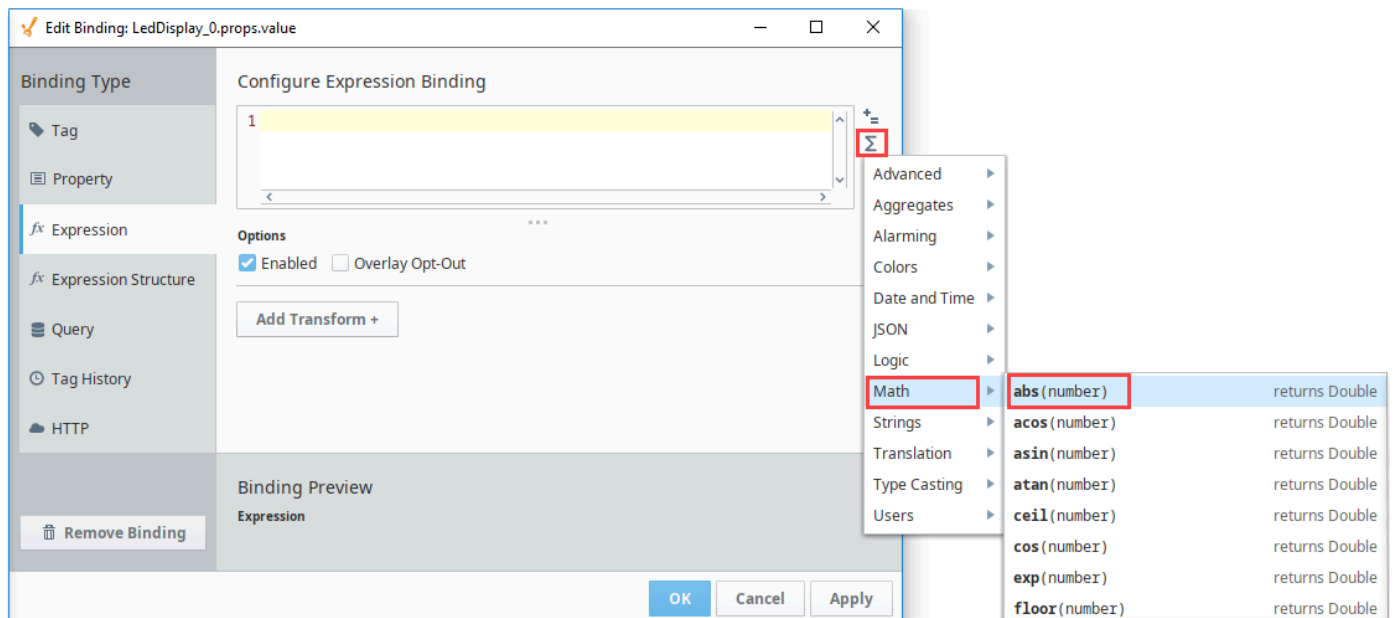
Now when this Expression is run, the value of this tag will be added to the first tag. Note that a preview of the Expression binding value is shown on the lower left.



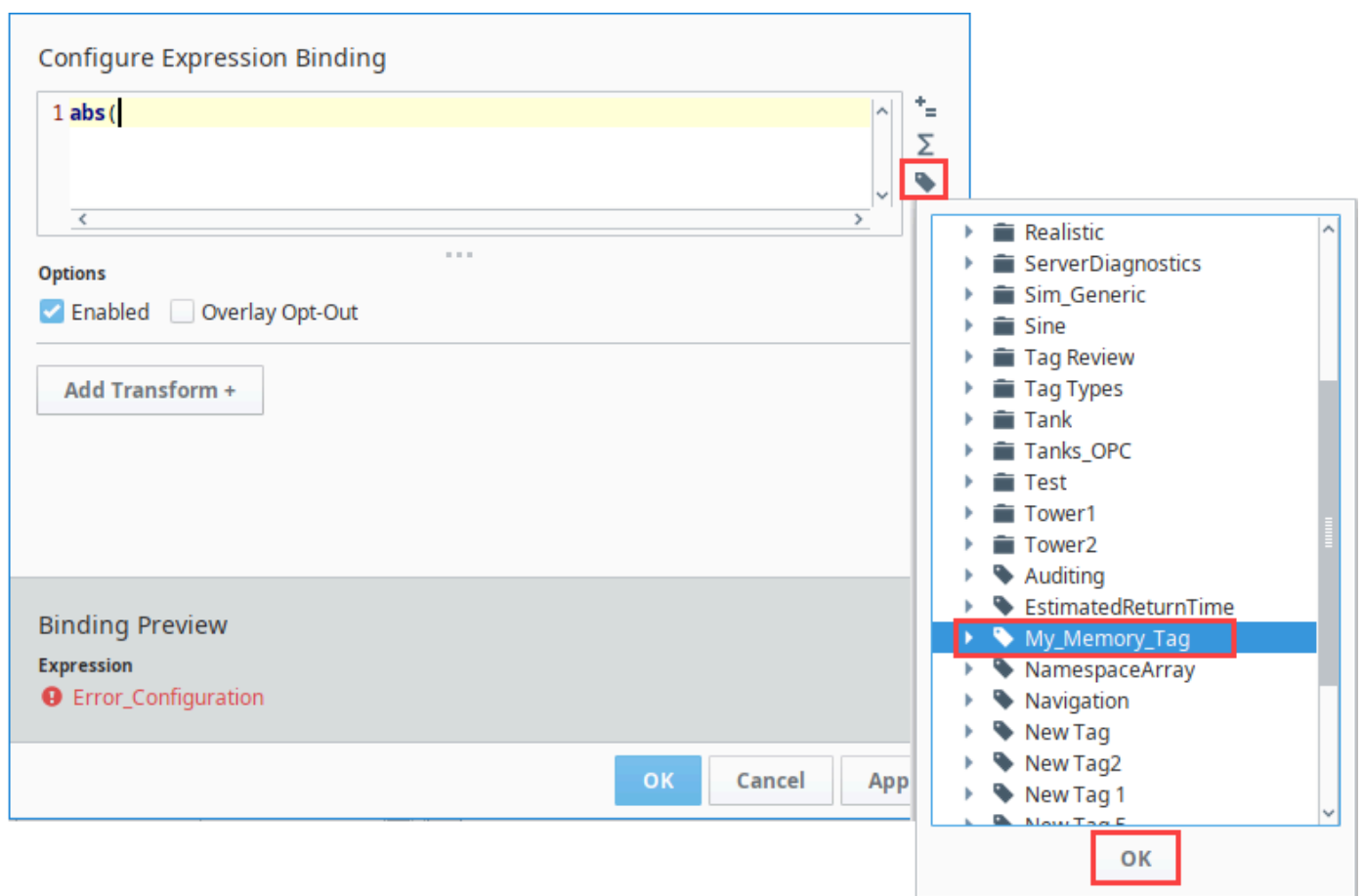
Example 2

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

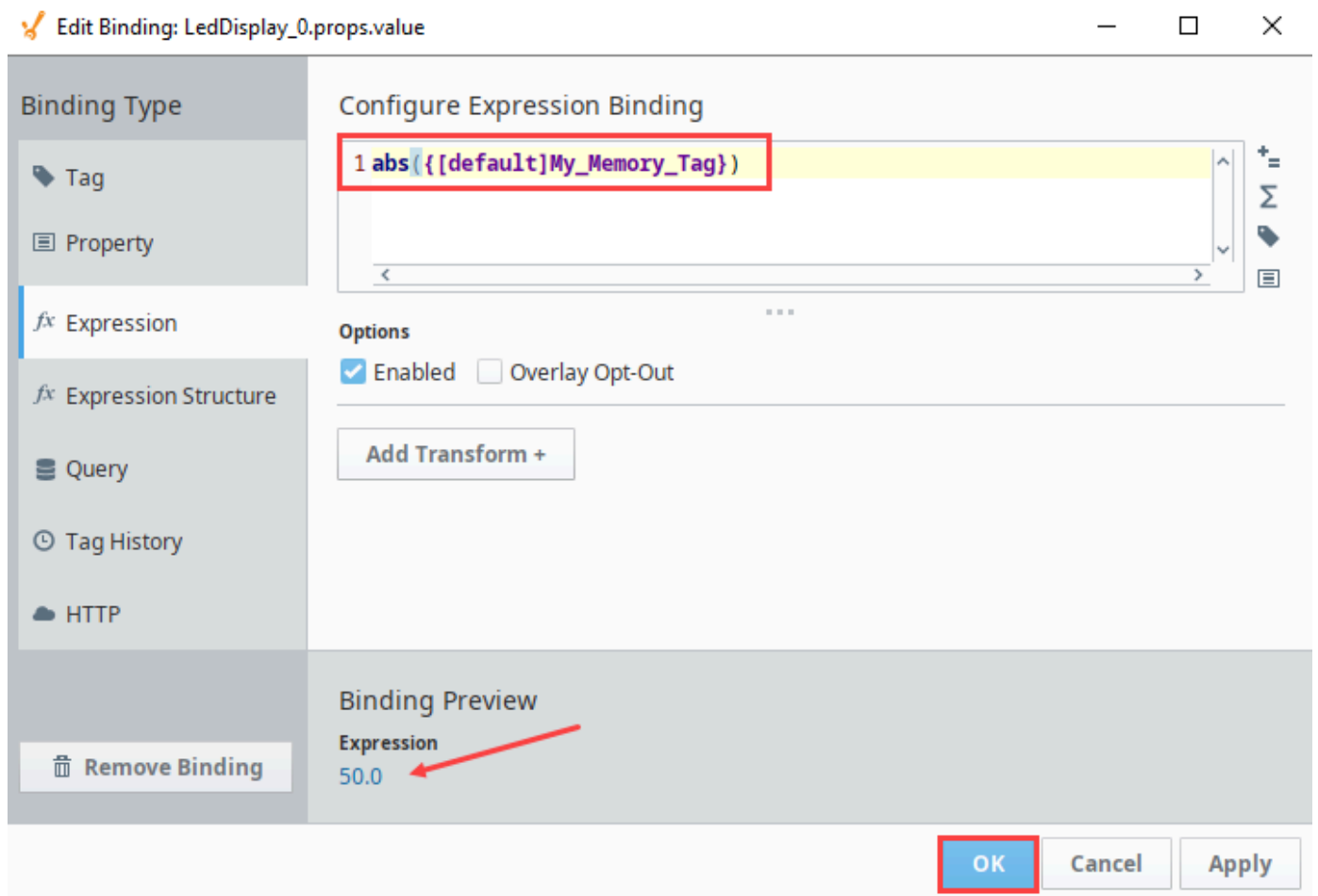
1. Select the second LED component and click on the **Binding**  icon for the value property.
2. Select Expression the binding type.
3. In the Configure Expression Binding section, click on the **Functions**  icon. Scroll down to the **Math** functions and then select the **abs** (number) function.



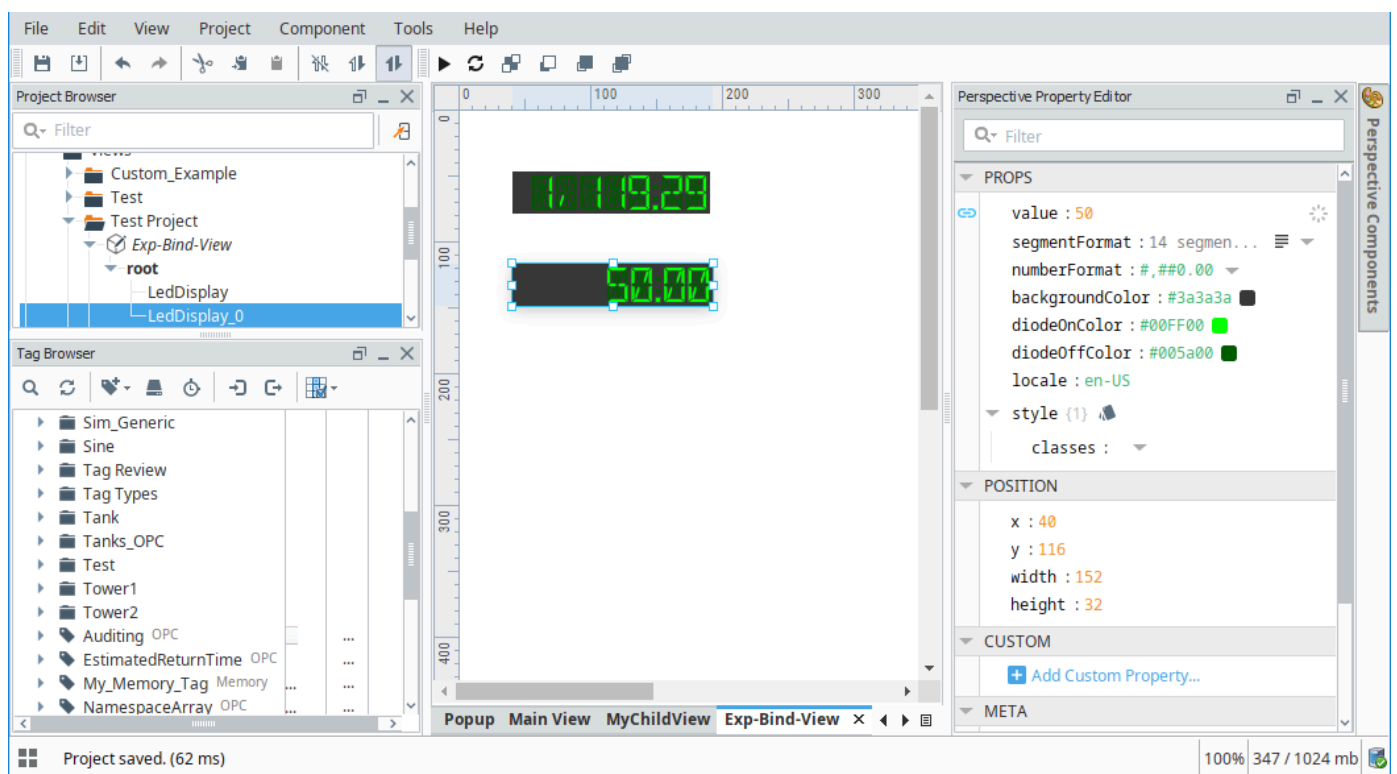
4. Next click the **Tag**  icon and select the Memory tag. Click **OK**.



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**.



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



Expression Structure Bindings in Perspective



INDUCTIVE
UNIVERSITY

Expression Structure Binding

[Watch the video](#)

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

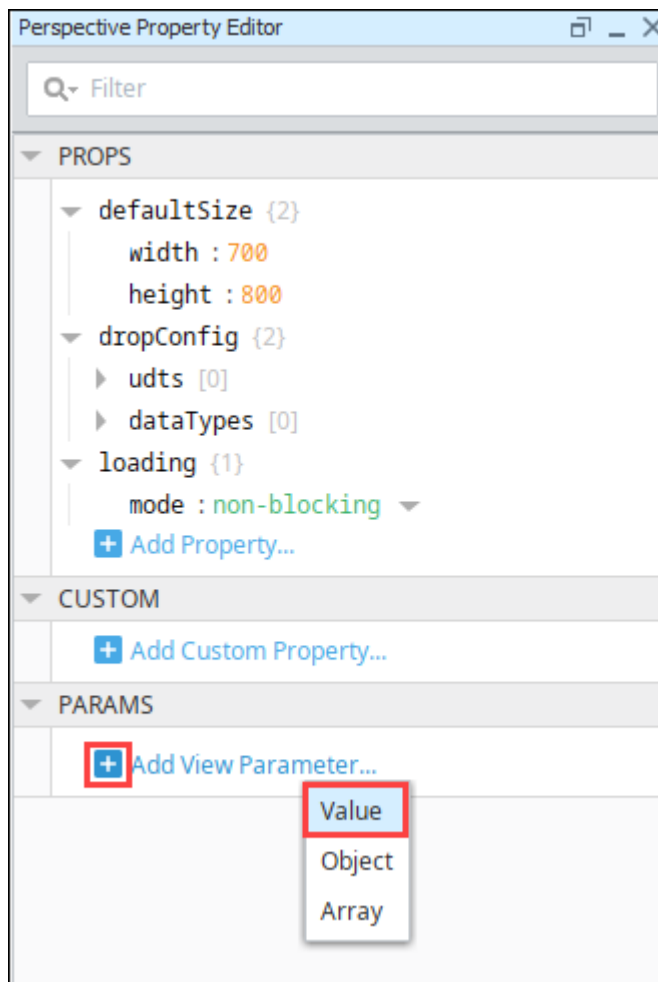
The Expression Structure binding is configured similarly to any object in Perspective, except that every **value** property in the object is evaluated via an individual expression.

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**.
 - i. Place a Carousel component and a Label component on the view.
 - ii. 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.
 - i. In the Property Editor under PARAMS, click **Add View** parameter and select the Value option.



ii. Change the word "key" to the parameter name we want, which is "iconPath."

iii. Click **Add View Parameter** again.

iv. Name the second view parameter "labelText."

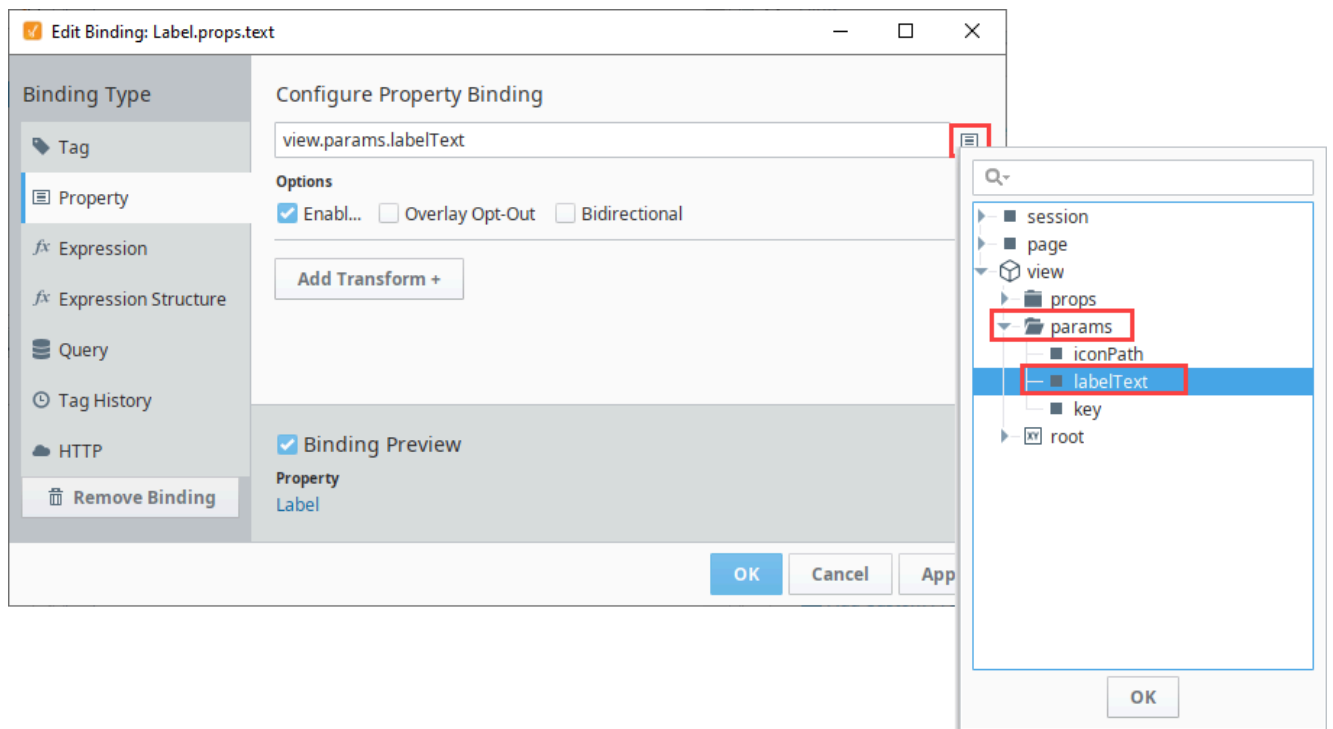
7. Next we'll bind the components to the view parameters. On MyChildView, select the Label component.

i. Click the **Binding**  icon next to the text property.

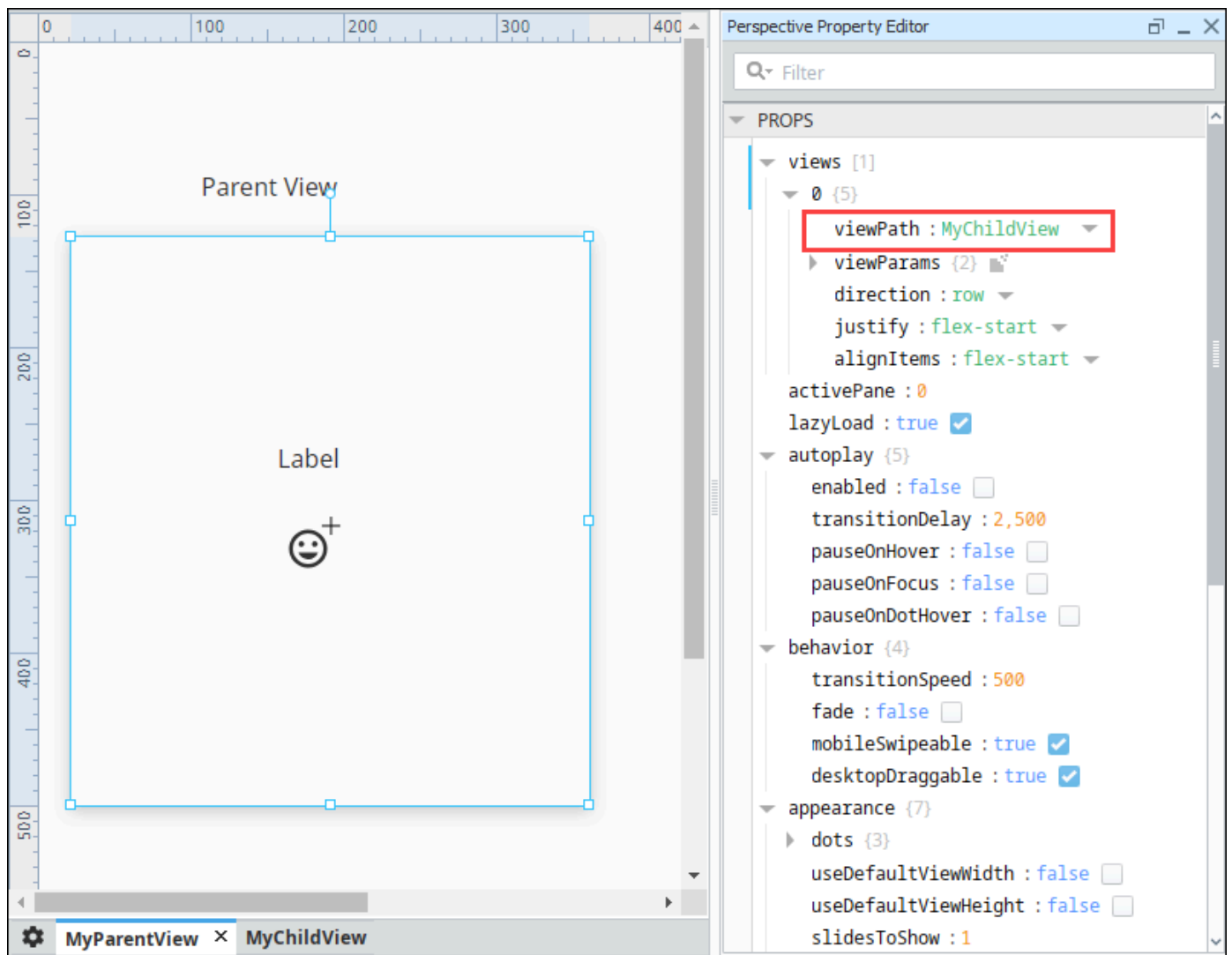
ii. Select the Property binding type.

iii. Click the **Property Editor**  icon.

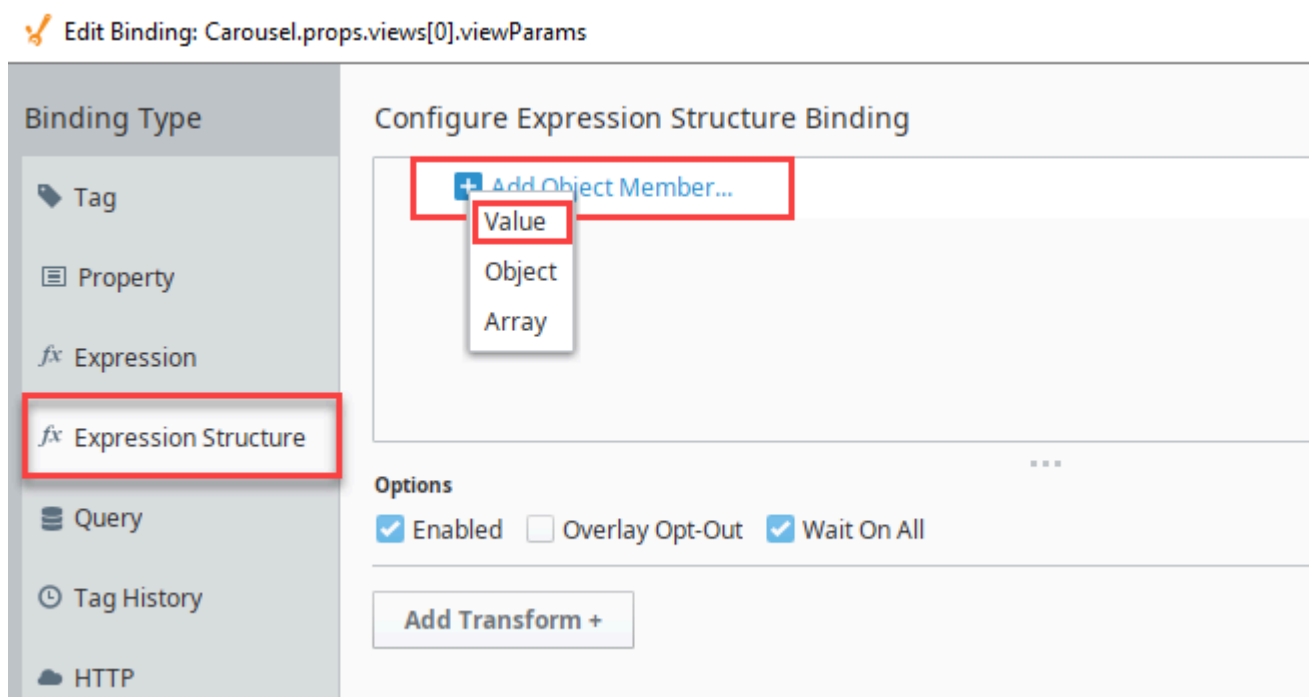
iv. Scroll down to the labelText view parameter. Click **OK**. Click **OK** again to save the binding.



- v. Select the **Icon** component.
 - vi. Click the **Binding**  icon next to the path property.
 - vii. Select the Property binding type.
 - viii. Click the **Property Editor**  icon.
 - ix. 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**.

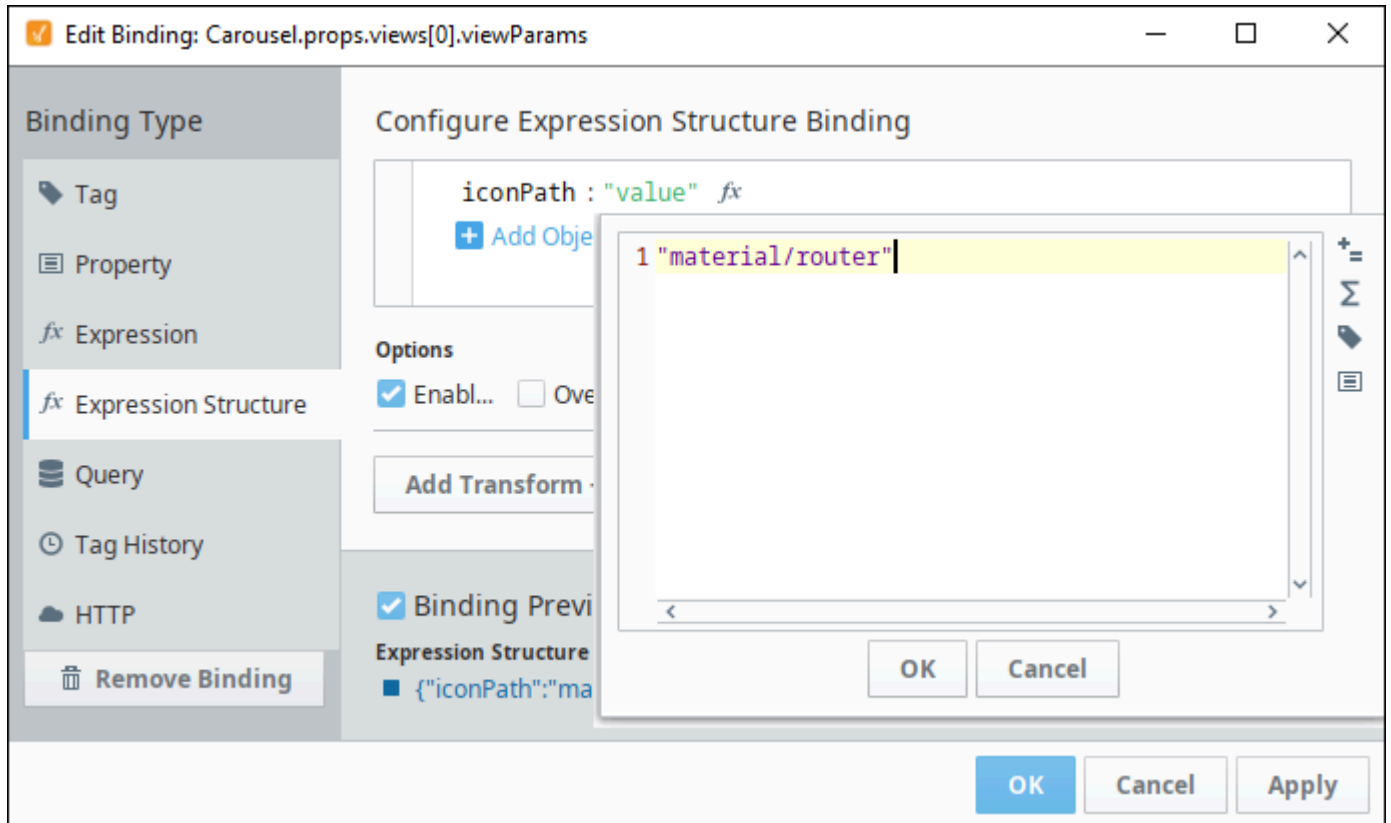


9. Next we need to add two view parameters. Still on the MyParentView, click the **Binding**  icon next to **viewParams**. The Edit Binding screen is displayed.
10. Choose Expression Structure as the binding type. Click **Add Object Member...** and select **Value**.



11. Name this parameter **iconPath**.

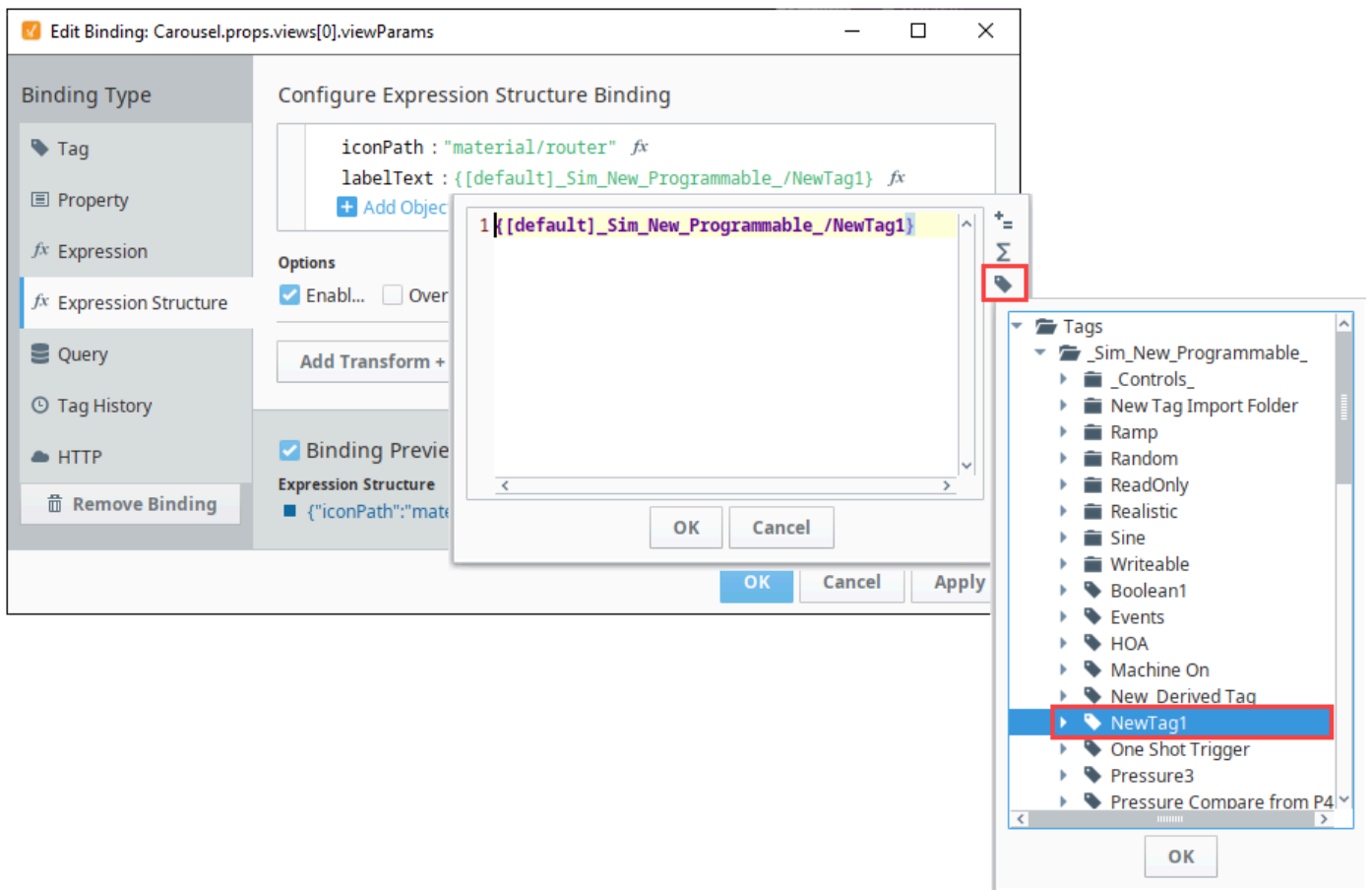
12. Click the **Expression** *fx* icon, then enter "material/insert_emoticon" as the expression. Click **OK**.



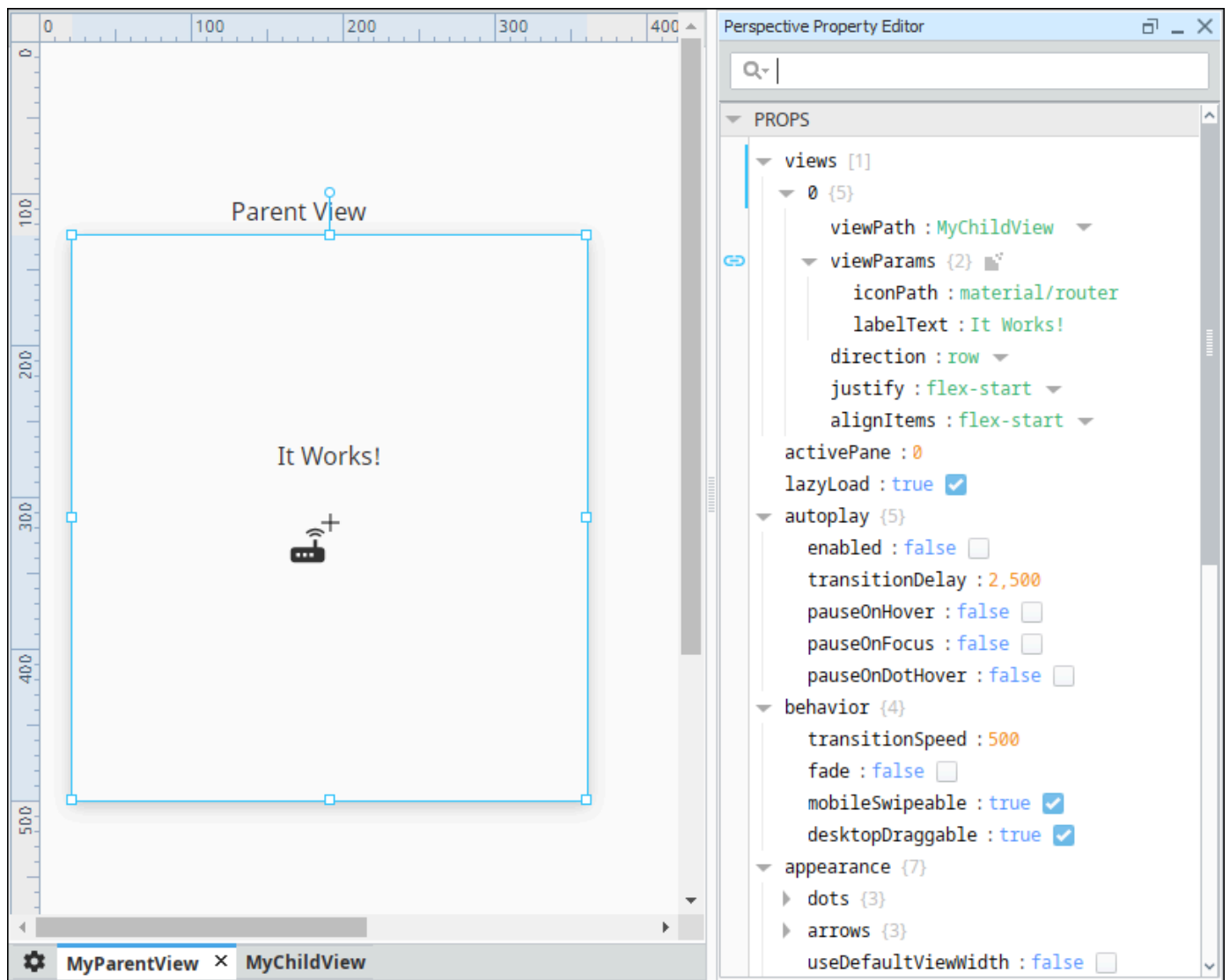
13. Click the **Expand**  icon and select Value to add another parameter.

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**.



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).



Query Bindings in Perspective



INDUCTIVE
UNIVERSITY

Query Bindings

[Watch the video](#)

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: Label.props.text

Binding Type

Tag

Property

fx Expression

fx Expression Structure

Query

Tag History

HTTP

Remove Binding

Configure Query Binding

Path Mode

☒ Direct ☐ Expression

Path

test

Parameters

No parameters defined

Query

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

Return Format

auto

Options

☒ Enabled ☐ Overlay Opt-Out ☒ Cache & Share ☒ Designer Limit

☐ Polling 30

fx

 sec

Add Transform +

☒ Binding Preview

Query

Dataset[0 rows, 1 cols]

OK

Cancel

Apply

Binding Properties

Property Name	Description
Path Mode	New in 8.3.7 Allows users to choose between binding a direct path to a Named Query, or using an expression to make the binding dynamic.
Path	The path to the Named Query the binding will use. Clicking on the Search  icon will produce a list of all available Named Queries.
Parameters	Displays a table of all defined Named Query parameters. This field will be different depending on if you select Direct or Expression for the Path Mode. Differences include: • Direct: Parameters will be automatically populated with the parameters of the selected Named Query. You can still pass in property or tag values to the parameters by first highlighting the parameter, then selecting either the Property icon or the Tag icon. • Expression: Users will be able to both set their own parameters on the Query binding itself, and pass in property or tag values. Note: The fields under the Value column are evaluated as expressions, so strings will require quotations marks.
Return Format	Specifies how the query results are returned. Options are: • auto: Query results are returned in the format native to the database, which is 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.
Options	A set of various options related to the Query binding itself. See the Options table below for more information.
Binding Preview	A toggleable field that gives a preview of what the Query Binding will look like.

Options

Property Name	Description
Enabled	Allows the component to be enabled and interactive (depending on the component) on the screen.
Overlay Opt-Out	Indicates whether the component should reflect the quality code with a component overlay if the target of the binding returns a non-good quality code.
Cache & Share	When enabled, the query will utilize a polling engine that is shared across all running Perspective sessions. This optimization is helpful for consolidating multiple identical polling tasks, particularly when you have many Perspective sessions running at once. The polling engine will run the query once, cache the value, and then 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. Disabling this option will bypass the cache.
Designer Limit	Forces the results of the query to be limited to a few rows when run in the Designer.
Polling	Sets the polling mode of the Named Query binding based on the polling rate, in seconds.

Tag History Bindings in Perspective



INDUCTIVE
UNIVERSITY

Tag History Binding

[Watch the video](#)

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.

✗ Edit Binding: Sensor.props.value.text

✕

Binding Type

Tag

Property

fx Expression

fx Expression Structure

Query

Tag History

HTTP

Remove Binding

Configure Tag History Binding

Return Format

☒ Wide

☐ Tall

☐ Calculations

Query Mode

Point Count

PointCount

100

fx

Time Range

Start Date

End Date

Historical

dateArithmetic(now(0), -1, fx)

now(0)

fx

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

Add Transform +

☒ Binding Preview

Tag History

Dataset[192 rows, 2 cols]

OK

Cancel

Apply

Configuration Property	Description
Return Format	Allows you to select the return format of the data. • Wide: Every column is a different tag, and each row is their values at different times. • Tall: There are columns for Value, Quality, Timestamp, and Path, and each row is a new tag value at a specific time. • Calculations: Will perform a calculation on the returned data, and return the calculated values instead. For example, using the Average calculation will generate an average of each tags value over the time range selected.
Query Mode	Determines how data is queried. • PointCount: Will return the number of records defined in the Point Count property. Note that setting this property to -1 will retrieve pen data points as they are stored in the database. In other words, this is the same as setting the Query Mode to AsStored. • Periodic: Will return records separated by an amount of time specified in the Period property. • AsStored: Will return the records as stored in the database. While querying data with this mode, multiple value changes at the same timestamp will result in multiple rows, one row for each unique value.
Time Range	The time range to pull data values from. See Time Range .
Select Tags	An area to select the tags to trend. Tag paths can be defined directly or using an expression. See Using Dynamic Tag Paths below about Expression mode. Users may browse for and select tags from both Realtime and Historical Tag Providers.

Configuration Property	Description						
	Select Tags Direct Expression <table><tr><th>Tag Path</th><th>Alias</th><th>Aggregation Mode</th></tr><tr><td colspan="3"></td></tr></table> Default Aggregation Mode MinMax Select Tag Path Realtime Providers - Tags - default - Sample_Tags - System Historical Providers - Sample_SQLite_Database	Tag Path	Alias	Aggregation Mode			
Tag Path	Alias	Aggregation Mode					
Aggregation Mode	The aggregation mode that will be applied to each time slice, unless a more specific aggregation mode is defined on a tag path. For more information on how time slices are populated, see Aggregation Mode .						
Default Aggregation Mode	Aggregation mode to use as a default if the Select Tags are set to Default Aggregation mode.						
Options	Allows you to specify various options that will apply to the binding. See Options .						

Time Range

Property	Description
Realtime	The start date will go back as far from the current time as the Most Recent property specified and the end date will be the current time when the binding 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 <i>fx</i> icon to use operators, expressions, tags, or properties. Time Range Realtime Most Recent 1 <i>fx</i> HOUR Polling ☐ Polling <i>fx</i> sec
Historical	You can specify the Start and End Date in an expression. Click the Functions <i>fx</i> icon to use operators, expressions, tags, or properties. No polling; times only changed if bound to something that changes. Time Range Historical Start Date dateArithmetic(now(), -1, <i>fx</i> End Date now() <i>fx</i> Caution: Please note that intervals returned by Historical queries are inclusive of the End Date, including when the End Date is set to now(). This means you may see one additional interval than expected that only contains future dates, which get interpolated to 0 and can cause trending issues. For example, if you want data from 10am - 11am in 1 minute windows, you'll need to set your query from 10am-10:59am. Querying to 11am would create an interval to contain it and that window will often return 0 since there is typically no future value. Additionally, if you went on to add the results of two queries of adjoining times, such as 10am - 11am and 11am - 12pm, the first window of the second period would have duplicate data to the last window of the first period.

Aggregation Mode

Aggregation Mode	Description
(default)	Use the mode set in the Default Aggregation Mode field.
Average	The values are averaged together, weighted for the amount of time they cover within a time slice.
MinMax	The minimum and maximum values will be returned for the window. In other words, two rows will be returned. If only one value is seen in the time slice, only one row will be returned.
LastValue	The value closest to the ending time of the time slice will be returned.
SimpleAverage	The values are summed together and divided by the number of values.
Sum	The values in the time slice are summed together.
Minimum	The minimum value in the time slice.
Maximum	The maximum value in the time slice.
DurationOn	Returns the total time, in seconds, that the value was recorded as non-zero.
DurationOff	Returns the total time, in seconds, that the value was recorded as zero.
CountOn	Returns the number of times the tag's value went from a zero value to non-zero.
CountOff	Returns the number of times the tag's value went from a non-zero value to zero.
Count	Returns the number of times a value was recorded.

Aggregation Mode	Description
Range	Returns the range between the highest and lowest value for the period. Range mode will return "0" if the historical tag value remains static over the given Time Range.
Variance	Returns the variance of values. Similar in concept to standard deviation. Only good quality values are used when calculating.
StdDev	Returns the standard deviation of values, or how much spread is present in the data. Low standard deviation shows that values are close to the mean, and high standard deviation shows that the data points are spread out over a large range of values. Only good quality values are used when calculating.
PctGood	Time-weighted percentage of good values over the date range.
PctBad	Time-weighted percentage of bad values over the date range.

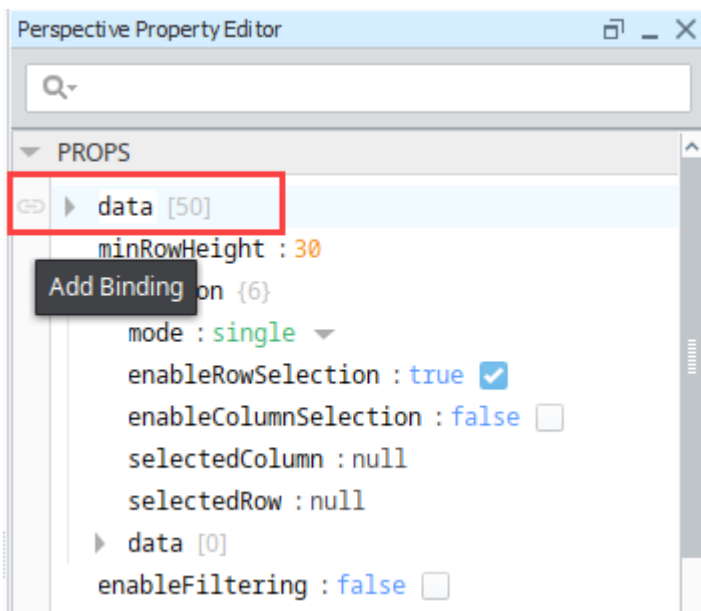
Options

Option	Description
Enabled	Enable these options.
Overlay Opt-Out	Opt out of displaying the Tag quality overlay .
Ignore Bad Quality	Only data with "good" quality will be loaded from the data source.
Prevent Interpolation	Requests that values not be interpolated, if the row would normally require it. Also instructs the system to not write result rows that would only contain interpolated values. In other words, if the raw data does not provide any new values for a certain window, that window will not be included in the result dataset.
Cache & Share	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.
Value Format	Can be Dataset or Document.

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.

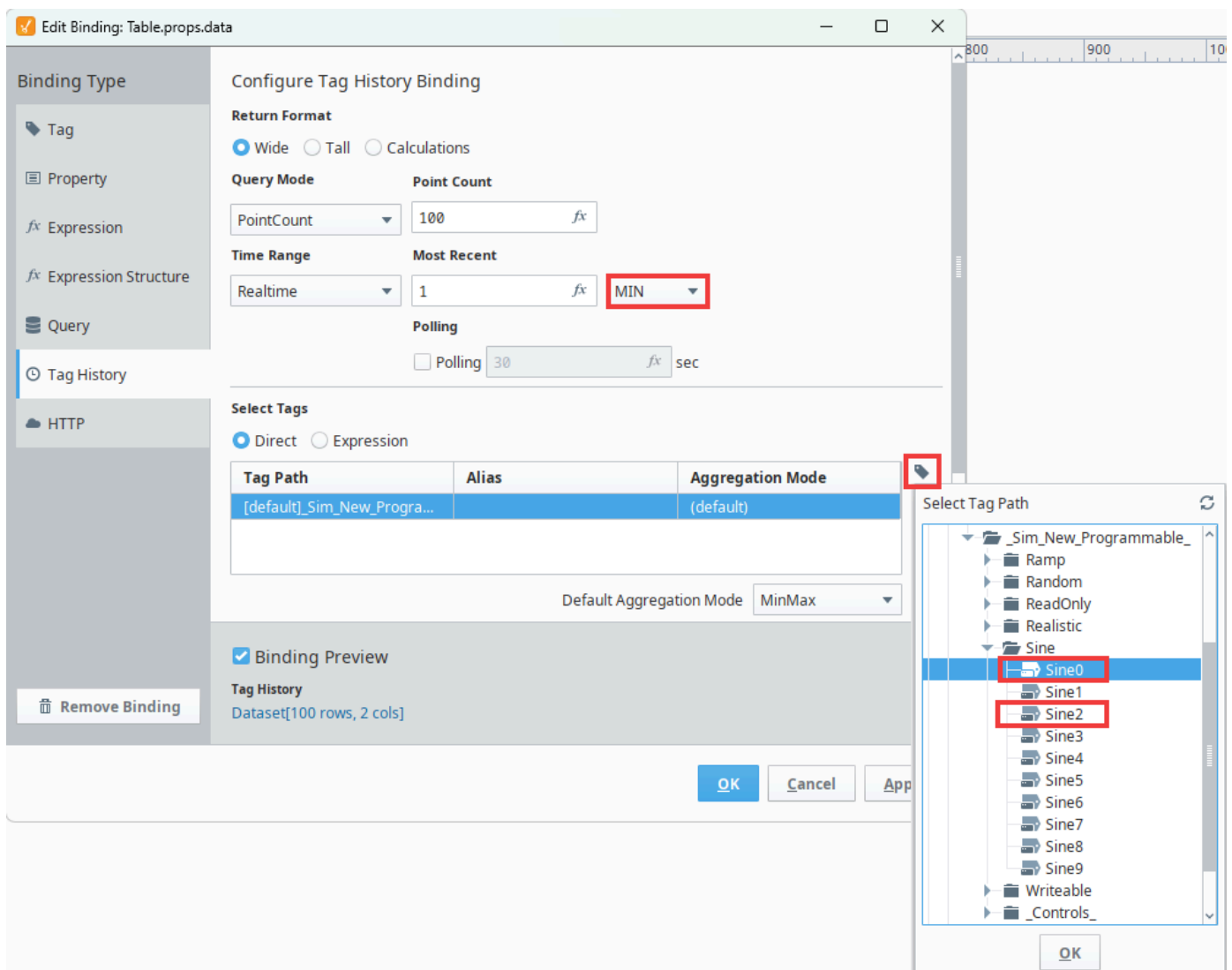
1. Drag a Table component onto a view. Select the Table component.
2. In the Property Editor, click on the **Binding**  icon next to the **data** property. The Edit Binding screen is displayed.



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.



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.

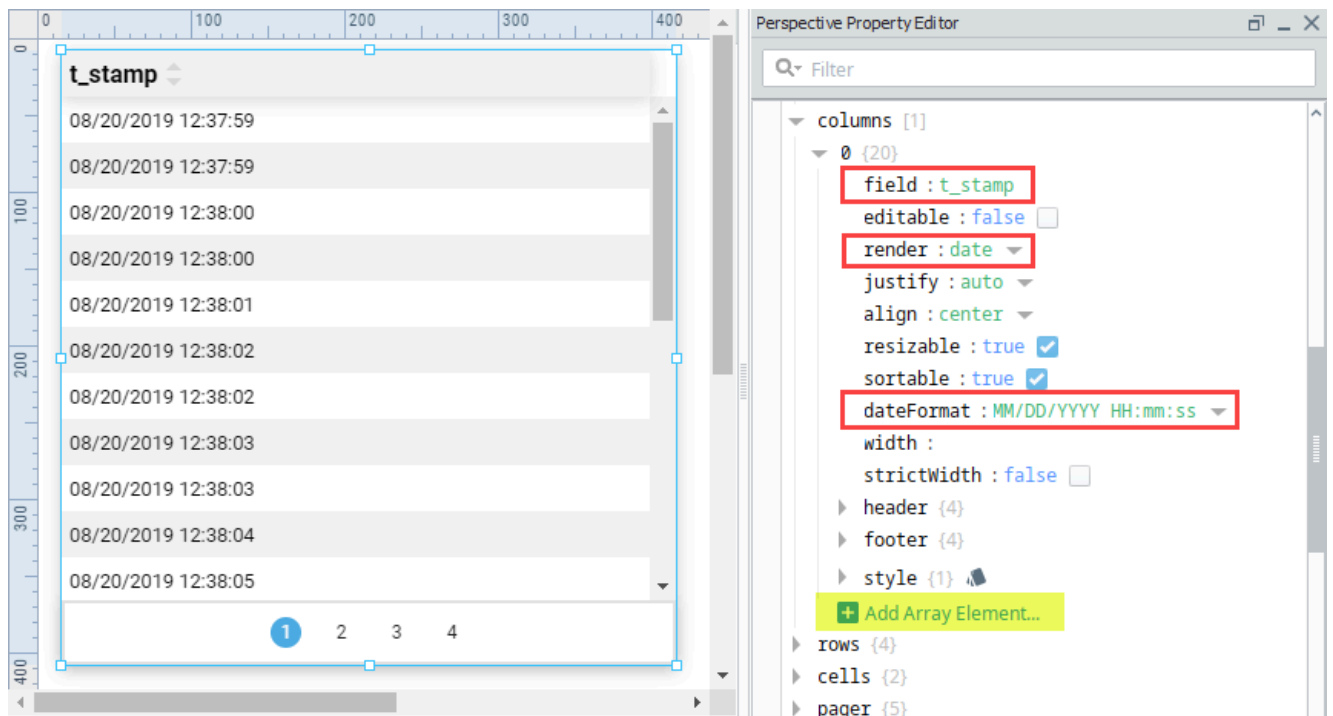


t_stamp	Sine0	Sine2
1,566,328,579,173	87.67	47.31
1,566,328,579,773	86.16	46.20
1,566,328,580,373	81.03	39.78
1,566,328,580,973	80.37	38.96
1,566,328,581,573	73.70	29.30
1,566,328,582,173	68.59	22.19
1,566,328,582,773	66.22	18.89
1,566,328,583,373	58.98	10.66
1,566,328,583,973	58	9.55
1,566,328,584,573	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....**

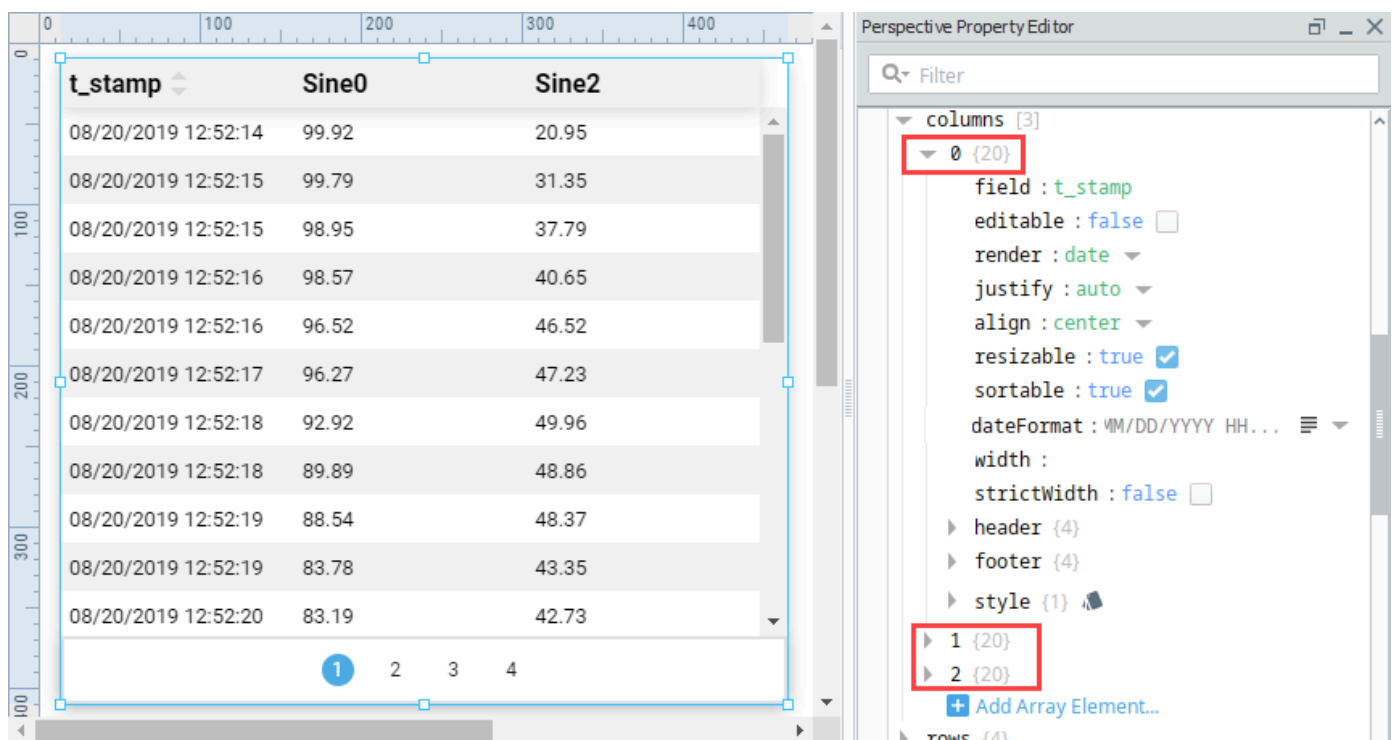


9. You'll notice that once the Array Element is added, the other two columns (Sine0 and Sine2) disappeared. We'll add them back in Step 10.
- In the **field** property, enter **t_stamp** (which is the column name).
 - For the **render** property, select **date** from the dropdown.
 - Select a **dateFormat** from the dropdown. In this example, we used the date time format.



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.



Tag History Binding - Expression



INDUCTIVE
UNIVERSITY

Tag History Binding - Expression

[Watch the video](#)

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
 - ...

NOTE

The **aggregate** in your nested Tag History object is optional. If not specified, it will use the default mode for direct Tag History bindings and use the aggregation mode set for the overall binding.

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



INDUCTIVE
UNIVERSITY

HTTP Binding

[Watch the video](#)

The HTTP Binding type allows you to use HTTP get/post protocols to interface with other API's.

Edit Binding: Sensor.props.value.text

Binding Type

Tag

Property

fx Expression

fx Expression Structure

Query

Tag History

HTTP

Remove Binding

Configure HTTP Binding

URL

"https://postman-echo.com/get?foo1=bar1&foo2=bar2"*fx*

Method

GET

Headers

Key	Value
-----	-------

Body

fx

Authentication Type

None

Authentication Value

fx

Connect Timeout

60

sec

Socket Timeout

60

sec

Options

☒ Enabled

☐ Overlay Opt-Out

☒ Allow Cookies

☒ Cache & Share

☐ Polling

30

fx

sec

Add Transform +

☒ Binding Preview

HTTP

■ {"args":{"foo1":"bar1","foo...

OK

Cancel

Apply

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

Binding Property	Description
URL	An expression indicating what web address to reference on the binding. If entering a static URL, quotation marks must be used.
Method	Any HTTP method. Used to send HTTP requests to the specified URL. See Method .
Headers	Used to pass key/value pairs in the header of our HTTP requests. See Headers .
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.

Binding Property	Description
Allow Cookies	Indicates whether to allow the remote web server to store cookies.
Cache & Share	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.

Method

Method	Definition
GET	The GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URI.
HEAD	The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response.
POST	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.
PUT	The PUT method requests that the enclosed entity be stored under the supplied Request-URI.
DELETE	The DELETE method requests that the origin server delete the resource identified by the Request-URI.
TRACE	The TRACE method is used to invoke a remote, application-layer loop-back of the request message.
CONNECT	The CONNECT method starts two-way communications with the requested resource. It can be used to open a tunnel.

Headers

Field	Definition
Key	Allows dropdown selection from common header keys, or the ability to specify a custom one.
Value	The Value field is an expression.

MongoDB Bindings in Perspective

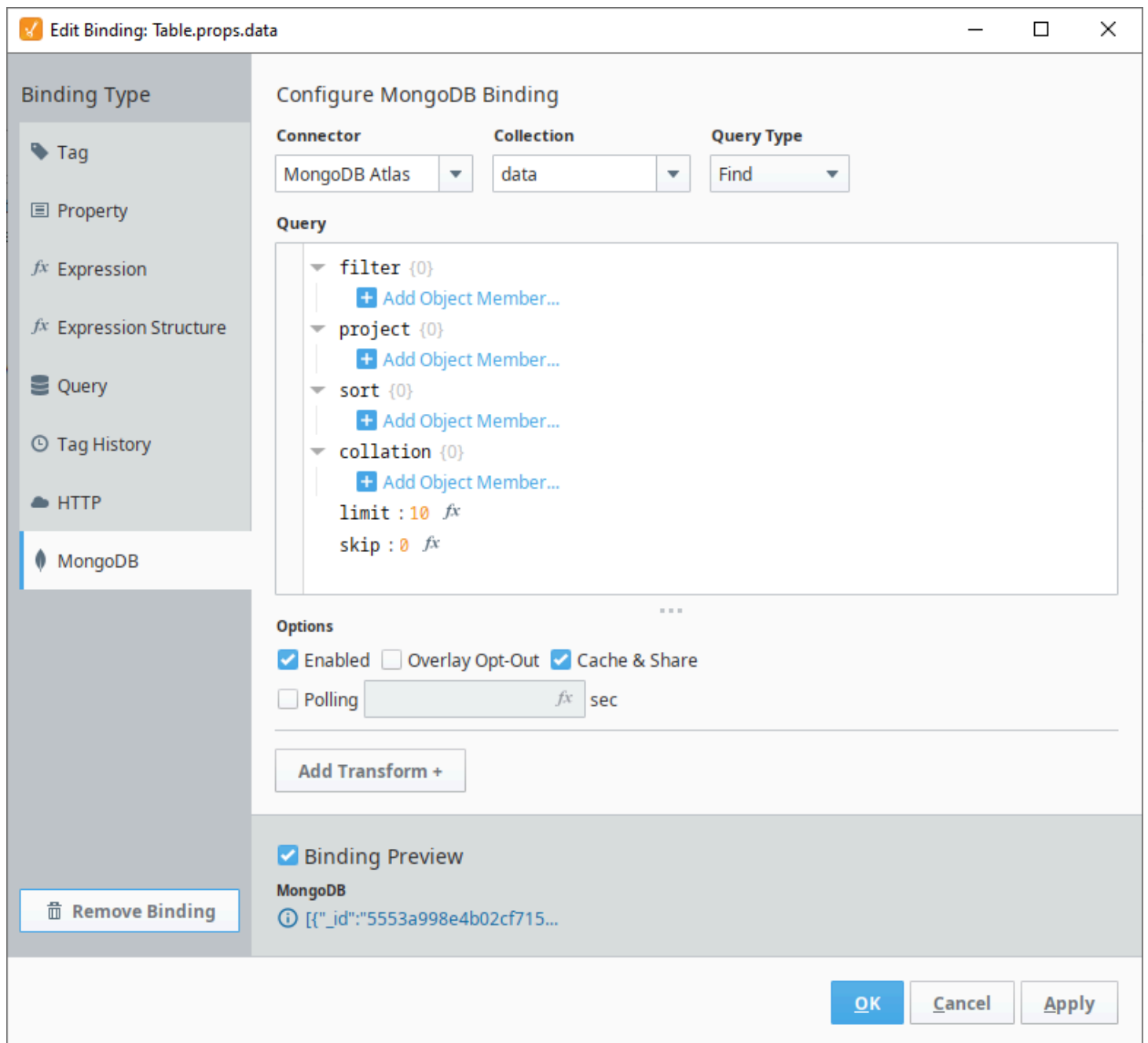
The MongoDB Cloud Connector module introduces a Perspective Binding type called MongoDB. The MongoDB binding follows the same format as the [Perspective Expression Structure binding](#) by allowing users to bind JSON data to individual aspects of a Perspective project such as properties. In other words, 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.

MongoDB Binding Interface

The image and table below detail various elements of the MongoDB binding user interface.



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 fx

skip : 0 fx

- 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.

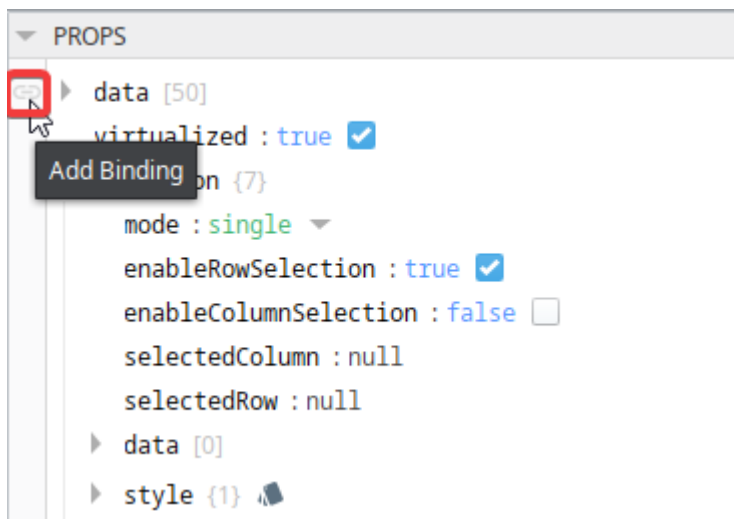
MongoDB Expression Functions

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.

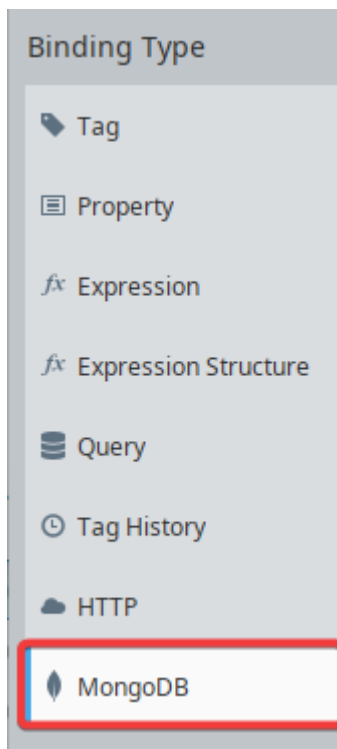
Example

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.



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**.

Edit Binding: Table.props.data

Binding Type

Tag

Property

Expression

Expression Structure

Query

Tag History

HTTP

MongoDB

Remove Binding

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 fx

skip : 0 fx

Options

☒ Enabled

☐ Overlay Opt-Out

☒ Cache & Share

☐ Polling

fx

sec

Add Transform +

☒ Binding Preview

MongoDB

info

 [{"_id": "621ff30d2a3e781873..."}]

OK

Cancel

Apply

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}

25 rows
1

NOTE

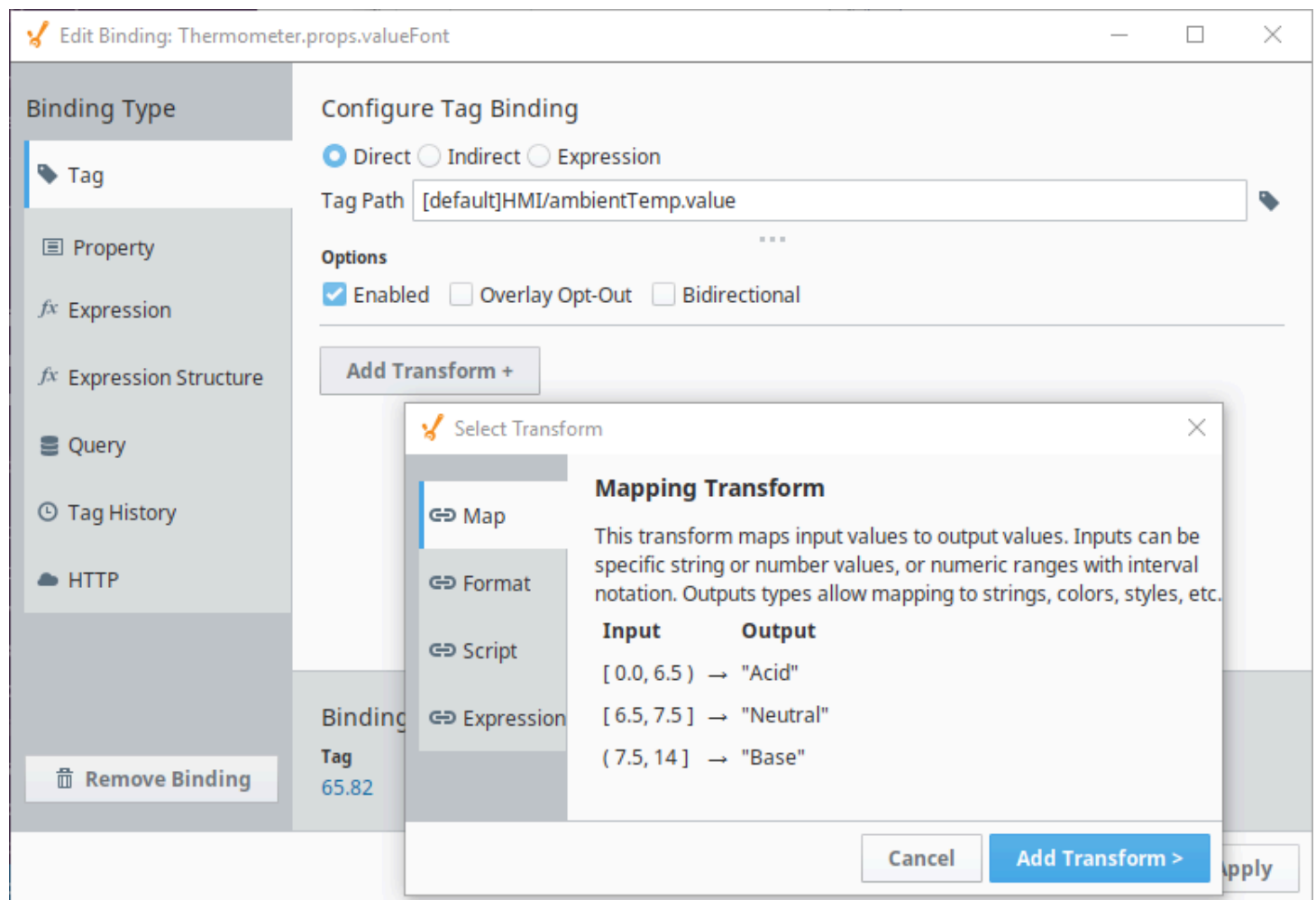
You may need to add the column names from your target collection in order for users to be able to perform actions such as column resizing or reordering. See the column row from the [Properties section](#) of the Perspective Table User Manual page for more details.

Transforms

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.



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.

Map Transform



INDUCTIVE
UNIVERSITY

Map Transform

[Watch the video](#)

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.

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 none 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

Edit Binding: Tag_Direct.props.value

Binding Type

Tag

Property

fx Expression

fx Expression Structure

Query

Tag History

HTTP

Remove Binding

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

Add Transform +

Binding Preview

Property

0

Map

99

OK

Cancel

Apply

Numeric Range to String Mapping - Range Values Omitted

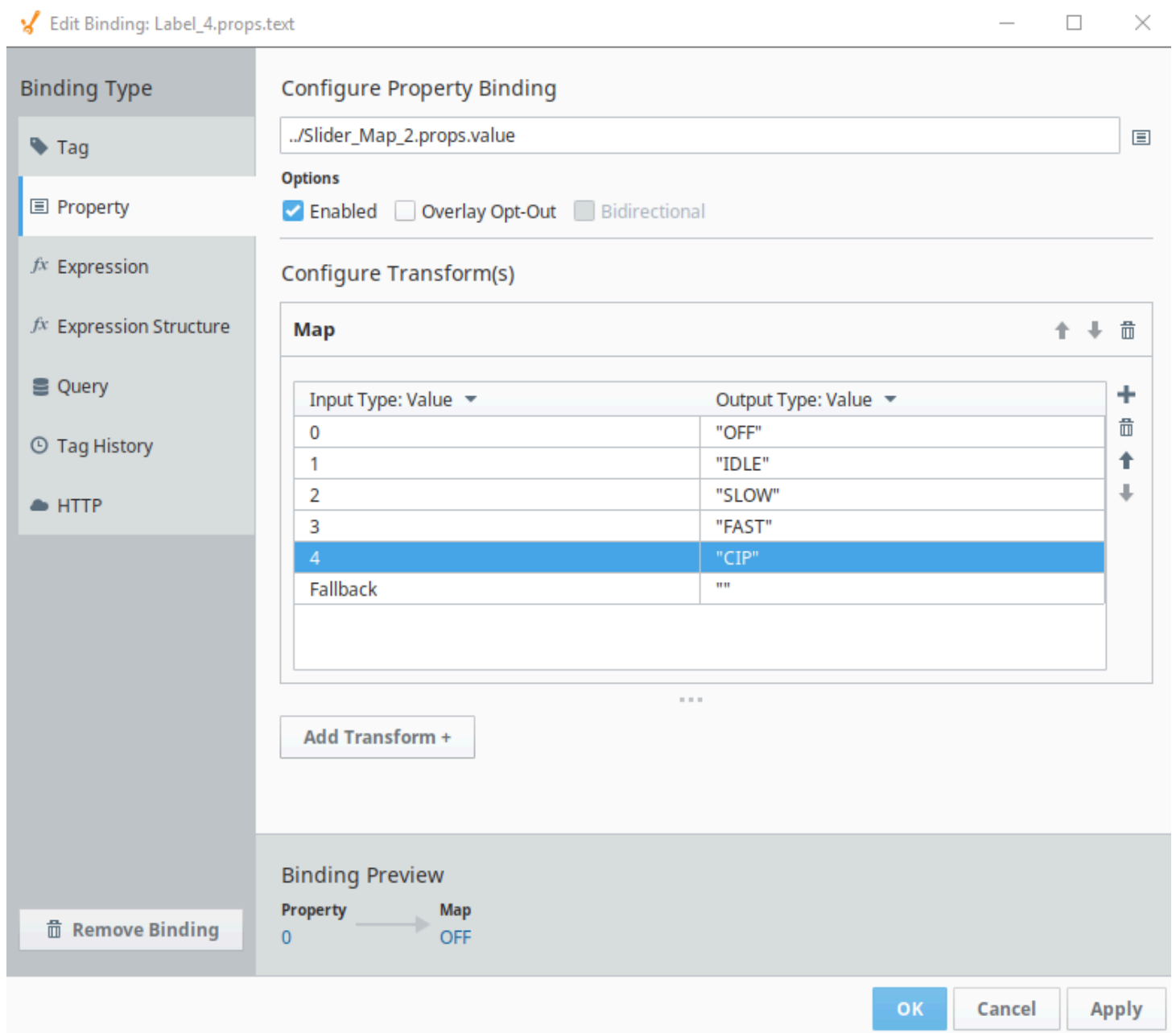
(,25]	"Below 25"
(25,)	"Above 25"
Fallback	"Invalid Value"

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".



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.

Edit Binding: Label_4_0.props.style.backgroundColor

Binding Type

Tag

Property

Expression

Expression Structure

Query

Tag History

HTTP

Remove Binding

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

Add Transform +

Binding Preview

Property

15

Map

#000000

OK

Cancel

Apply

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.

Binding Type

Tag

Property

Expression

Expression Structure

Query

Tag History

HTTP

Remove Binding

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')

Add Transform +

Binding Preview

Property

11 →

Map

51

OK

Cancel

Apply

Format Transform



INDUCTIVE
UNIVERSITY

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. The return type for all Format Transforms is a string.

When configuring a Format Transform, you'll see different settings based on the Format Type selected:

- **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 based on the Format Pattern entry. The option is helpful when wanting to add leading zeros on a display.
 - **Format Pattern:** Uses `0` and `#` to set the pattern.
- **Integer:** Rounds input (float) to integer.
- **Number:** Formats number based on Locale setting, and will then place the commas or other number separators as applicable.
- **Percent:** Converts the input number into percent. For example, 0.123 would format to 12.3%.
- **Currency:** Uses the currency specified by the Locale 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 Session. Use the dropdown to select a different language.

Time Zone (Datetime only)

- **auto:** Default time zone of Session. Use the dropdown to select a different time zone.

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 **-35.893925**.

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 **-35.89**, since any digits beyond the first two decimal places are ignored by the transform.

Edit Binding: Label.props.text

Binding Type

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

Configure Tag Binding

☒ Direct ☐ Indirect ☐ Expression

Tag Path [Sample_Tags]Sine/Sine0

Options

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

Fallback Delay 2.5

Configure Transform(s)

Format

Select Format Type

☐ Datetime ☒ Numeric

Format **Format Pattern**

Pattern #.00

Locale

auto

Add Transform +

☒ Binding Preview

Tag -35.893925 → Format -35.89

Remove Binding

OK Cancel Apply

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

fx Expression

fx 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

DateTime

Date

Short

Time

Short

Locale

auto

Time Zone

Japan

Add Transform +

Binding Preview

Property

1551823366022

Format

3/6/19, 7:02 AM

OK

Cancel

Apply

Script Transform



INDUCTIVE
UNIVERSITY

Script Transform

[Watch the video](#)

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.

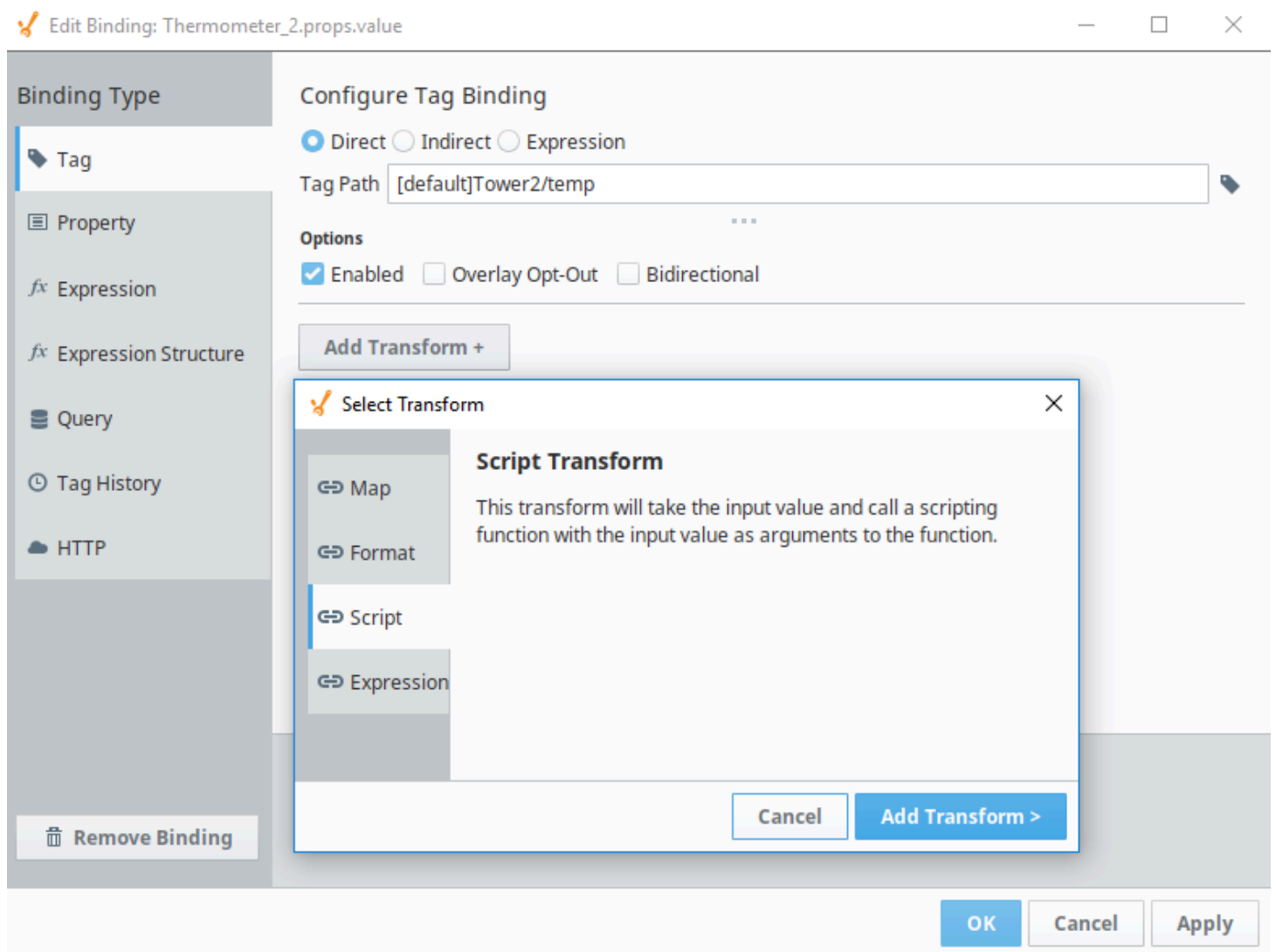
When a script transform is first created it generates a template script with a few assumed input Arguments (see below). From this template you can write your own script and reference an incoming value as well as a few other parameters. It is assumed the custom script will end with a Return comment that 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.



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

```
# Get the header names directly from the dataset
header = value.getColumnNames()

# Create a blank List so we can append later
newList = []

# Step through the rows in the dataset
for row in value:
    # 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 the 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"]

# Create a blank list so we can append later
newList = []

# Step through the rows in the dataset
for row in value:
    # 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 the 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.

The screenshot shows the InfluxDB UI configuration for a Sparkline chart. On the left, the 'Binding Type' is set to 'Tag History'. The 'Options' section on the right shows 'Enabled' checked, and 'Value Format' is set to 'DATASET'. The 'Script' section contains the following Python code:

```

1  """
2
3  # convert the incoming value data
4  pyData = system.dataset.toPyDataSet(value)
5  # create a blank list so we can append later
6  newList = []
7
8  # step through the rows
9  for row in pyData:
10     # append the dictionary to list
11     if row[1] is not None:
12         newList.append(row[1])
13
14  # return the results

```

Tag History to List of Data Points

```
# Create a blank List for collected values
newList = []

# Step through the rows in the dataset
for row in value:
    if row[1] is not None:
        newList.append(row[1])

# Return the results
return newList
```

Expression Transform



INDUCTIVE
UNIVERSITY

Expression Transform

[Watch the video](#)

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

Edit Binding: Label_0_0_0_0_0.props.text

Binding Type

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

Configure Property Binding

../Slider.props.value

Options

☒ Enabled ☐ Overlay Opt-Out ☐ Bidirectional

Configure Transform(s)

Expression

```
1 //Convert an intger to Hexadecimal and put in leading 0's
2 switch(
3   len(toHex({value})),           // determine length of string
4   0,1,2,3,                       // possible lengths
5   'n/a',                         // 0 - results to display
6   concat('000', toHex({value})), // 1 - results to display
7   concat('00', toHex({value})),  // 2 - results to display
8   concat('0', toHex({value})),   // 3 - results to display
9   toHex({value})                 // Failover
10 )
```

Add Transform +

Binding Preview

Property	Expression
419	01A3

OK Cancel Apply

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
)
```

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.	<pre>// Absolute path. Sequential slashes allow // for movement into a container /root/LabelA.position.x /root/Sub_Container1/ButtonA.position.y</pre>
.	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	<pre>/root/LabelA.meta.foo /root/LabelA.meta.rotate.angle</pre>

Operator/Keyword	Description	Example
	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.	
[]	Brackets - When referencing an array property, brackets allow you to specify an individual index within the array.	<pre>/root/LabelA.props.complex.bar[5]</pre>
../	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	<pre>// From ButtonA, we can use this operator // quickly move to a sibling component ../ButtonB.position.x</pre> Move Up Multiple Parent Containers <pre>// Moving up multiple parent containers ../../LabelA.position.x</pre> // Also moves up: each additional dot is // another parent container .../LabelA.position.x

Operator/Keyword	Description	Example
	dots to move up levels. Each additional dot moves up another level.	
./	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	parent.props.complex.foo or parent.position

Operator/Keyword	Description	Example
	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 "./"	

Operator/Keyword	Description	Example
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	

Operator/Keyword	Description	Example
	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



INDUCTIVE
UNIVERSITY

Events and Actions

[Watch the video](#)

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:

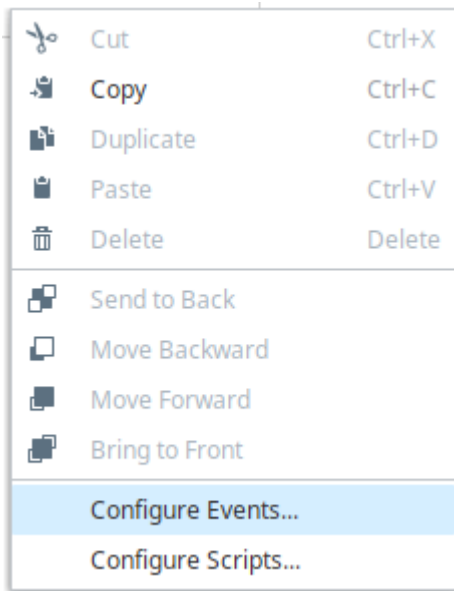
- Navigating to a new page when the user presses a button.
- Opening a popup containing details on a specific PLC when a user double-clicks on it in a diagram.
- Logging the user out of the Session when they press Ctrl + L on the keyboard.
- Scanning a barcode from a mobile device, and sending it to the Gateway.

In this section, we'll cover the basics on how to configure actions and events to suit your project's needs.

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**.

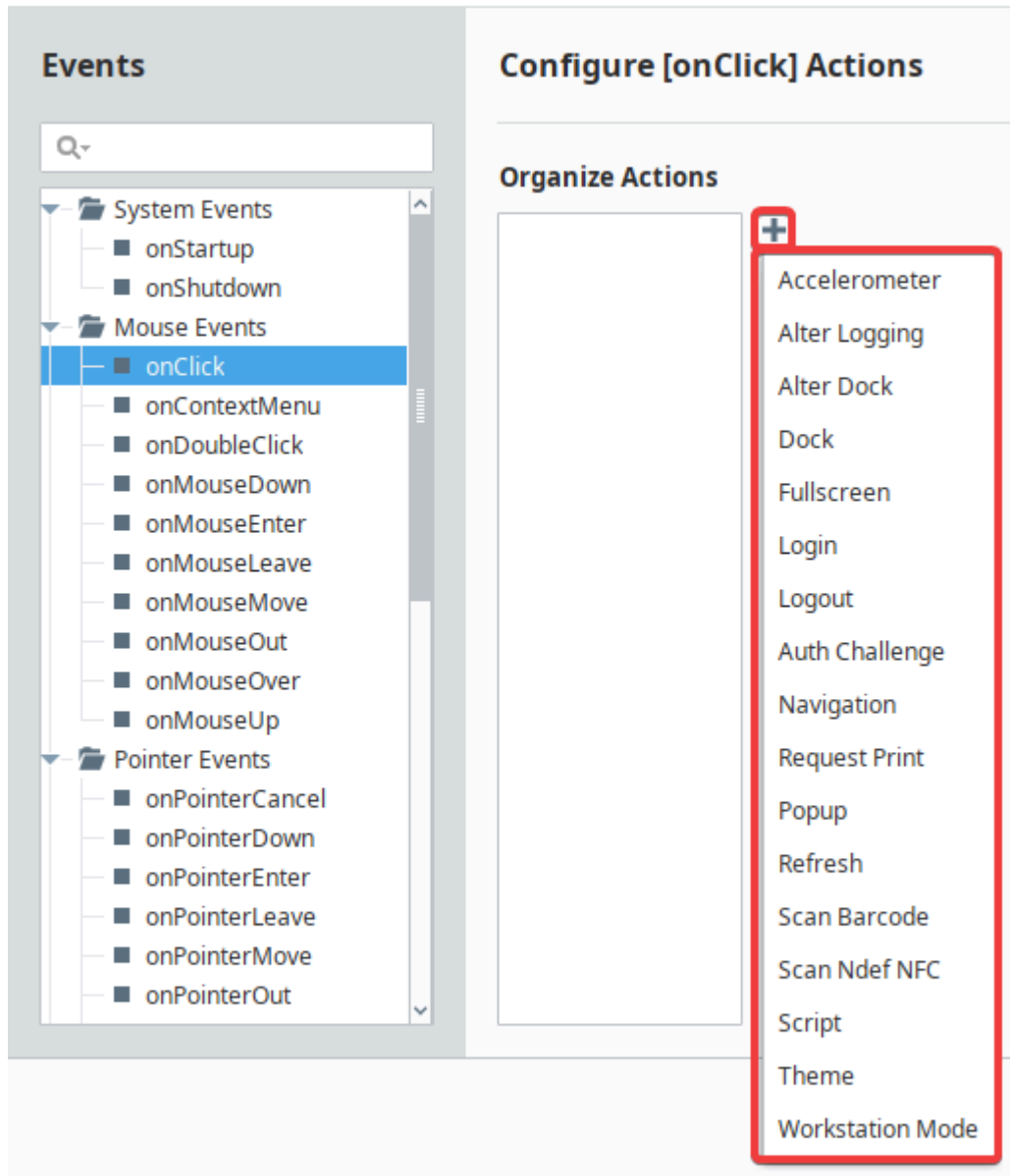


3. Choose an event from the left-hand side.

i 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**.

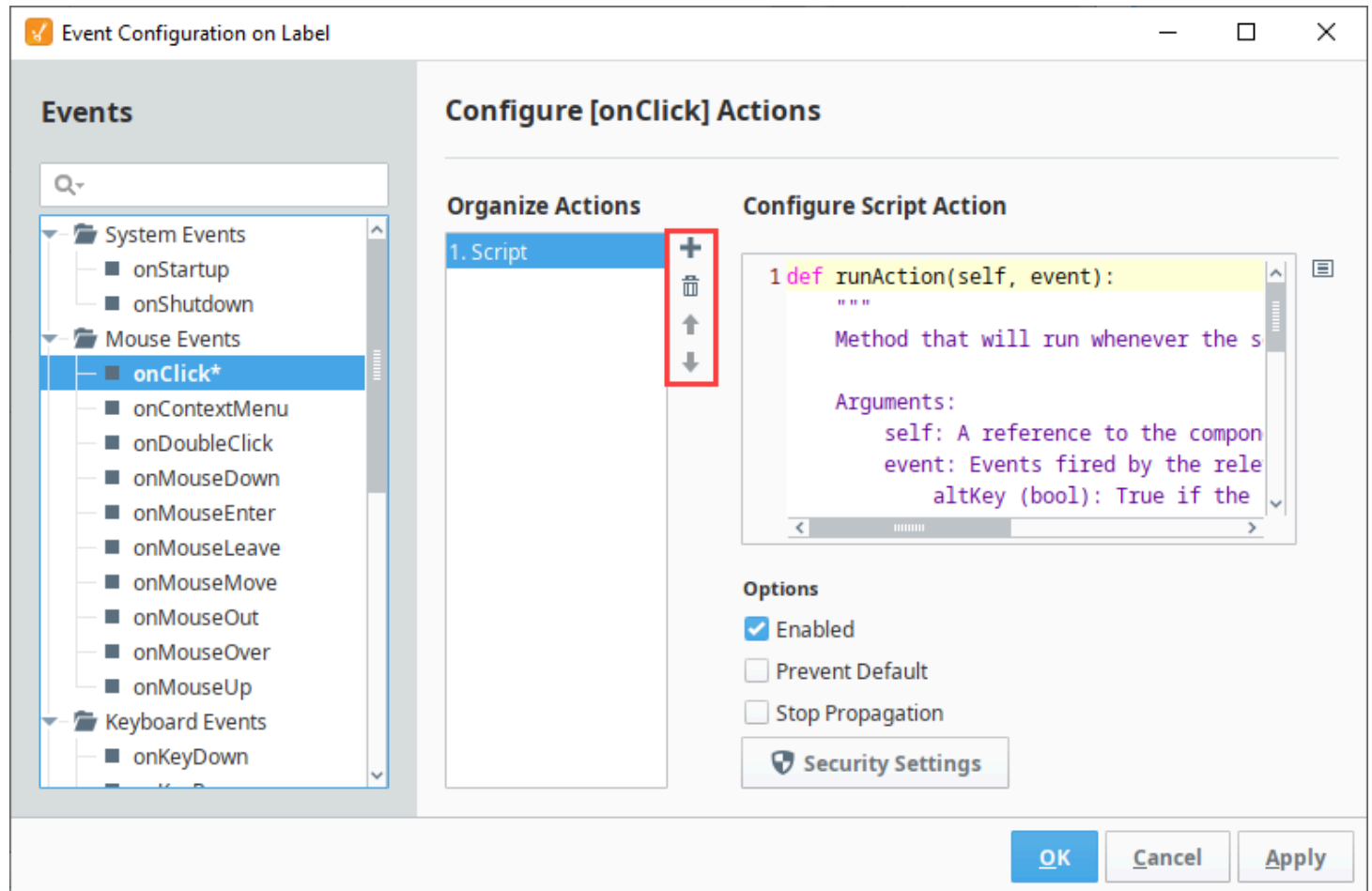
Perspective Events

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.

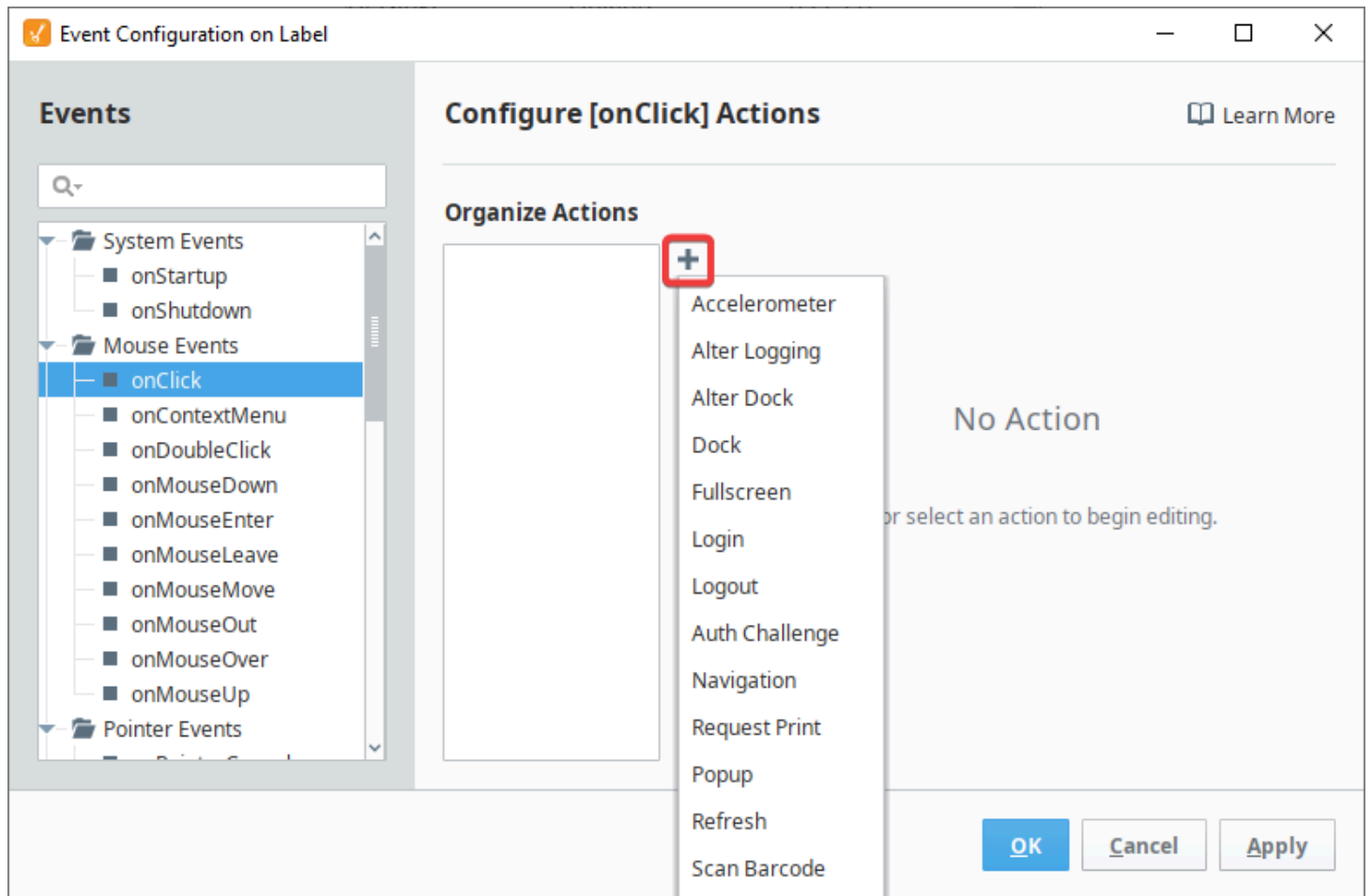
Action Types

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.



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.

Option	Description
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 <pre>1 def runAction(self, event): """ Fired after the event has been deleted Arguments: self: A reference to the component that is invoking this function event: An object with the following attributes: end (int float): The end date of the deleted event. """</pre> 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  Security Settings

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 Perspective 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 logs. Note: For all messages to be visible, the Perspective.Client logger must be set to the same logging level as the level indicated in this action. You can change this logger's level on the Gateway's Diagnostics > Logs page by clicking the settings icon and searching for the logger.
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

Allows you to alter the configuration of a docked view. Any action settings left blank will remain unchanged.

Action Setting	Description
View	The currently selected view. Changing this will change which view is mounted to this position.
Display	This property allows you to show or hide the docked view. • visible: The docked view is always expanded/displayed. • onDemand: The docked view is collapsed, but allows the user to display the view by clicking on the docked view's handle. • auto: 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).
Resizable?	Determines whether the docked view may be resized or not.
Modal?	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 Display property is set to onDemand .
Content	Determines how the docked view interacts with other views on the page. • push: Opening or closing the docked view causes the content in the center to resize: the center view will be 'pushed' out of the way. • cover: 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. • auto: 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.
Anchor	Allows you to make a view always visible while scrolling. Only available on North docked view configurations. • fixed: 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. • scrollable: 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

Action Setting	Description
	move along with the page as the user scroll down.
Size	Determines the size, in pixels, of the view. • If the view is docked to the North or South edge, then size determines the height. • If the view is docked to the East or West edge, then size determines the width.
Auto Breakpoint	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 Display property is set to auto.
Dock ID	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.
Handle	Allows you to show or hide a handle for users to expand/collapse the view. • Show: Show handle at all times. • Hide: Hide handle at all times. • AutoHide: Hide handle when page is not active.
Handle Icon	Path to an icon used to identify the view when the view is hidden.
View Parameters	Allows specific parameter values to be passed to the docked view when navigating to the page.

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 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 [onAuthChallengeCompleted](#) 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.

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.

USING A SECONDARY IDP

A separate IdP tailored specifically for the secondary users' authentication challenges may be specified under the **Identity Provider** action setting. So long as both this special IdP is only used for these secondary user authentication challenges (and not used anywhere else), and 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 Session event.
Framing	How the challenge should be presented to the user's current page: • Same Tab / Window: User is navigated away from the Session and to the IdP in the same window/tab. Once authentication is complete at the IdP, they are redirected back to the Session. • New Tab / Window: A new tab is opened in the user's browser, leaving the current Session tab open while it waits for the new tab to complete the authentication challenge at the IdP. • Embedded Frame: The redirect to the IdP is embedded in an Iframe within the current Session's page. Some IDPs may require Content-Security-Policy or other configuration changes to display their login page in the Session's Iframe. Note: Perspective Workstation Sessions do not support New Tab / Window and will fall back to Same Tab / Window. Mobile Sessions only support Same Tab / Window.

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:

Type	Description
Page	Navigates to a separate page. • Set Page: A string denoting the page URL of the target page. See Page URLs. • Open in new tab: Specifies whether the newly opened page should replace the current page, or open in a new browser tab.
View	Replaces the current main View with a new main View. See View .
Url	Navigates to an external web address. See Url .
Browser	Navigates either forward or backward using browser history. See Browser .

View

Setting	Description
Select View	The path to the view that should be used.
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

Url

Setting	Description
Enter Url	The URL that the action should navigate to. Example: <code>https://inductiveautomation.com/</code>
Open in new tab:	Specifies whether the newly opened page should replace the current page, or open in a new browser tab.

Browser

Setting	Description
Navigate	Specifies whether to go forward or backwards in browser history when the action is triggered.
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.

Request Print Action

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.

Action Setting	Description
	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	If selected, 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 Perspective 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.

 CAUTION

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	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.

⚠ CAUTION

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.

Event Configuration on Label_0

Events

- System Events
 - onStartup
 - onShutdown
- Mouse Events
 - onClick***
 - onContextMenu
 - onDoubleClick
 - onMouseDown
 - onMouseEnter
 - onMouseLeave
 - onMouseMove
 - onMouseOut
 - onMouseOver
 - onMouseUp
- Keyboard Events
 - onKeyDown
 - onKeyPress
 - onKeyUp
- Text Composition Events
 - onCompositionEnd
 - onCompositionStart
 - onCompositionUpdate
- Focus Events

Configure [onClick] Actions

Organize Actions

- 1. Script

Configure Script Action

```
1 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 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 event was fired.  
        buttons (int | float): The buttons being depressed when  
            the event was fired.  
        clientX (int | float): The X coordinate in local coordinates  
        clientY (int | float): The Y coordinate in local coordinates  
        ctrlKey (bool): True if the 'ctrl' key was held down when  
            the event was fired.  
    """
```

Options

- ☒ Enabled
- ☐ Prevent Default
- ☐ Stop Propagation

[Security Settings](#)

OK Cancel Apply

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

 NOTE

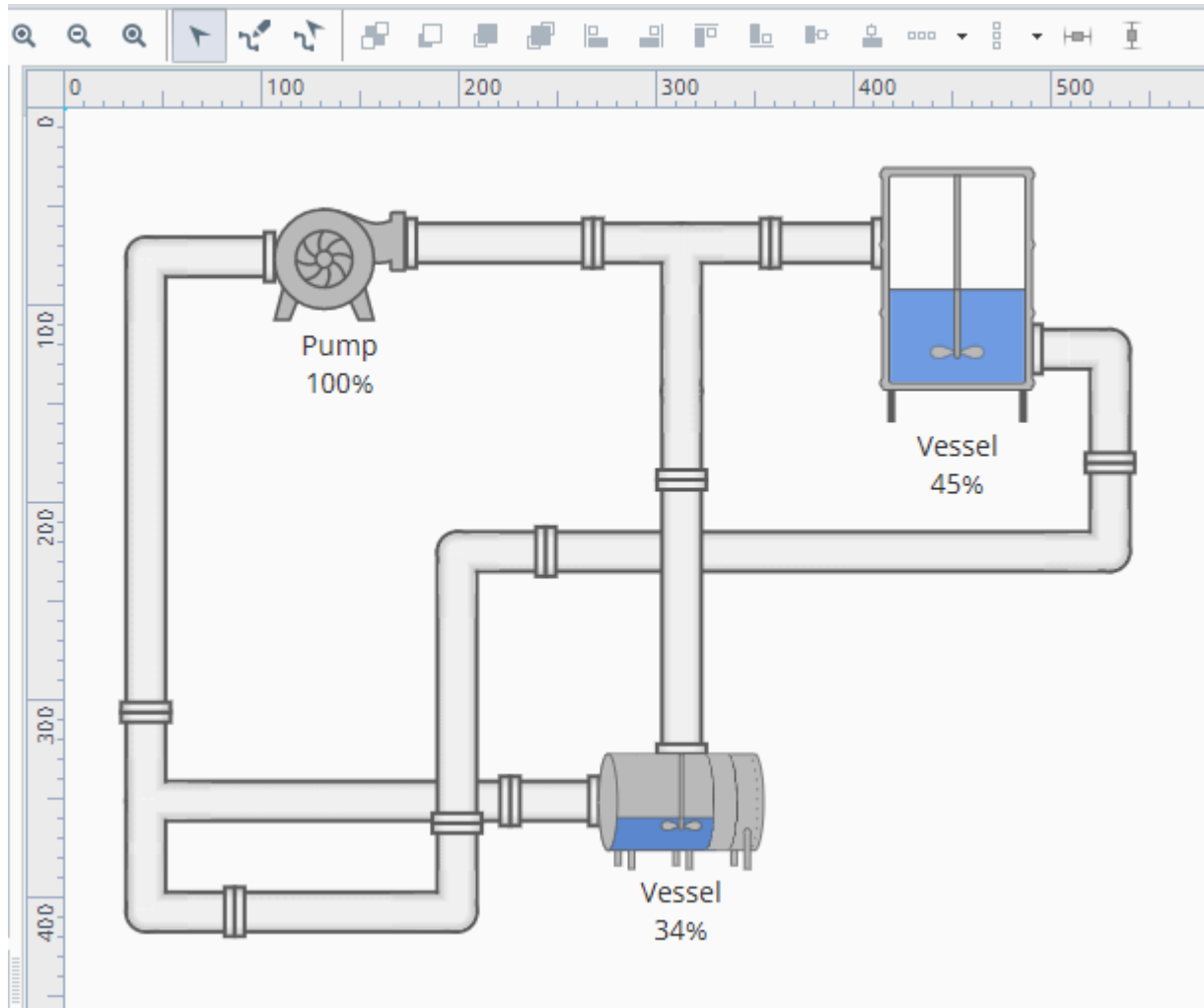
This action will only take effect if called from a Session running in Workstation.

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.

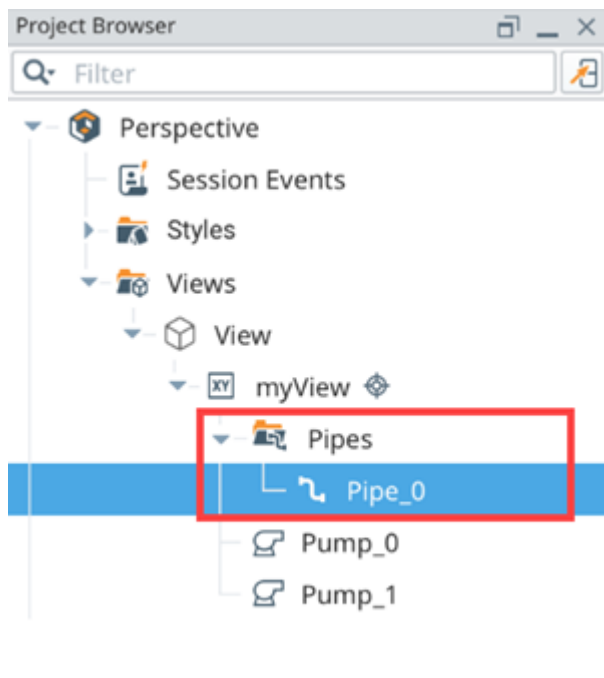
Perspective Pipes

Pipes can be drawn within a coordinate container. These pipes can be used to visualize flow between separate objects on a view.

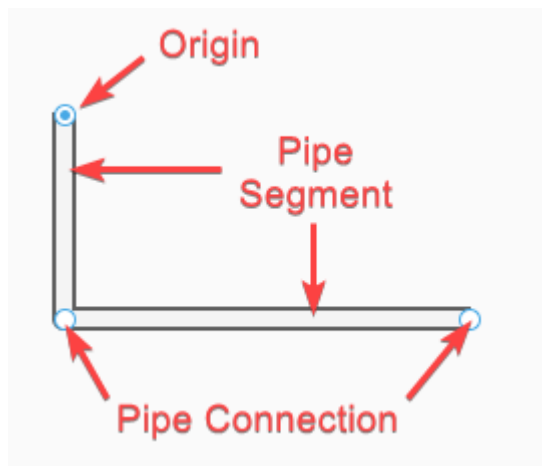


Pipes are created by a dedicated **drawing tool** on the toolbar, as opposed to being dragged onto a container like other components. Pipes are effectively drawn on the bottom of the z-order in a coordinate container, hence any components added into the container will always be placed above the pipes.

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.

Pipe Connection

An XY coordinate within the pipe. Visualized as a circle.

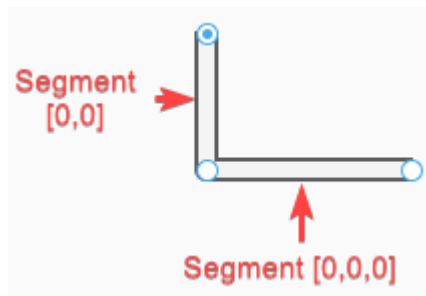
Origin

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.

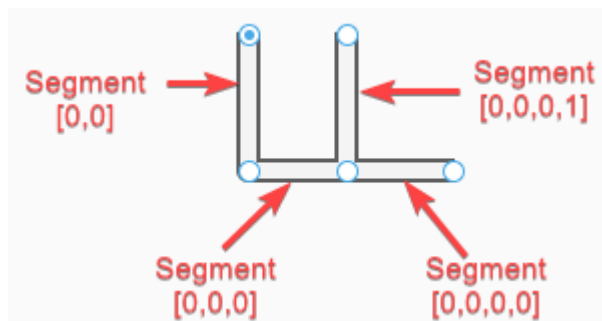
Pipe Segment

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
                }
              ]
            }
          ]
        }
      ]
    }
  }
]
```

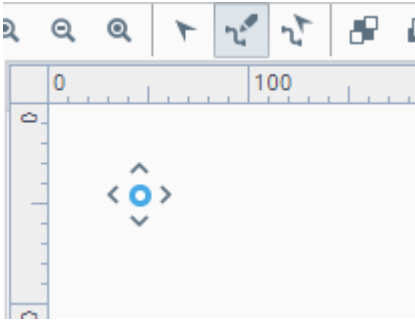
Creating Pipes

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.

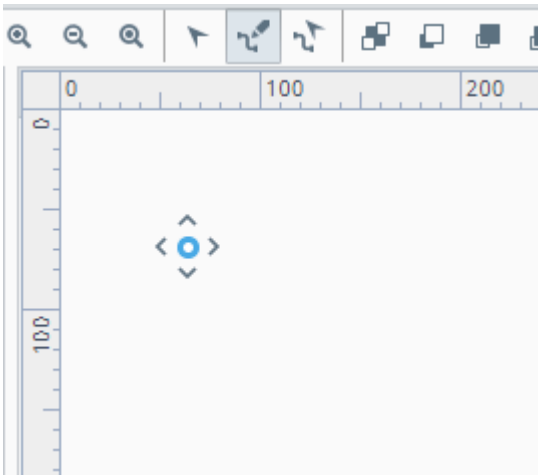
NOTE

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.

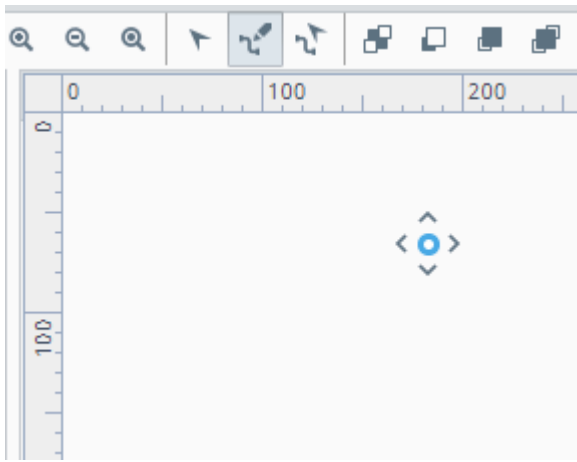
By default, a Coordinate Container's `aspectRatio` property does not have a value. As such, setting your container's `aspectRatio` to a value like 1:1 when creating Pipes is recommended. This helps Pipes remain anchored to components if the components change size in proportion with the container.



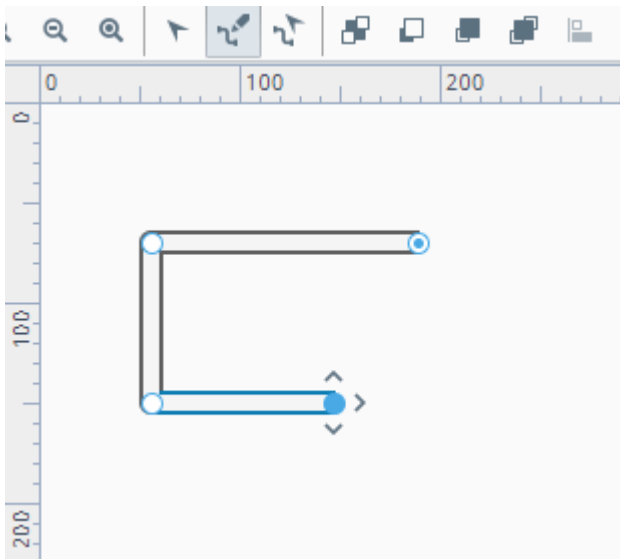
All connections, including the origin, can be repositioned by clicking and dragging.



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.

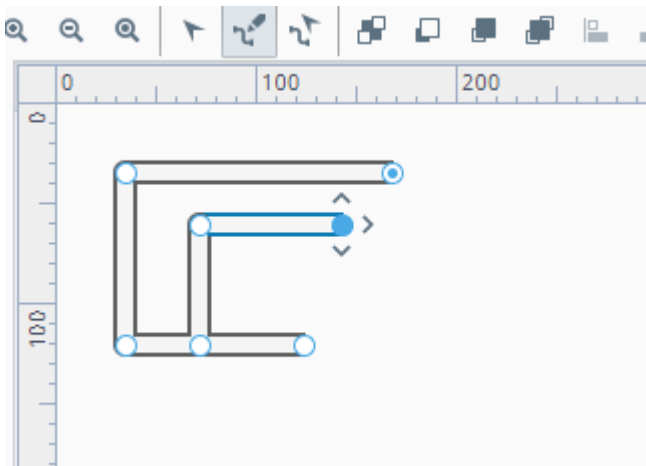


New connections can be added to existing segments. Doing so allows the segment to split off into a new segment.

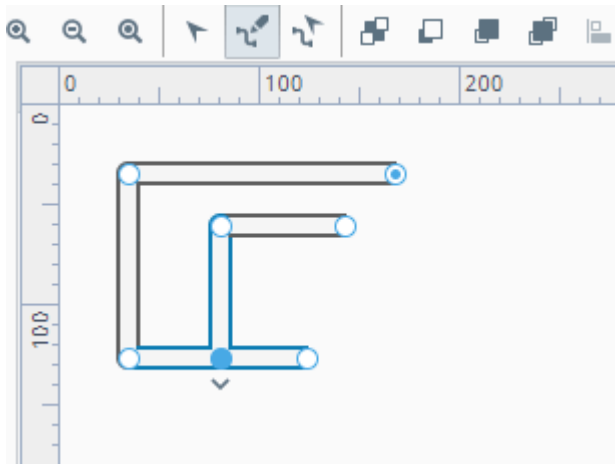


To delete a segment, click on a connection in the segment and press the **Delete** key, or by right-clicking and selecting "Delete".

Existing connections can be moved by dragging. All attached segments will move with the origin.

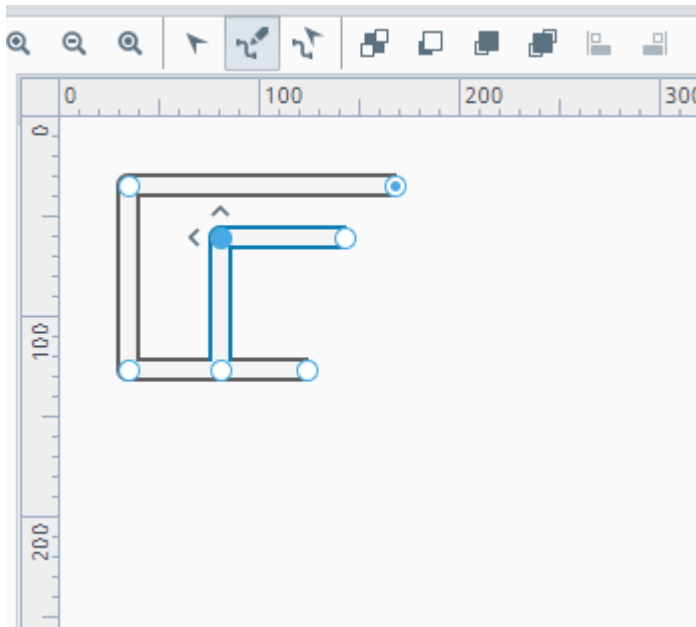


If **Shift** is held while repositioning a connection, then connecting segments will bend.



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:

Q 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 appearances.

Perspective Property Editor

Editing 2 pipes

Style: P&ID
Width: 10

Q 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.

Q 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}

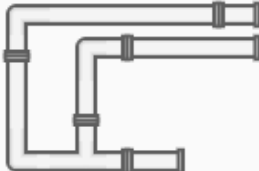
x : 36

y : 212.61

▼ connections [1]

▶ 0 {3}

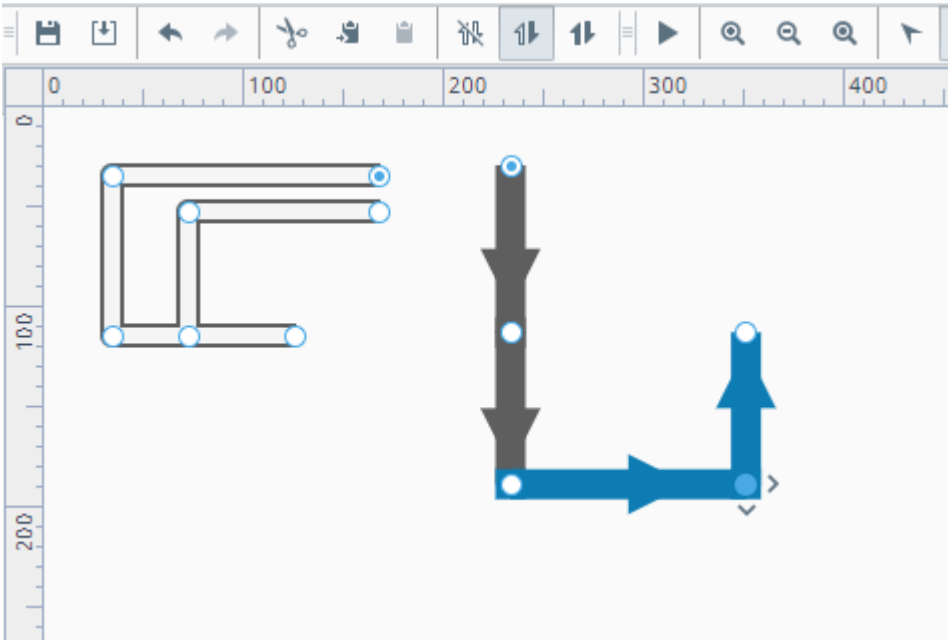
Pipe Properties

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.  SimpleMimicP&ID In addition to the appearance options, there is an auto value will will cause the set to use the appearance specified under <code>session.props.pipes.autoAppearance</code>.
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.  SolidDashedMid ArrowWavy
Line Color (P&ID Appearance Only)	The color to use for the pipe set.

Property	Description
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	Determines whether the pipe set is visible or not. Default setting is True.

Combining Pipes

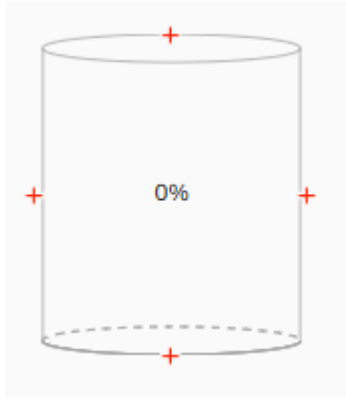
Dragging a pipe to another will combine the pipes. Combining pipes will cause the pipes to share the same appearance. However this combination can be overridden by pressing the Undo button (or by pressing **Ctrl-Z/Command-Z**) once. This allows different pipes to overlap without actually combining.



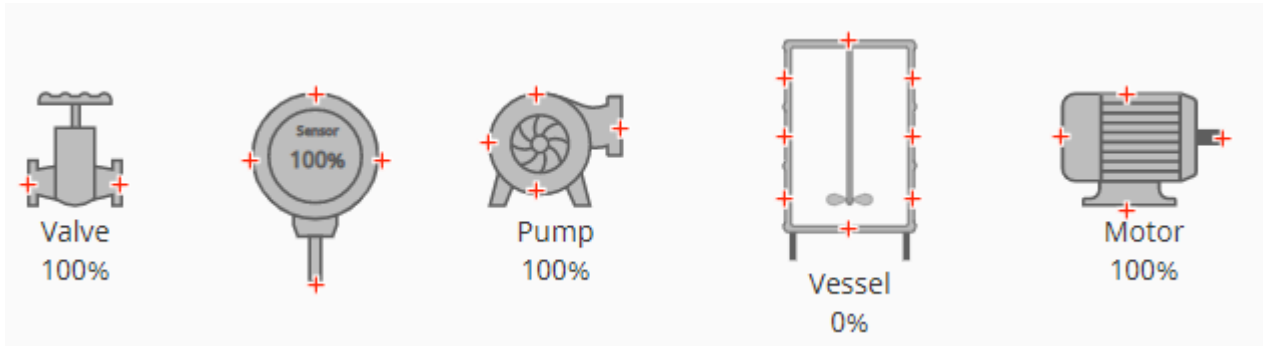
Additionally, a connection can be dragged onto it's immediate sibling or parent, which effectively removes the dragged connection.

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.

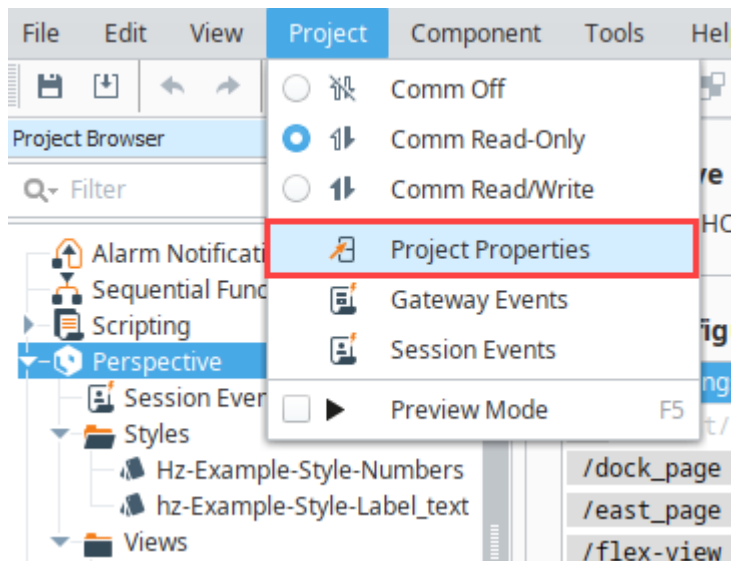


Dragging a segment near the anchor will cause the segment to snap to the anchor (unless the **Shift** key is currently being held). This can be useful when leading segments into different components. Note that anchors are only used to guide the segment in the designer. The pipes are not "connected" to the component, so moving the component later will not cause the pipe segment to move.

Perspective Project Properties

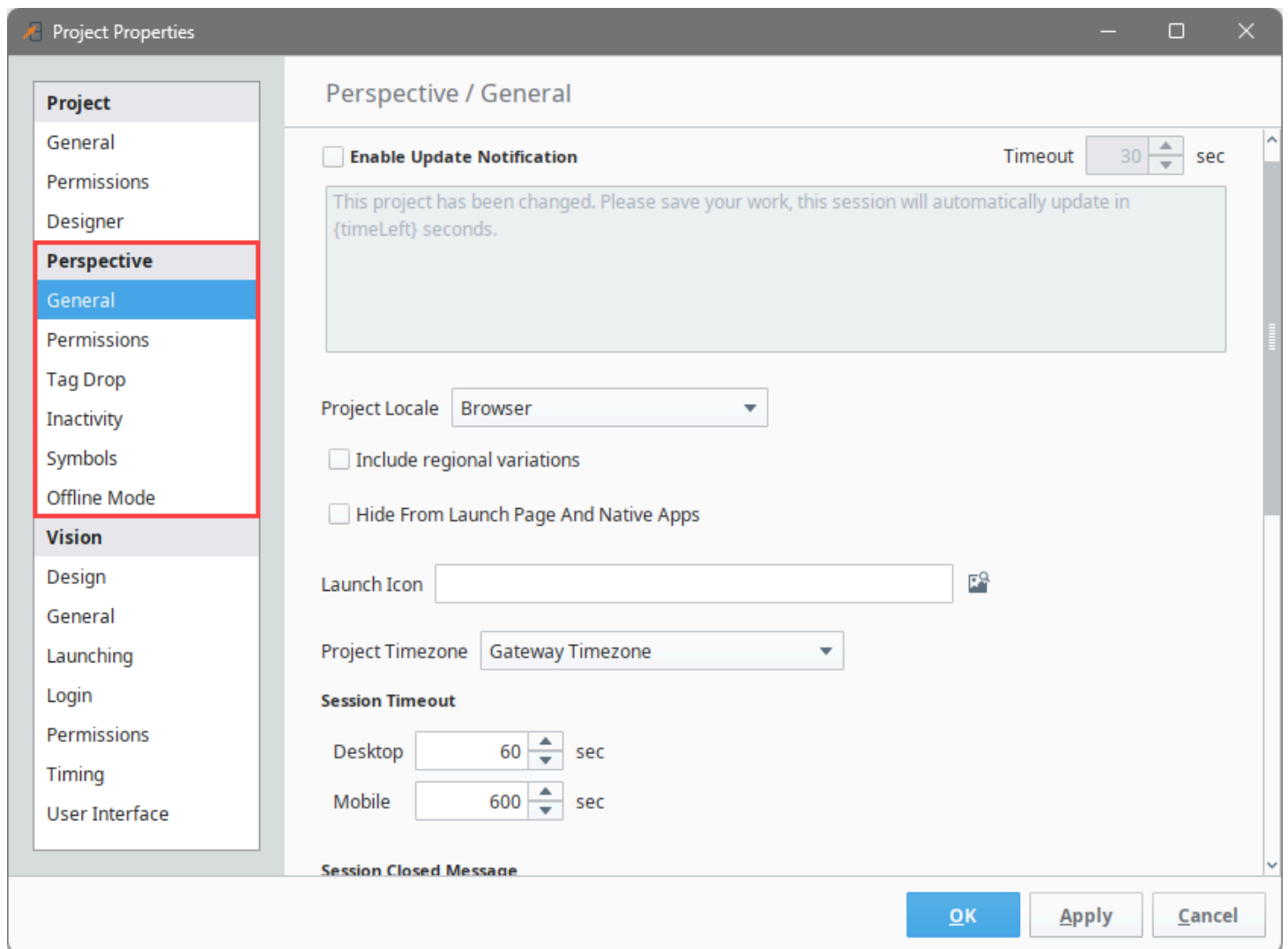
There are a number of properties you can set for your Perspective projects within the Designer. For example, there are properties for setting security levels, configuring how the sessions receive updates, and more.

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, including:

- **General**
- **Permissions**
- **Tag Drop**
- **Inactivity**
- **Symbols**
- **Offline Mode**



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 <code>{timeLeft}</code> 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.
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 <code>system.perspective.closeSession</code> is invoked.
Page Closed Message	Message to be displayed if <code>system.perspective.closePage</code> is invoked.
Logged Out Message	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.

Project Properties

Project

General

Permissions

Designer

Perspective

General

Permissions

Tag Drop

Inactivity

Symbols

Vision

Design

General

Launching

Login

Permissions

Timing

User Interface

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

OK

Apply

Cancel

Perspective Tag Drop Properties



INDUCTIVE
UNIVERSITY

Tag Drop

[Watch the video](#)

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 **Add**  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.
 - The Tag Drop interface includes **Bidirectional** and **Coalesce** settings when adding components where these settings can be applied.

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.

Project Properties

Project

General

Permissions

Designer

Vision

Design

General

Launching

Login

Permissions

Timing

User Interface

Perspective

General

Permissions

Tag Drop

Inactivity

Symbols

Perspective / Tag Drop

Data Type Configuration

Data TypesInteger

Components

Cylindrical Tank

Gauge

Label

LED Display

Progress

Simple Gauge

Slider

Component Bindings Configuration

Component TypeSlider

Tag Property	Prop Path	Bidirectional	Coalesce
value	props.value	☒	☐

OK

Apply

Cancel

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	A dropdown list of the available data types in Ignition. Choose a datatype to see a list of components associated with it.
Components	A 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	A dropdown list of the available components in Perspective.
Tag Property	The name of a tag's property to be used in a binding. <code>value</code> is most commonly used.
Prop Path	The property path on the component where the binding will be created. <code>props.value</code> or <code>props.text</code> are most commonly used.
Bidirectional	A check box to indicate if the binding should be bidirectional. The Bidirectional property is set by default when adding Input Components.
Coalesce	A check box that when checked ensures target property changes of child properties are combined into a single write back to the source. This setting applies to Bidirectional bindings only.

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.

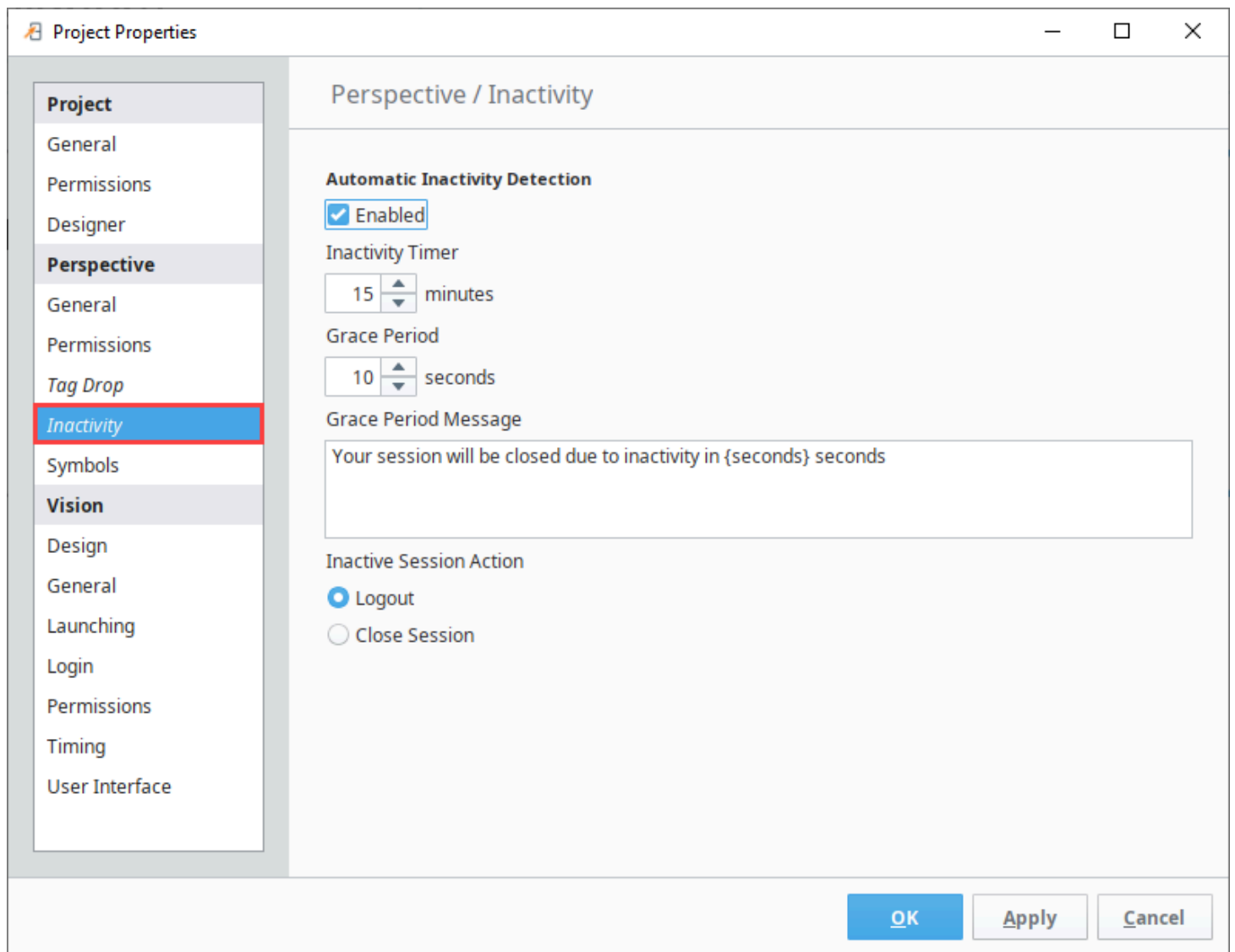
 NOTE

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.

Inactivity

The following are the properties for the Inactivity Timeout:

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 <code>{seconds}</code> 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. - Close Session - Close the session.



Perspective Symbols Properties

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.

Symbols

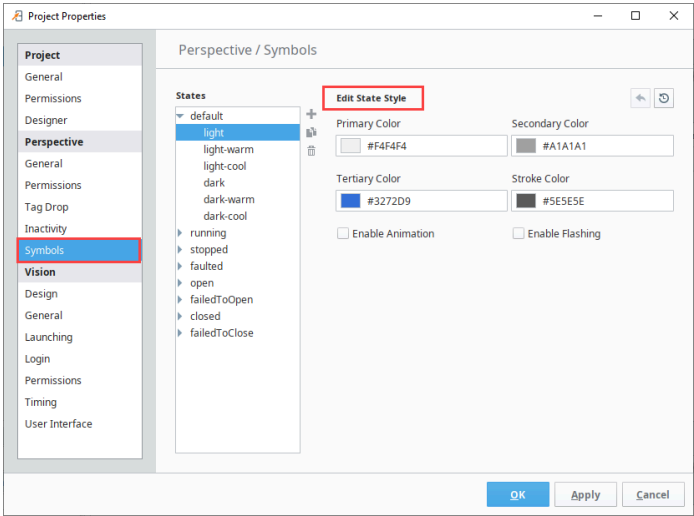
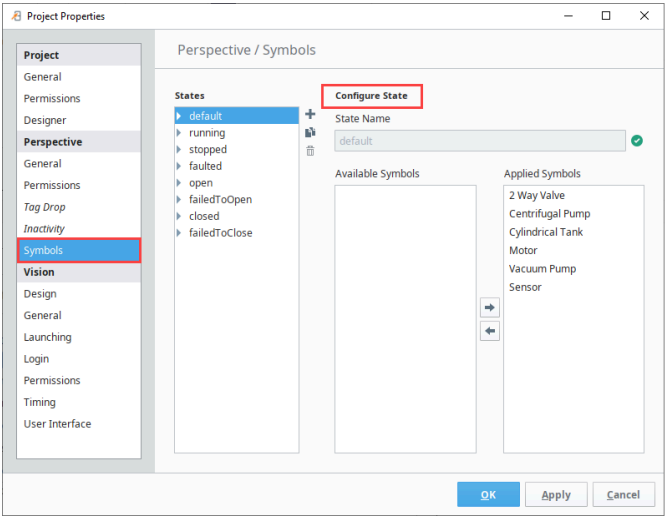
The following table lists the properties for the Symbols States:

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.

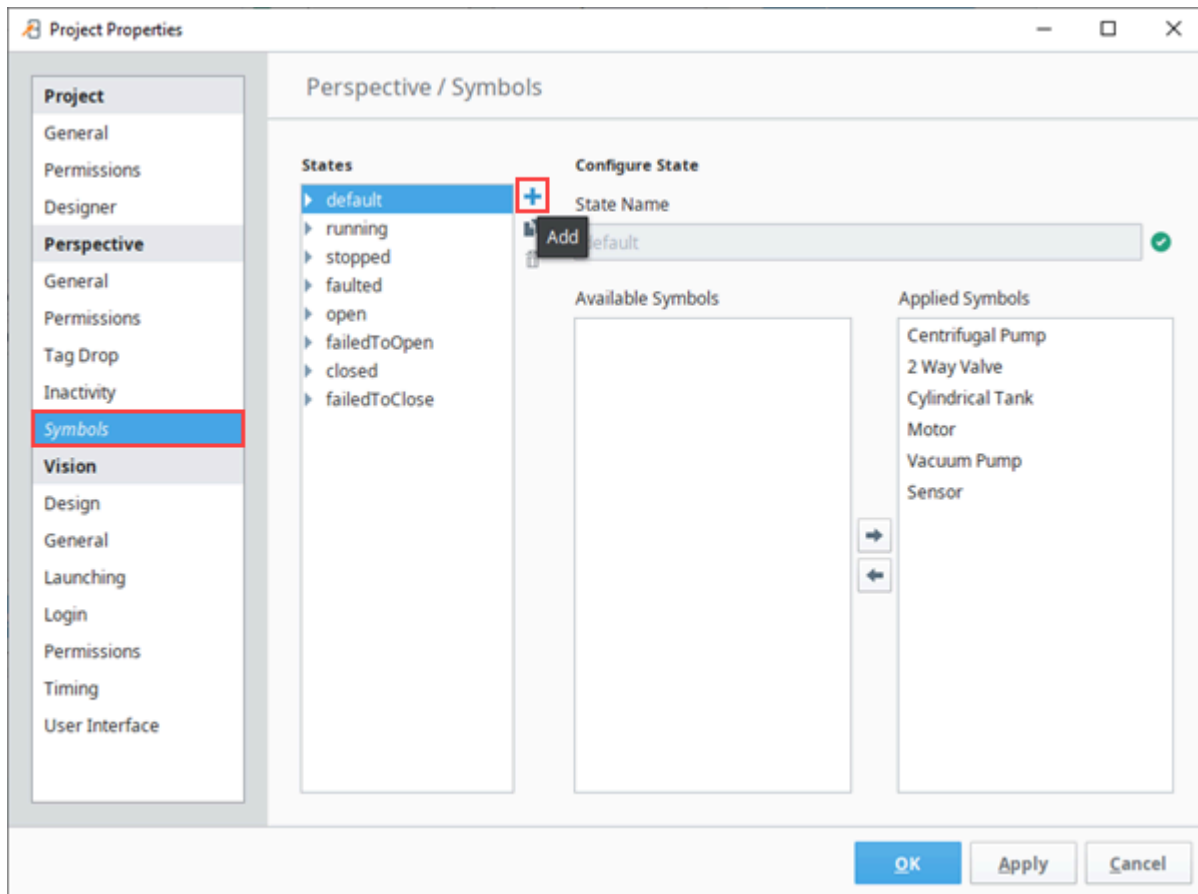
Edit State Style (State Name > Visual State Selected)

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.

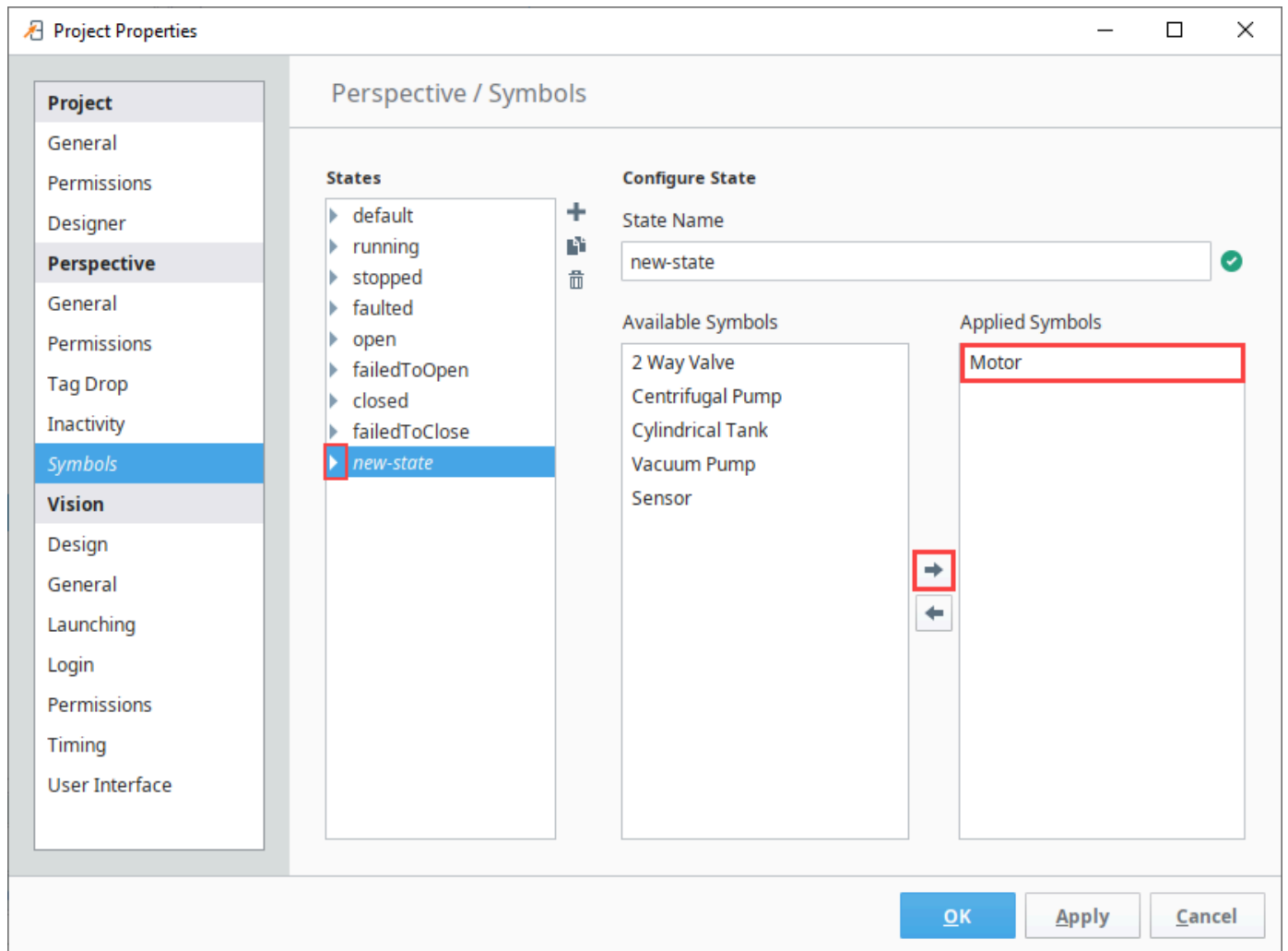


Create and Apply a Custom State

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.

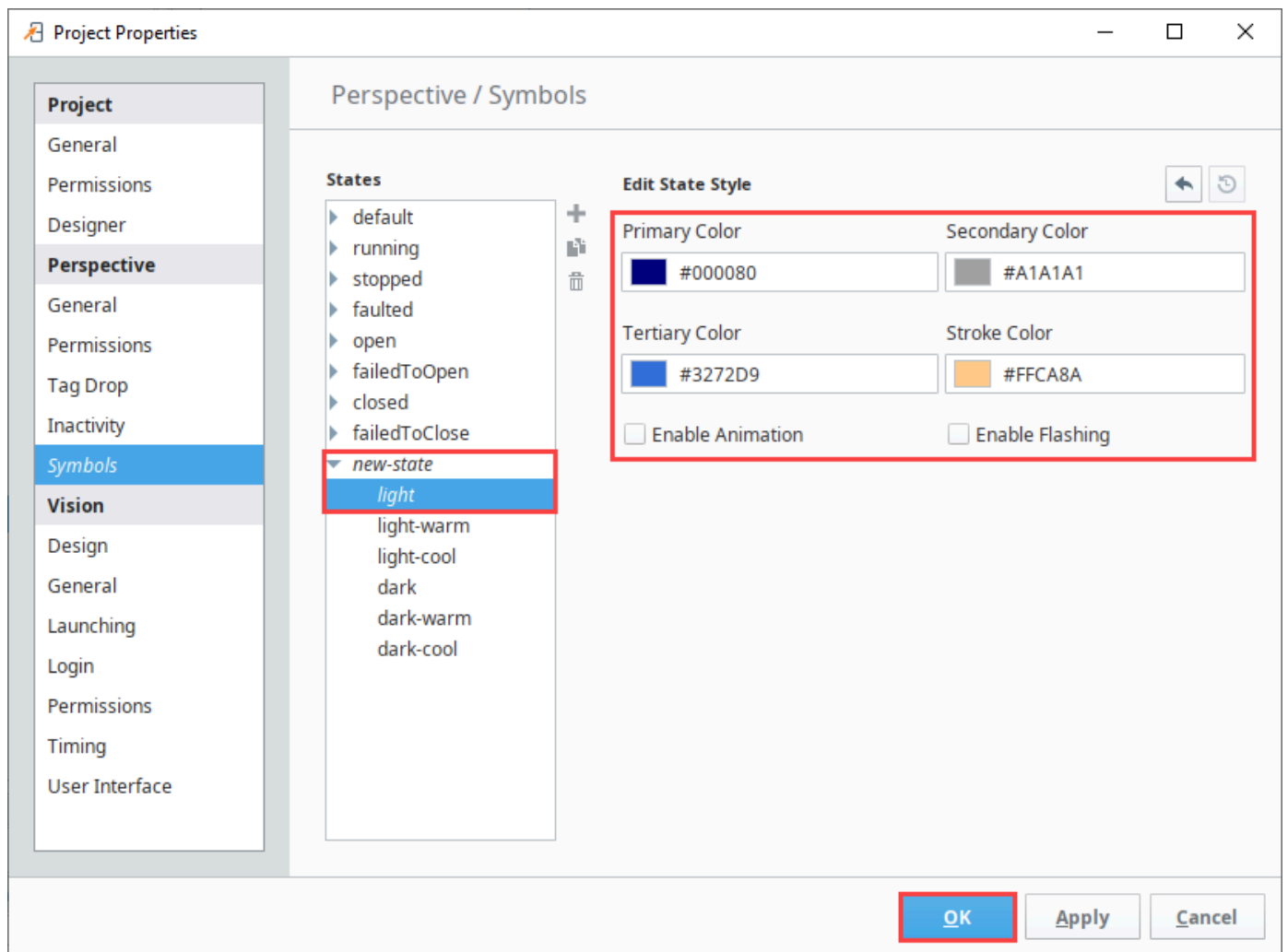


4. Use the **Add Applied Symbol**  arrow to select and apply your new state to desired symbols. This example will make this state applicable to the **Motor** symbol.
5. Use the state dropdown to select the visual themes you want to edit. This example will edit the light visual state.

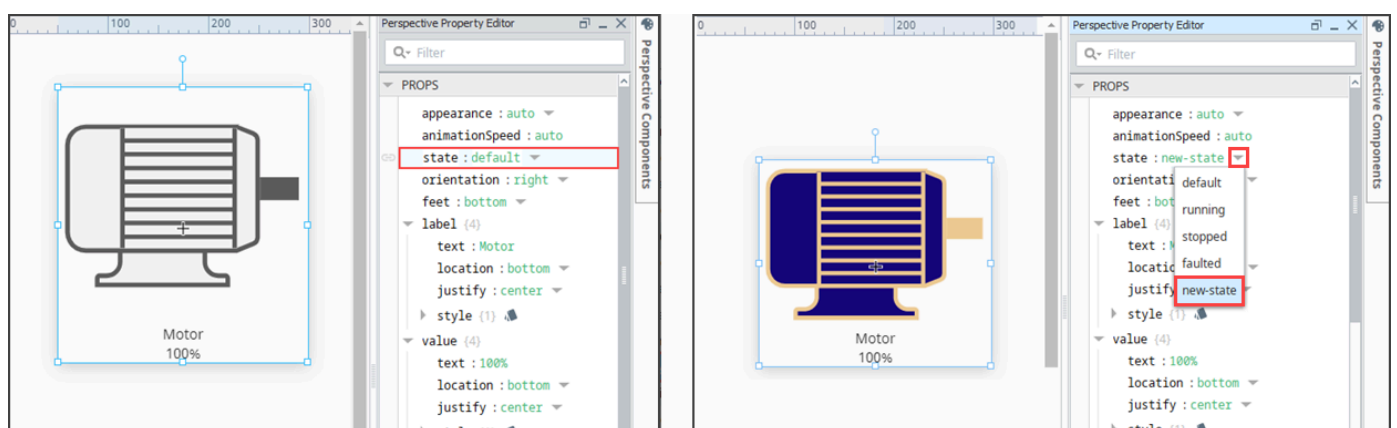


6. Set your desired color, animation, and flashing configurations under Edit State Style.

7. Click **OK**.



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.



Perspective Offline Mode Properties

Offline Mode allows users to interact with a project even when the Gateway is unavailable. When enabled, the Perspective App on supported mobile devices stores project resources locally so projects can be launched offline as well as continued during temporary disconnections. This includes selected

views, supported components, authentication tokens, themes, and localization assets. Offline Mode is particularly useful for mobile and remote scenarios where network connectivity may be unreliable or intermittent.

To use Offline Mode, you'll need to enable it in the Designer. Once enabled, any view a user visits during an online session will be available offline as well. Users can interact with components such as forms, which in Offline Mode will queue submissions for later transmission. However, Gateway-dependent features such as tag values, database queries, and script execution will not function while offline.

NOTE

While Offline Mode must be enabled in the Designer, the feature only works in the Perspective App on mobile devices and is not supported in desktop browsers. See [Offline Mode](#) for full details.

Offline Mode

The following properties control caching behavior and user access for Offline Mode.

Property	Description
Enabled	Enables Offline Mode for the project. When enabled, compatible mobile devices using the Perspective App will cache project resources locally for offline access.
Security Levels	Defines which users are allowed to use Offline Mode. You can specify one or more required security levels. Supports All (user must match all levels) or Any (user must match at least one level).
Authentication Token Expiration	Duration (in days) the cached authentication token remains valid for offline use. After this period, the user must reconnect online to re-authenticate. Default is 7 .
Always ask IdP to re-authenticate	Forces the user to re-authenticate with the Identity Provider every time the Perspective App connects online, even if the cached token is still valid.
Themes	Specifies which project themes should be bundled for offline use. Only selected themes will be available to users when disconnected.
Languages	Specifies which languages should be included in the offline cache. This limits localization assets to reduce bundle size.

Project Properties

Project

General

Permissions

Designer

Perspective

General

Permissions

Tag Drop

Inactivity

Symbols

Offline Mode

Vision

Design

General

Launching

Login

Permissions

Timing

User Interface

Perspective / Offline Mode

☒ Enabled

💡 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

☐ 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

Authentication Token Expiration

7

 days

☐ Always ask the IdP to re-authenticate

Themes

☐ light

OK

Apply

Cancel

Styles



INDUCTIVE
UNIVERSITY

Styles

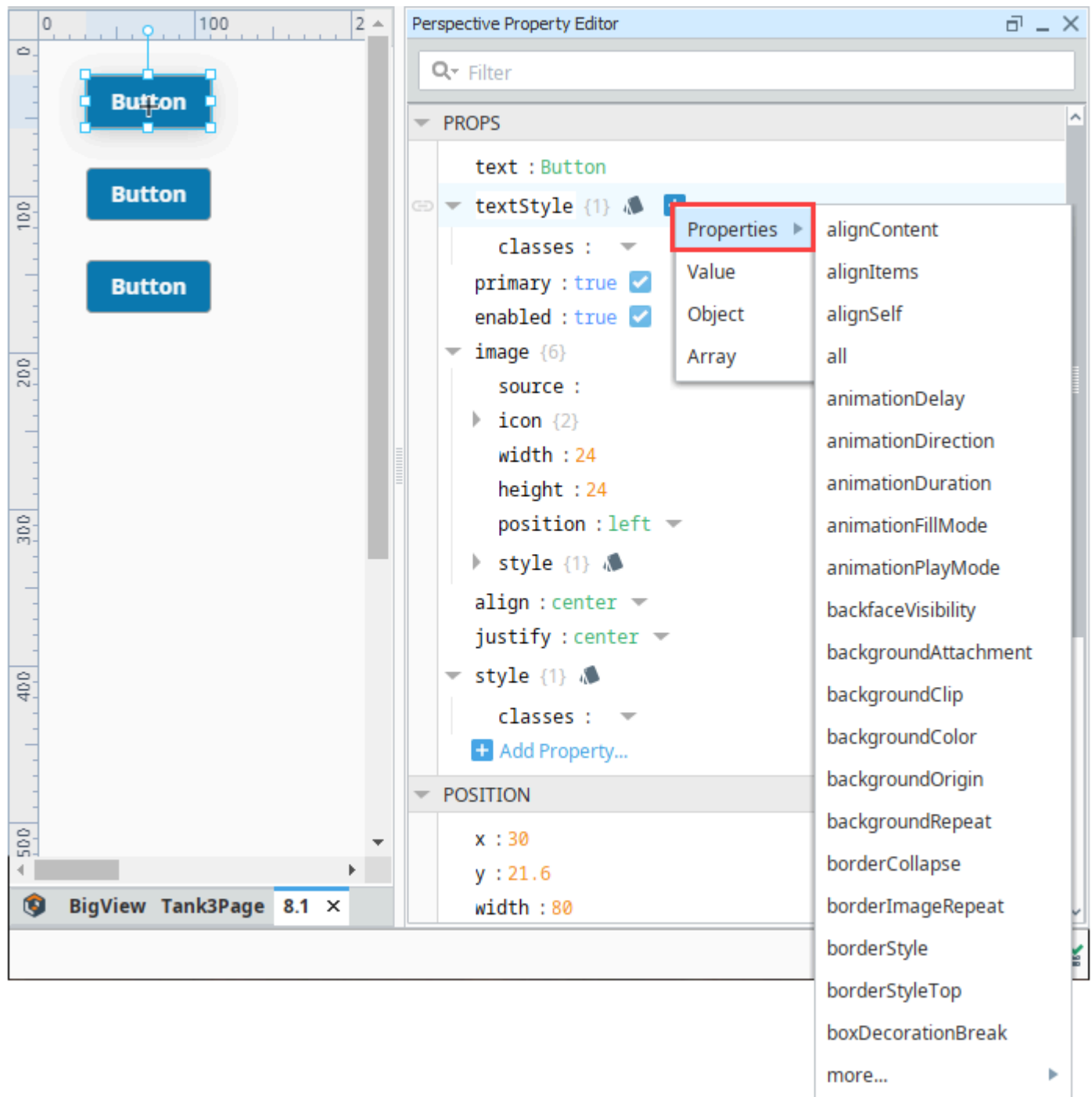
[Watch the video](#)

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](#).

CSS properties are also available through a properties dropdown menu. This menu provides access to all available properties in the schema.

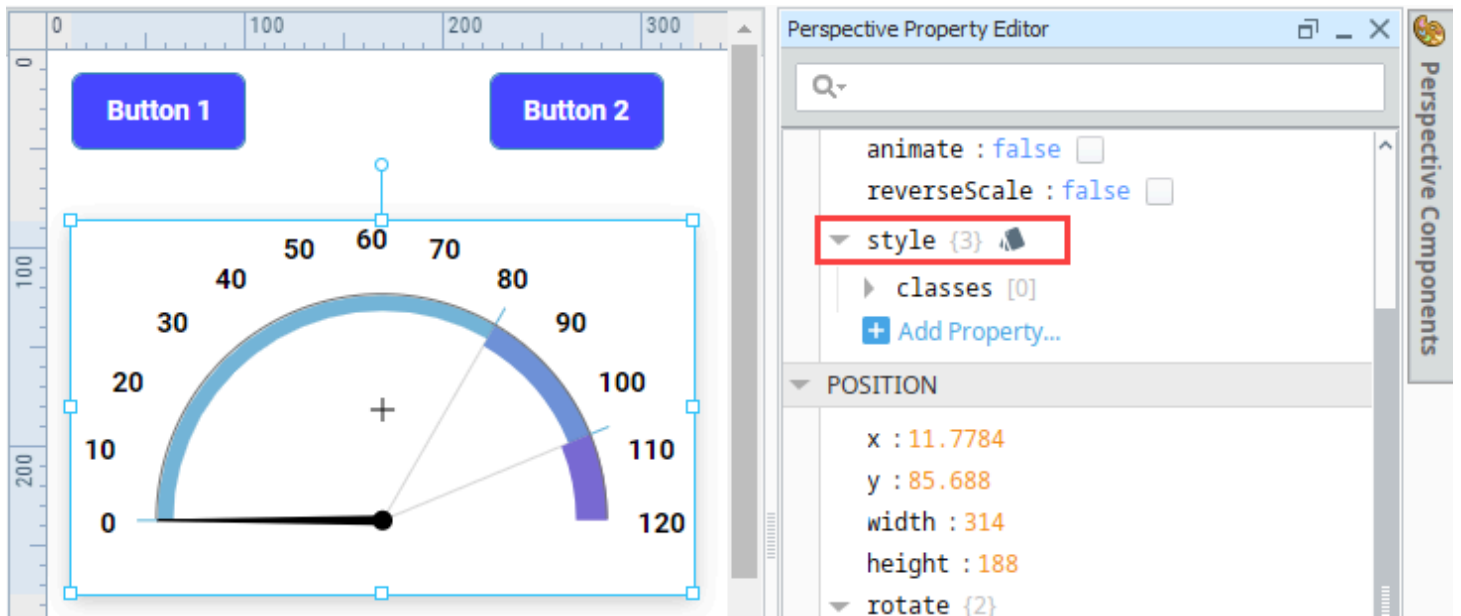
Properties Example



A group of Style settings can be saved together and given a name as a **Style Class**.

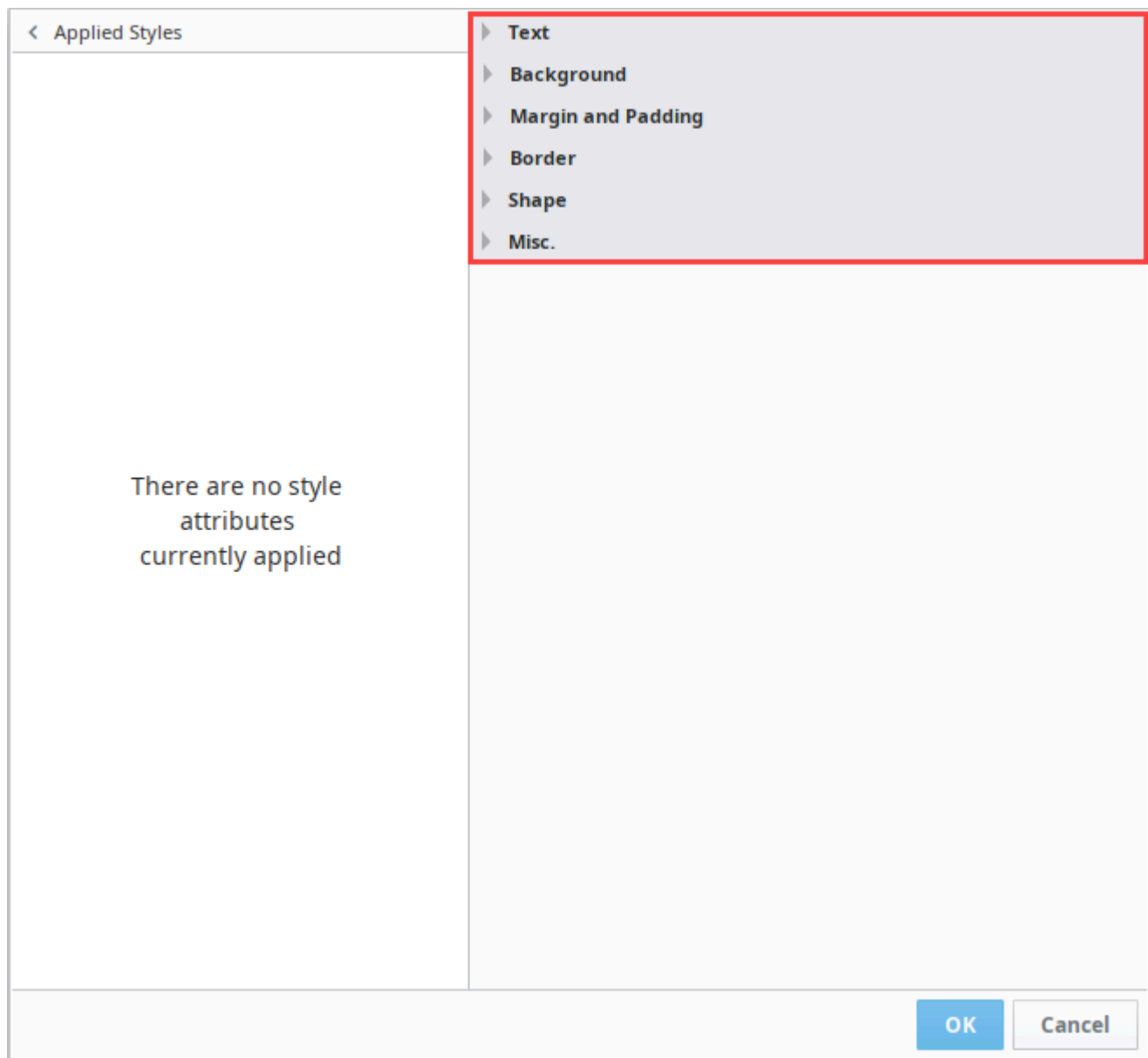
Style Editor

Wherever the style property and the **Styles**  icon appear in the Property Editor, you can click on the icon to display the style editor.



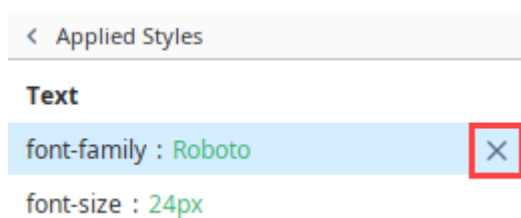
The following menus are available in the style editor:

- Text
- Background
- Margin and Padding
- Border
- Shape
- Misc

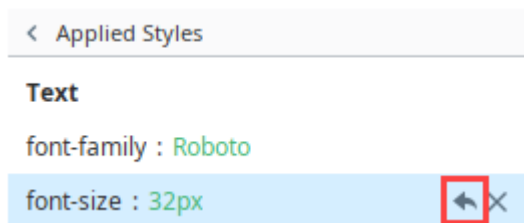


You can quickly make adjustments to the styles by clicking on them in the Applied Styles panel.

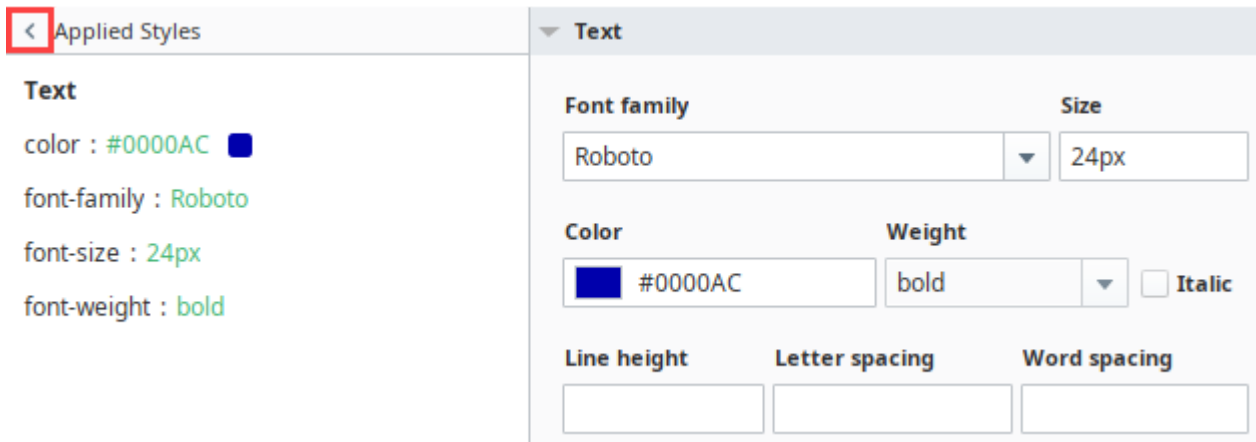
When you hover over a style listed in the panel, a **Delete**  icon is displayed. Click on it to delete the style.



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.



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.



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

< Applied Styles

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

Alignment

Text align

center

Indent

White space

Text Options

Transform

Tt TT tt

Decoration

S ~~S~~ S

☐ Break-word

Shadow ☐

OK

Cancel

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.

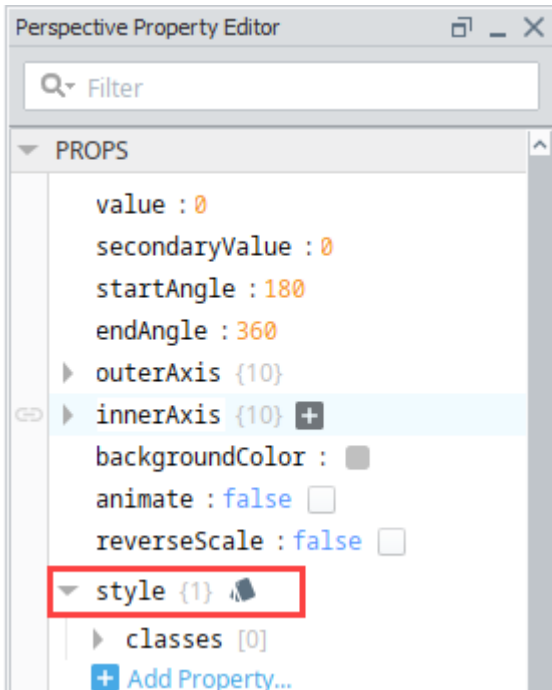
CSS Lengths

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

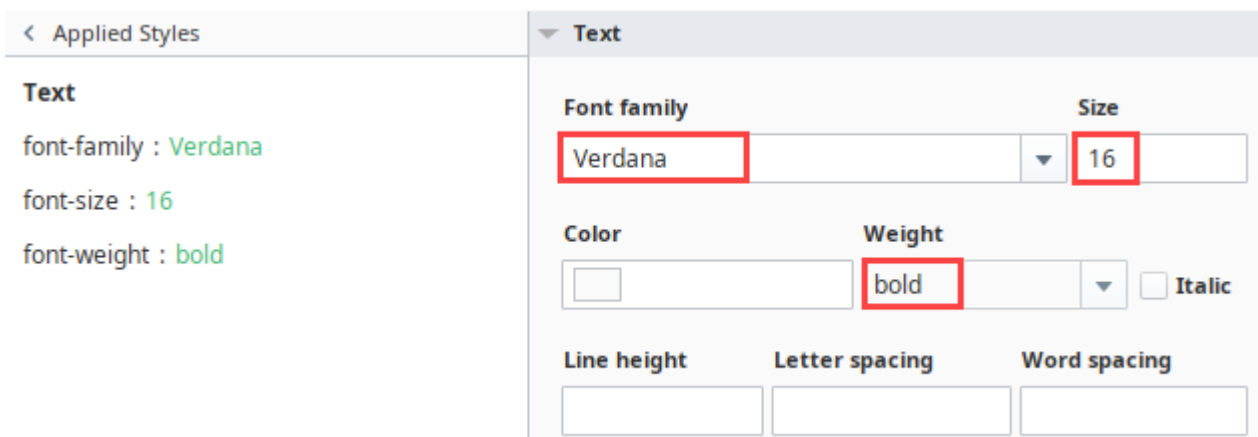
1. In a coordinate view, drag a Gauge component and a Label component.
2. Select the Gauge component and click the **Styles**  icon in the properties for the component.



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.

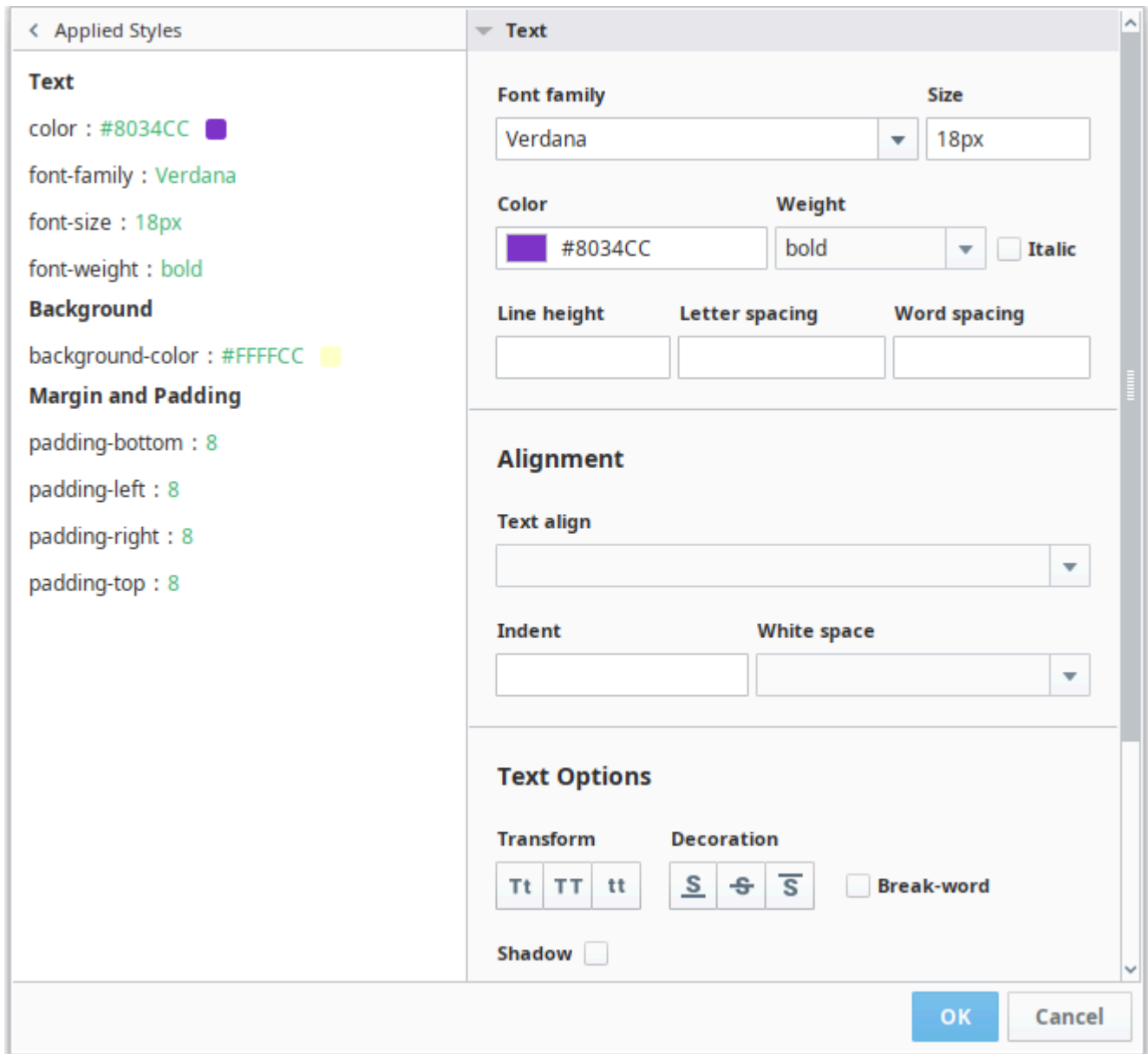


4. Click **OK**.

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

Line height Letter spacing Word spacing

Alignment

Text align

Indent White space

Text Options

Transform Tt TT tt Decoration S ~~S~~ S ☐ Break-word

Shadow ☐

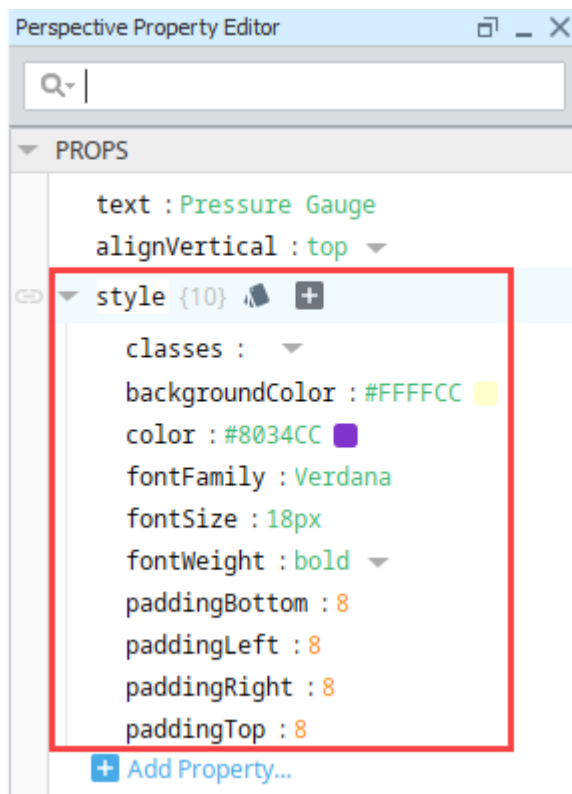
OK Cancel

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 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:

Base

- light
- dark

Derived

- light-warm
- light-cool
- dark-warm
- dark-cool

Built-in themes are considered system collection resources, which means the best way to view any theme files or definitions is by using API Keys. The API endpoints for Perspective Themes are located in the OpenAPI docs at `<gateway_address>/openapi#tag/config-perspective-themes`.

For instance, you can view the config for the light theme by using the GET endpoint at `/data/api/v1/resources/find/com.inductiveautomation.perspective/themes/light`. It is also possible to define your own variable or make modifications to the built-in theme files, using the Modify or Create API endpoints.

Built-in Theme Modifications

Derived Themes

The derived themes, like light-cool and dark-cool, can be modified freely. Be aware that changes to these built-in themes will remain overridden through restarts and upgrades, so any updates from

Inductive Automation will not be applied. For this reason, it is highly recommended that you create custom themes with their own entry point and various CSS files.

Base Themes

Both the light and dark themes are system config resources. This means that they can't be altered. The only way to override light and dark, is to create a `overrides-[light | dark]` theme resource. If an override resource is present, Ignition will redirect the entry point of the light or dark theme to the entry point of the `overrides-light` or `overrides-dark` theme resource accordingly. The `overrides-[light | dark]/index.css` file should contain one or several import statements that import CSS from the light/dark theme. A simple and complex example are included below to demonstrate this structure.

Simple Light Theme Override Example

```
@import "../light/index.css";

:root {
  --customVariable: "value";
}
```

Complex Light Theme Override Example

```
@import "../variables.css";
@import "../light/fonts.css";
@import "../globals.css";
@import "../light/app/index.css";
@import "../light/common/index.css";
@import "../light/designer/index.css";
@import "../light/palette/index.css";
```

Note that when creating an `overrides` theme resource, the theme config property `isPrivate` is set to **true** in order to hide the theme from themes list.

POST Payload Example

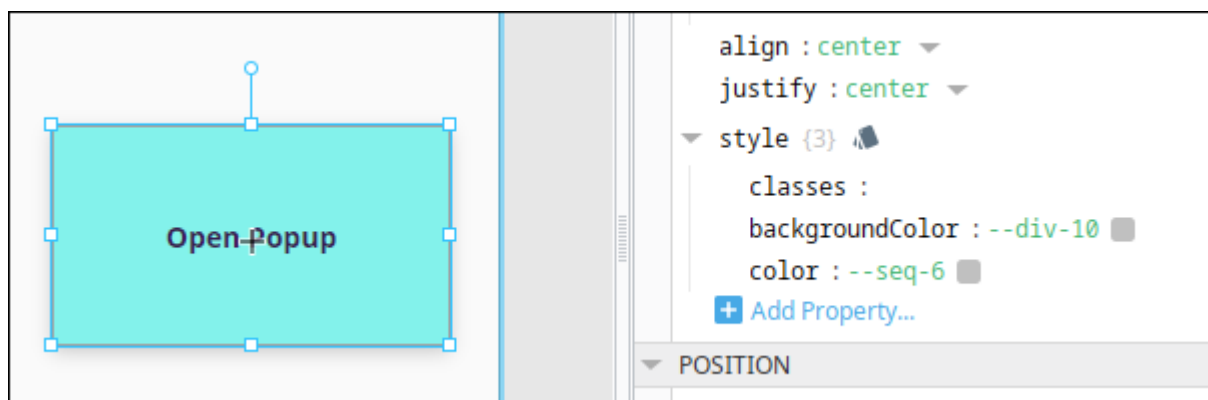
```
[
  {
    "name": "overrides-light",
    "collection": "core",
    "enabled": true,
    "description": "Overrides light theme.",
    "config": {
      "entrypoint": "index.css",
      "isPrivate": true
    }
  }
]
```

Refer to the [Creating and Using Custom Perspective Themes](#) page for more information on theme customization.

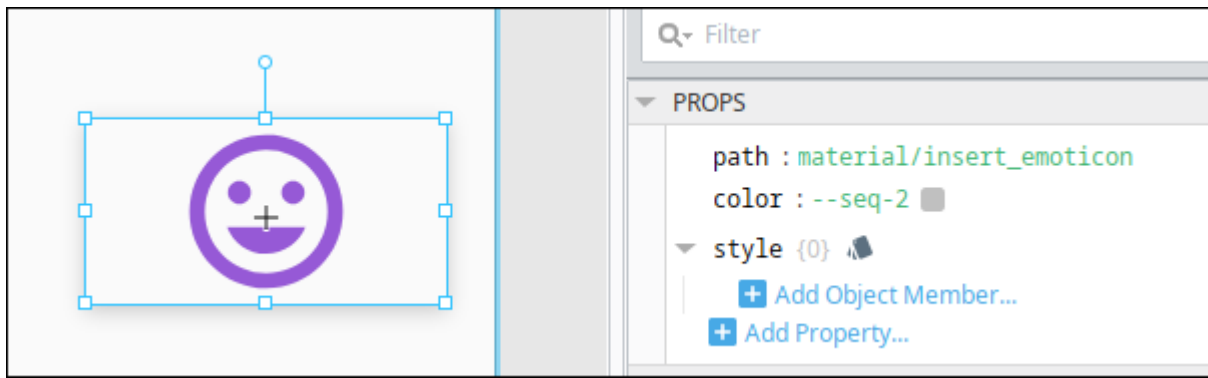
Theme Colors

The built-in themes make heavy use of CSS variables for colors. Colors are defined in a default theme `variables.css` file. They are also listed below in the [Built-in Theme Colors](#) table.

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 stating the variable name for the appropriate styling properties on the component's **style** object.

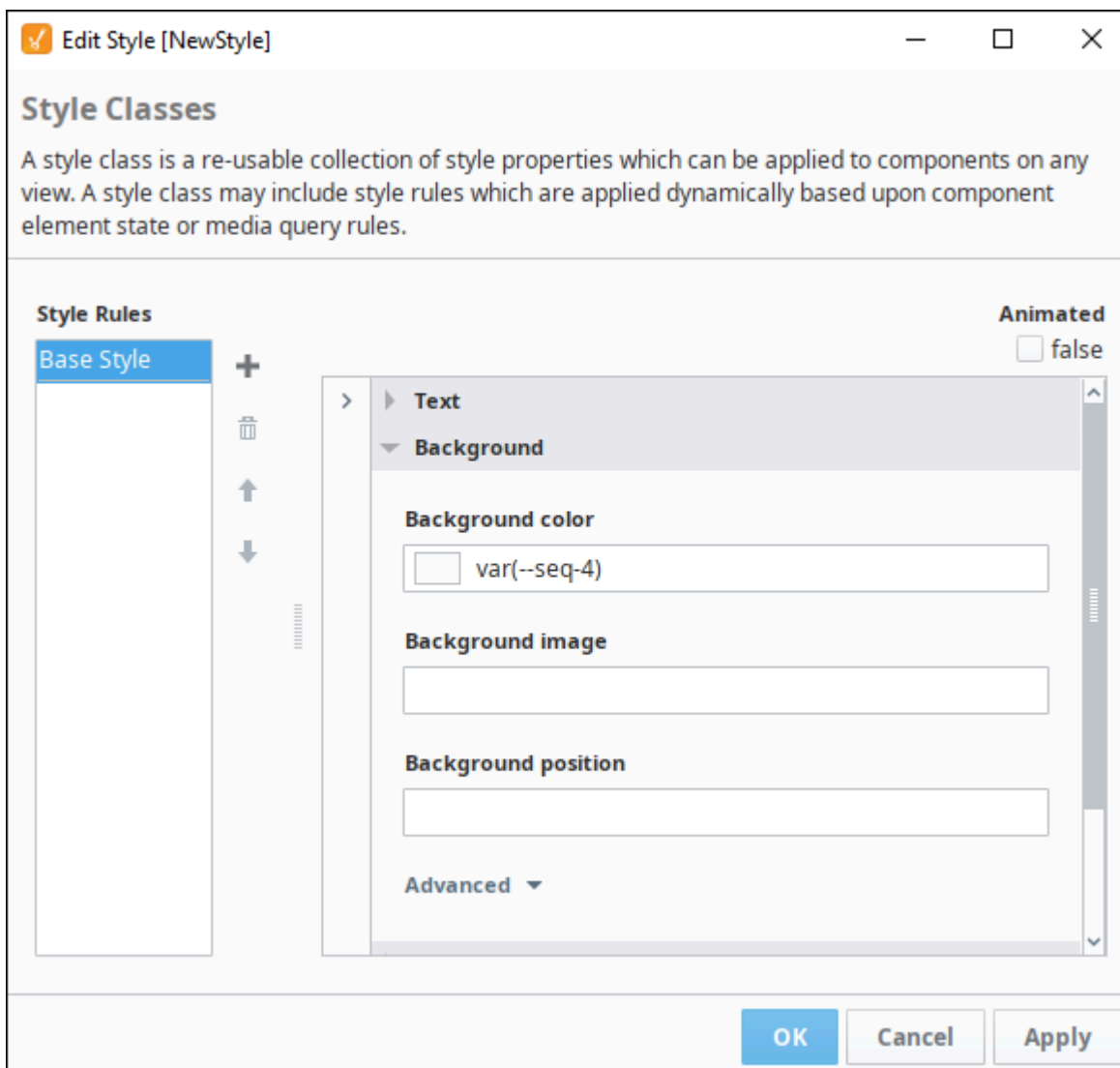


If a component has a **color** property outside of a style object, such as the Icon component, the same rules apply; simply set the value of the color property to the name of the variable.



Style Classes

When using a Theme Color in a Style Class, the variable must be wrapped in the `var()` method. So, the `--seq-4` variable would be entered as `var(--seq-4)`.



Built-in Theme Colors

The following color swatch represents the built-in color variables for each Inductive Automation provided theme.

Variable Name	light	light-cool	light-warm	dark	dark-cool
--neutral-10	#fafafa	#f2f4f8	#f7f3f2	#161616	#121212
--neutral-20	#f4f4f4	#dde1e6	#e5e0df	#323232	#212121
--neutral-30	#d8d8d8	#c1c7cd	#cac5c4	#515151	#343434
--neutral-40	#bdbdbd	#a2a9b0	#ada8a8	#5e5e5e	#4d5358
--neutral-50	#a1a1a1	#878d96	#8f8b8b	#767676	#697077
--neutral-60	#767676	#697077	#736f6f	#a1a1a1	#878d96
--neutral-70	#5e5e5e	#4d5358	#565151	#bdbdbd	#a2a9b0
--neutral-80	#515151	#343a3f	#3c3838	#d8d8d8	#c1c7cd
--neutral-90	#323232	#21272a	#272525	#f4f4f4	#dde1e6

Variable Name	light	light-cool	light-warm	dark	dark-co
--neutral-100	#161616	#121619	#171414	#fafafa	#f2f
--seq-1	#ae74e8	#ae74e8	#ae74e8	#9659d6	#965
--seq-2	#9659d6	#9659d6	#9659d6	#ae74e8	#ae7
--seq-3	#7d3cbd	#7d3cbd	#7d3cbd	#c79bf2	#c79
--seq-4	#642b9e	#642b9e	#642b9e	#dabcf7	#dab
--seq-5	#4b2175	#4b2175	#4b2175	#ead9fa	#ead
--seq-6	#371c52	#371c52	#371c52	#f5f0fa	#f5f
--div-1	#642b9e	#642b9e	#642b9e	#642b9e	#642
--div-2	#7d3cbd	#7d3cbd	#7d3cbd	#7d3cbd	#7d3

Variable Name	light	light-cool	light-warm	dark	dark-cool
--div-3	#9656d6	#9656d6	#9656d6	#9656d6	#9656d6
--div-4	#ae74e8	#ae74e8	#ae74e8	#ae74e8	#ae74e8
--div-5	#c79bf2	#c79bf2	#c79bf2	#c79bf2	#c79bf2
--div-6	#dabcf7	#dabcf7	#dabcf7	#dabcf7	#dabcf7
--div-7	#ead9fa	#ead9fa	#ead9fa	#ead9fa	#ead9fa
--div-8	#f5f0fa	#f5f0fa	#f5f0fa	#f5f0fa	#f5f0fa
--div-9	#d7faf8	#d7faf8	#d7faf8	#d7faf8	#d7faf8
--div-10	#83f2eb	#83f2eb	#83f2eb	#83f2eb	#83f2eb
--div-11	#43ded3	#43ded3	#43ded3	#43ded3	#43ded3

Variable Name	light	light-cool	light-warm	dark	dark-co
--div-12	#21c2b7	#21c2b7	#21c2b7	#21c2b7	#21c
--div-13	#0ea197	#0ea197	#0ea197	#0ea197	#0ea
--div-14	#08827a	#08827a	#08827a	#08827a	#088
--div-15	#086962	#086962	#086962	#086962	#086
--div-16	#09524d	#09524d	#09524d	#09524d	#095
--qual-1	#7d3cbd	#7d3cbd	#7d3cbd	#9656d6	#965
--qual-2	#229ad6	#229ad6	#229ad6	#53baed	#53b
--qual-3	#086962	#086962	#086962	#08827a	#088
--qual-4	#b01355	#b01355	#b01355	#f78bb8	#f78

Variable Name	light	light-cool	light-warm	dark	dark-cool
--qual-5	#f55353	#f55353	#f55353	#f55353	#f55353
--qual-6	#661414	#661414	#661414	#faf0f0	#faf0f0
--qual-7	#038537	#038537	#038537	#46e385	#46e385
--qual-8	#114599	#114599	#114599	#5691f0	#5691f0
--qual-9	#ed5393	#ed5393	#ed5393	#ed5393	#ed5393
--qual-10	#cf7911	#cf7911	#cf7911	#e89c3f	#e89c3f
--callToAction	#0c7bb3	#0c7bb3	#0c7bb3	#229ad6	#229ad6
--callToActionHighlight	#e8f4fa	#e8f4fa	#e8f4fa	#0c2938	#0c2938
--callToAction--hover	#229ad6	#229ad6	#229ad6	#0c7bb3	#0c7bb3

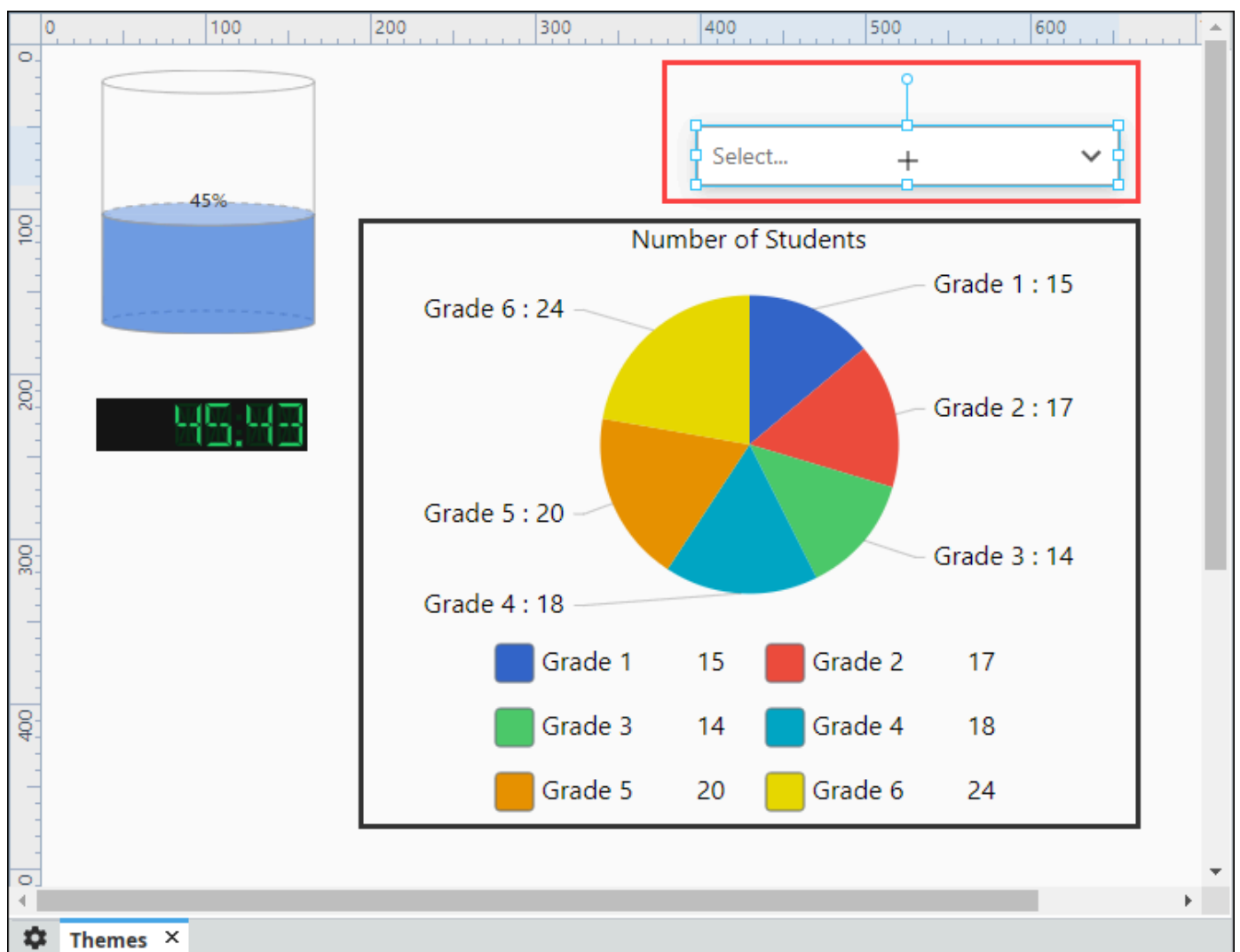
Variable Name	light	light-cool	light-warm	dark	dark-cool
--callToAction-- active	#066391	#066391	#066391	#53baed	#53baed
--callToAction-- disabled	#a1a1a1	#a1a1a1	#a1a1a1	#a1a1a1	#a1a1a1
--error	#de1b1b	#de1b1b	#de1b1b	#f55353	#f55353
--info	#3272d9	#3272d9	#3272d9	#5691f0	#5691f0
--infoSecondary	#accbfc	#accbfc	#accbfc	#114599	#114599
--warning	#ad5f00	#ad5f00	#ad5f00	#cf7911	#cf7911
--warningSecondary	#f5bc76	#f5bc76	#f5bc76	#693d07	#693d07
--success	#038537	#038537	#038537	#0aa648	#0aa648
--indicator	#1ec963	#1ec963	#1ec963	#1ec963	#1ec963

Variable Name	light	light-cool	light-warm	dark	dark-co
--indicatorOff	#0a2e18	#0a2e18	#0a2e18	#0a2e18	#0a2e18

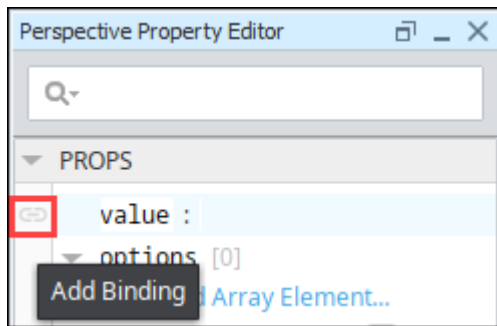
Changing the Theme from a Session

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.

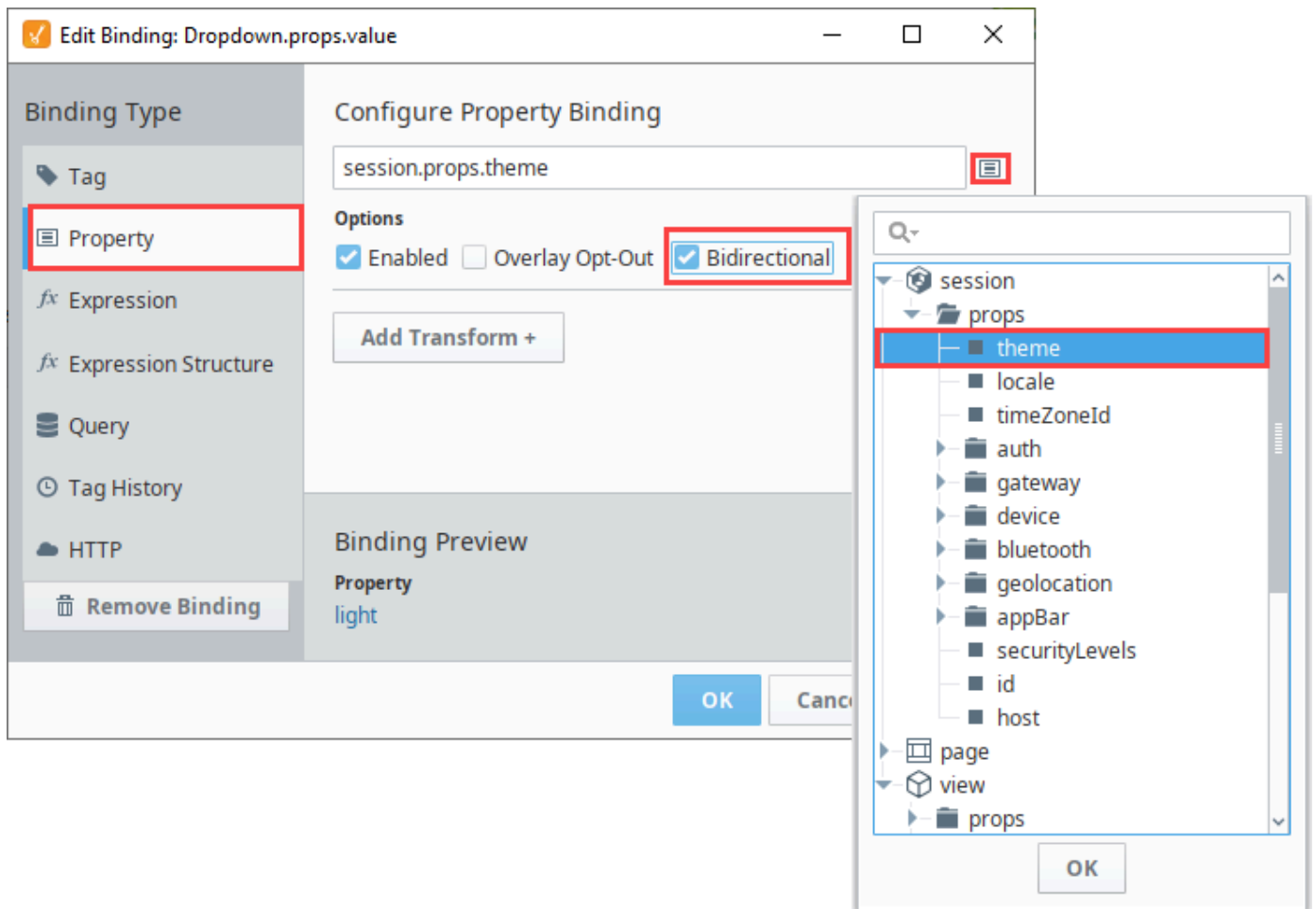
1. Open a view in the Designer that contains a few components and drag in a Dropdown component.



2. With the Dropdown component selected, click on the **Binding**  icon to create a property binding on the **value** property



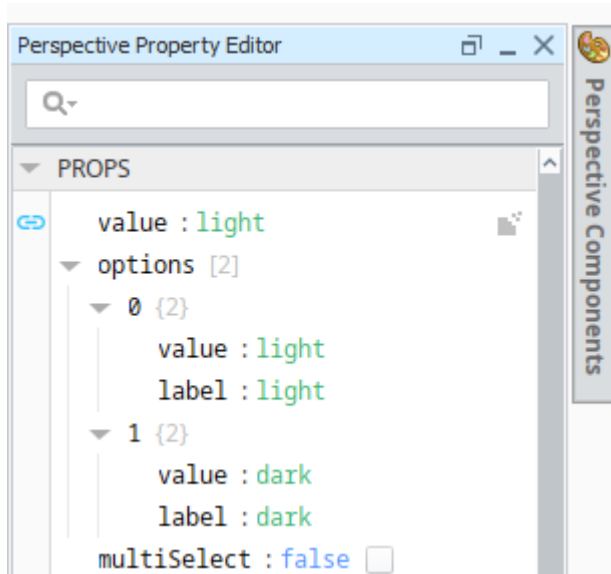
3. The Edit Binding window will open. The theme property is a session property. Click on the **Property**  icon and expand **session > props**, and select **theme**. Click **OK**.
4. Check **Bidirectional** and click **OK** to save your property binding.



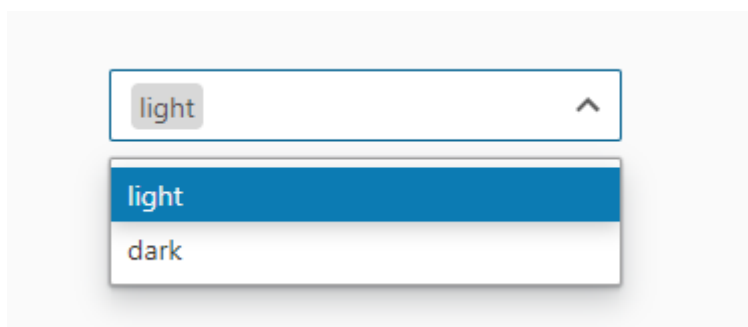
5. With the Dropdown component still selected, locate the **props.options** property in the Property Editor.

Paste the following JSON values onto the **options** property of the Dropdown component.

```
[
  {
    "value": "light",
    "label": "light"
  },
  {
    "value": "dark",
    "label": "dark"
  }
]
```



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



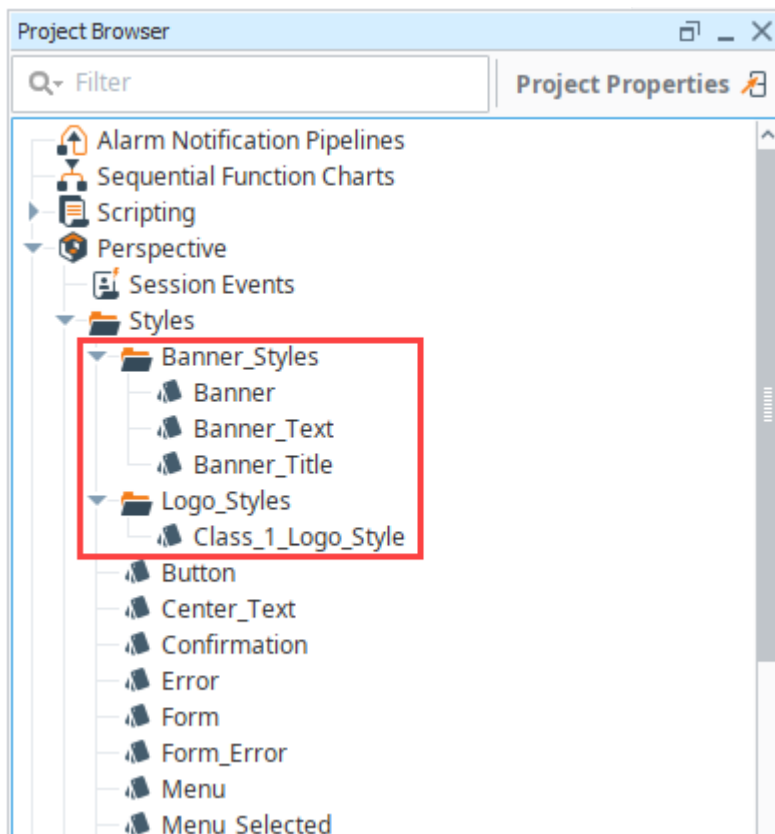
INDUCTIVE
UNIVERSITY

Style Classes

[Watch the video](#)

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.

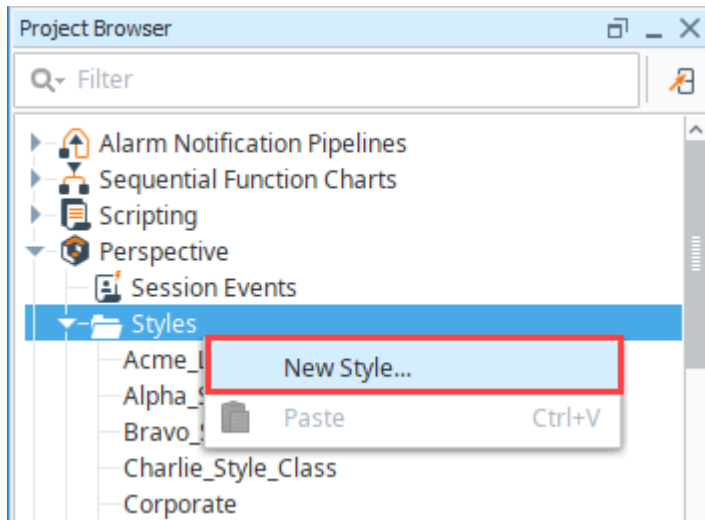


The prefix `ia_` is reserved for the base, predefined styling that is built into Perspective, and should not be used when naming user-created Style Classes. Using the `ia_` prefix for user-created Style Classes may result in unintended behavior.

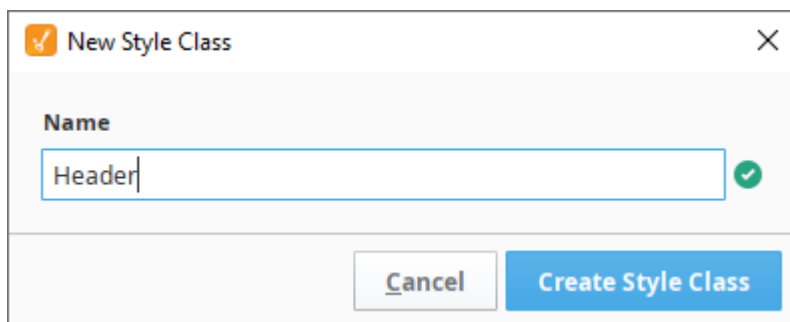
Creating a Style Class

In this example we'll create a 35 point Italic font, and center the text.

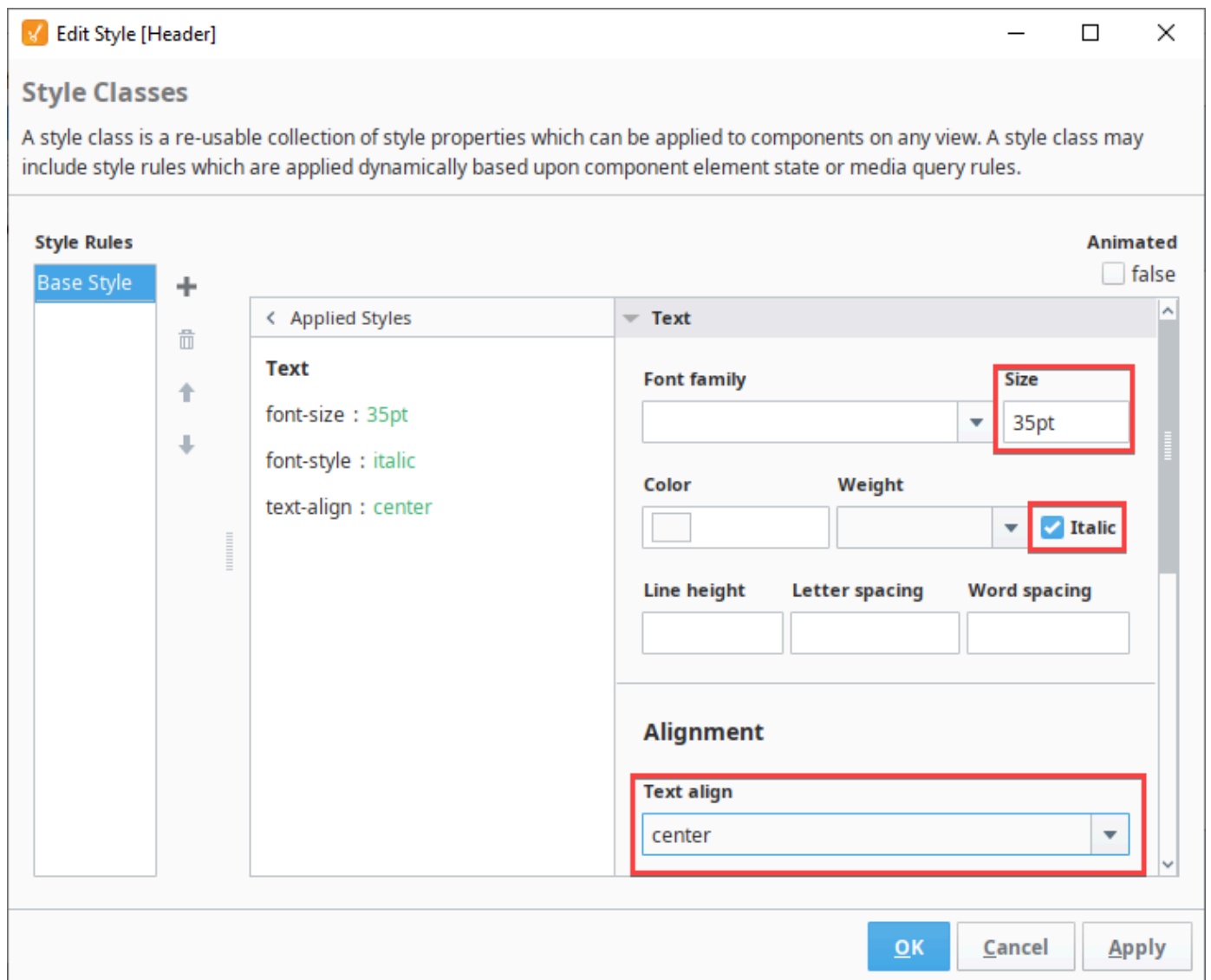
1. To create a Style Class, right-click on the Styles folder and select **New Style**.



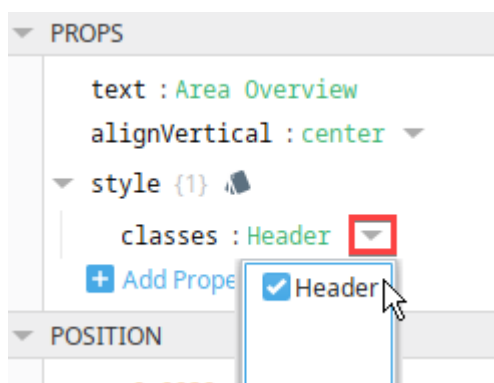
2. Enter a name for the new Style Class. Style Class names must be unique to the project. You will see a green check icon if the name you enter is acceptable. In this example we'll name ours "Header".



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



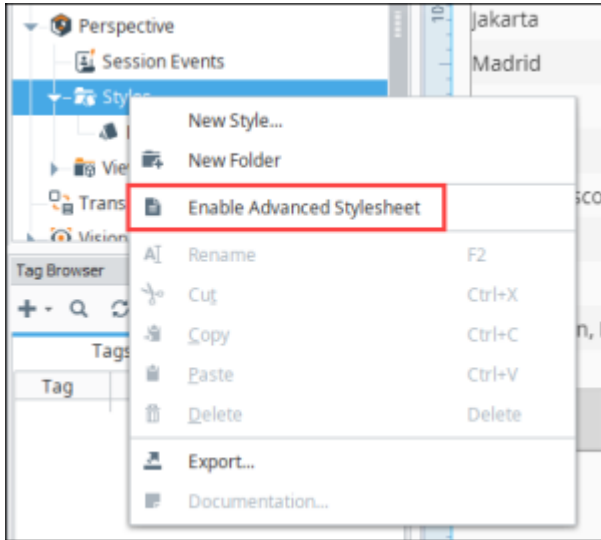
5. Click **OK** to save the Style Class.
6. Once a Style Class has been created, it can then be applied to a component. In the component's Style property, there will be a classes property with a dropdown list of all available Style Classes that can be applied to the component.



Enabling the Advanced Stylesheet

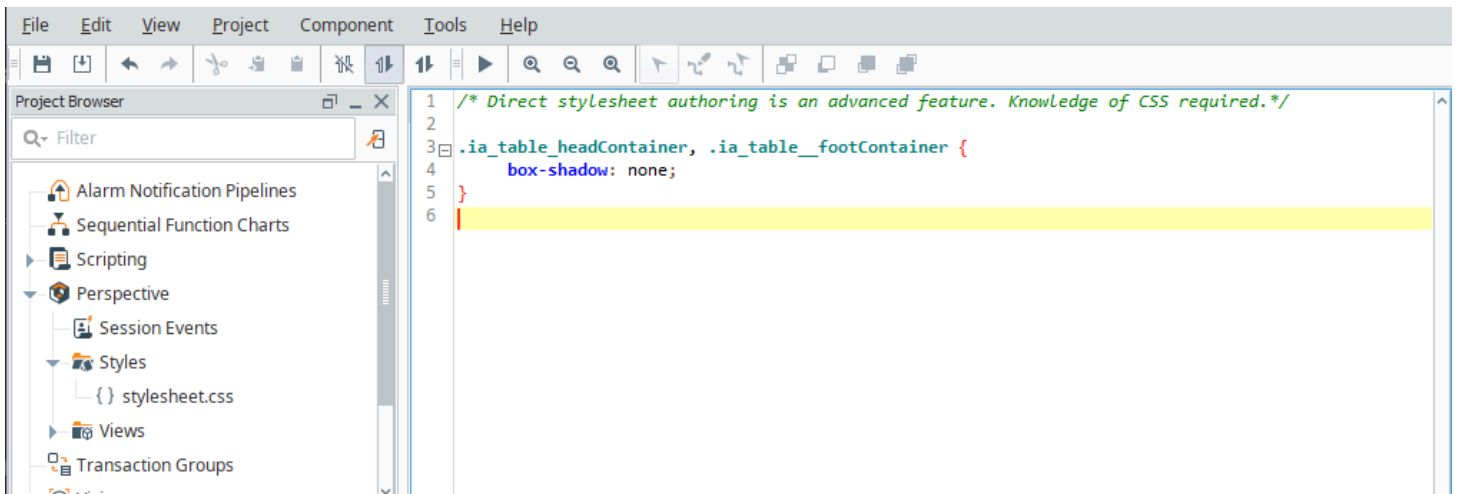
Users with extensive background working with CSS are now able to fine tune styling by modifying the Advanced Stylesheet resource. The stylesheet is an advanced feature and is not placed by default. To

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.

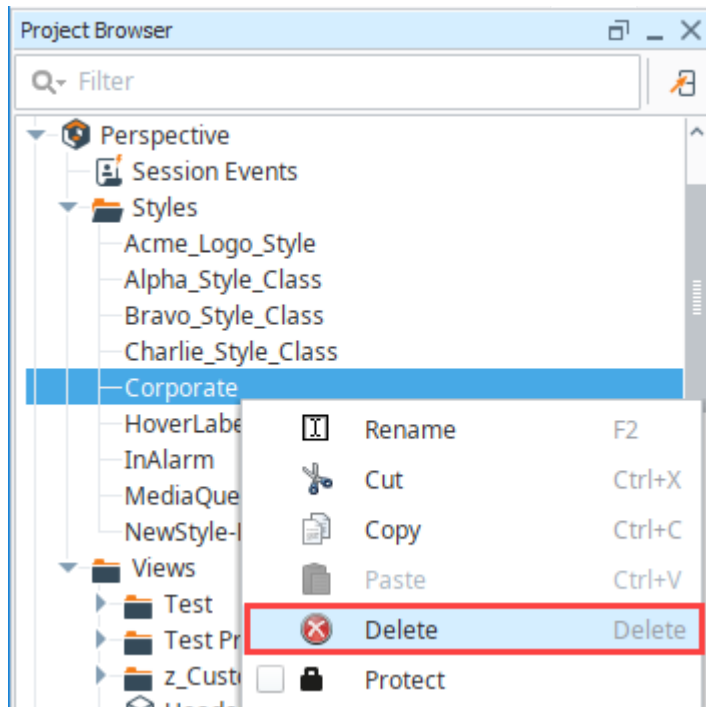


The rules defined in this resource are inserted into the Perspective project between the Theme files of the Gateway and the Style Classes of the project. This means the resource is scoped to a project, unlike Theming files being Gateway-scoped.

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

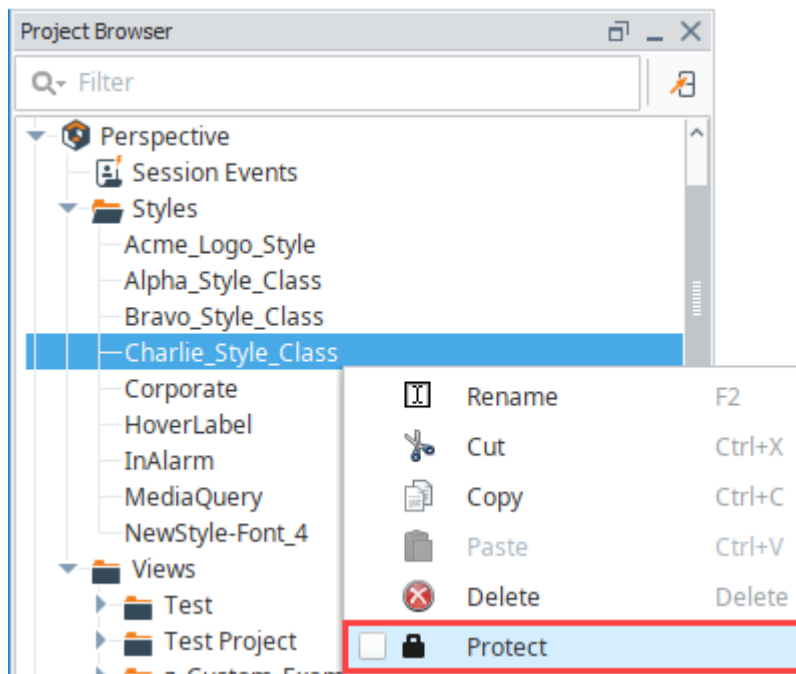
To rename a Style Class, right-click on the Style Class name in the Project Browser, then select **Rename**.

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](#).

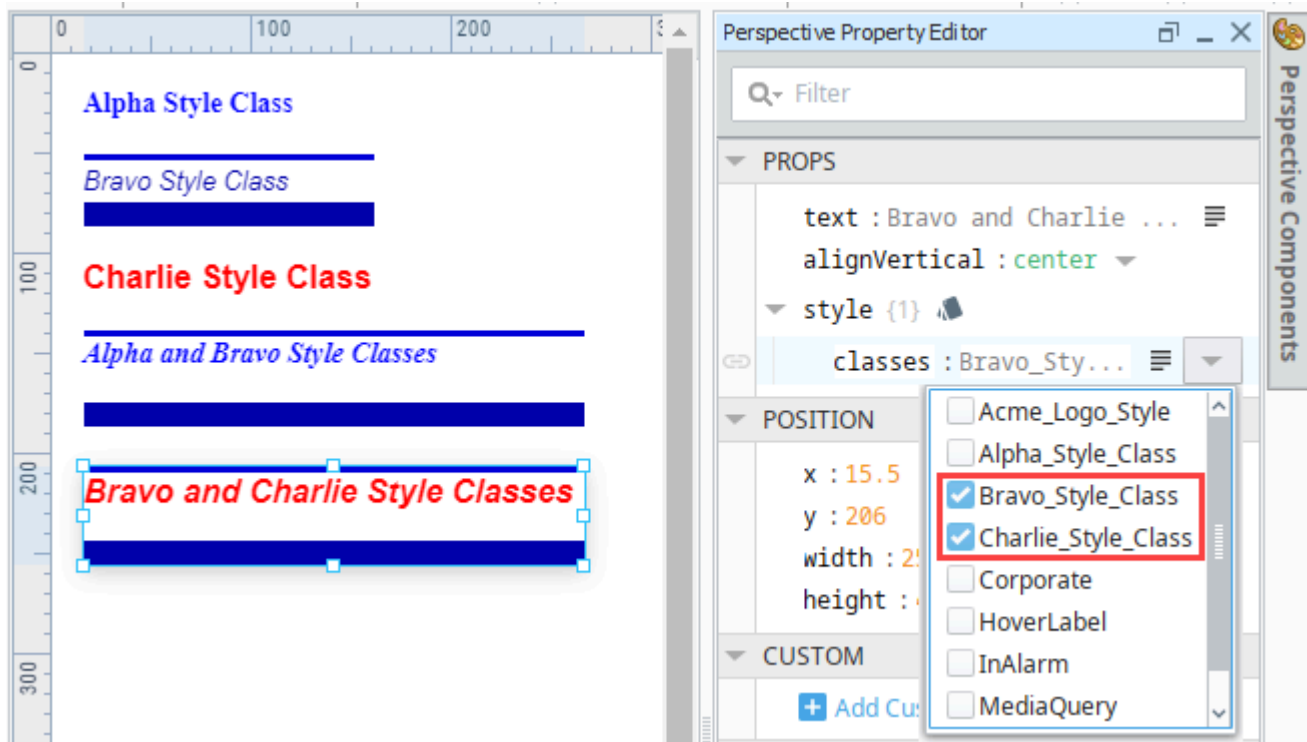


Multiple Style Classes

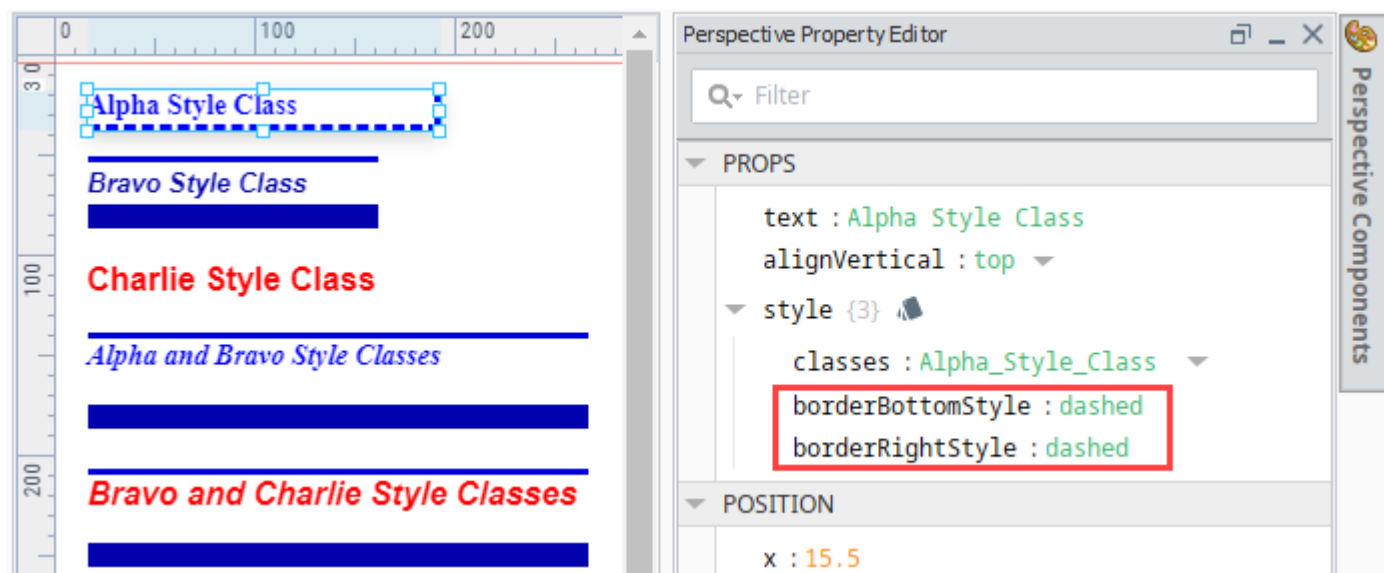
Perspective allows you to select more than one Style Class for a component. Multiple Style Classes are applied to a component in alphabetical order. Style Classes further along in the alphabet will override earlier Style Classes.

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.



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.



Animated Style Classes

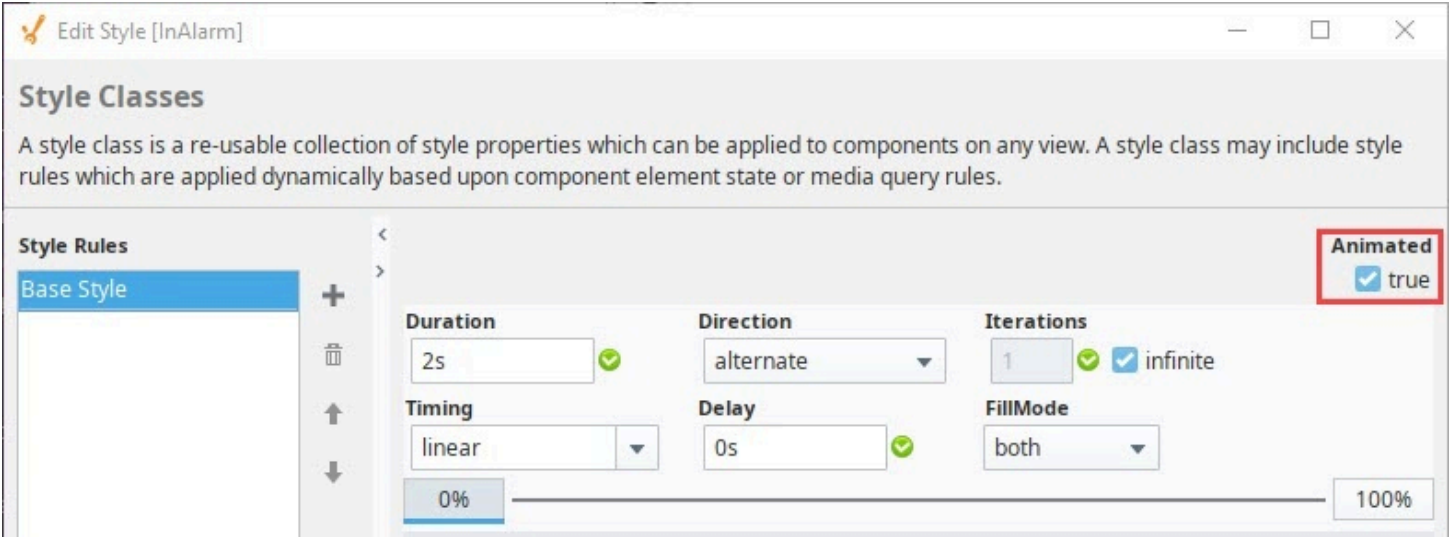


INDUCTIVE
UNIVERSITY

Animated Style Classes

[Watch the video](#)

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 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



INDUCTIVE
UNIVERSITY

Dynamic Style Classes

[Watch the video](#)

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.

Element States on Style Classes



INDUCTIVE
UNIVERSITY

Element States

[Watch the video](#)

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.

Name	Description
Element State	State of the component. See the Element State table for state options.
Animated	If checked, the Element State can be animated. The animate options are as follows: See Animated .

Element State

State	Description
active	The component is being activated by the user. For example, clicking on it with a mouse.
checked	The component is checked or toggled to an "on" state. Applies to checkboxes, radio buttons, etc.
default-choice	The component is the default selection in a group of related options.
disabled	The component is disabled and it cannot be selected, clicked on, typed into, or accept focus.
empty	Represents any element that has no children. Children can be either element nodes or text (including whitespace).
enabled	The component is enabled and can be selected, clicked on, typed into, or accept focus.
first-child	Changes the style for the first element among a group of sibling elements.
fullscreen	Automatically adjust the size, style, or layout of content when elements switch back and forth between full-screen and traditional presentations.
focus	The component receives focus. It is generally triggered when the user clicks or taps on an element or selects it with the keyboard's <code>tab</code> key.
hover	User hovers mouse over the component.
in-range	Element whose current value is within the range limits specified by the <code>min</code> and <code>max</code> attributes.
invalid	The component has an invalid input or selection based on the input type or validation rules.
last-child	The component is the last element among a group of sibling elements.

State	Description
link	The component acts as a hyperlink, navigating the user to a different location when activated.
only-child	The component is the only child of its parent
out-of-range	The component's current value is outside the range limits specified by the <code>min</code> and <code>max</code> attributes
read-only	The component is read-only.
read-write	The component is available for read and write.
required	The component must be filled before submitting a form.
valid	The component has a valid input or selection based on the input type or validation rules.
visited	The component links that the user has already visited.

Animated

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.
Style Settings	Sets a style animation stop. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous.

Media Query on Style Classes



INDUCTIVE
UNIVERSITY

Media Query

[Watch the video](#)

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.

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

Media [min-width]



Media Query

min-width



= 300px

min-width

max-width

orientation

min-aspect-ratio

max-aspect-ratio

hover

Misc

Animated

☐ false

OK

Cancel

Apply

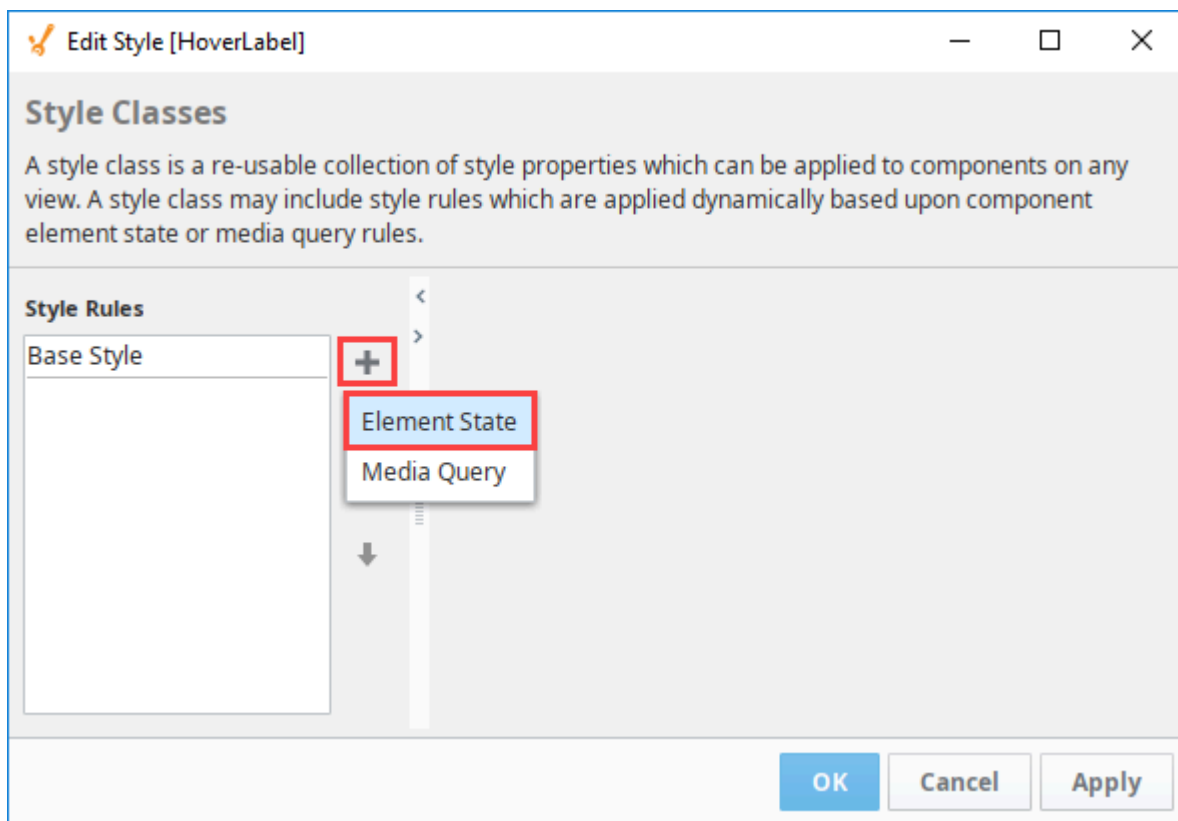
Media Query Settings

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.

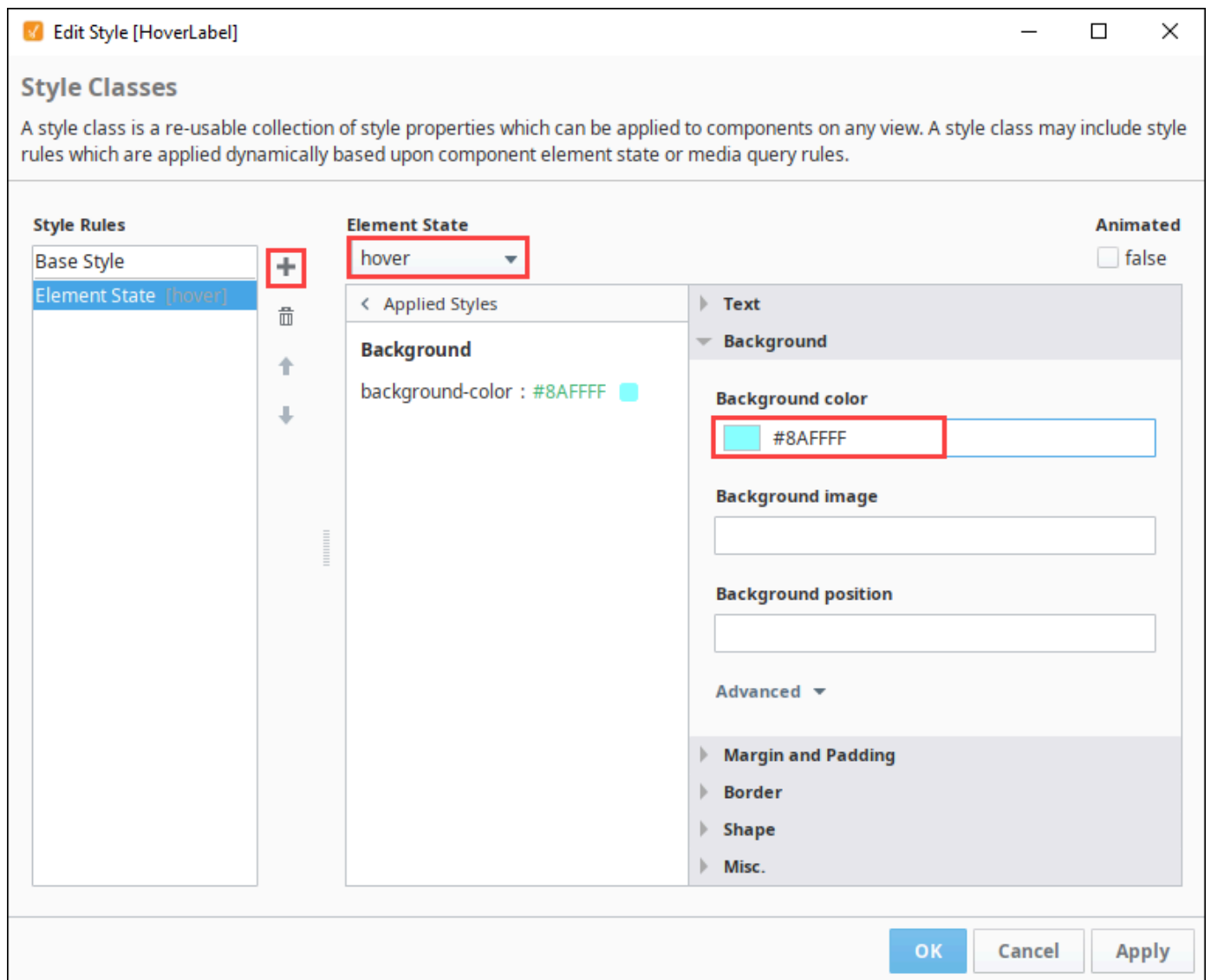
How to Change Style on Hover

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**.

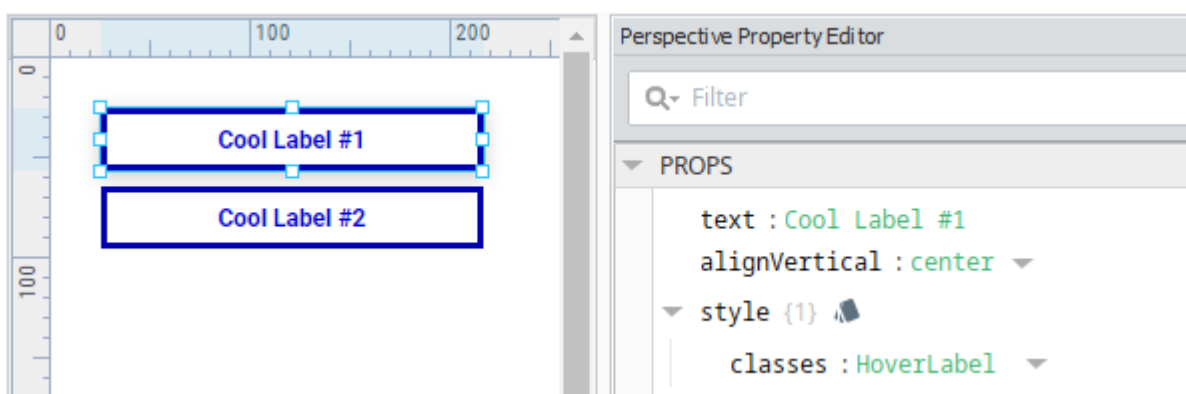


3. On the Element State dropdown, scroll down and select **hover**.
4. Click on the **Expand ►** icon next to Background to see the Background settings.
5. In the **background color** field, enter a color code or click on the color wheel to select a background color. We chose **#8AFFFF**, a light blue.



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.



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.

Cool Label #1

Cool Label #2

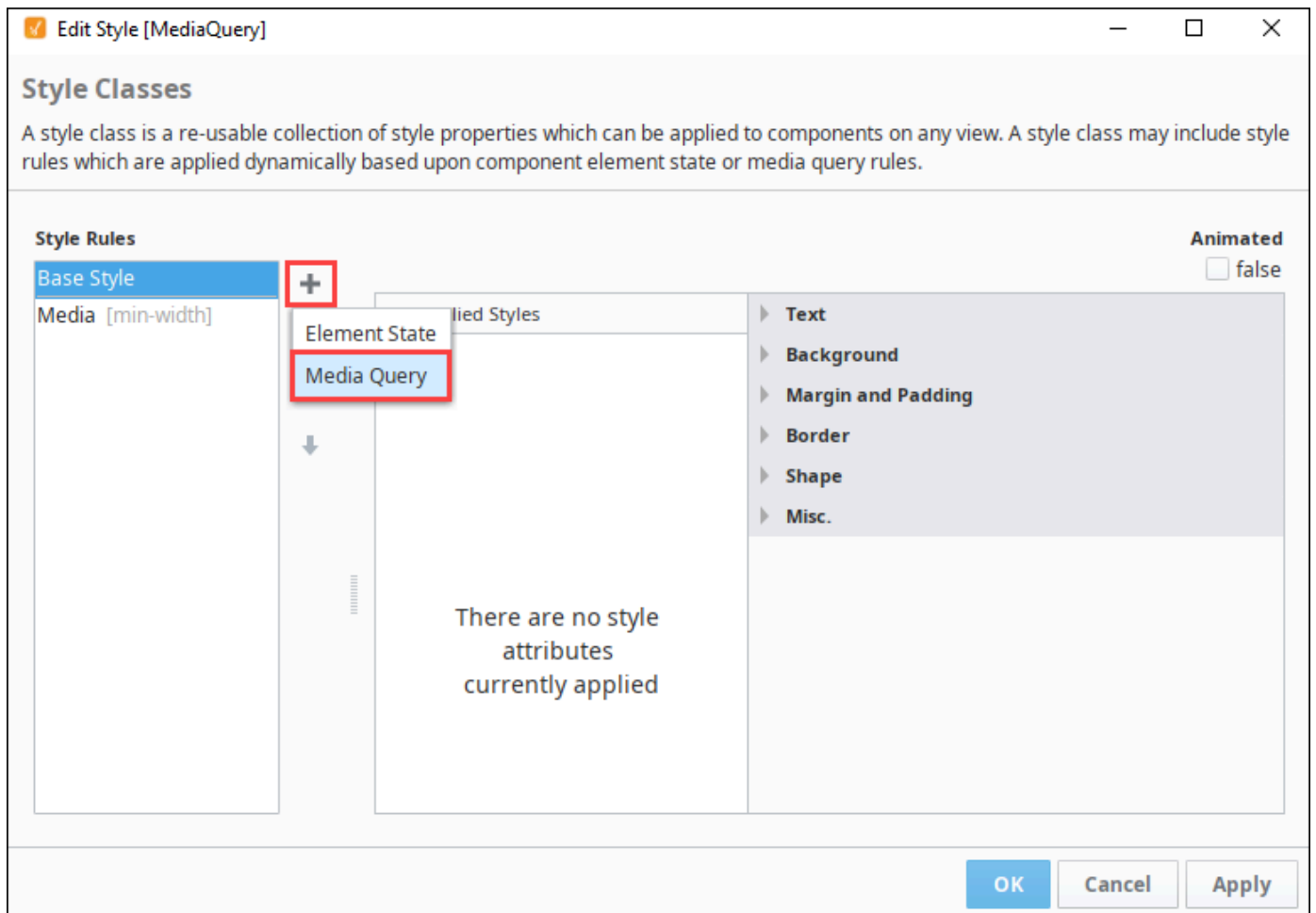


How to Change Text Size Based on Width

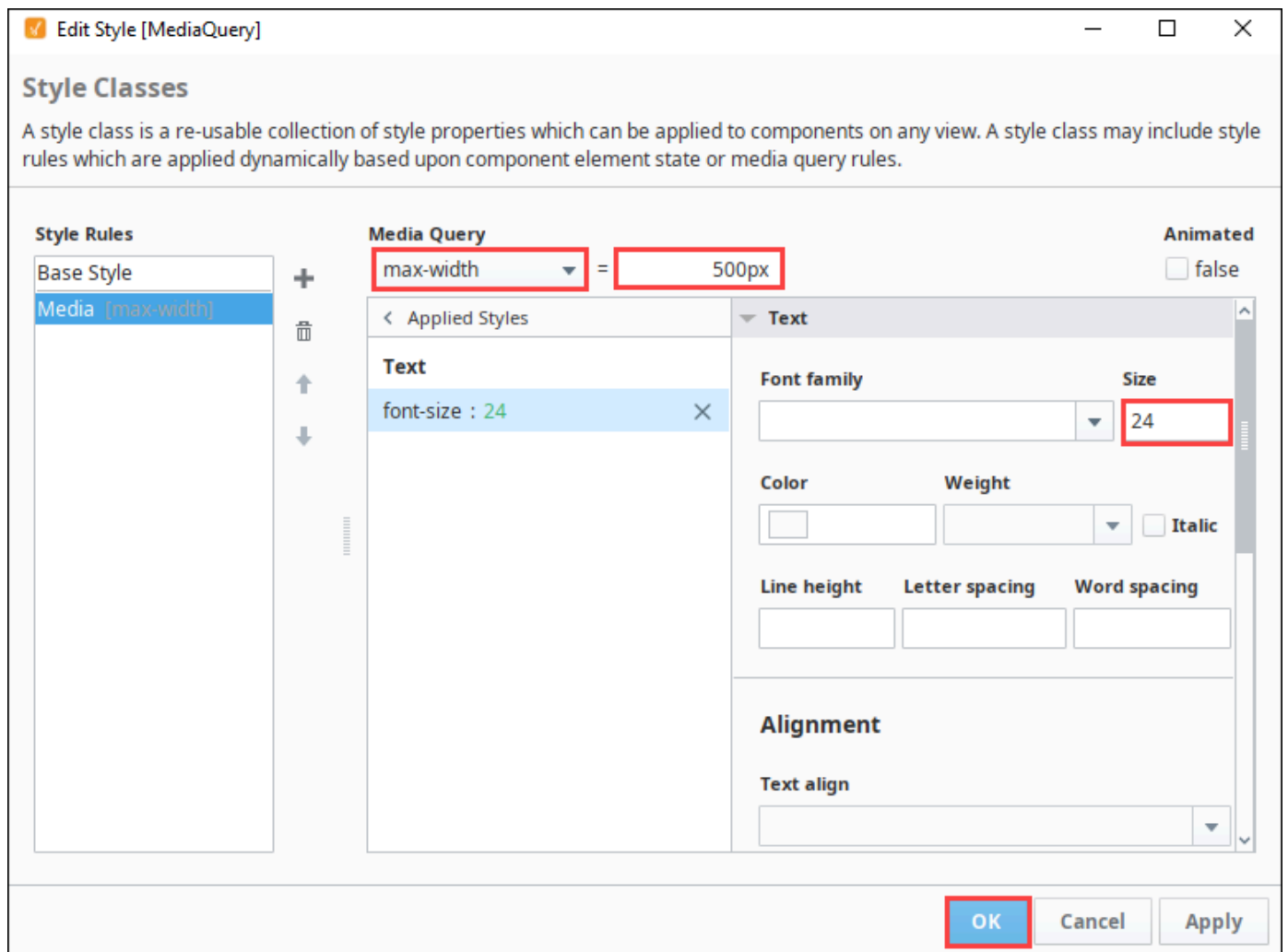
Since media queries react to viewport changes, we can use them to change the text size of a style class. Imagine a scenario where text needed to be larger on a smaller display, such as phone, to improve readability. That same text could likely be smaller on a desktop.

In this example, we will configure a media query that will change the text size for the Style Class when the screen is smaller.

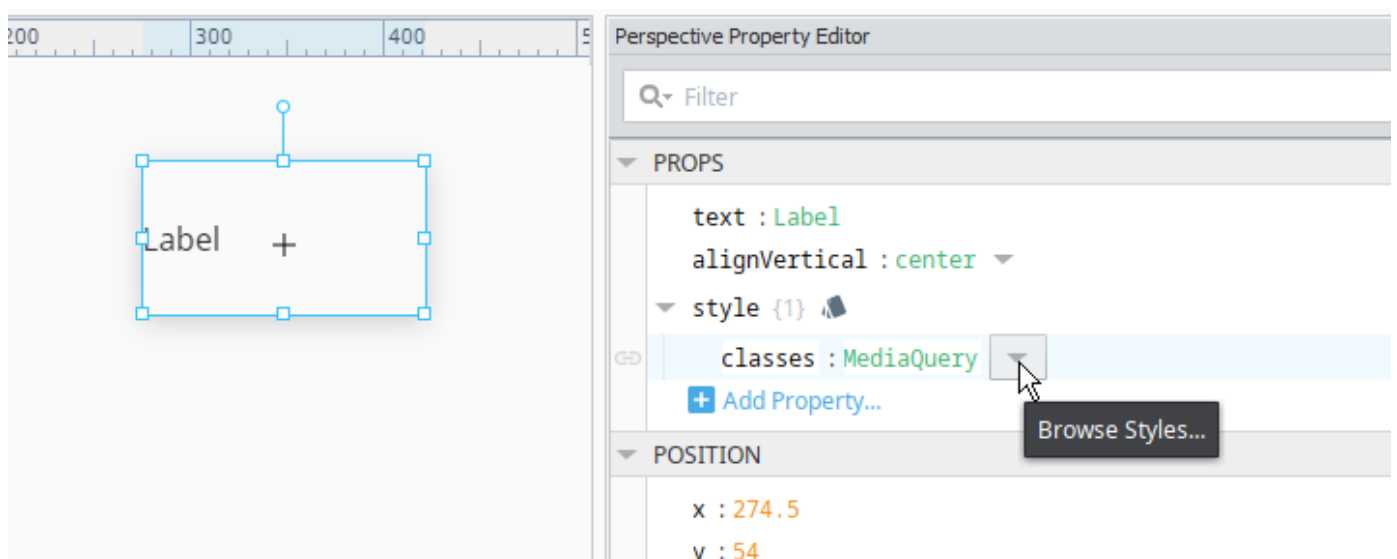
1. Create a new Style Class. We called ours **MediaQuery**.
2. Under Style Rules, click on the **Add +** icon and select **Media Query**.



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.

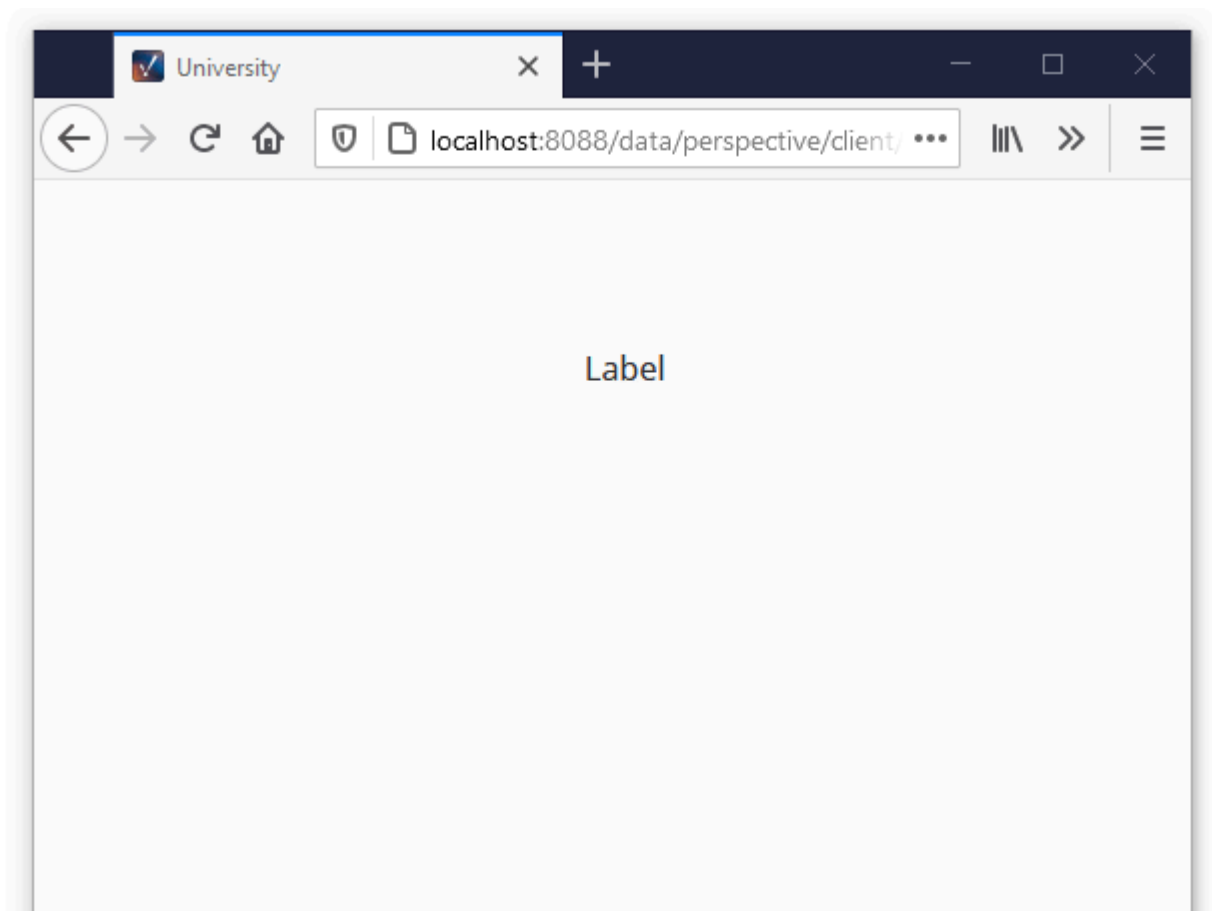


5. Create a new component that contains text, or select an existing component within a container. In this example, we created a new Label component and placed it on a view.
6. Assign the **MediaQuery** Style Class we just created to the Label component using the props.style.classes dropdown to browse 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.



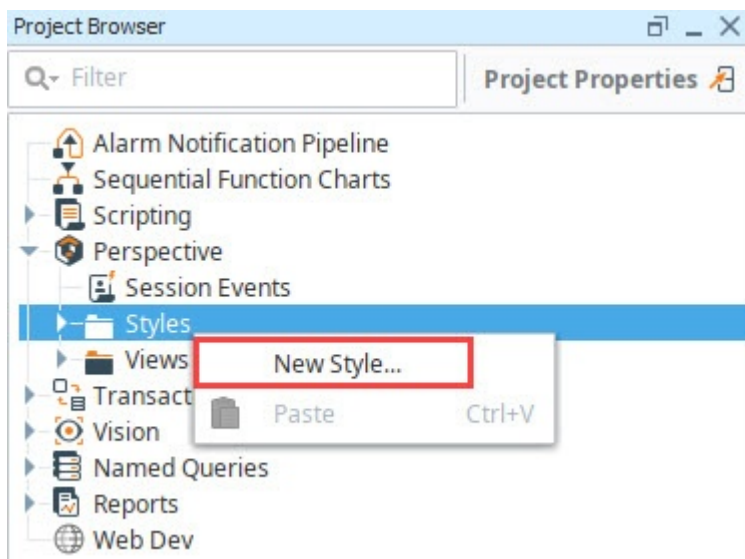
How to Create a Flashing Component

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 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.

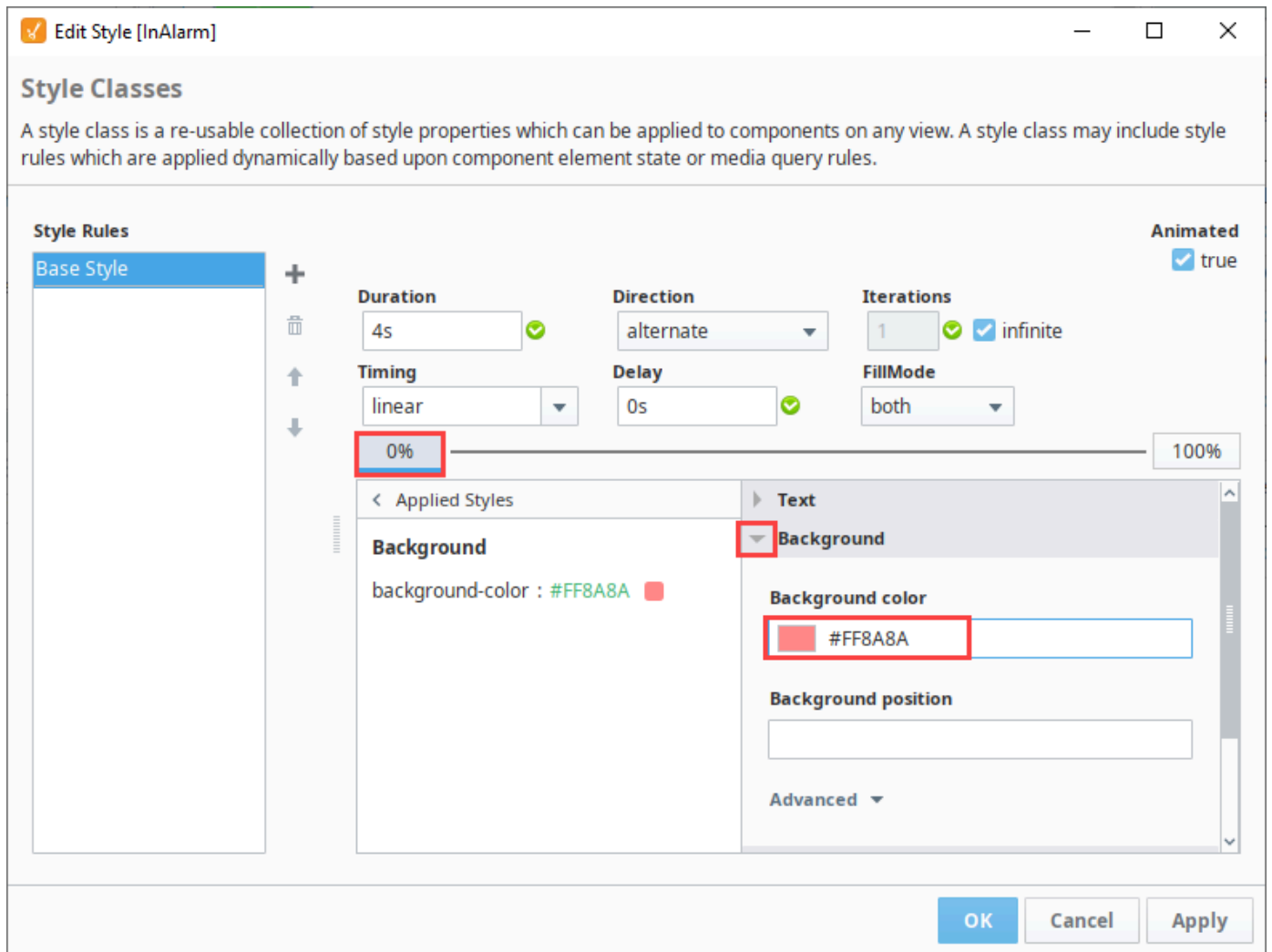
Part 1 - Creating an Animated Style Class

1. In the Project Browser, right-click Styles and select **New Style**.

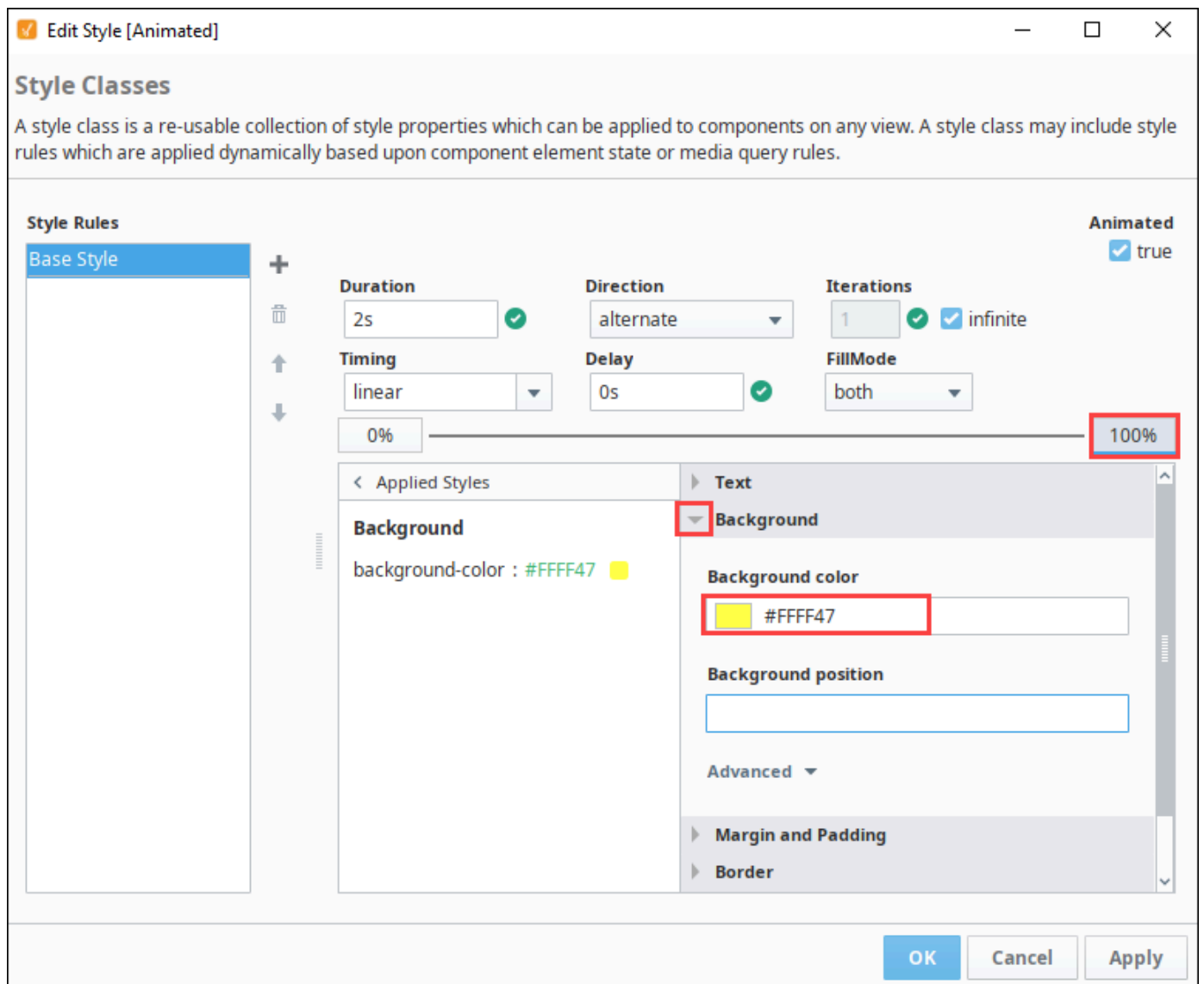


2. Enter a Style Class name, for this example we used "InAlarm", and click **Create Style Class**.
3. On the Edit Style screen, check the **Animated** option in the upper right corner. For this example, we'll leave the default settings in the fields.
4. Next click on **0%** to set the style for the beginning 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.



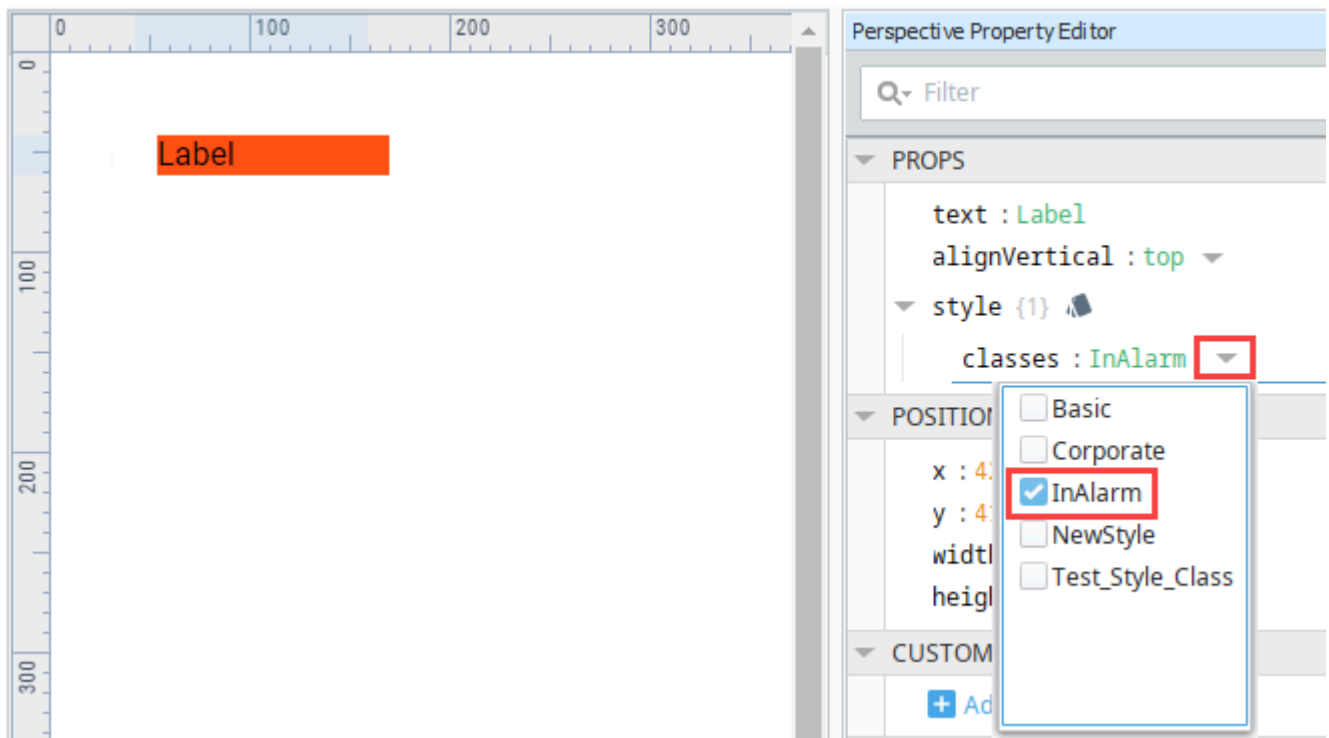
7. Next click on **100%** to set the style for the end of the animation.
8. Click on the **Expand** icon next to Background to see the Background settings.
9. In the Background color field, enter a color code or click on the color wheel to select a background color for the first animation stop. We chose **#FFFF47**, a bright yellow.



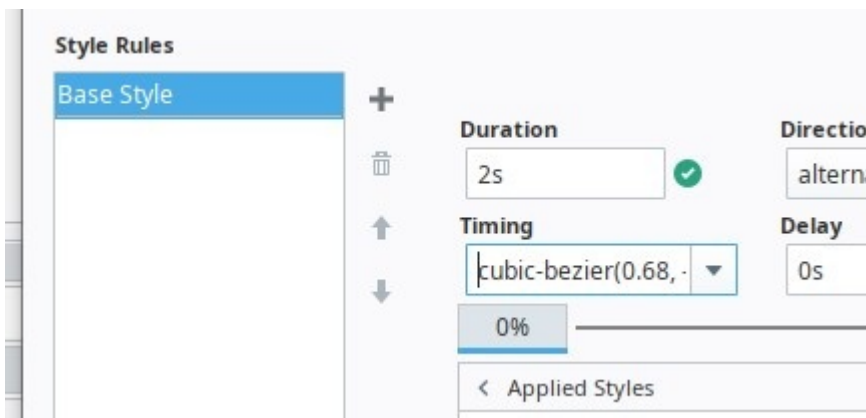
10. Click **OK** to save the Style Class.

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.



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.



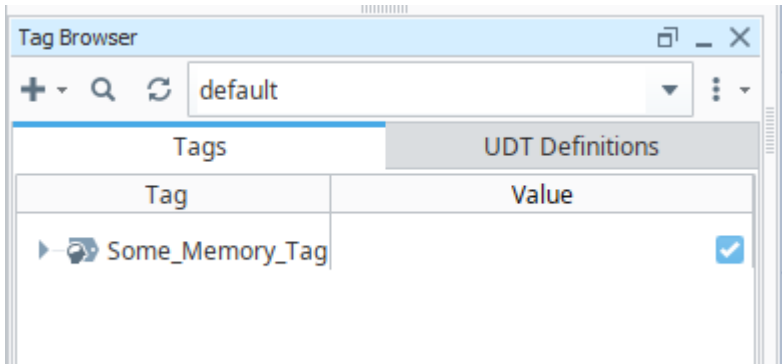
In addition, we can also reduce the overall duration of the animation, making the transition between the two colors occur more frequently. Regardless, we now have an animated style class.

Part 2 - Dynamically Using the Style Class

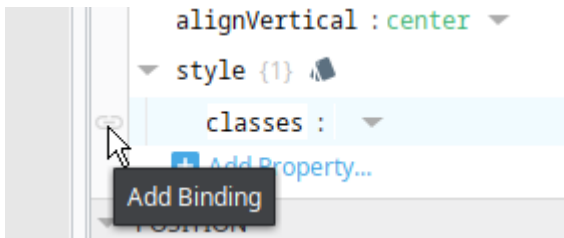
In this part, we can use a binding to conditionally add or remove our animated style class. First, we need to identify what should cause the label to flash. In this example, we'll use a simple tag value on a

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.

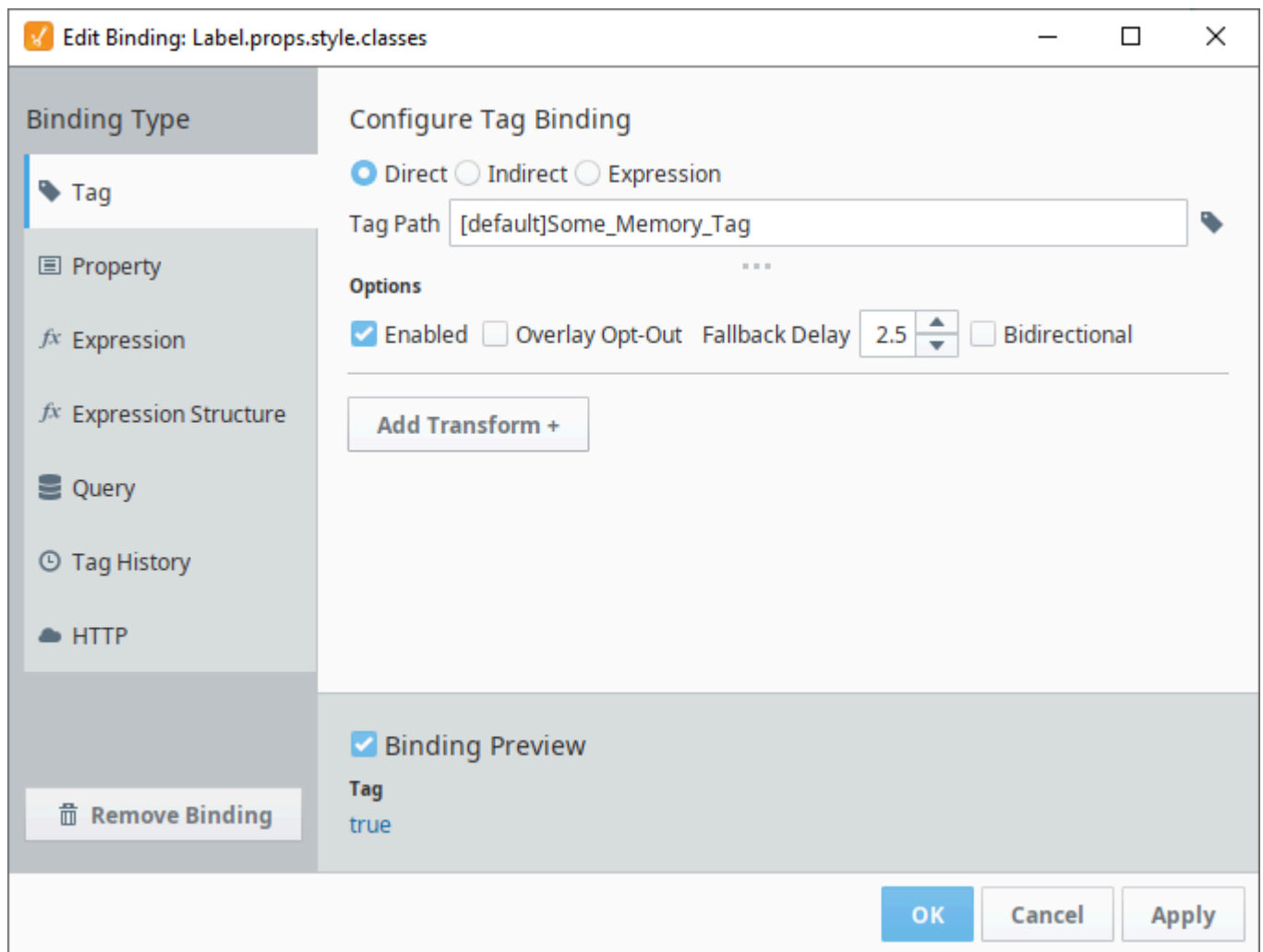
1. Identity the criteria that will cause our component to flash. In our case, we have a Memory Tag with a Boolean data type. When the value is True on the tag, we will cause the label to flash.



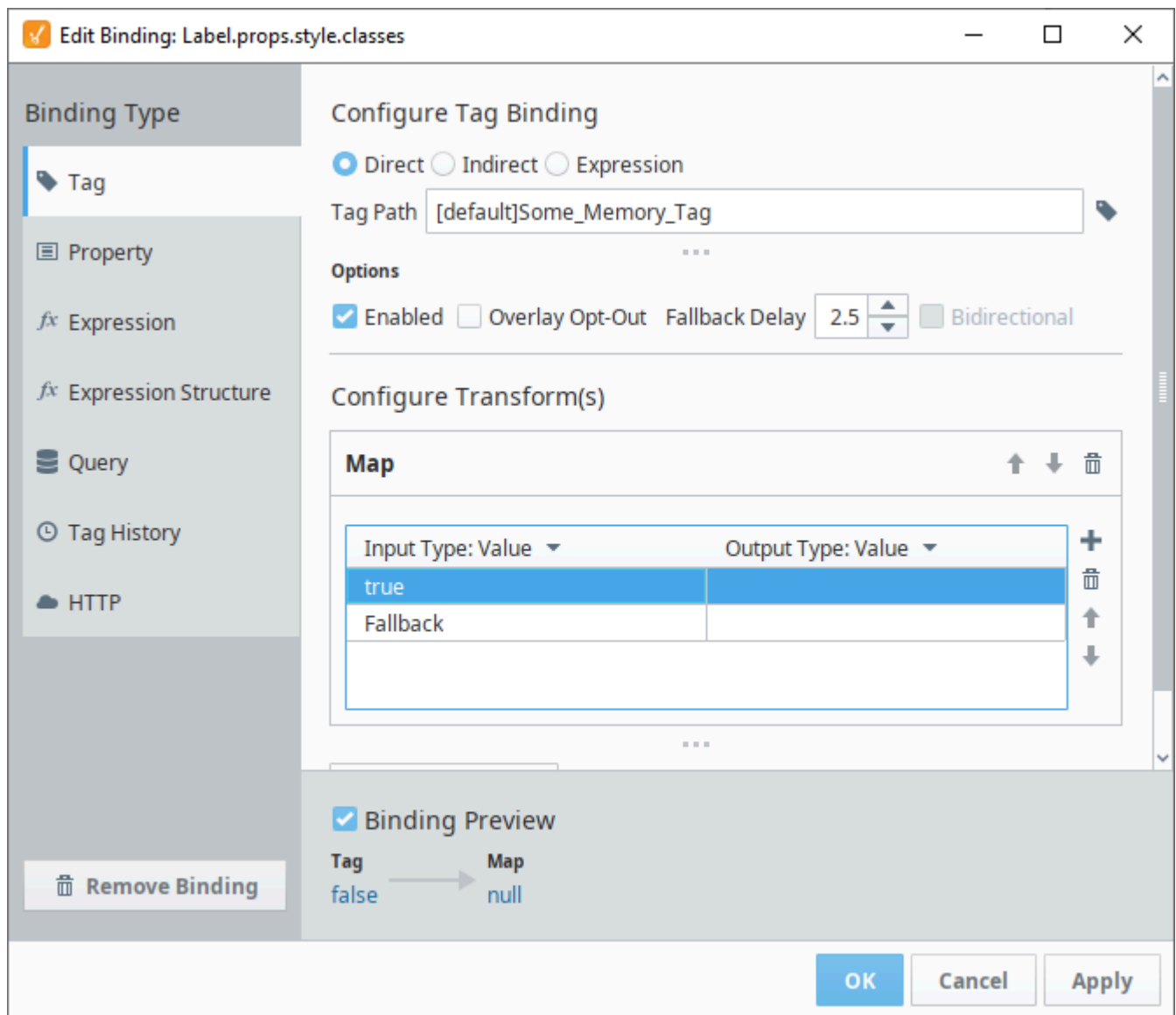
2. With our component selected, place a binding on the **style.classes** property.



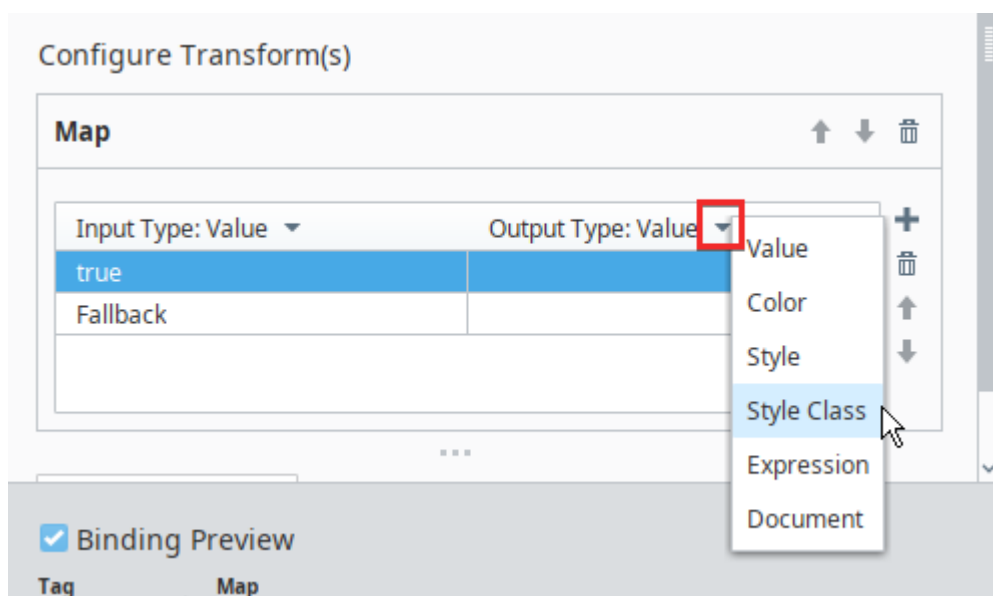
3. Since we want a tag value to determine the flash, we'll select a **Tag Binding**, and set the path to our desired tag.



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.

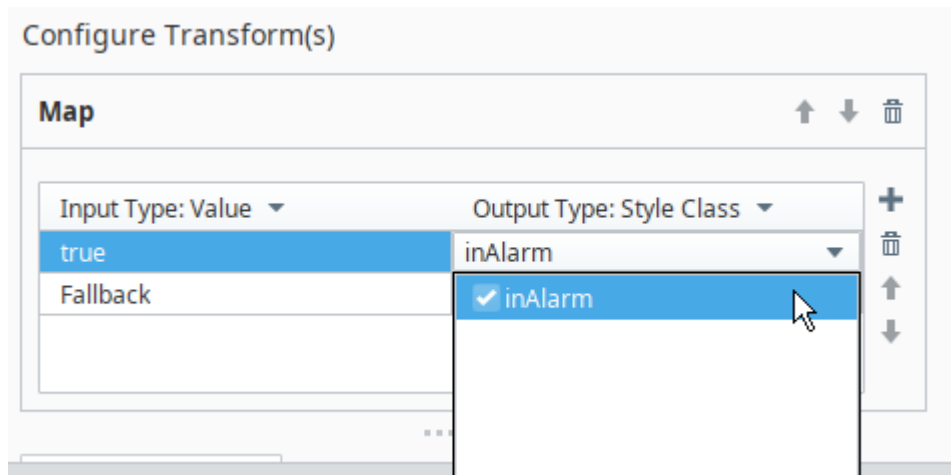


- Change the Output Type column to **Style Class** by clicking on the header over the Output Type column, and selecting Style Class.



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.

- 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".



- Press **OK** to close the binding window.
- 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

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 default themes, which you can learn about on the [Perspective Built-In Themes](#) page.

In some cases, the default themes may not meet your requirements. For these situations, you can create a custom theme.

Creating a Custom Theme

Structure

Before creating your own theme, we recommend that you familiarize yourself with the structure of one of our provided themes, such as `dark-cool`. A valid theme has the following structure:

```
- custom-theme
|- index.css // theme entry configurable in config.json
|- config.json // theme configuration file
|- resource.json // internal resources file
```

Two important properties are within the `config.json`, **entrypoint** and **isPrivate**. By default, the entrypoint of your theme is `index.css`. The `isPrivate` property determines if the theme should be listed as an available theme. By default, this is true. Setting this to false is particularly useful if your theme is shared between multiple themes that are listed as available themes.

Features

CSS Variables

CSS variables have a central role in theme authoring. We recommend leveraging this built-in feature of CSS to aid in modular theme development and ease frustrations of long term maintenance. We use CSS variables throughout our theme stylesheets, so whenever there may be a need to change something like a color, it only needs to be changed in a single file.

Declaring variables

```
--corporateColor: #FF5D00;
```

Using variables

```
background-color: var(--corporateColor);
```

Descriptive CSS Classes

Component authors can specify unique CSS class selectors that give direct access to styling particular elements. Each class selector is unique enough to prevent clashing, and descriptive enough to minimize any guessing surrounding its dedicated purpose. We follow a standard naming convention called ABEM (Atomic Block Element Modifier).

Syntax Structure

```
atomicPrefix_blockName__elementName--modifierName
```

IA Component Example Syntax

```
ia_cylindricalTankComponent__liquid--animation
```

Keep in mind that these unique classes give you direct access to specific elements. You can find a list of available selectors, and their defined rules, in their respective files. If you can not find the selector you need, then it is likely not defined. You may either request it or use one of the many other ways to style elements in Perspective.

CSS Imports

Imports provide the ability to develop or author themes in a modular fashion. They also make overriding easy. There are two rules that we expect you to follow when using imports.

- Imports should be declared at the top of each CSS file.

- Each import statement must begin with the CSS at-rule `@import`, specify the relative path to the file to import, and terminate with a semicolon. Imports that do not meet this criteria may be ignored. You may use single or double quotes surrounding the import path. For example:

```
@import "../button.css";  
@import './variables.css';
```

Using the API

Custom themes can be created via the API documented in the Gateway's [API Documentation](#) interface (/openapi). These themes will be stored within a folder that contains all of the CSS files, as well as metadata files needed for resource collections.

Below is a sample API POST request at

`<gateway_address>/data/api/v1/resources/com.inductiveautomation.perspective/themes` that will create a custom theme with an `index.css` file as an entrypoint.

```
[  
  {  
    "name": "test",  
    "collection": "core",  
    "enabled": true,  
    "description": "test descriptions",  
    "config": {  
      "entrypoint": "index.css"  
    }  
  }  
]
```

POST ▾ {{gateway_address}}/data/api/v1/resources/com.inductiveautomation.perspective/themes

Params

Body

Headers

Auth

Vars

Script

Assert

Tests

Docs

JSON ▾

Prettify

1 ▾

[

2 ▾

{

3

"name": "test",

4

"collection": "core",

5

"enabled": true,

6

"description": "test description",

7 ▾

"config": {

8

"entrypoint": "index.css"

9

}

10

}

11

]

Response

Head

1 ▾

{

2

"suc

3 ▾

"cha

4 ▾

{

5

"

6

"

7

"

8

8f3033

9

}

10

],

11

"prol

12

}

Once the entrypoint is created, you will use the Update Theme Data File endpoint

`<gateway_address>/data/api/v1/resources/datafile/com.inductiveautomation.perspective/themes/test/index.css` to add a custom `index.css` file.

It's recommended to use one of the bundled themes as an example. Any derivative themes (i.e, light-cool, dark, dark-warm) will serve as ideal examples for most use cases. These themes simply extend the base theme, but override the values of the CSS variables that define fundamental colors. If any of the rules defined in the provided base theme are missing from your custom theme, the result will be reflected in your project. For example, a Button component missing a border. When creating your own custom themes, make sure that rulesets declared in the base theme exist in your custom theme at all times.

When you have ensured that the structure is correct and that an entry file has been defined in the theme resource's `config.json`, you can start adding your own changes to your custom theme file. For example, if you wanted to modify the `.ia_button--primary` background and text color, you would use the Update Theme Data File endpoint again with the button changes updated in the payload. Note that order matters for defined rules. Duplicate rules that come after existing rules will override or supplement depending on declared properties between the rules.

i REFERENCING BASE THEMES

Base themes light and dark can be copied into the core resources folder by submitting a POST HTTP request

`<gateway_address>/data/api/v1/resources/com.inductiveautomation.perspective/themes/copy-base-themes`. This is to make referencing the Inductive Automation base themes easier.

The API token used in this request must have write permissions. Any modifications to these copied base themes are ignored.

Using the File System

In addition to the using Ignition's HTTP OpenAPI, you can create the necessary structure directly in the config file system, in the core resource collection. Follow the steps below to create a custom theme without using the API.

1. Navigate to the **themes** folder, located at

```
%IgnitionInstallationDirectory%\data\config\resources\core\com.inductiveautomation.perspective\themes
```

If you do not have this folder, create it and name it **themes**.

2. Within the **themes** folder, create a new folder with the name of your theme. We will name our folder **new-dark-cool**.
3. While we can create the files necessary for our theme manually, we can also simply use the files in existing theme folders and adjust the contents to fit our needs. In our example, we'll copy the files in the **dark-cool** folder (config.json, index.css, resource.json, variables.css) to our **new-dark-cool** folder.

NOTE

If you are creating your theme files manually, consider the following:

- Your structure should match the files in the [Structure](#) section and an entry file should be defined in the theme resource's `config.json` file.
- The order in which you implement your rules matter, since sequential rules will override or supplement existing rules.
- The `light` theme is the base theme, and it is recommended to use it as a foundation for your custom themes. If any rules defined in the provided base theme are missing from your custom theme, those missing properties will be reflected in your project.

4. Modify your files to represent the theme you want and save them. We will keep the default CSS and JSON files for this example.
5. Once you are done configuring your theme, Ignition needs to rescan the file system to bring in your changes. To do this, go to your Gateway Webpage and navigate to **Platform > Overview**, and then click on **Scan File System**.

You can now use your custom theme in your Perspective project.

Overriding or Adapting Themes

Overriding or adapting themes is done by leveraging the "C" in CSS. Simply add your own CSS import pointing to your own custom style sheet containing the rulesets that you want to override or adapt after any existing imports. For example, to override something declared in the built-in light-cool theme, add your own import in the light-cool/index.css entry file like so:

```
@import "../light/index.css"  
@import "../custom/overrides.css"
```

Ignition will only attempt to rewrite this file on startup if it does not exist.

Adding a Custom Font Example

This example demonstrates how to add a custom font to a default theme, with the assumption that the font has already been downloaded. Ignition comes packaged with Roboto, Merriweather, and NotoSans. Here, we will be adding Bitcount to the dark-cool theme.

! INFO

Custom fonts can also be added using `font` [API routes](#) that will automatically build the files needed in this process.

1. Create a **fonts** folder in

```
%IgnitionInstallationDirectory%\data\config\resources\core\com.inductiveautomation  
.perspective
```

 if one doesn't already exist.

2. Copy over the `.tff` files from the downloaded font into the new fonts folder.

3. In the **fonts** folder, create a `resource.json` and `config.json` pair that lists out all of the font files for the resource system.

- The `resource.json` should list each of the font variations and be formatted as shown below.

```

{
  "scope": "G",
  "version": 1,
  "restricted": false,
  "overridable": true,
  "files": [
    "config.json",
    "BitcountPropSingle-Light.ttf",
    "BitcountPropSingle_Cursive-Bold.ttf",
    "BitcountPropSingle_Cursive-Light.ttf",
    "BitcountPropSingle_Cursive-ExtraLight.ttf",
    "BitcountPropSingle-SemiBold.ttf",
    "BitcountPropSingle_Roman-Light.ttf",
    "BitcountPropSingle-Regular.ttf",
    "BitcountPropSingle_Roman-ExtraBold.ttf",
    "BitcountPropSingle_Roman-SemiBold.ttf",
    "BitcountPropSingle_Cursive-Medium.ttf",
    "BitcountPropSingle_Cursive-ExtraBold.ttf",
    "BitcountPropSingle-Medium.ttf",
    "BitcountPropSingle_Roman-Thin.ttf",
    "BitcountPropSingle_Cursive-Black.ttf",
    "BitcountPropSingle_Cursive-Thin.ttf",
    "BitcountPropSingle_Roman-Regular.ttf",
    "BitcountPropSingle-ExtraLight.ttf",
    "BitcountPropSingle_Roman-Bold.ttf",
    "BitcountPropSingle_Roman-Medium.ttf",
    "BitcountPropSingle-Thin.ttf",
    "BitcountPropSingle-Black.ttf",
    "BitcountPropSingle_Cursive-SemiBold.ttf",
    "BitcountPropSingle_Cursive-Regular.ttf",
    "BitcountPropSingle_Roman-ExtraLight.ttf",
    "BitcountPropSingle-Bold.ttf",
    "BitcountPropSingle_Roman-Black.ttf",
    "BitcountPropSingle-ExtraBold.ttf"
  ],
  "attributes": {
    "lastModification": {
      "actor": "theme-manager",
      "timestamp": "2025-11-12T13:41:04Z"
    },
    "lastModificationSignature":
    "724d740b276edf98109505e83b86e00d64d2f869b9a633962c7acb1fc58ca92d"
  }
}

```

- The `config.json` is needed for folder structure requirements, but will be empty `{}` for now.

4. On the Gateway, navigate to **Platform > Overview** and click **Scan File System** to sync the Gateway with the updated system files.
5. Verify the font file is now available by navigating to the font file path. In this instance, we can test it at `http://localhost:8088/data/perspective/fonts/BitcountPropSingle-Black.ttf`.

TIP

If you are running into a 404 error when testing your font with the link above, try recreating `resource.json` and `config.json` outside the root directory, and then manually moving both `.json` files into the fonts folder.

6. Return to the **fonts** folder to create a corresponding `fonts.css` file to define available fonts with `@font-face` details and included `src` paths.

Available Fonts CSS Snapshot

```
@font-face {
  font-family: 'Bitcount Prop Single';
  src: url('/data/perspective/fonts/BitcountPropSingle-Thin.ttf');
  format('truetype');
  font-weight: 100;
  font-style: normal;
}

@font-face {
  font-family: 'Bitcount Prop Single';
  src: url('/data/perspective/fonts/BitcountPropSingle-ExtraLight.ttf');
  format('truetype');
  font-weight: 200;
  font-style: normal;
}

@font-face {
  font-family: 'Bitcount Prop Single';
  src: url('/data/perspective/fonts/BitcountPropSingle-Light.ttf');
  format('truetype');
  font-weight: 300;
  font-style: normal;
}
```

7. Add references to the `fonts.css` file within the dark-cool theme's `resource.json` and `index.css` files, located in the theme's folder at

```
%IgnitionInstallationDirectory%\data\config\resources\core\com.inductiveautomation  
.perspective\themes\dark-cool.
```

- o resource.json:

```
{  
  "scope": "G",  
  "description": "The dark-cool theme for Perspective.",  
  "version": 1,  
  "restricted": false,  
  "overridable": true,  
  "files": [  
    "config.json",  
    "index.css",  
    "variables.css",  
    "fonts.css"  
  ],  
  "attributes": {  
    "lastModification": {  
      "actor": "theme-manager",  
      "timestamp": "2025-11-12T13:41:04Z"  
    },  
    "lastModificationSignature": "724d740b276edf98109505e83b86e00d64d2f869b9a633962c7acb1fc58ca92d"  
  }  
}
```

- o index.css:

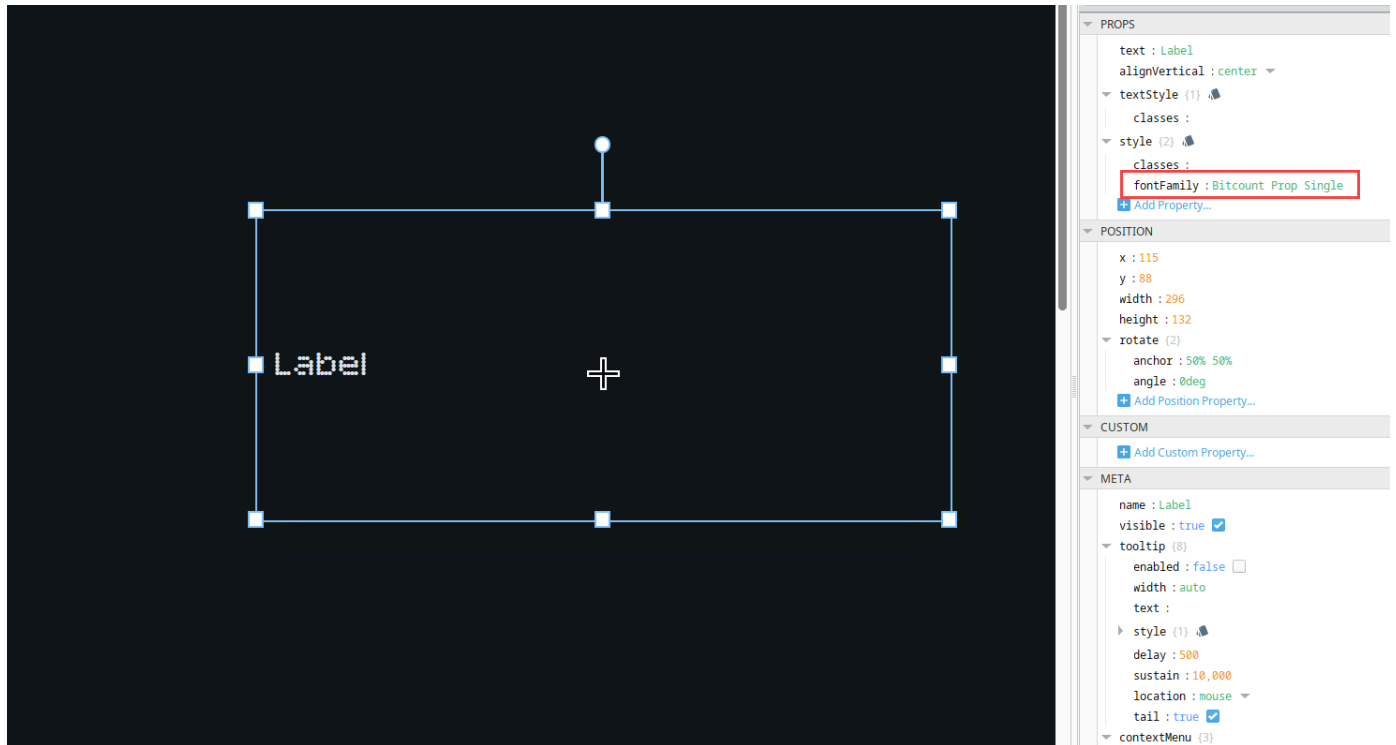
```
1  @import "../variables.css";  
2  @import "../light/fonts.css";  
3  @import "../fonts.css";  
4  @import "../dark/globals.css";  
5  @import "../light/app/index.css";  
6  @import "../light/common/index.css";  
7  @import "../light/designer/index.css";  
8  @import "../light/palette/index.css";  
9  @import "../dark/palette/index.css";  
10
```

Test the Custom Font in the Designer

1. Open the Designer, and navigate to Perspective in the Project Browser.
2. Make sure the dark-cool theme is selected as the `theme` property under Perspective Session Props.
3. Place a **Label** component in a view.

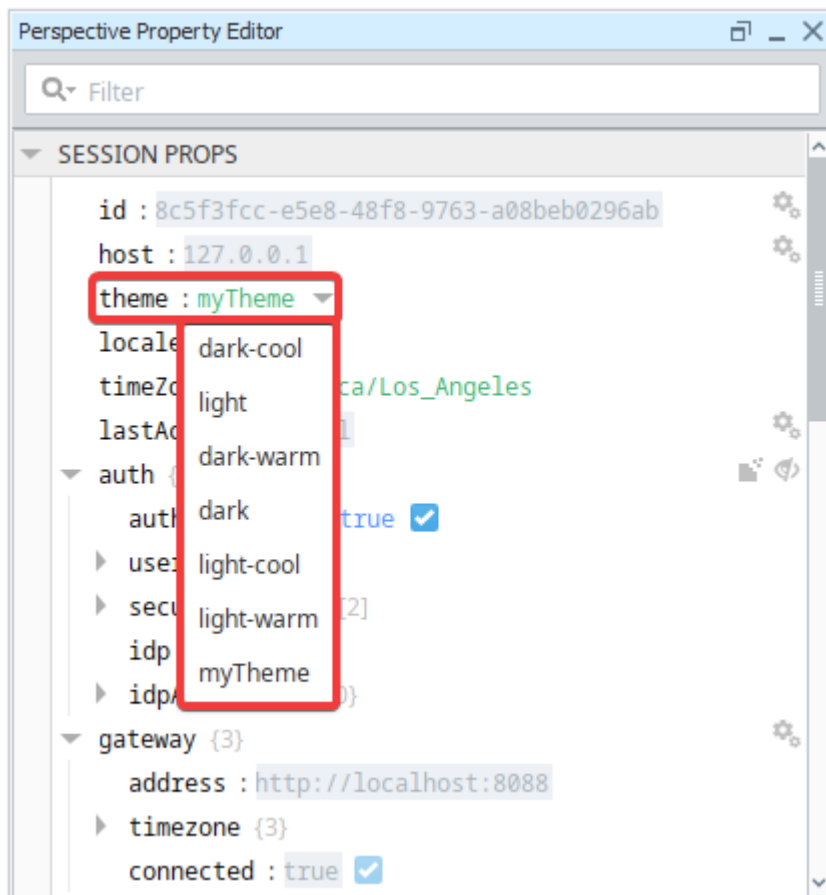
4. In the component property panel, select the **plus**  icon for style and select **Properties > fontFamily**.

5. Enter **Bitcount Prop Single**.



Using a 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. Keep in mind that changes made to the **theme** property will become active only after a Gateway restart.



Once you select your custom theme and restart your Gateway, your components will change appearance according to your `.css` file.



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.

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 scripts can be run in either the Gateway or Perspective Session scopes. All Perspective scripting is run on the Gateway, however session-specific functions will only affect a single session. Perspective does not have a Client scope, because unlike Vision, Perspective does not have Clients.

See the [System Function](#) section for more information about the available scopes for each function.

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.	<pre>self.props.text = "My Text" self.props.startAngle = 5</pre>
Object	An object is structured like a Python Dictionary, holding any number of elements. Each element can be a value, object, or an array.	<pre>motorObject = {"motorNum":1,"motorState":"Running"} self.custom.myObject = {"operationNum":15,"motorObject":motorObject}</pre>
Array	An array is structured like a Python List. Unlike an object	<pre>rowObject1 = {"city":"Folsom","country":"United States", "population":77271} rowObject2 = {"city":"Helsinki","country":"Finland","population":651000} self.props.data = [rowObject1,rowObject2]</pre>

Type	Description	Example
	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.	

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 popup for 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.

Perspective Component Methods

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**:

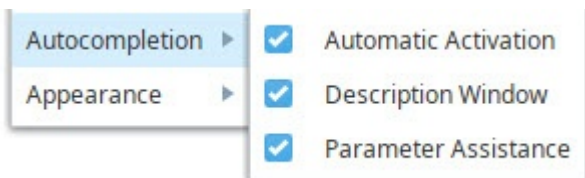
```
output = self.myMethod(param1,param2)
```

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

A complete list of component methods (with their definitions) is available from the autocomplete popup. 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**.



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.	<pre>self.children</pre>
.getChildren()	Functionally similar to ".children" above.	<pre>self.getChildren()</pre>
.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.	<pre>self.parent</pre>
.getParent()	Functionally similar to ".parent" above.	<pre>self.getParent()</pre>
.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.	<pre>self.getChild('Label_0') self.getChild('Container/Label_0')</pre>
.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').	<pre>self.getSibling('Label')</pre>

Method/Property	Description	Example
.view	Calling this from anywhere within a view will return the parent view of the object.	<pre>self.view</pre>
.getView()	A method that will return the view, similar to ".view" above.	<pre>self.getView()</pre>
.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.	<pre>page = self.page pageID = page.props.pageId pagePath = page.props.path</pre>
.getPage()	A method that will return the page, similar to ".page" above.	<pre>self.getPage()</pre>
.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.	<pre>session = self.session sesName = session.props.gateway.sesName sesProp = session.custom.property</pre>
.getSession()	A method that will return the session, similar to ".session" above.	<pre>self.getSession()</pre>

Session Objects

Method/Property	Description	Example
<code>.close()</code> or <code>.close(message)</code>	When called will close the session. Optionally accepts a string message which will be displayed after the session closes.	<pre>session = self.getSession() session.close("The session ha</pre>
<code>.getInfo()</code>	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.	<pre>self.session.getInfo()["pageI</pre>
<code>.getPages()</code>	Returns a list of page objects.	<pre>self.session.getPages()</pre>
<code>.getPage(ID)</code>	Returns the page associated with the given string ID parameter, if it exists. Page ID values can be determined with the <code>.getInfo()</code> method.	<pre>self.session.getPage("2b2eb64</pre>
<code>.getProjectInfo()</code>	Returns a dictionary of project meta data, including name, title, description, lastModified, lastModifiedBy, views, and pageConfigs.	<pre>self.session.getProjectInfo()</pre>

View Objects

Method/Property	Description	Example
.rootContainer	Returns the root container of the View.	<pre>view.rootContainer</pre>
.id	Returns a string that uniquely identifies the view instance. For example: <pre>Perspective/Views/path/to/view</pre> <pre>yields path/to/view@C</pre>	<pre>view.id</pre>
.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.	<pre>session = view.session sesName = session.gateway.address sesProp = session.custom.properties</pre>

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 `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 `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 `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')
```

The following example demonstrates how to use the `requestPrint` method for the various expected targets of component, view, or page. If left blank, the default target will be component:

```
self.requestPrint(target='component', documentTitle='MyTitle')
self.requestPrint(target='view', documentTitle='MyTitle')
self.requestPrint(target='page', documentTitle='MyTitle')
```

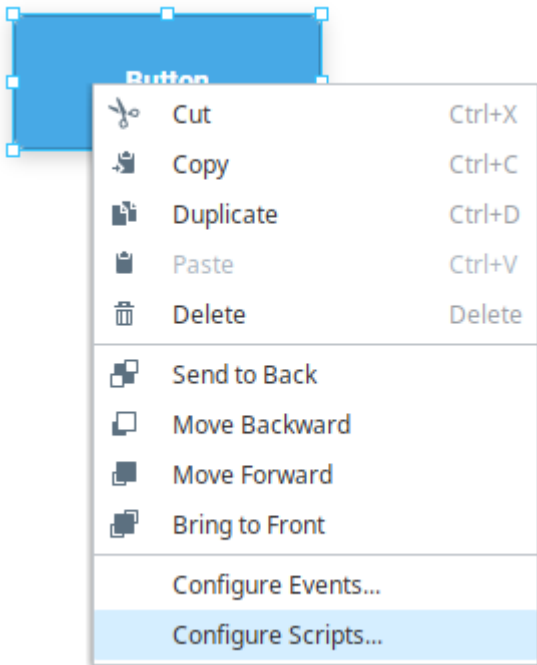
The following example demonstrates how to invoke the `requestPrint` method on other components:

```
self.getSibling("ToggleSwitch").requestPrint(target='component', title='MySwitch')
```

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....**

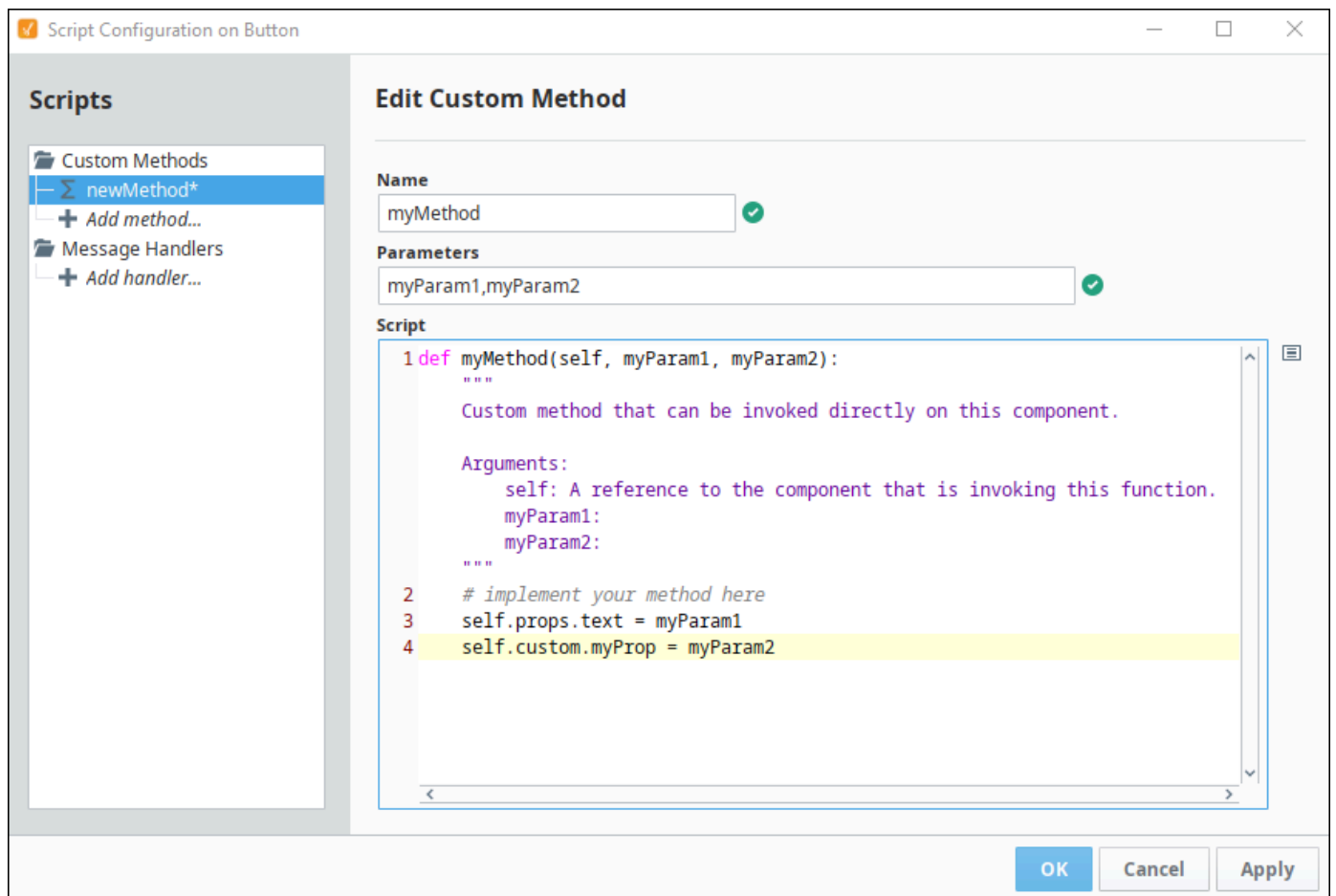


2. Under **Custom Methods**, double click on **+ Add method....**
3. Enter a **Name** for your method, which will be used to call the method. We used **myMethod** in this example.
4. Enter any number of **Parameters** your method will need, separated by commas. For this instance, we have two parameters: **myParam1** and **myParam2**.
5. Add code to implement your method.

```
self.props.text = myParam1
self.custom.myProp = myParam2
```

i NOTE

A self object is provided in every custom method, but should not be provided as a parameter when calling the function.



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")
```

Component Message Handlers



INDUCTIVE
UNIVERSITY

Message Handlers

[Watch the video](#)

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.

Object Traversal



INDUCTIVE
UNIVERSITY

Component Paths

[Watch the video](#)

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

A message type is a string used to identify a Message Handler.

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 types are case-sensitive, so message types `foo`, `Foo`, and `F00` are considered separate types.

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.

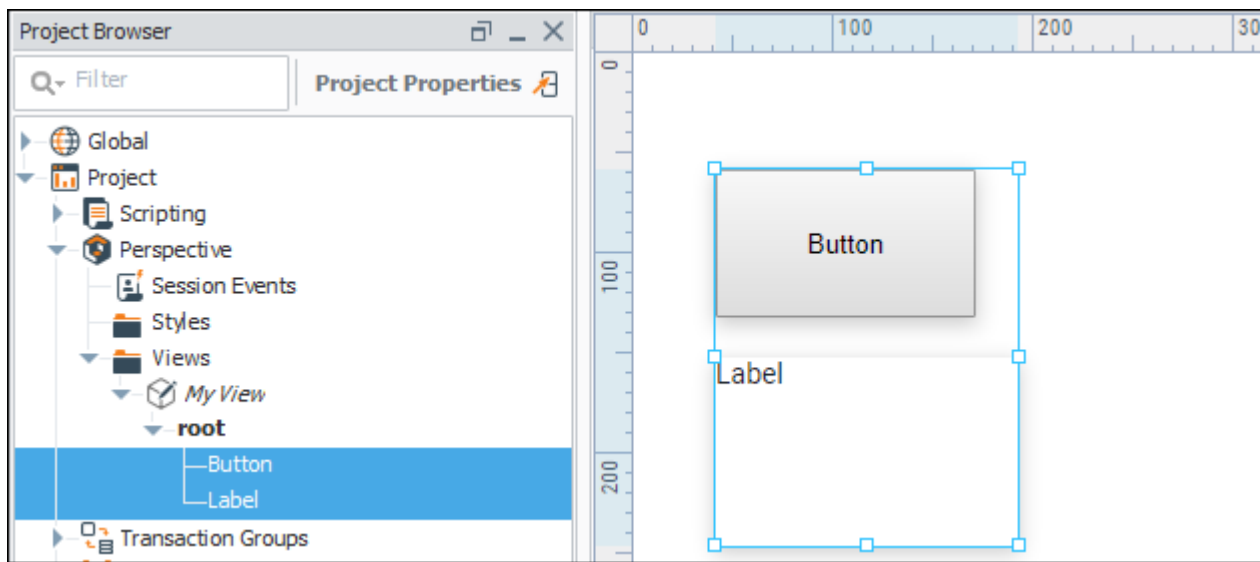
There are two ways to limit the scope of a message:

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.



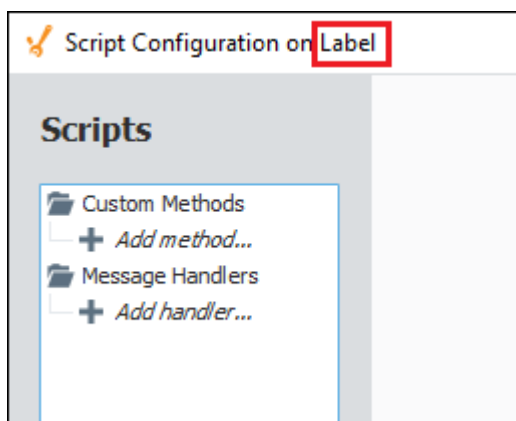
Step 2 - Create A Message Handler

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.

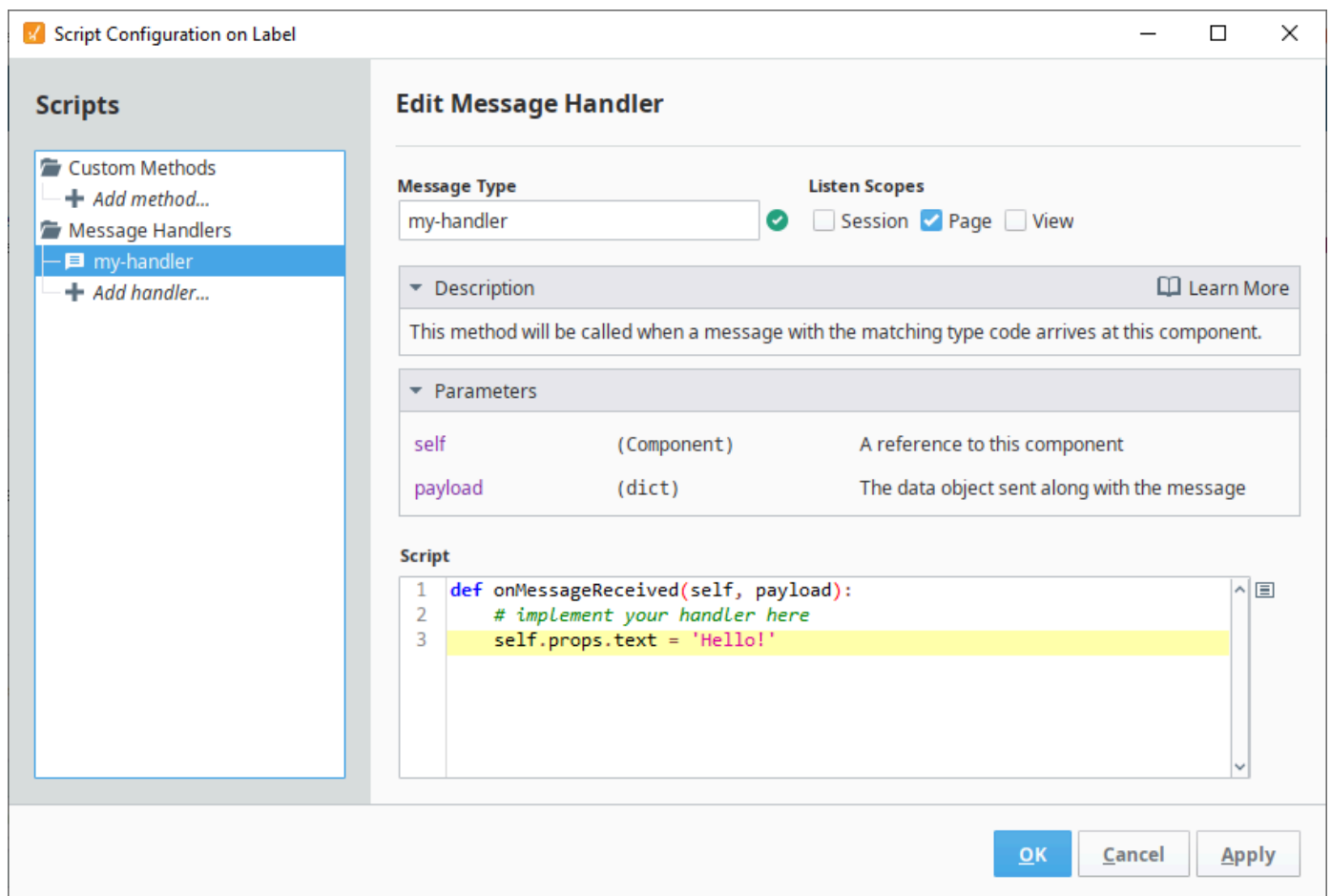


- On the left side of the **Script Configuration** window, you will see a tree. Double-click on the **Add handler...** item.
- A new handler named "type-name-here" will appear. Under **Message Type** type `my-handler`. Note that we're using all lowercase characters: **Message Handlers are case-sensitive**. 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.
- 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!'
```

- 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.



- 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

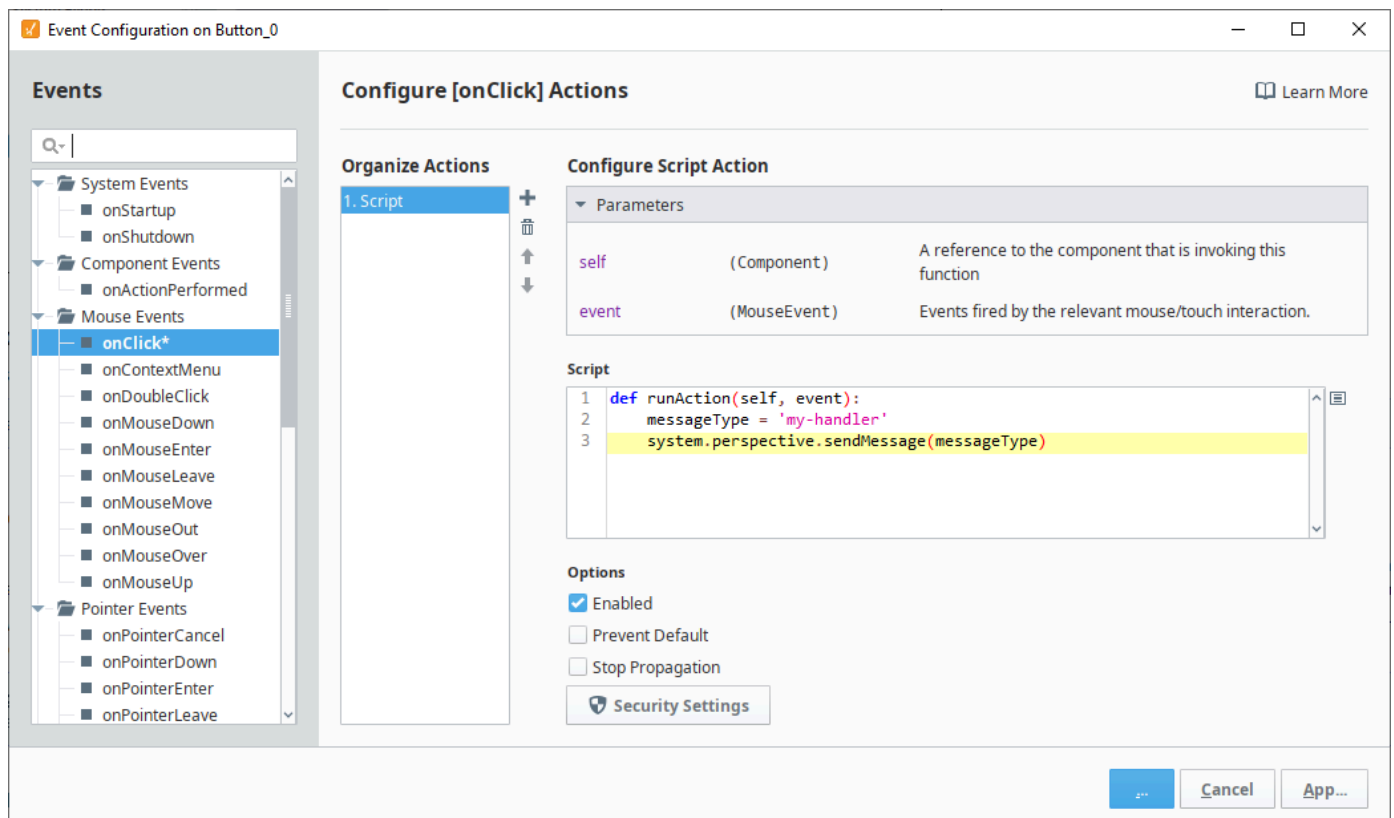
In this step, we place a script on the Button that will call our Message Handler.

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.



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.

Messages Execute Asynchronously

When sending messages from a script, the called Message Handler will execute in a separate thread. This means the script that sent the message will **not** wait for the Message Handler to execute its own script. Additionally, this means a script where `system.perspective.sendMessage` was invoked will continue executing, running, and possibly completing before the script on the Message Handler.

For example, consider the following script action on a button's `onActionPerformed` event:

```
system.perspective.sendMessage('my-handler')
system.perspective.print('running')
```

The print function will likely be invoked before the `my-handler` Message Handler finishes. This is because the component script is not waiting for a response from `my-handler`. Since a message was broadcast, the component script continued to the next line.

NOTE

Due to the non-deterministic timing, it is still possible for the Message Handler to complete before the original script executes. Just as you cannot rely on the Message Handler to block the invoking thread, you cannot rely on the invoking thread to block the Message Handler.

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.

Configure [onClick] Actions

Organize Actions	Configure Script Action
1. Script	<pre>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</pre>

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.

Perspective Property Change Scripts



INDUCTIVE
UNIVERSITY

Property Change Scripts

[Watch the video](#)

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.

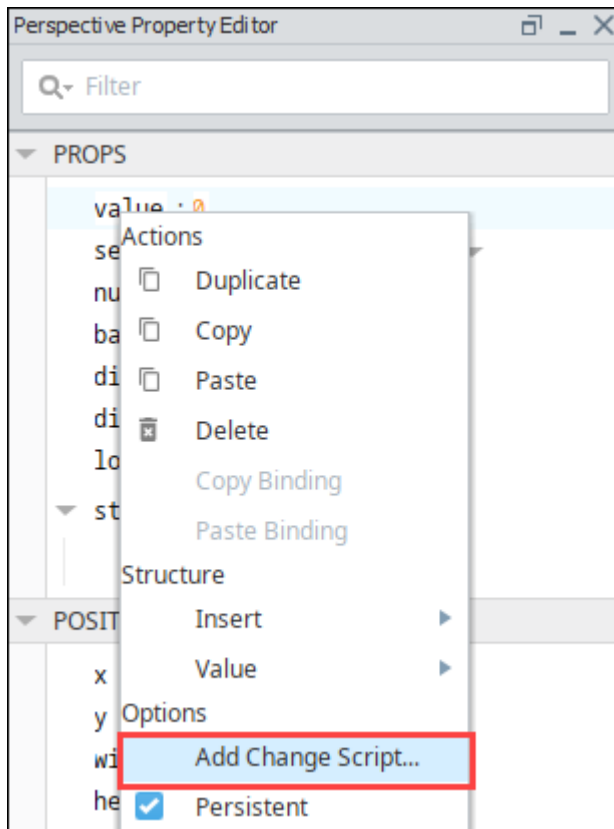
A very common example of a property change script would be to take the dataset from a binding and modify it into a new dataset using other information on screen. This can be accomplished with a Script Transform instead.

NOTE

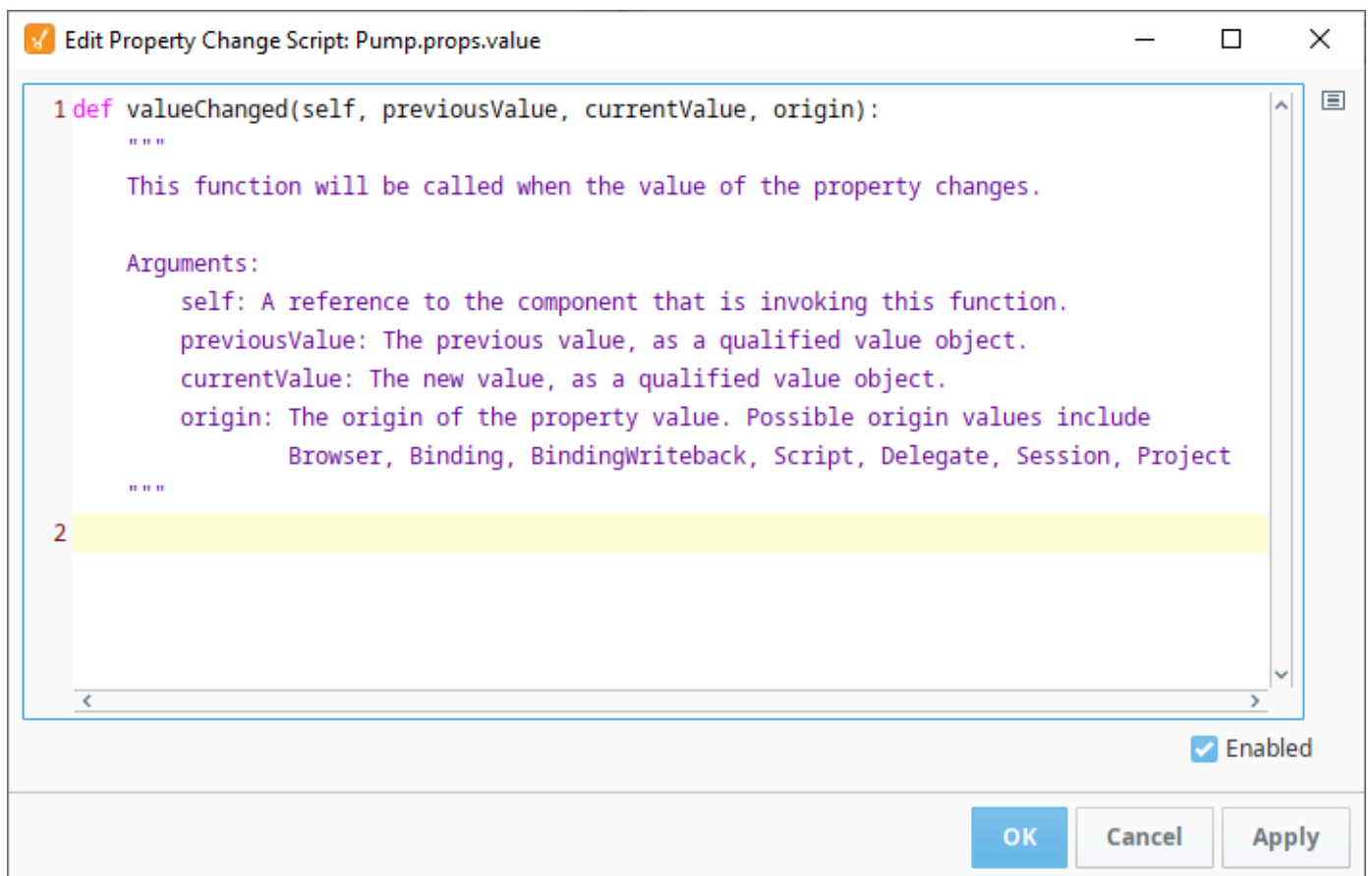
Expression bindings are generally quicker and have less impact on session performance than a script. Before creating a property change script, consider if the goal of the script can be accomplished with an expression binding instead.

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.



3. Type in the script that you want to run and click **OK**.

Property Change Arguments

`self`

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.

`previousValue`

The previous value, as a QualifiedValue object. QualifiedValue objects have a value, quality, and timestamp. See also [Scripting Object Reference](#).

`currentValue`

The new value, as a QualifiedValue object. QualifiedValue objects have a value, quality, and timestamp. See also [Scripting Object Reference](#).

`origin`

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:

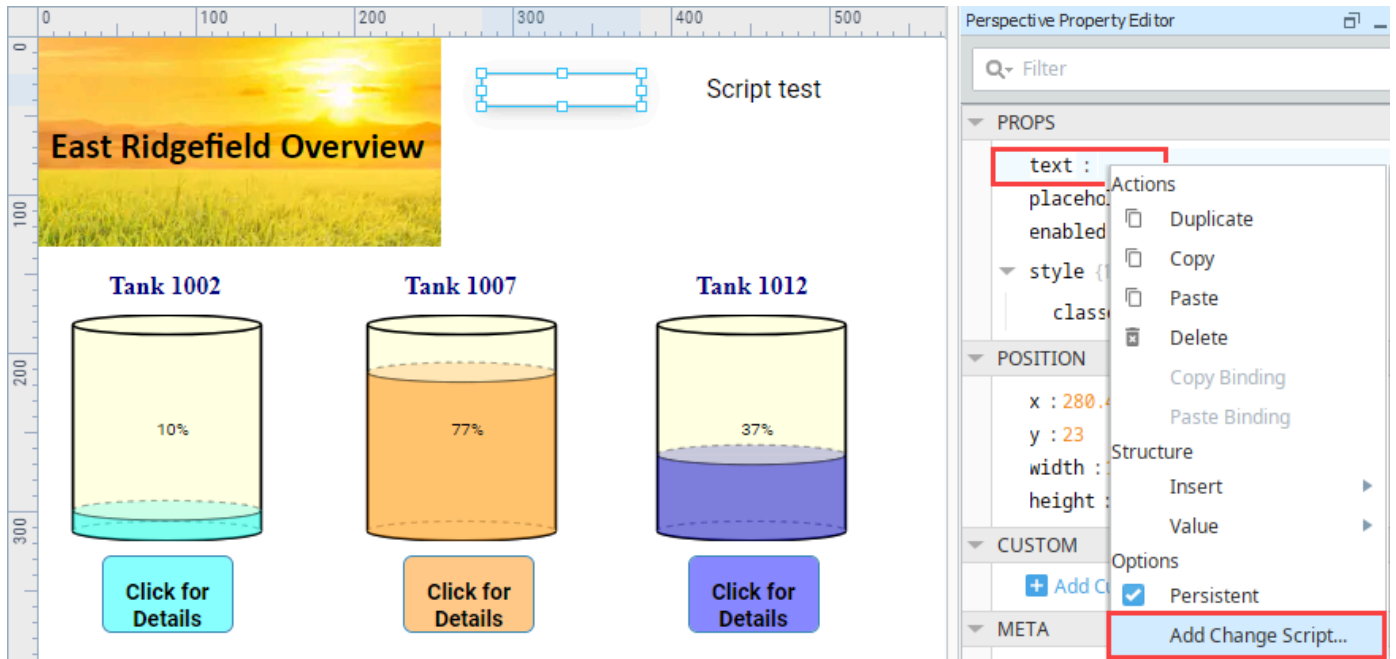
Name	Description	Example
Browser	Used when the change comes from the Browser interface.	The user changes the text property on a text field by typing a word into the field.
Binding	Used when the change comes from a binding (or transform) generating a new value.	A tag changes value, and a property with a binding to that tag is updated.
BindingWriteback	Used when the change comes from a bidirectional binding writing back to its source.	ComponentB's value property has a bidirectional binding to the value property on ComponentA. If ComponentB's value changes, then a property change script on ComponentA will have an origin of BindingWriteback .
Script	Used when the change comes from a script.	A user presses a button, and a script on the button assigns a new value to a custom property.
Delegate	Used when a change to a property comes from something intrinsic to the component's design.	A complex component that automatically fills itself with data, like the alarm status table component.
Session	Used when the session itself causes the property change.	A change in user privileges causes access to be revoked, resulting in a change in the auth session property.
Project	Used when the default property value is changed in the designer and saved.	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.

missedEvents

A flag indicating that some events have been skipped due to event overflow.

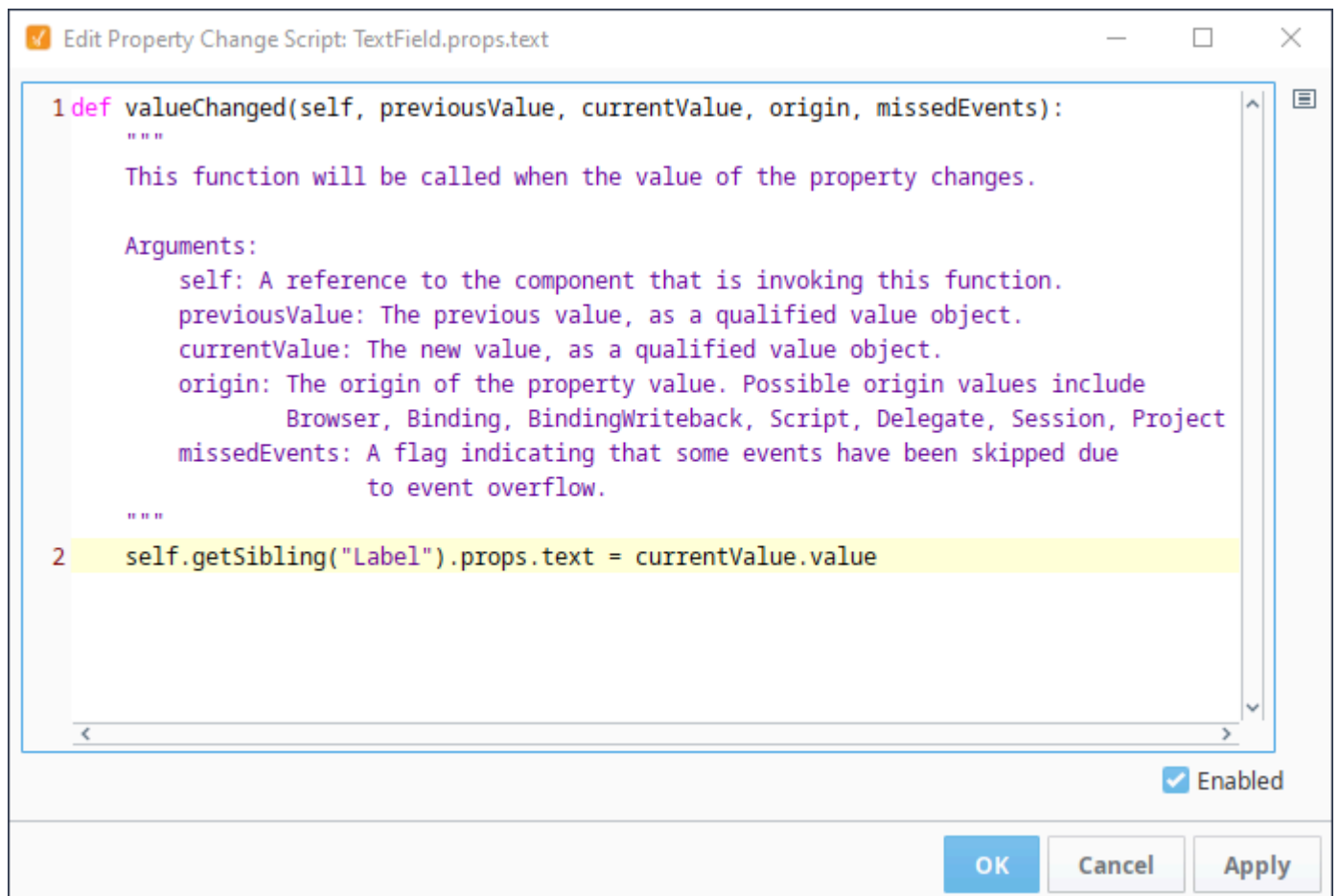
Change Script Example

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
```



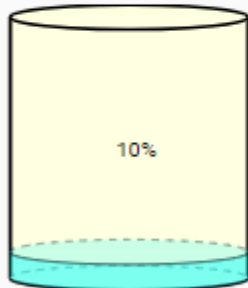
5. Click **OK** to commit the script. You'll see that the Property Editor now shows a **Change Script**  icon next to the text property.
6. **Save** your project.
7. In a Perspective Session, enter some text into the Text Field component and hit Return. You'll see that the contents are repeated to the Label component.



East Ridgefield Overview

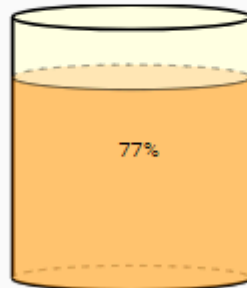
Script test

Tank 1002



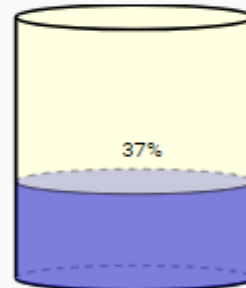
Click for
Details

Tank 1007



Click for
Details

Tank 1012



Click for
Details

Perspective Event Scripts

Perspective Event Scripts let you configure project level scripting that responds to key events in Perspective Sessions or on the Gateway. These scripts enable interaction between the client session, mobile devices, and the Gateway at various points in a session's lifecycle.

Scripts fall into two categories:

- **Gateway Event Scripts:** Events handled at the Gateway level, useful for centralized processing or offline functionality, like form submissions.
- **Session Event Scripts:** Events tied to an individual Perspective Session. For example, session startup, shutdown, and keystroke.

Gateway Event Scripts

Gateway Event Scripts run in response to Perspective events that require centralized, Gateway-scoped execution. These events are often related to device independent workflows or features that support Offline Mode. Gateway Event Scripts are not tied to a specific active session context and may occur outside of a live connection (e.g., queued form submission from a disconnected mobile device).

For a full list of available Gateway Event Scripts across the platform, see [Gateway Event Scripts](#).

Form Submission

The Form Submission event script runs in the Gateway when a form is submitted from a Perspective Session. Unlike other events, this submission process is managed by an internal Form Submission Manager, which allows for offline submission, automatic retries, and response handling.

Project Browser

Filter

- Alarm Notification Pipelines
- Sequential Function Charts
- Scripting
- Perspective
 - Gateway Events
 - Form Submission**
 - formSubmissionExample*
 - Session Events
 - Keystroke
 - Message
 - Accelerometer**
 - Authentication Challenge
 - Barcode**
 - Bluetooth
 - NFC**
 - Page Startup
 - Shutdown
 - Startup
 - Styles
 - Views
 - Transaction Groups
 - Vision
 - Event Streams
 - Named Queries
 - Reports

Enabled ☒

▼ Description [Learn More](#)

Form submission handler scripts that run whenever the Gateway receives a form submission.

▼ Parameters

session	(Session)	The Perspective session attached to this form submission. Value is null if submission originated offline.
name		The name of the submitted form.
data	(dict)	A dictionary containing form data.
formContext	(dict)	A dictionary of additional context relevant to the submission.
sessionContext	(dict)	A dictionary of session data automatically captured at the moment of submission: timestamp, location, user, device info etc.
retry	(bool)	A boolean value indicating that this is a resubmission.

Script

```
1 def handleSubmission(session, name, data, formContext, sessionContext, retry):
2
```

formSubmissionExample x Accelerometer

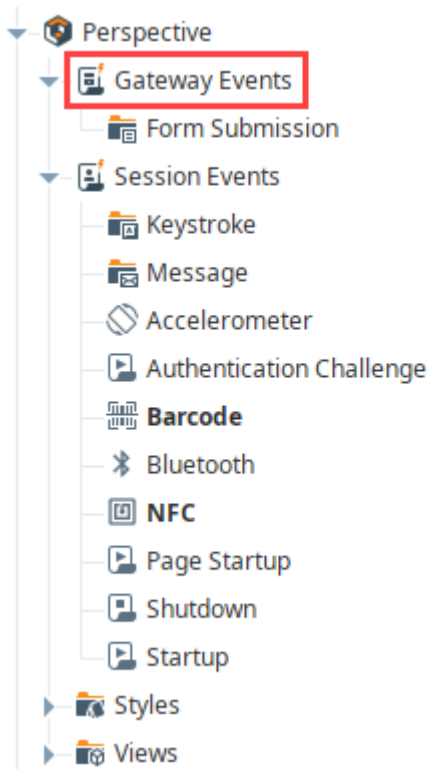
Form Submission Parameters

Name	Description
session	The Perspective Session handling the form submission event.
name	The name of the submitted form.
data	A dictionary containing the submitted form data, with key-value pairs matching the form fields.
formContext	A dictionary containing additional context relevant to the submission, as defined by the user.
sessionContext	A dictionary of session-related data captured at the time of submission, such as: • Timestamp: The time when the event occurred. • User information: Details about the user associated with the event. • Device details: Information about the device used. • Location data: The geographical location of the device (if available).
retry	A boolean value indicating whether this is a resubmitted form due to a previous failure.

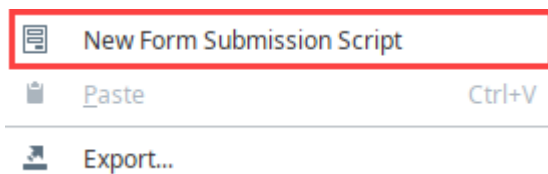
Form Submission Example

This example logs form data in the Gateway and returns a validation error if a required field is missing.

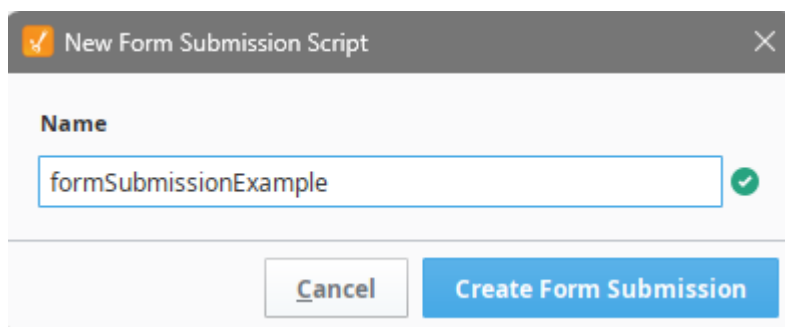
1. In the Project Browser, navigate to **Perspective > Gateway Events**.



2. Right-click **Form Submission** and select **New Form Submission Script**.



3. Enter **formSubmissionExample** into the Name field, and click **Create Form Submission**.



4. Add the following script to the editor and click **Save**.


```

# Example script for handling form submissions
logger = system.util.getLogger("formSubmissionExample")

# Log submission data
logger.info("Received form submission: " + str(data))

# Example validation: Ensure 'email' field is provided
if 'email' not in data or not data['email']:
    response = {
        "success": False,
        "title": "Missing Required Field",
        "message": "Email address is required.",
        "fieldErrors": {"email": {"title": "Required", "message": "Please
enter your email."}}
    }
else:
    response = {"success": True, "title": "Form Submitted", "message": "Thank
you for your submission!"}

# Return response
response

```

Test the Example

To test the form submission:

1. Add a Form component to a view.
2. Ensure the Form component name matches the Form Submission event name.
3. Configure the form with an Email input field and a Submit button.
4. Launch the Perspective Session and submit the form.
5. Verify the response:
 - i. If the email field is empty, an error will appear.
 - ii. If the email is provided, a success message will be displayed.

Session Event Scripts

Session Event Scripts run in the Gateway scope and respond to events initiated by a Perspective Session.

Session Events include:

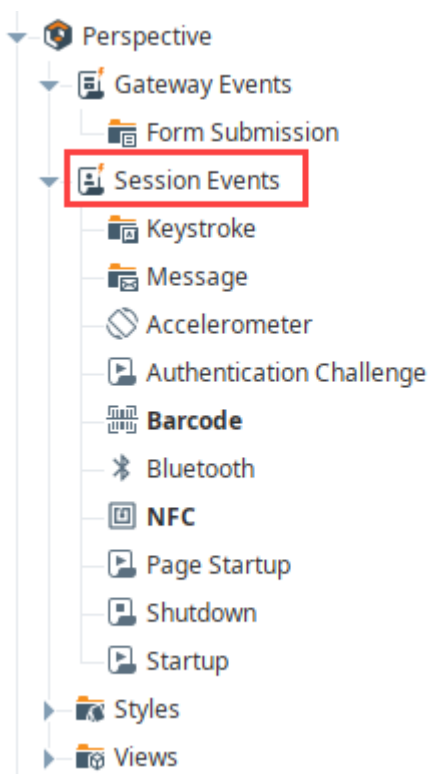
- **Startup**

- **Shutdown**
- **Page Startup**
- **Authentication Challenge**
- **Message**
- **Keystroke**
- **Barcode**
- **Bluetooth**
- **Accelerometer**
- **NFC**

Configuring Session Events

To configure a Session Event Script:

1. In the **Project Browser**, navigate to **Perspective > Session Events**.



2. Select the event type you want to configure.

NOTE

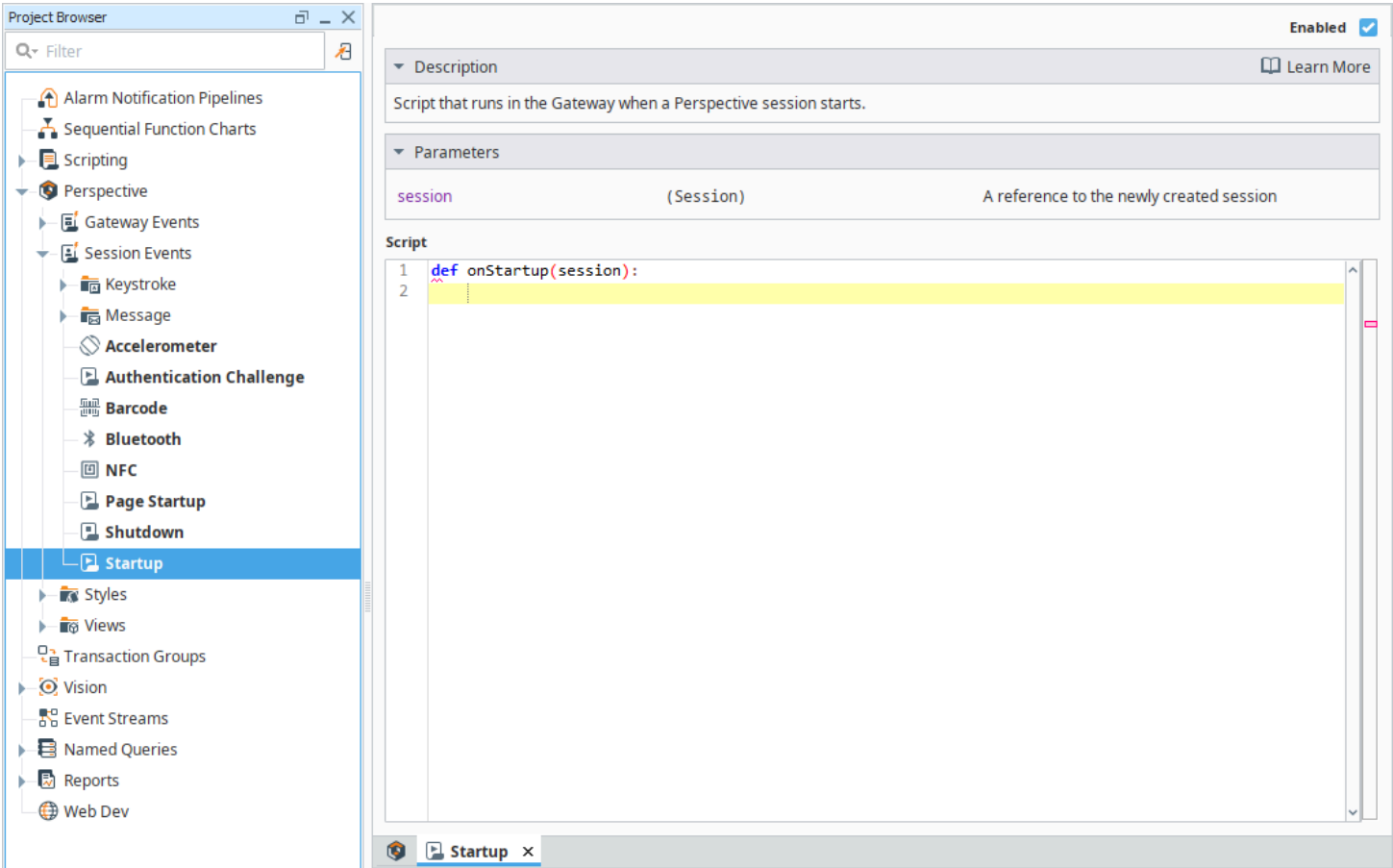
If configuring **Keystroke** or **Message** scripts, right-click the event folder and select **New Script**.

3. Add your script in the provided editor.

4. Click **Save**.

Startup

The Startup event script runs when a Session starts. The Gateway has access to the session object and all its properties.



You can use custom Session properties to pass information between the Session and the Gateway.

Startup Parameters

Name	Description
session	Represents the Perspective Session that is starting. The session object contains all Session properties.

Startup Example

This example logs the Session start time in a database.

Create a Named Query

1. In **Named Queries**, create a new query called **Startup Query**.

2. Set the **Query Type** to **Update Query**.
3. Add a **Value type parameter** named `SessionID` (string).
4. Enter the following query:

Startup Query

```
INSERT INTO sessions (session_id, start_time)
VALUES (:SessionID, CURRENT_TIMESTAMP)
```

Add the Startup Script

1. In the **Project Browser**, navigate to **Perspective > Session Events > Startup**.
2. Enter the following script:

Startup Script

```
# Log the Session start time.
queryParams = {'sessionID': session.props.id}
system.db.runNamedQuery('My Project', 'Startup Query', queryParams)
```

3. Click **Save**.

Test the Example

1. Open a Perspective Session on a mobile device or browser.
2. Verify the database has a new entry in the **sessions** table with the Session ID and start time.

Shutdown

The Shutdown event script runs when a Session ends. The Gateway has access to the session object and all its properties.

You can use custom Session properties to pass information between the Session and the Gateway.

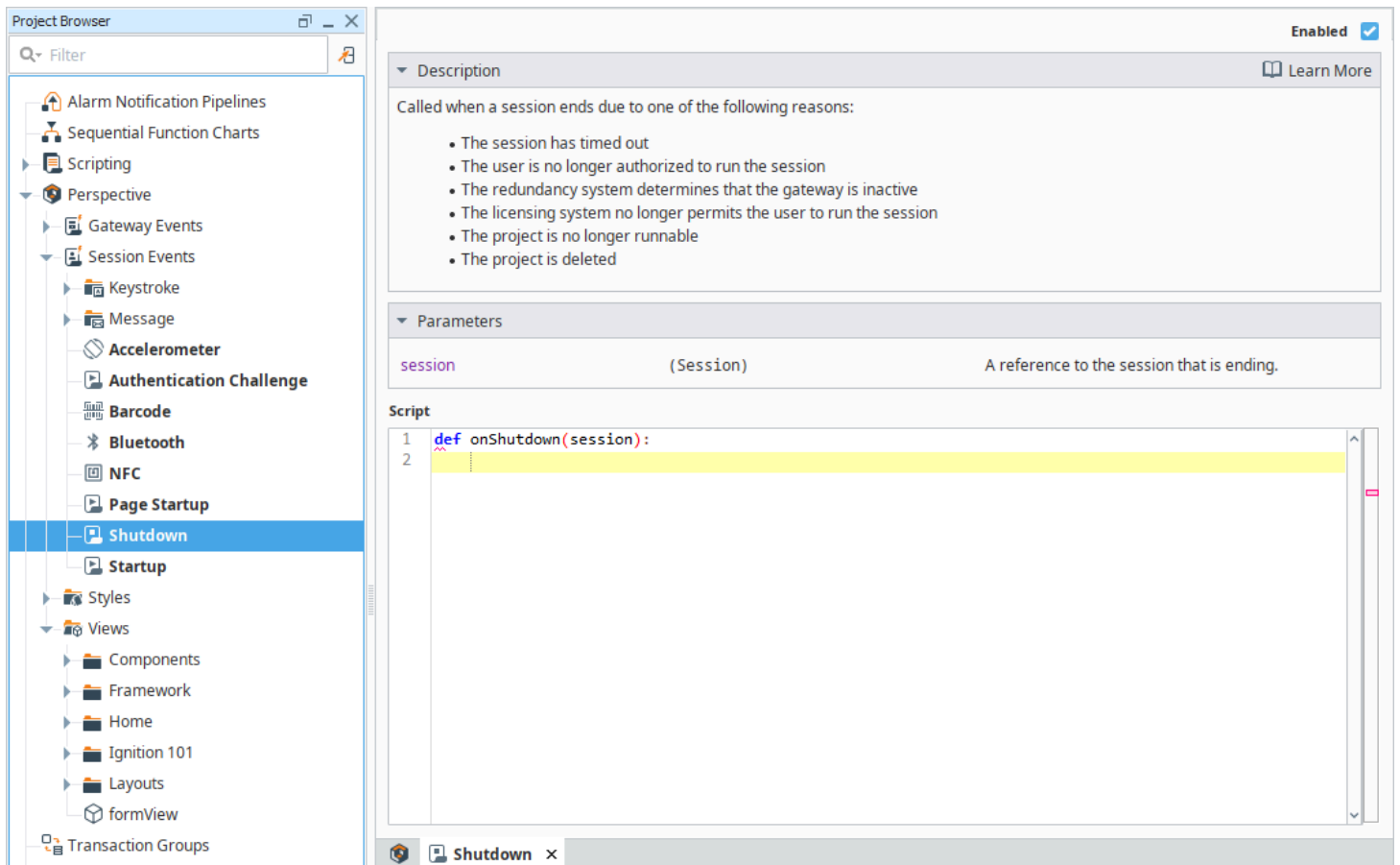
A Shutdown script executes when a Session ends due to:

- The Session closes due to a timeout. Session timeout is configurable in the **Perspective > General** section of **Project Properties**.
- The user losing authorization.

- Gateway redundancy failing over.
- Licensing issues preventing access.
- The project being deleted or made unrunnable.

NOTE

Closing a browser tab **does not immediately close a Session**. The Session will remain open until it **times out**.



Shutdown Parameters

Name	Description
session	Represents the Perspective Session that is ending. The session object contains all Session properties.

Shutdown Example

This example logs the Session end time in a database.

Create a Named Query

1. In **Named Queries**, create a new query called **Shutdown Query**.
2. Set the **Query Type** to **Update Query**.
3. Add a **Value type parameter** named SessionID (string).
4. Enter the following query:

Shutdown Query

```
UPDATE sessions
SET end_time = CURRENT_TIMESTAMP
WHERE session_id = :SessionID
```

Add the Shutdown Script

1. In the **Project Browser**, navigate to **Perspective > Session Events > Shutdown**.
2. Enter the following script:

Shutdown Script

```
# Log the Session end time.
queryParams = {'sessionID': session.props.id}
system.db.runNamedQuery('My Project', 'Shutdown Query', queryParams)
```

3. Click **Save**.

Test the Example

1. Open a Perspective Session on a mobile device or browser.
2. Close the Session using one of the methods described above and wait for the timeout period to pass..
3. Verify the database entry for the Session ID has an updated end time.

Page Startup



INDUCTIVE
UNIVERSITY

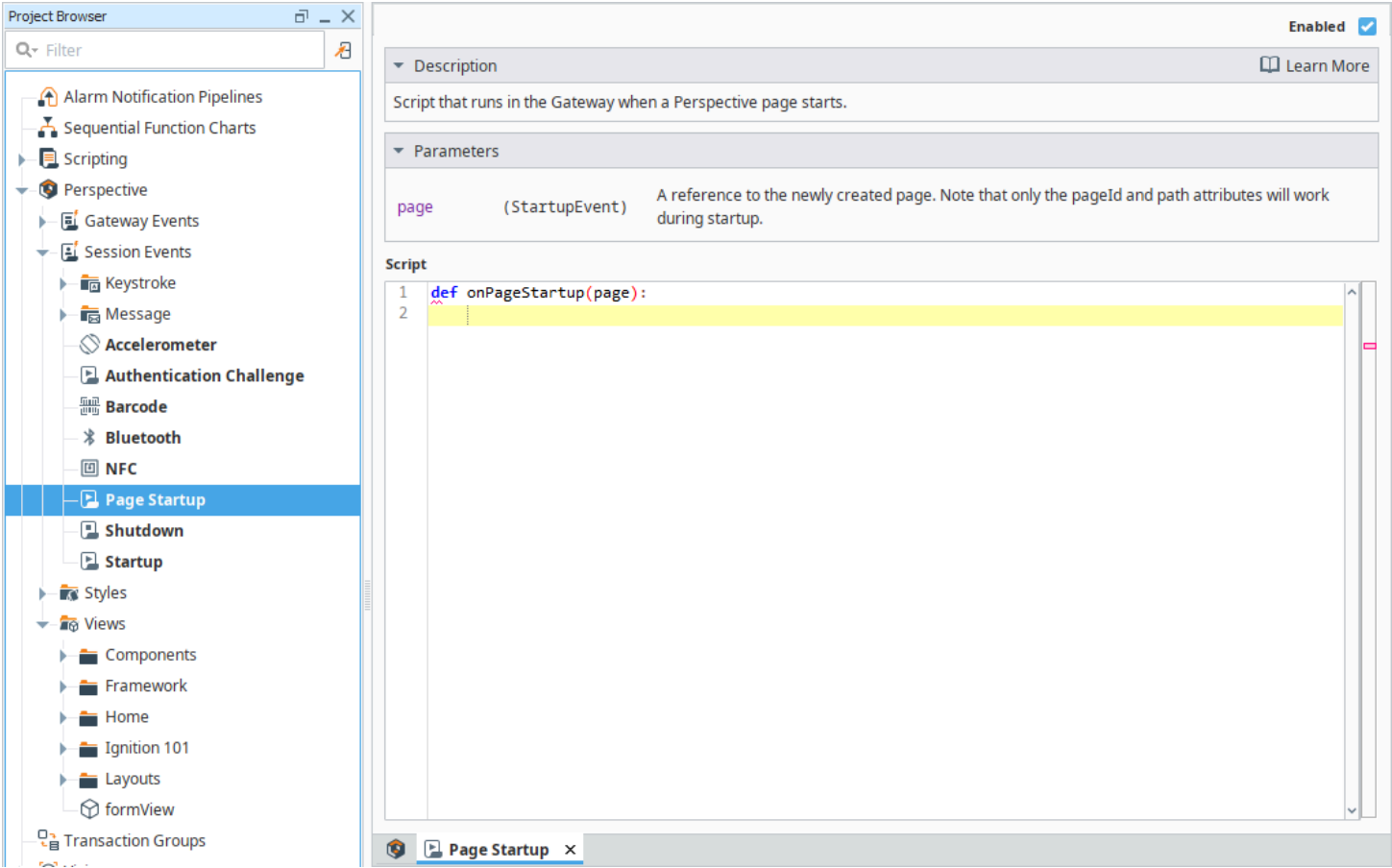
Page Startup

[Watch the video](#)

This script runs in the Gateway when a Perspective page starts in a new tab or window.

NOTE

Navigating to an already open page does not trigger the Page Startup event (such as when using `system.perspective.navigate()`).



Page Startup Parameters

Name	Description
page	Represents the newly created page. Use <code>.props.</code> to access page properties.

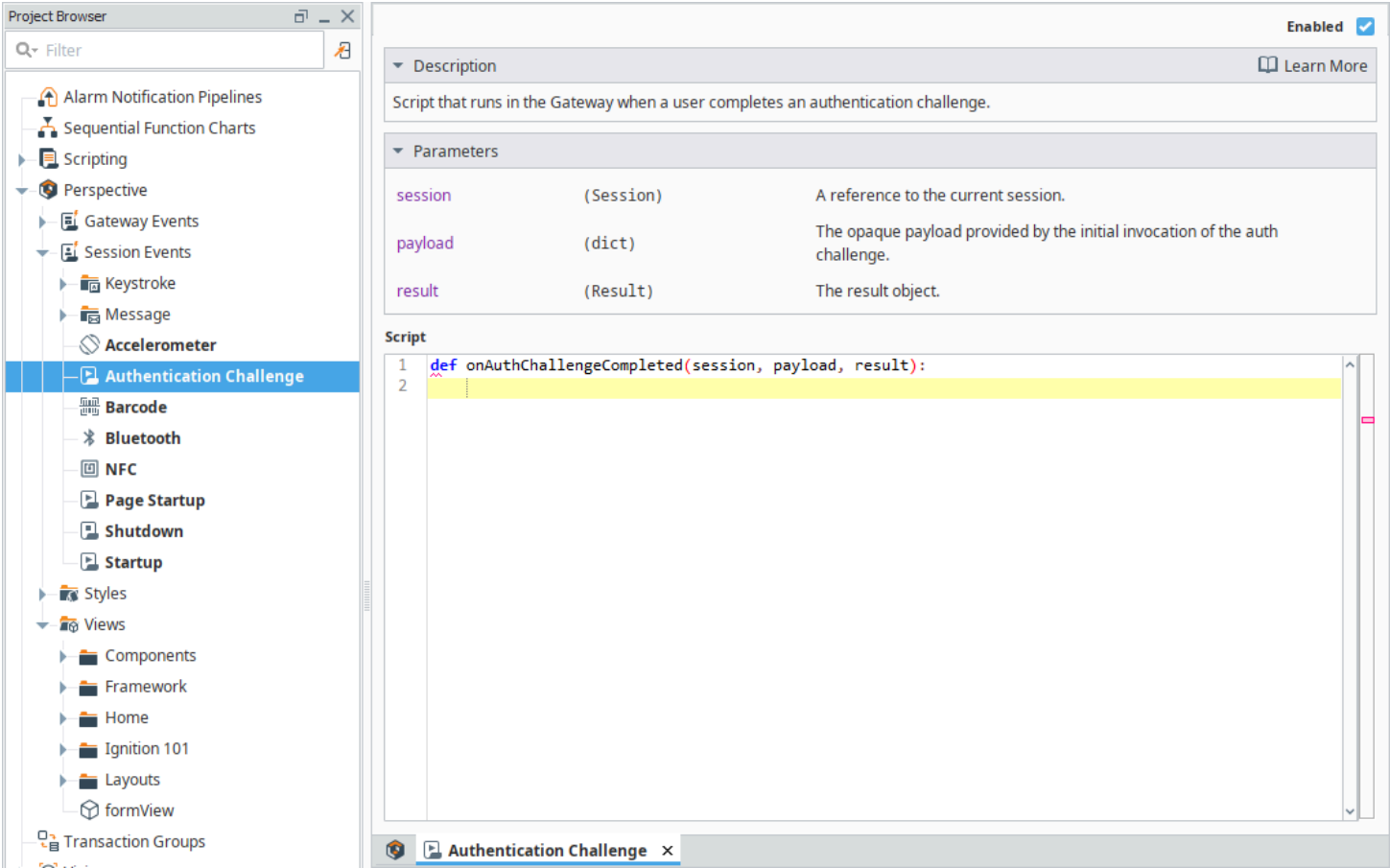
Page Startup Example

Accessing Page Properties

```
pageid = page.props.pageId # Retrieves the page ID.
pagePath = page.props.path # Retrieves the page path.
sessionId = page.session.props.id # Retrieves the session ID.
```

Authentication Challenge

This script runs in the Gateway when a user completes an authentication challenge.



Authentication Challenge Parameters

Name	Description
session	An object that references the Project Session that triggered the authentication challenge. Use this to identify the specific Session that triggered the authentication challenge.
payload	The opaque payload provided by the initial Authentication Challenge Action or Scripting function invocation.
result	A result object that can be parsed by the result functions below.

Results Functions

Function	Description
isSuccess()	Returns true if the authentication challenge was a success. If true, optionally call getAsSuccess() to get the result as a Result.Success object for further parsing. If false, optionally call getAsError() to get the result as a Result.Error object for specific details pertaining to the error.
getAsSuccess()	Returns the result as a Result.Success object if the result is successful, otherwise throws an UnsupportedOperationException if the result is an error. Check isSuccess() or isError() before calling this function to ensure you get the correct result type. Note: It is not necessary to call getAsSuccess(). If isSuccess() returns true, the result type is Result.Success, so all of the properties and functions of a Result.Success object are available on the original result reference.
isError()	Returns true if the authentication challenge resulted in an error. If true, optionally call getAsError() to get the result as a Result.Error object for specific details pertaining to the error.
getAsError()	Returns the result as a Result.Error object if the result is an error, otherwise throws an UnsupportedOperationException if the result is successful. Check isSuccess() or isError() before calling this function to ensure you get the correct result type. Note: It is not necessary to call getAsError(). If isError() returns true, the result type is Result.Error, so all of the properties and functions of a Result.Error object are available on the original result reference.

Result Objects

This object contains different properties depending on whether the authentication challenge result is a success or an error.

Result.Success

An object that represents a successful authentication challenge result. Call **getContext()** to return a **WebAuthUserContext** object with the following properties.

Property	Description
idp	The Identity Provider that challenged the user.
securityZones	A string array containing the names of the security zones associated with the challenged user.
user	A WebAuthUser object that contains the properties in the User Properties table below.
securityLevels	An array of SecurityLevelConfig objects representing all of the security levels associated with the challenged user.

User Properties

Property	Description
id	The unique identifier associated with the challenged user
userName	The challenged user’s username.
firstName	The challenged user’s first name, or null / None if not defined for this user.
lastName	The challenged user’s last name, or null / None if not defined for this user.
email	The challenged user’s email address, or null / None if not defined for this user.
roles	The challenged user’s roles as a string array.

Result.Error

An object that represents an error authentication result. It can be parsed by the following functions.

Function	Description
isGeneric()	Returns true if the error is not related to a timeout or user cancellation. Optionally call getAsGeneric() to get the Result.Error as a Result.Error.Generic object for specific details pertaining to the generic error.
getAsGeneric()	Returns the result as a Result.Error.Generic object if the error is generic, otherwise throws an UnsupportedOperationException . Check isGeneric() before calling this function to ensure you get the correct result type. Note: It is not necessary to call getAsGeneric() . If isGeneric() returns true, the result type is Result.Error.Generic , so all of the properties and functions of a Result.Error.Generic object are available on the original result reference.
isTimeout()	Returns true if the error is due to a timeout. Optionally call getAsTimeout() to get the Result.Error as a Result.Error.Timeout object for specific details pertaining to the timeout error.
getAsTimeout()	Returns the result as a Result.Error.Timeout object if the error is due to a timeout, otherwise throws an UnsupportedOperationException . Check isTimeout() before calling this function to ensure you get the correct result type. Note: It is not necessary to call getAsTimeout() . If isTimeout() returns true, the result type is Result.Error.Timeout , so all of the properties and functions of a Result.Error.Timeout object are available on the original result reference.
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.

Result.Error Objects

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 script determines whether a user can proceed with an action after a completed **Authentication Challenge Action**. The action is allowed only if the authenticated user has the **supervisor** role. If authorization fails or an error occurs, the script logs the result.

Handling an Authentication Challenge Result

```

if result.isSuccess():
    # Parse the authentication result
    context = result.getAsSuccess().getContext()
    user = context.user
    roles = user.roles

    # Check if the user has the "supervisor" role
    if "supervisor" in roles:
        proceedWithSomeAction() # Execute the action
    else:
        # Log unauthorized access
        logger = system.util.getLogger("Authentication Challenge")
        logger.info("%s is not authorized" % user.userName)

# Handle authentication failure
else:
    error = result.getAsError()
    logger = system.util.getLogger("Authentication Challenge")

    # Log the error based on its type
    if error.isGeneric():
        logger.info(error.getAsGeneric().getMessage())

    elif error.isTimeout():
        timeout = error.getAsTimeout().getTimeout()
        logger.info("Authentication challenge timed out after %d minutes." %
timeout)

    elif error.isCancelled():
        logger.info("Authentication challenge was cancelled.")

    else:
        logger.info("Unknown authentication error occurred.")

```

Message

The Message event script runs when a Session receives a message from [system.util.sendMessage](#) or [system.util.sendRequest](#).

NOTE

Session Message Handlers are different from [component-based message handlers](#), which are accessed with [system.perspective.sendMessage](#). Session Message Handlers cannot be called using `system.perspective.sendMessage`.



TIP

For user-driven operations, you can use [system.perspective.getSessionInfo](#) to retrieve session data and filter sessions by specific users.

Message Parameters

Name	Description
session	The Perspective Session that is handling this message.
payload	Retrieve them with a subscript, e.g., <code>myObject = payload['argumentName']</code>

Message Example

This example demonstrates how to configure a Session Message Handler to log messages received from `system.util.sendMessage`.

Configure a Message Handler

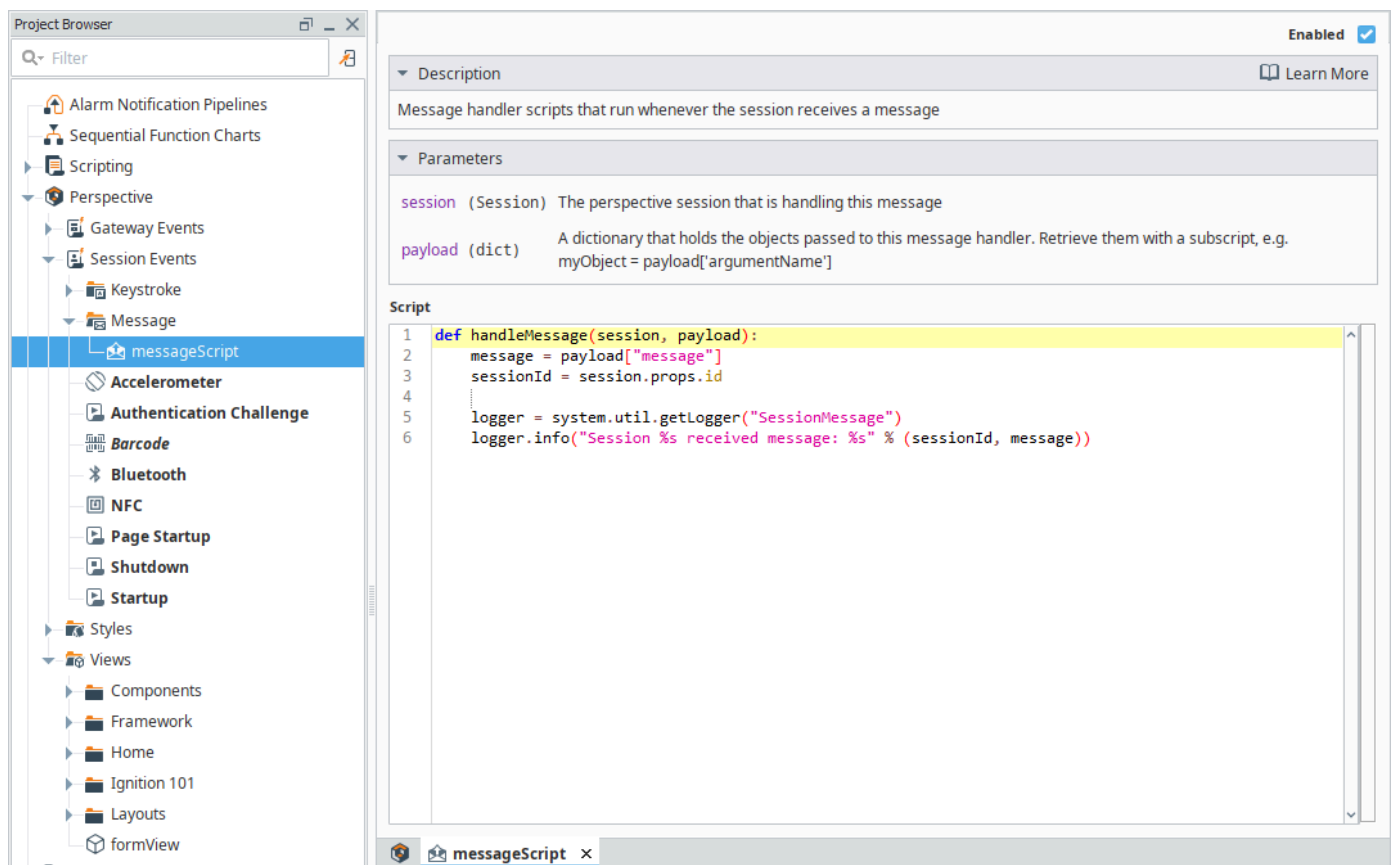
1. In the **Project Browser**, navigate to **Perspective > Session Events**.

2. Right-click the **Message** folder and select **New Message Script**.
3. Enter "messageScript" as the script name.
4. Click **Create Message**.
5. Add the following script:

Logging Incoming Messages

```
message = payload["message"]
sessionId = session.props.id

logger = system.util.getLogger("SessionMessage")
logger.info("Session %s received message: %s" % (sessionId, message))
```



6. Click **Save**.

Send a Message Using a Button

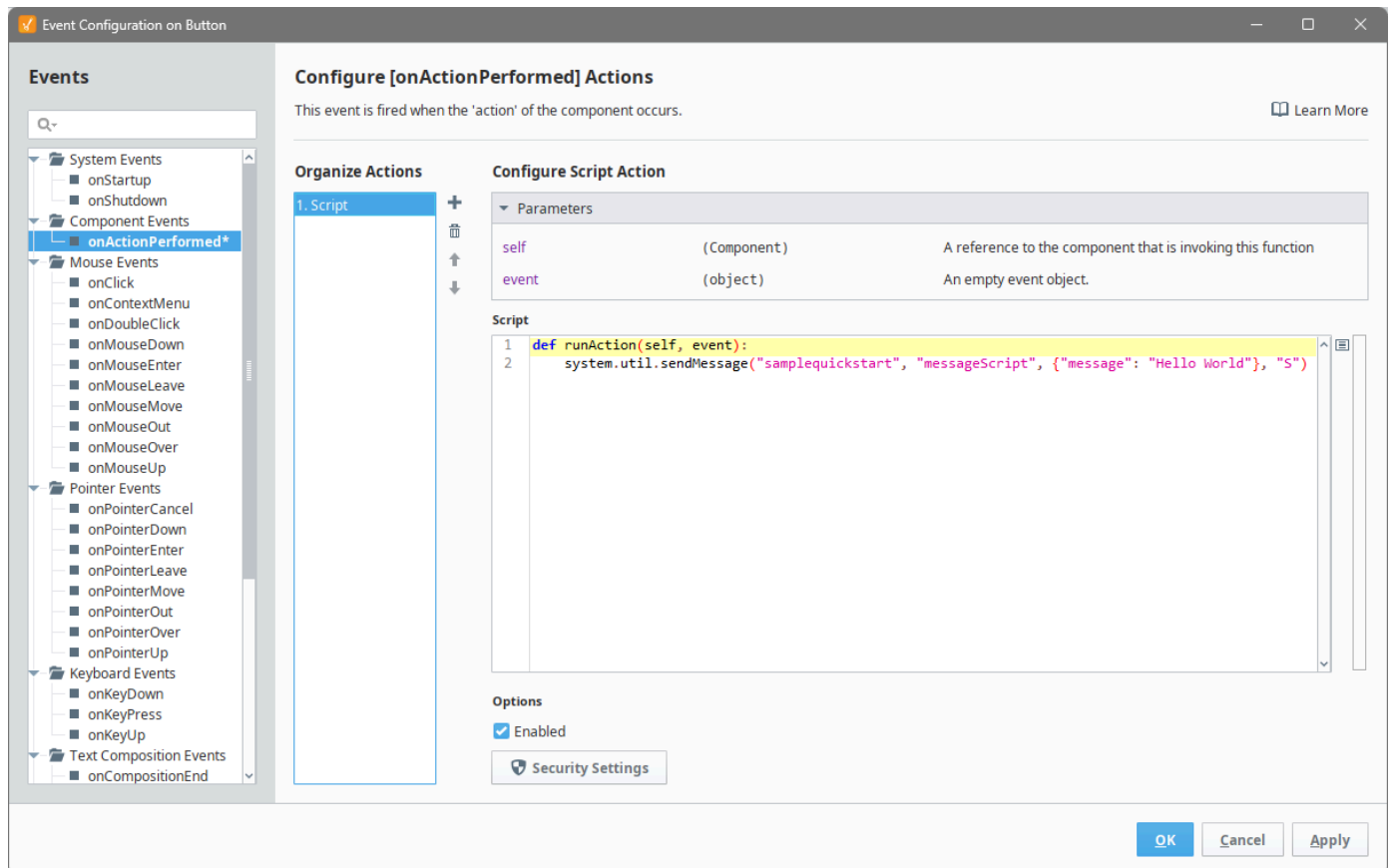
1. Add a **Button** component to a view.
2. Right-click the Button component and select **Configure Events**.
3. On the Event Configuration screen, select **Component Events** > **onActionPerformed**.

4. Click the **Add** icon and select **Script**.

5. Enter the following script:

Sending a Message to the Session

```
system.util.sendMessage("samplequickstart", "messageScript", {"message":  
"Hello World"}, "S")
```



6. Click **OK**.

7. Click **Save**.

Test the Example

1. Launch a Perspective Session.
2. Click the Button to send a message.
3. Navigate to **Gateway > Diagnostics > Logs** to verify that the message was received.

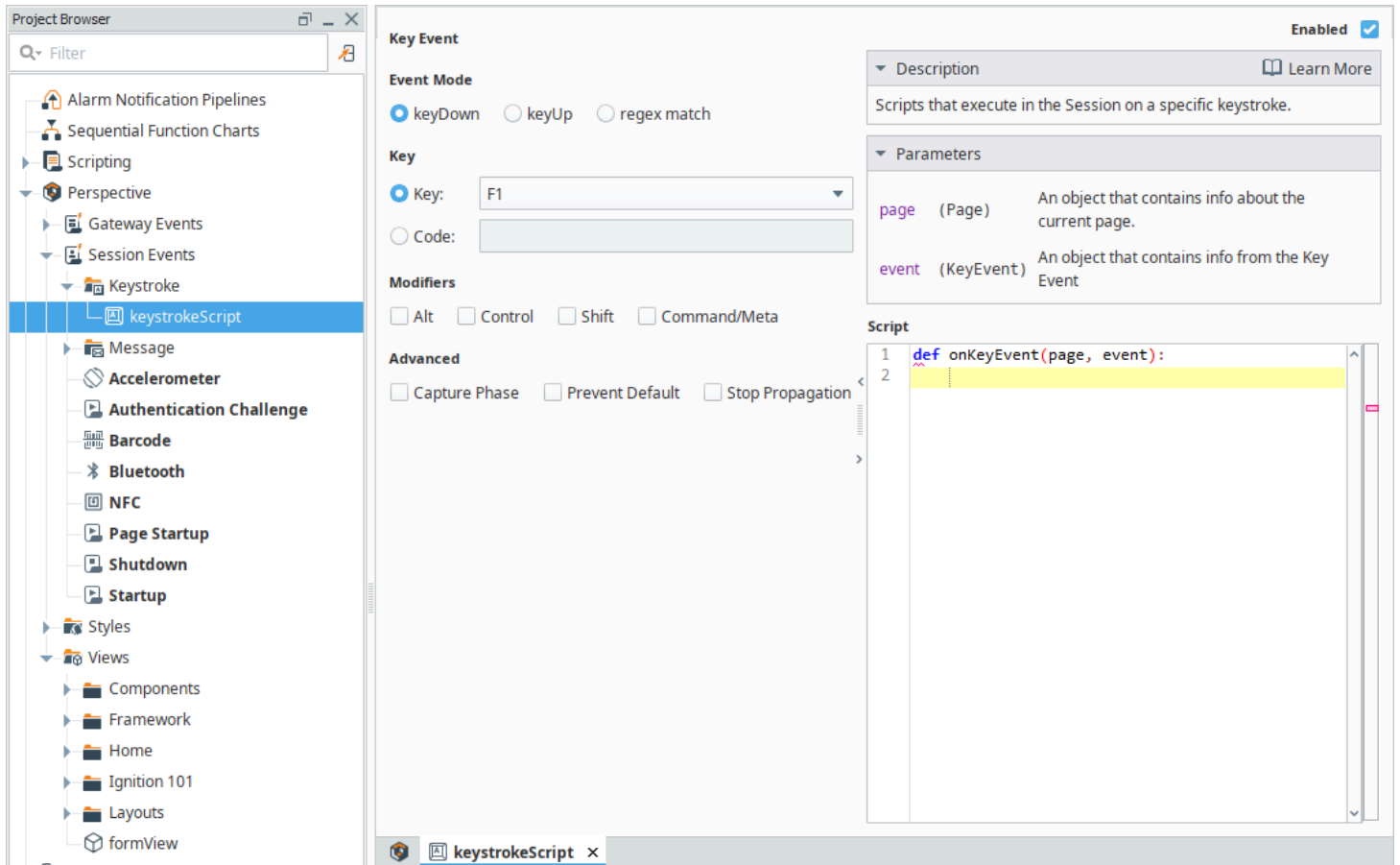
Keystroke

Keystroke scripts run when a Perspective Session detects a KeyboardEvent that matches either a single key or a regex pattern.

- **Single Key:** Matches on key press (keyDown) or key release (keyUp).
- **Regex Match:** Always detected on key release.

! INFO

Keystroke event scripts can be triggered by keyboards or any HID barcode scanner that generates KeyboardEvent data.



Keystroke detection is only active inside a Perspective Session and does not function in the Designer.

i NOTE

The device triggering the keystroke event must generate a KeyboardEvent. HID barcode scanners generate KeyboardEvents because they simulate physical keyboard input. Scanners and input devices that do not create KeyboardEvents will not trigger this logic.

Keystroke Parameters

Name	Description
page	Represents the current page. See the Page Arguments table below for parameter arguments.
event	Contains details about the Keyboard Event. See the Event Arguments table below for parameter arguments.

Page Arguments

Arguments for the Keystroke Page parameter.

Argument	Description
props	Properties of the page object.
session	Represents the current session.

Event Arguments

Arguments for the Keystroke Event parameter.

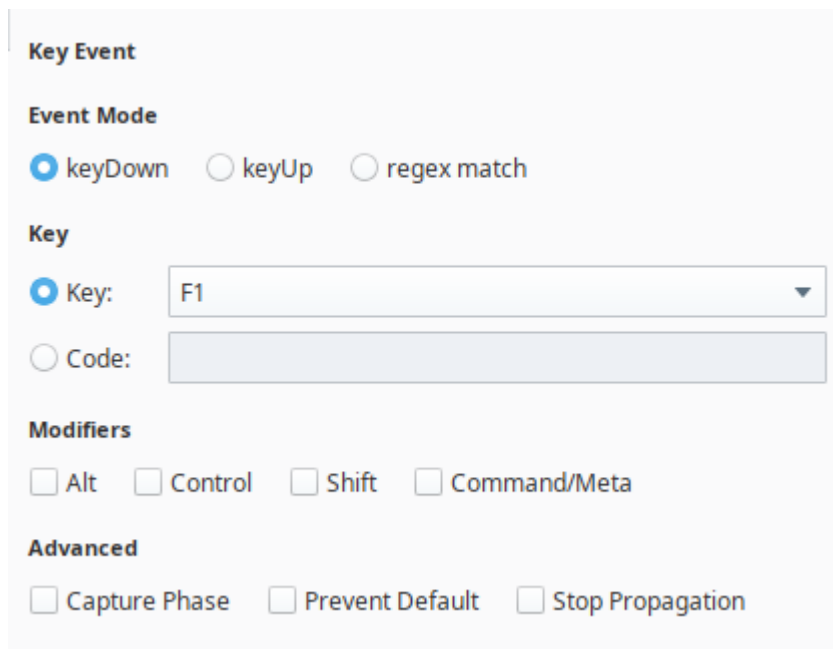
Argument	Description
altKey	True if the Alt (Option on macOS) key was active when the event was generated.
code	A string representing the physical key code.
ctrlKey	True if the Control (Ctrl) key was active when the event was generated.
isComposing	True if the event occurred between compositionstart and compositionend .
key	A string representing the key value of the event.
location	A number indicating the location of the key on the keyboard or input device.
metaKey	True if the Meta key (⌘ Command, ⌘ Windows) was active when the event was generated.
repeat	True if the key was held down long enough to repeat automatically.
shiftKey	True if the Shift key was active when the event was generated.
matches	A list where the first entry is the matched text for the regex, and all subsequent entries are capture group contents.

Single Key Match

For single key matches, the match can be either:

- A physical key typed
- The code generated by the key

Modifiers (Alt, Ctrl, Shift, Meta/Command) can be detected in the event data.



Key Event

Event Mode

☒ keyDown ☐ keyUp ☐ regex match

Key

☒ Key:

☐ Code:

Modifiers

☐ Alt ☐ Control ☐ Shift ☐ Command/Meta

Advanced

☐ Capture Phase ☐ Prevent Default ☐ Stop Propagation

Additionally, key event matches include advanced settings:

- **Capture Phase:** Executes the script before other interface events. By default, key events run during the bubbling phase, meaning they execute after other events in the Document Object Model (DOM).
- **Prevent Default:** Stops default browser actions (e.g., form submission, page navigation) for a given key press.
- **Stop Propagation:** Prevents the key event from affecting deeper DOM elements (e.g., text inputs).

Regex Match

For a regular expression (regex) match, the match is based on:

- A regex pattern
- A match buffer (stores the most recently typed characters)

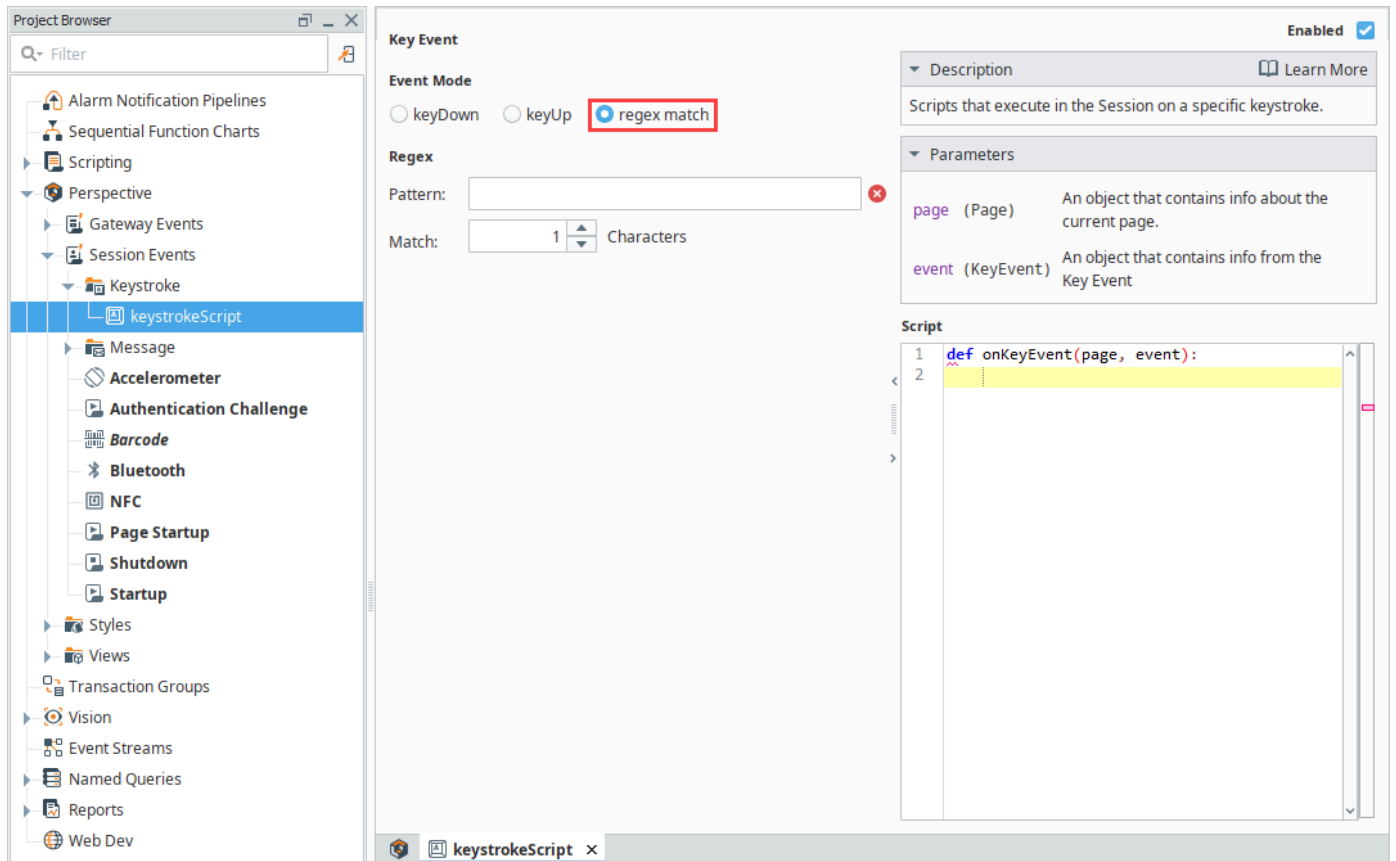
The buffer length should be set to the exact number of characters needed to match the regex.

Keystroke Example (Regex Match)

This example updates a custom Session property when a regex match is found.

1. Add a **Label** component to a view.
2. Create a **custom Session property** called `scannerInput`.
3. Bind the **Label's text** to the **custom Session property**.

4. In the Project Browser**, navigate to **Perspective > Session Events**.
5. Right-click **Keystroke** and select **New Key Event Script**.
6. Enter a name and click **Create Keystroke**.
7. Under **Event Mode**, select **Regex Match**.



8. Enter the **regex pattern** to detect:
 - Example: `[0-9]{12}@` (matches **12 digits followed by @**).
9. Set **Match Buffer Length** to `13` (minimum required characters).
10. Add the following script:

Updating a Session Property on a Regex Match

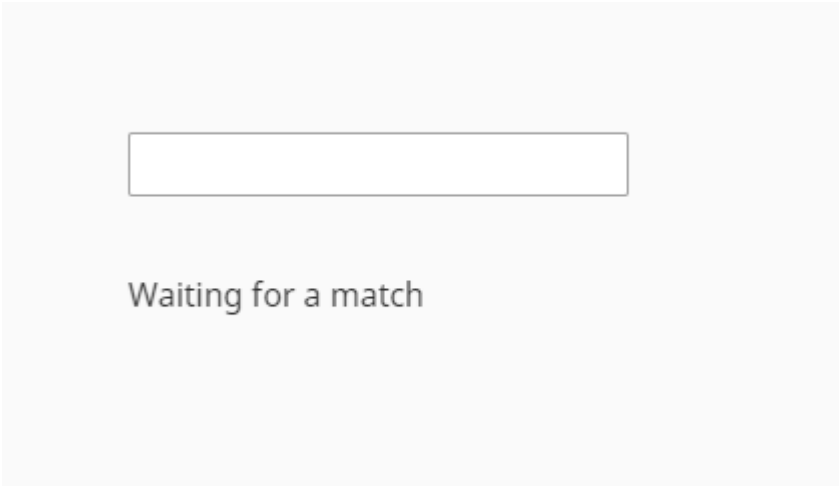
```
page.session.custom.scannerInput = "Match found"
```

11. Click **Save**.

Test the Example

1. Launch a Perspective Session.

2. Open the view with the Label.
3. Type characters matching the regex pattern.
4. The Label's text updates when a match is found.



Perspective 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 Perspective App Actions.

Barcode



The scanned barcode action can make use of a mobile device's built in camera.

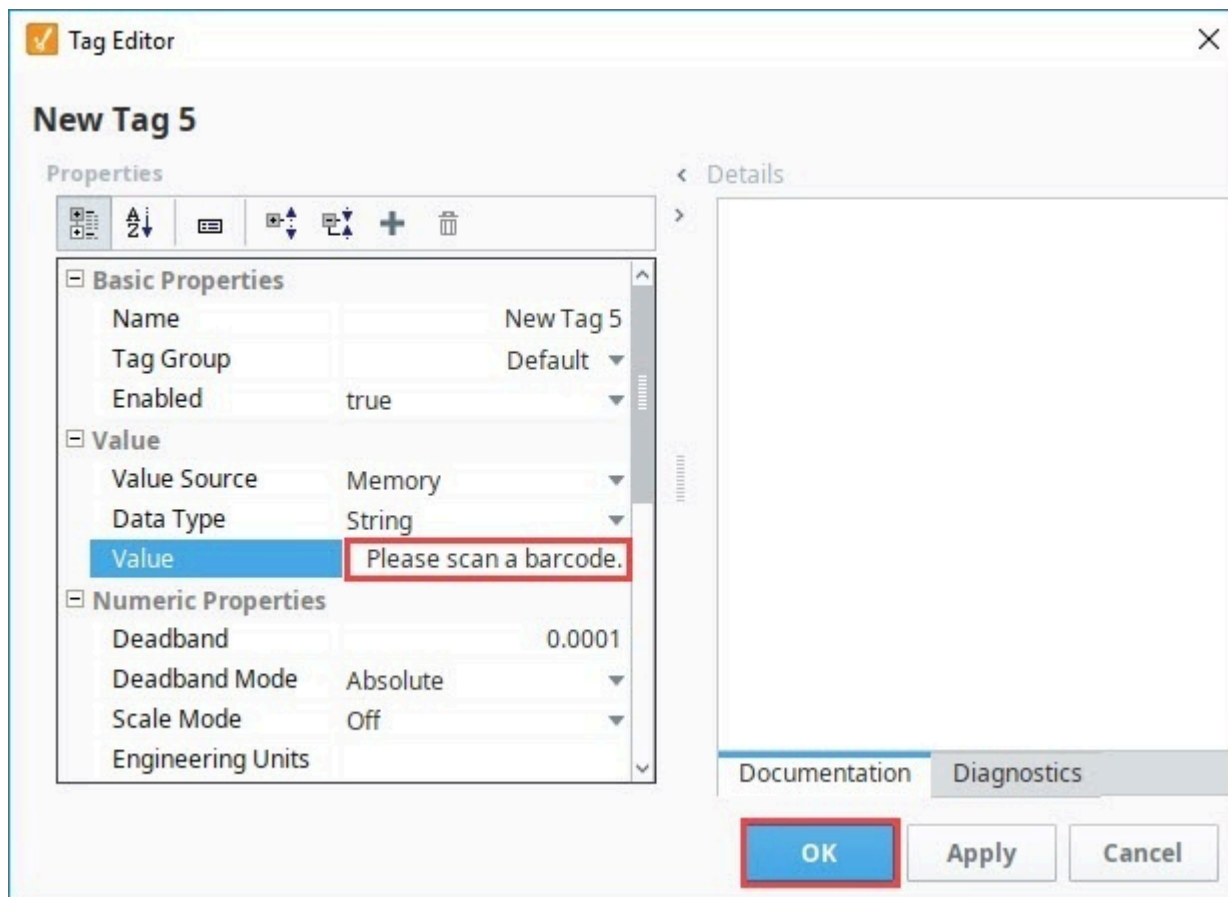
Barcode Parameters

Name	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: <code># The value that was scanned</code> <code>data.text</code> <code># An integer representing a unix timestamp of when the barcode was scanned</code> <code>data.timestamp</code> New in Perspective Mobile app version 0.98 Version 0.98 of the Perspective Mobile App added the following property: <code># The type of barcode that was scanned.</code> <code>data.barcodeType</code> }
context	The user defined context object that can be defined on the action.

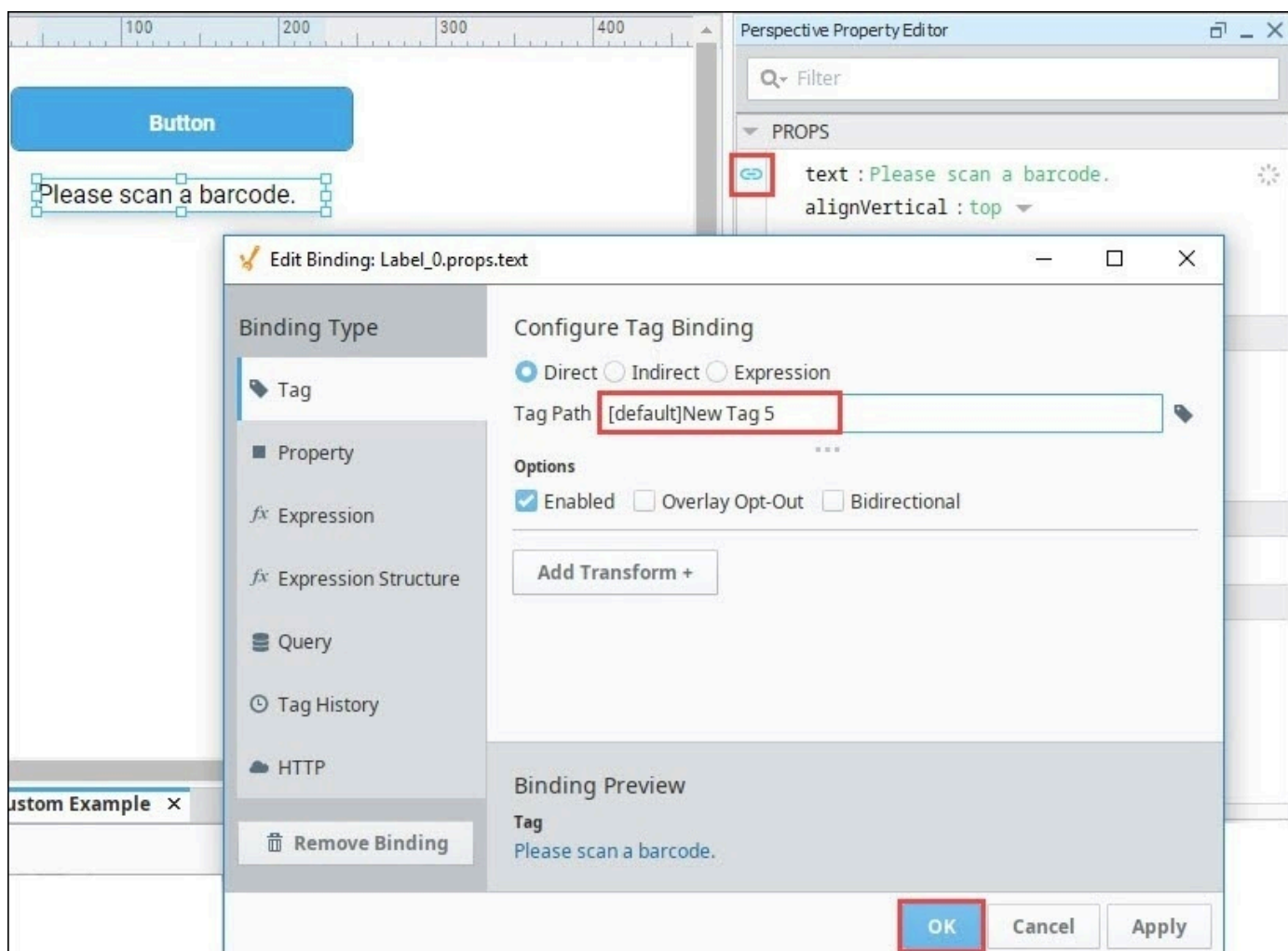
Barcode Example

This example will scan a barcode, and write its value to a tag.

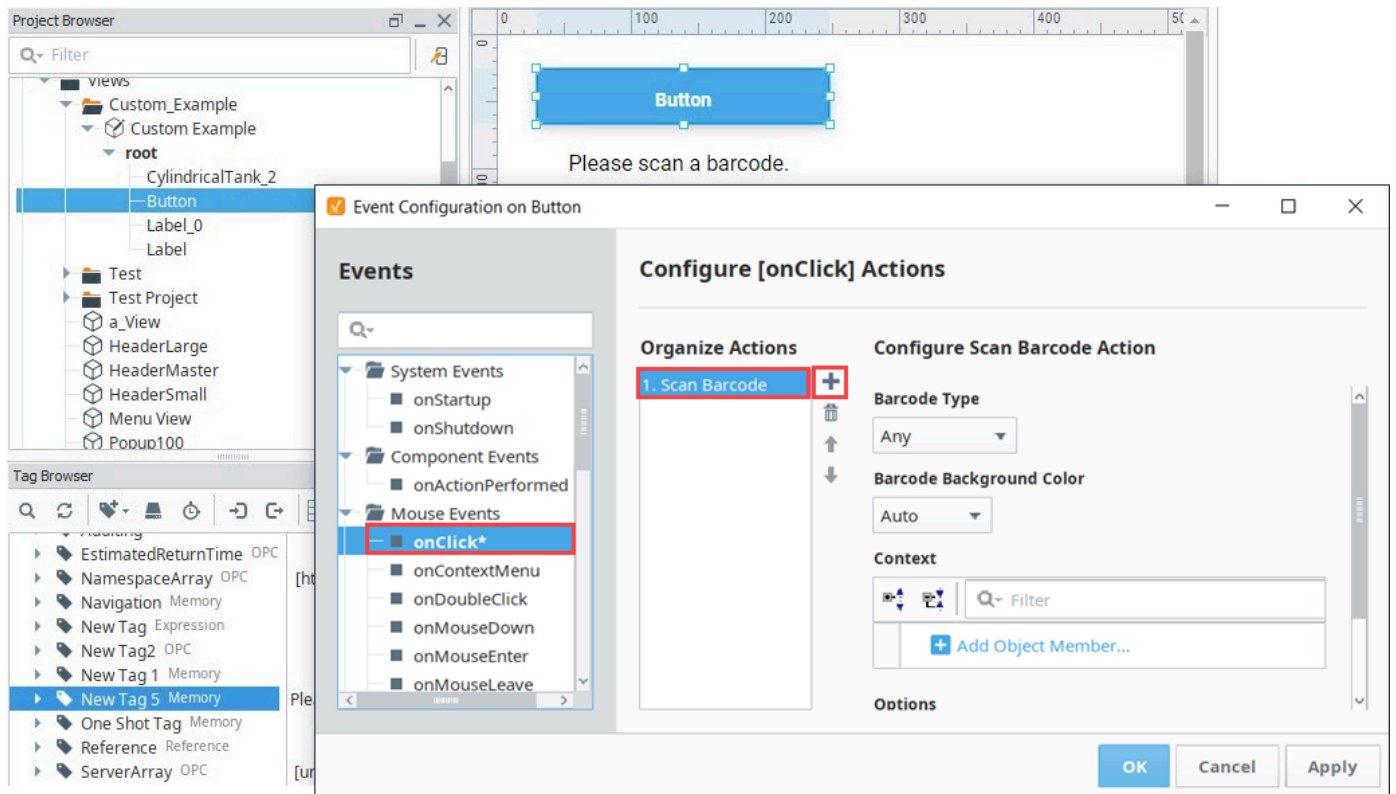
1. For this example, drag a Button component and a Label component onto a view.
2. Create a new Memory tag with a data type of String. Set the value to **Please scan a barcode**.



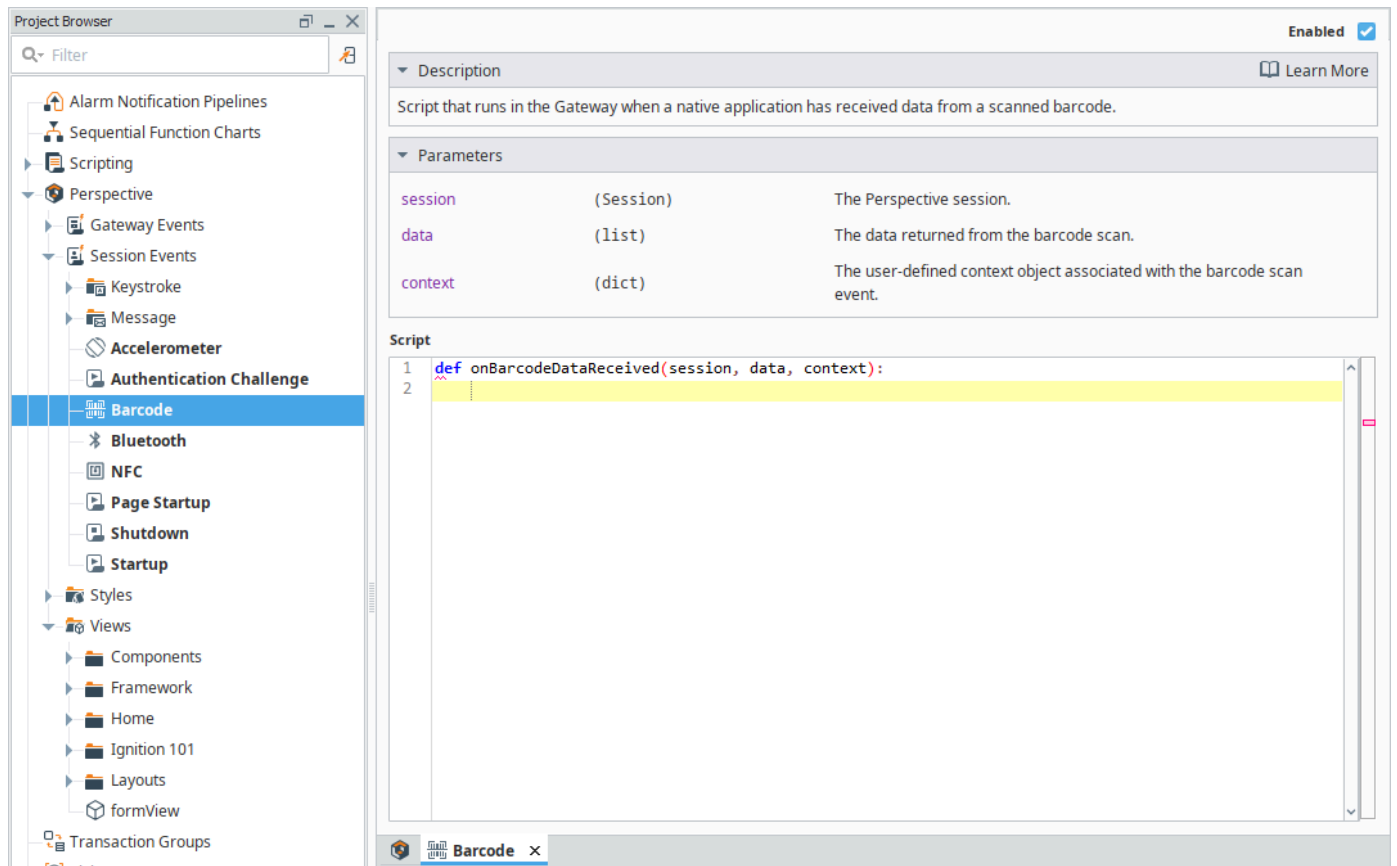
3. Bind the text of the Label to the new tag.



4. 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.



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 Browser navigate to **Perspective > Session Events**, and select **Barcode**.

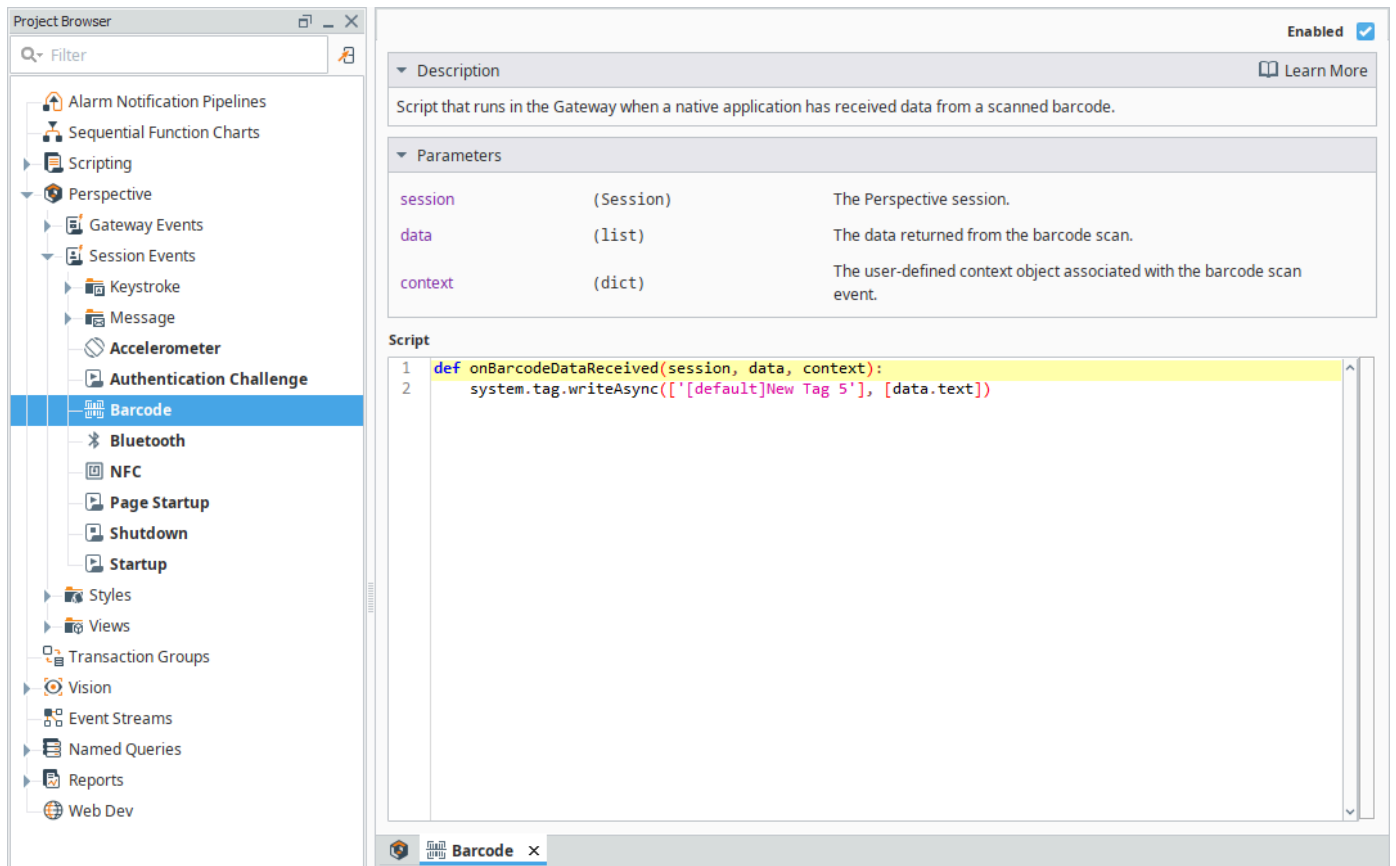


9. Add the following script to the page:

```
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.



10. Save your project.

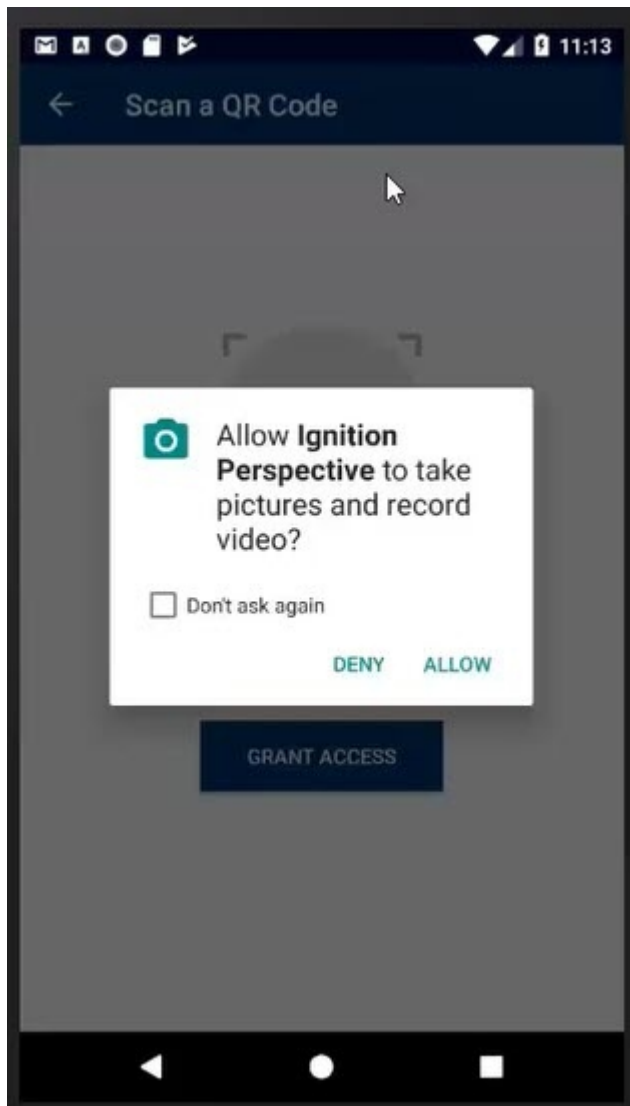
Test the Example

To test the example, open the Perspective App on your mobile device and load the project.

1. Click Scan Barcode.

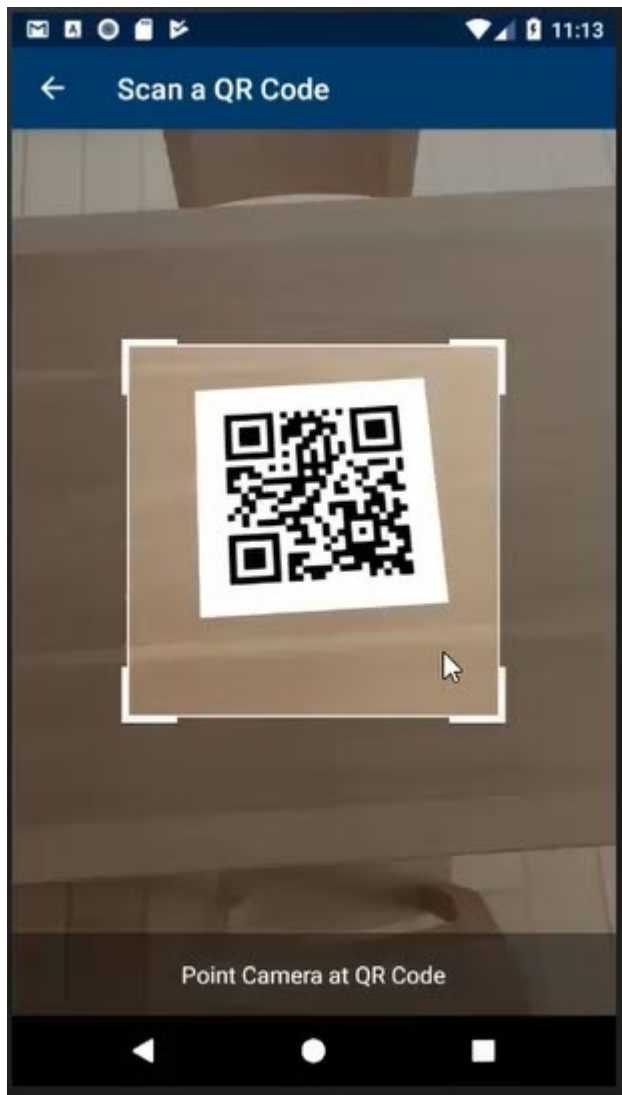


2. If this is the first time scanning a barcode, you'll get a message requesting permission for Perspective to take pictures and record.



3. Click Allow.

4. You can now use the camera on the mobile device to scan the barcode.



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.



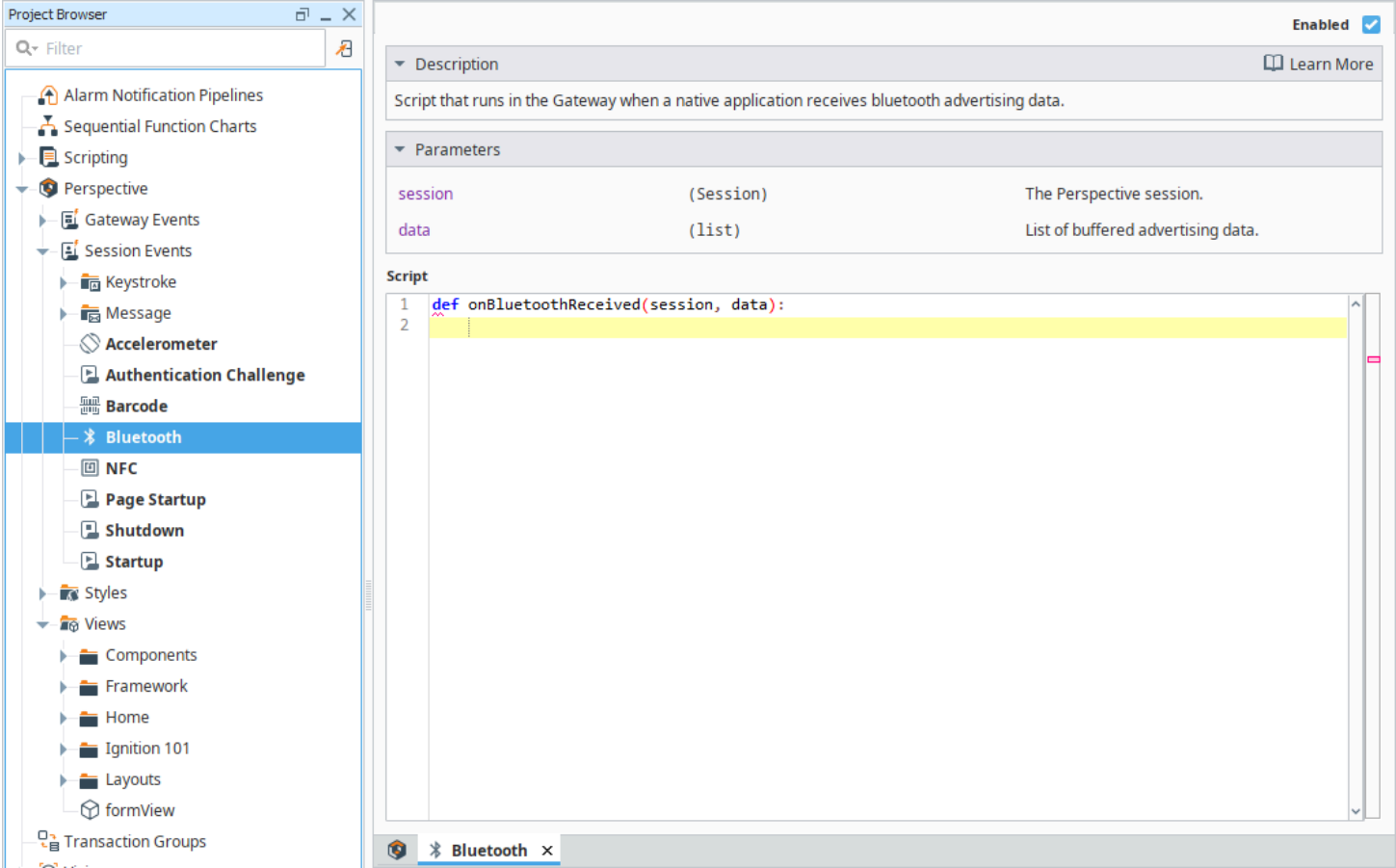
Bluetooth

This Session Event Script sends received 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 Parameters

Name	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. See the example output below.

Example Output


```
{
  "values":
  [
    {
      "rssi": -48,
      "timestamp": 1570147485167,
      "manufacturerData": {
        "companyId": 6,
        "dataBase64Encoded": "AQkgAl_edccdnHCIMvecMCiM--aAkLHAPibd"
      }
    },
    {
      "rssi": -48,
      "timestamp": 1570147486012,
      "serviceUUIDs": [
        "FEAA"
      ],
      "serviceData": {
        "uuid": "FEAA",
        "dataBase64Encoded": "AOyqqqqqqqqqqqqAAAAAAA"
      },
      "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": "vqwSNFZ4EjQSNBI0EjRWeJASAAAAA0wA"
      },
    }
  ]
}
```

```

    "AltBeacon":{
      "manufacturerId":65535,
      "uuid":"12345678-1234-1234-1234-123456789012",
      "instance":"00000000",
      "txPower":-20,
      "manufacturerReserved":"00"
    }
  ]
}

```

Accelerometer



INDUCTIVE
UNIVERSITY

Accelerometer Data Received

[Watch the video](#)

The Accelerometer event is only used when data is coming in from a batched [Accelerometer Action](#).

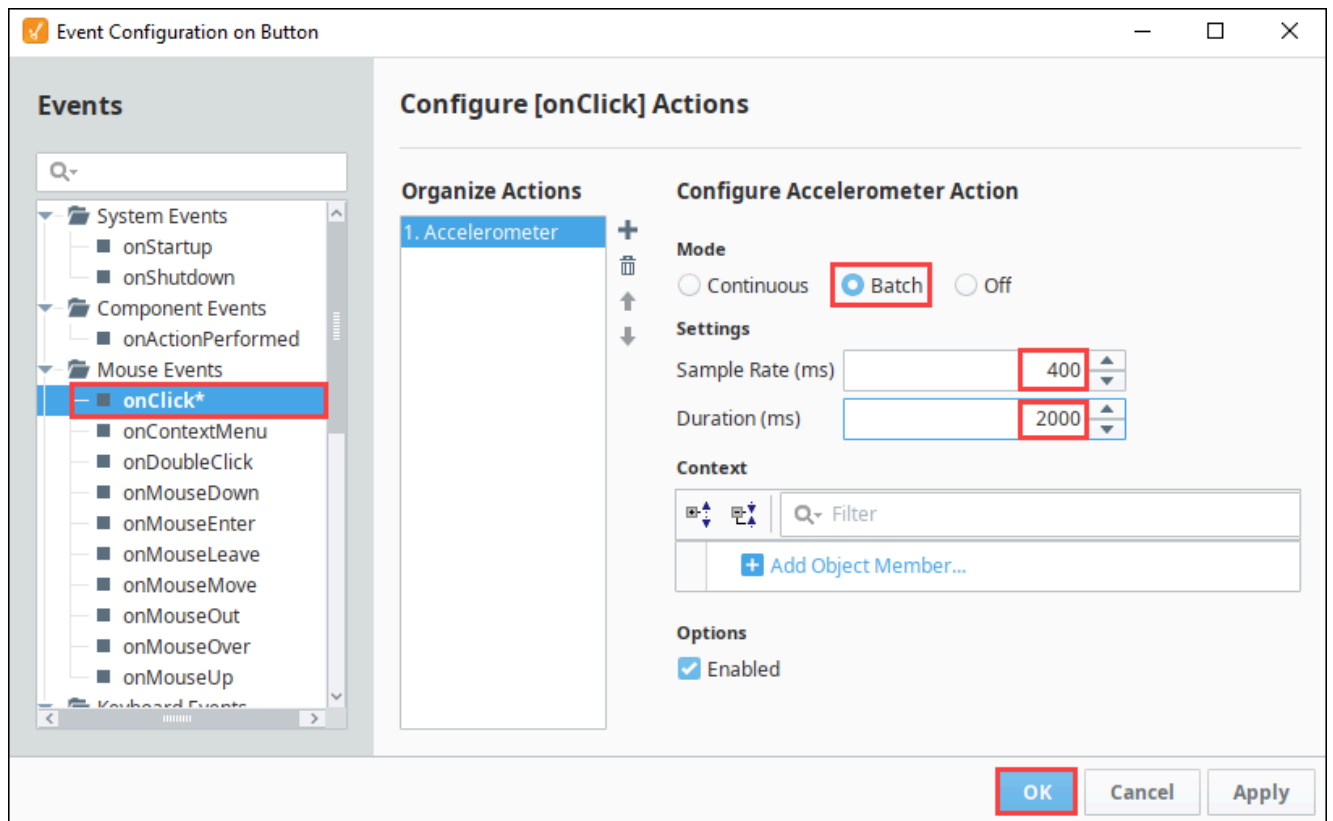
Accelerometer Properties

Name	Description
session	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.
data	The data returned from the batched accelerometer. The data comes in as a data object, which has various parts. Access the various parts like: <code>logger = system.util.logger('accelerometer')</code> <pre> 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>
context	The user defined context object that can be defined on the action.

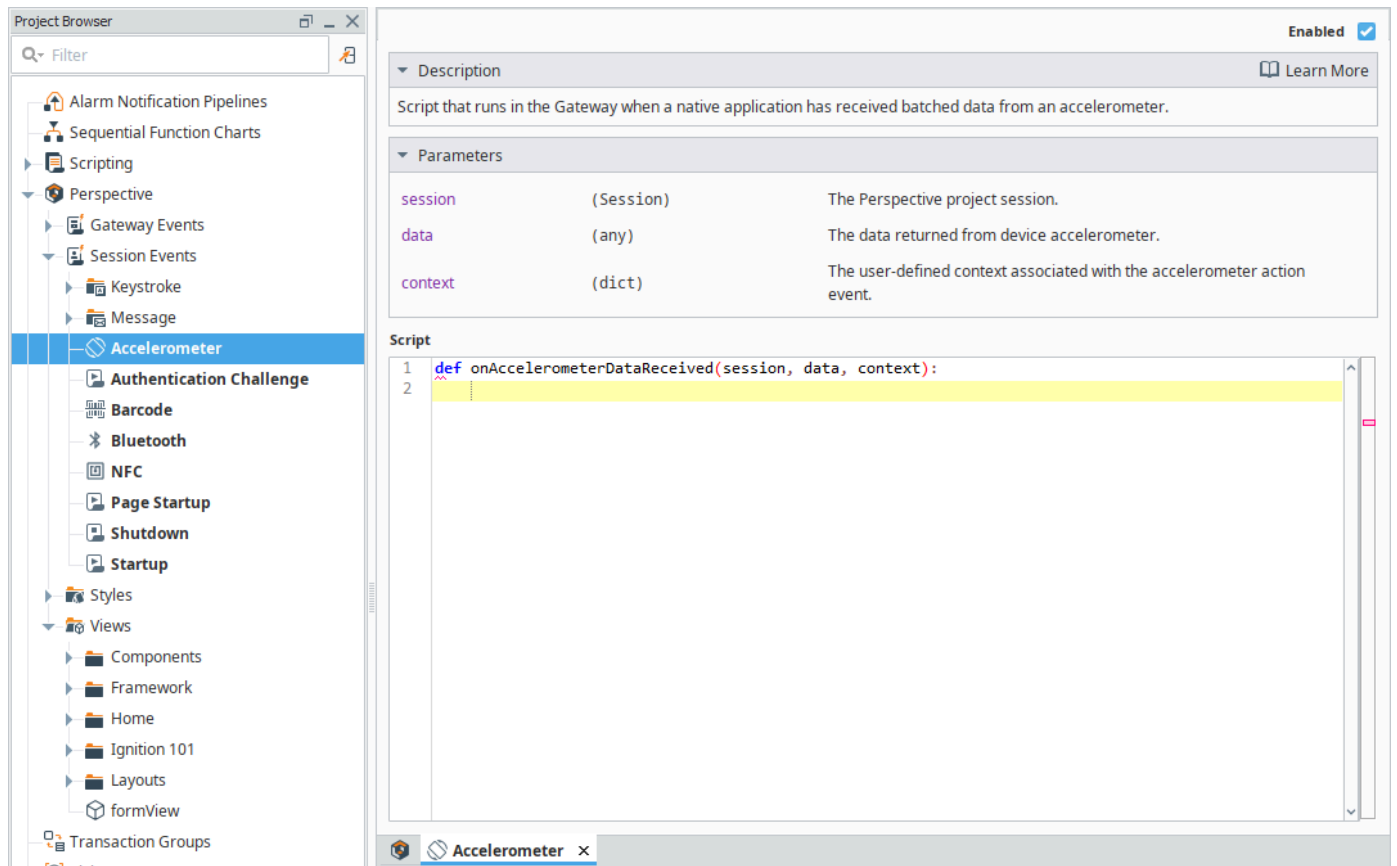
Accelerometer Example

This example will write accelerometer data to the Gateway logs.

1. Place 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.
 - i. Select **Batch** mode.
 - ii. Set a **Sample Rate** of 400.
 - iii. Set a **Duration** of 2000.



5. Click **OK**. Next we need to set up a Session Event so that Perspective knows how to interpret the accelerometer data.
6. Select **Accelerometer** under the **Perspective > Session Events** list.



7. 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']))
```

8. 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. The script will record accelerometer data for the next two seconds, so move the phone around.
3. After the two seconds, you should see the logged information appear in the Gateway logs.



The NFC event is used when the **NFC Action** is used and the mobile device scans an NFC tag.

NFC Properties

Name	Description
session	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.
data	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>
context	The user defined context object associated with the NFC scan event.

NFC Compatibility

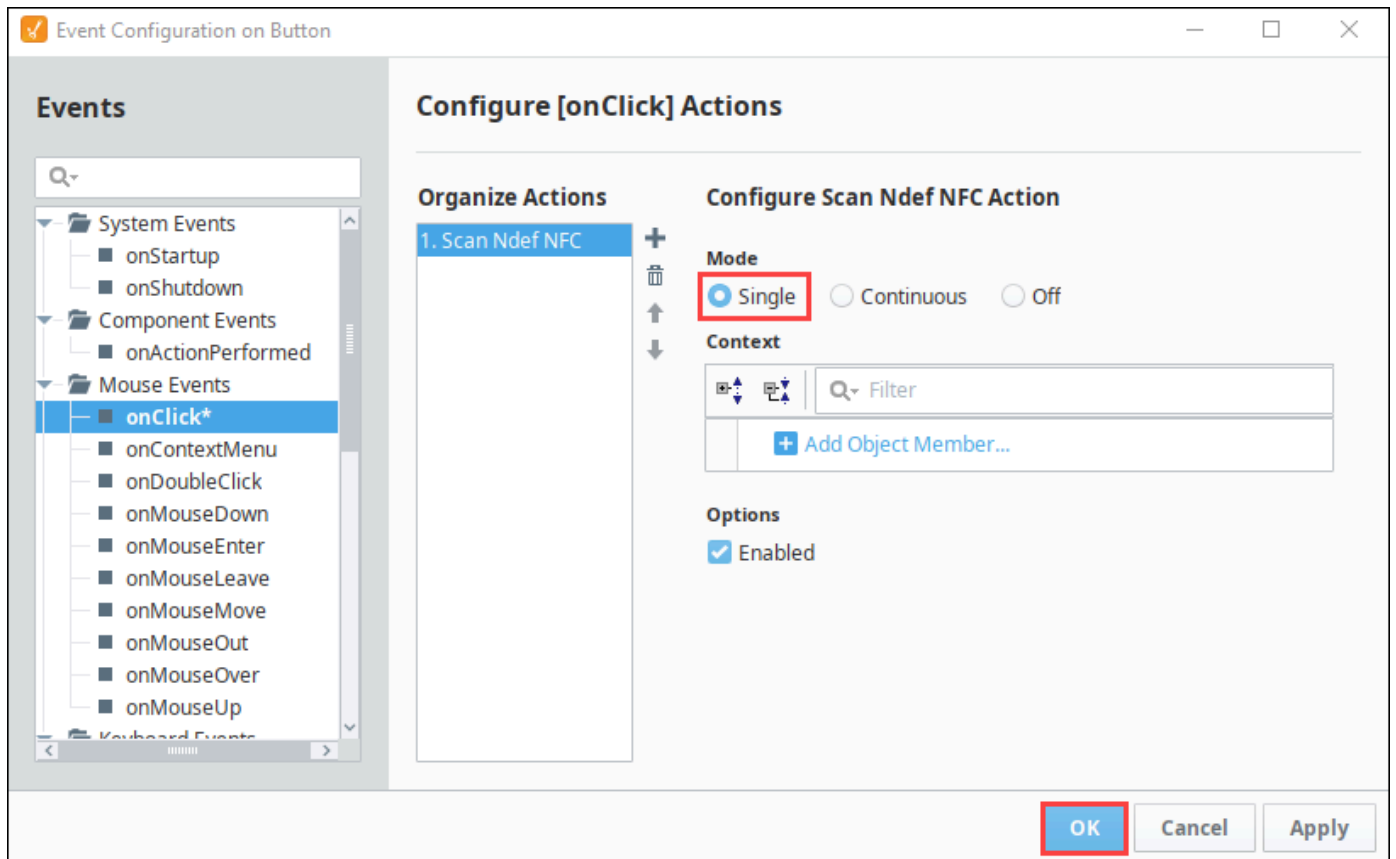
NFC tag types must be NDEF-formatted as one of the following types:

- NFC_FORUM_TYPE_1 based on ISO14443A
- NFC_FORUM_TYPE_2 based on ISO14443A
- NFC_FORUM_TYPE_3 based on JIS X 6319-4
- NFC_FORUM_TYPE_4 based on ISO14443A/B

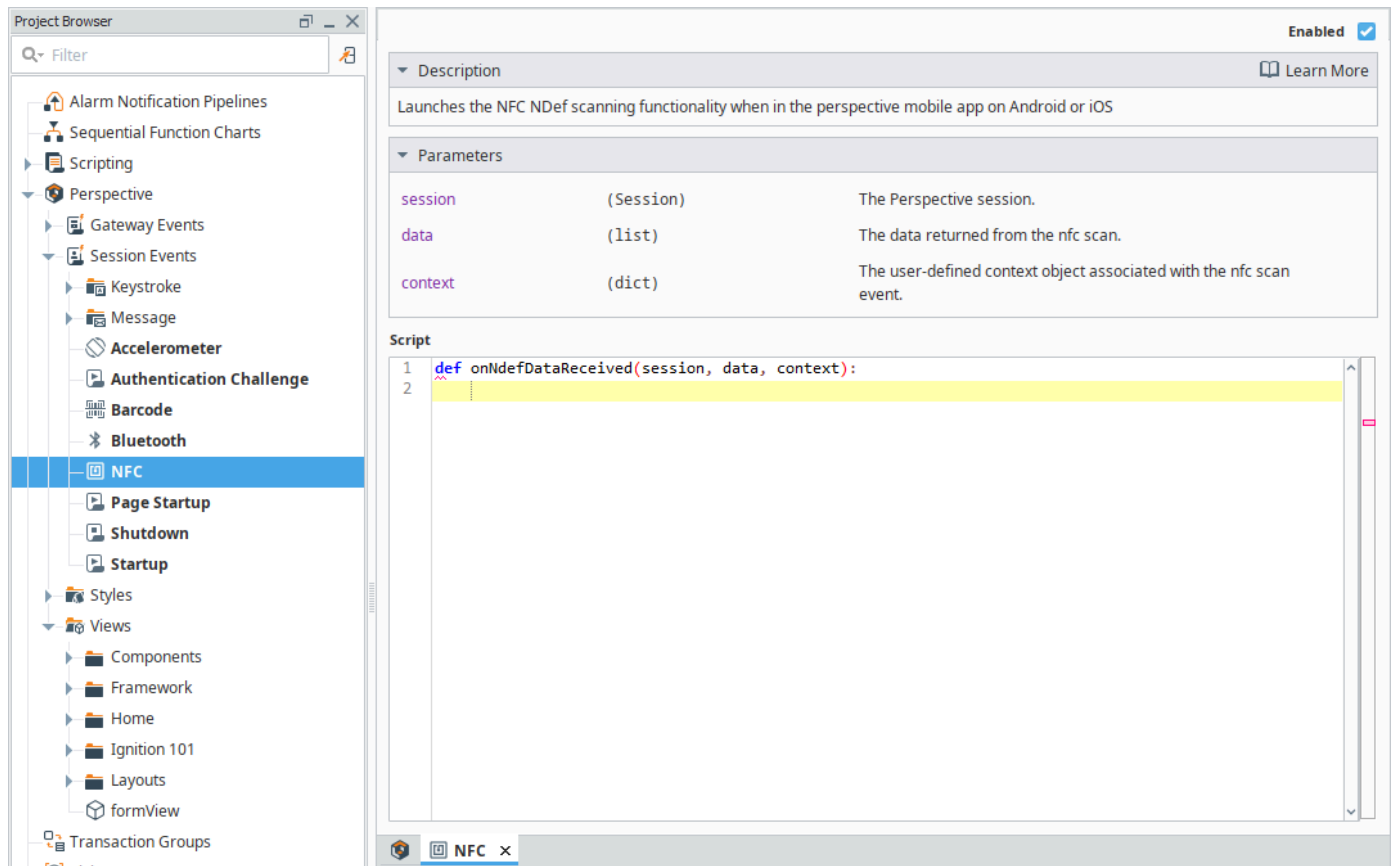
NFC Example

This example will write the NFC tag data to the Gateway logs.

1. Place a Button component onto a view.
2. 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**.



6. Next we need to set up a Session Event so that Perspective knows how to interpret the NFC data. Select **NFC** under the **Perspective > Session Events** list.



7. 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']))
```

8. 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.

- After clicking the button, the script will pass the next NFC tag scanned to the script to be handled.

2. After scanning an NFC tag, you should see the logged information appear in the Gateway logs.

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 with **Security Levels**, **Security Level Rules**, and **User Grants** you can utilize additional security features for the following items to further control user access:

- Perspective Sessions
- Perspective Views
- Event actions on Perspective components

Perspective Sessions Security



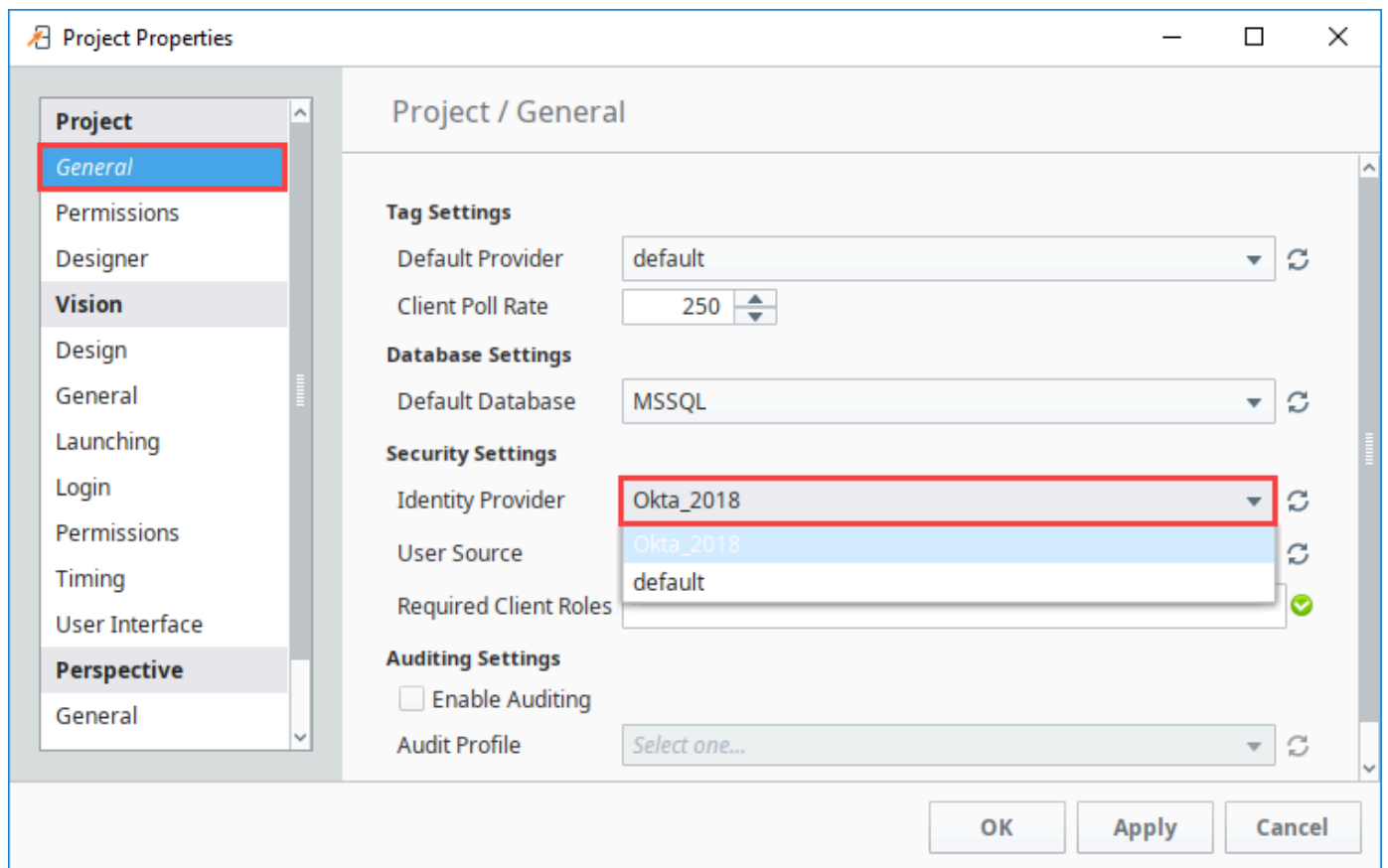
INDUCTIVE
UNIVERSITY

Requiring Authentication

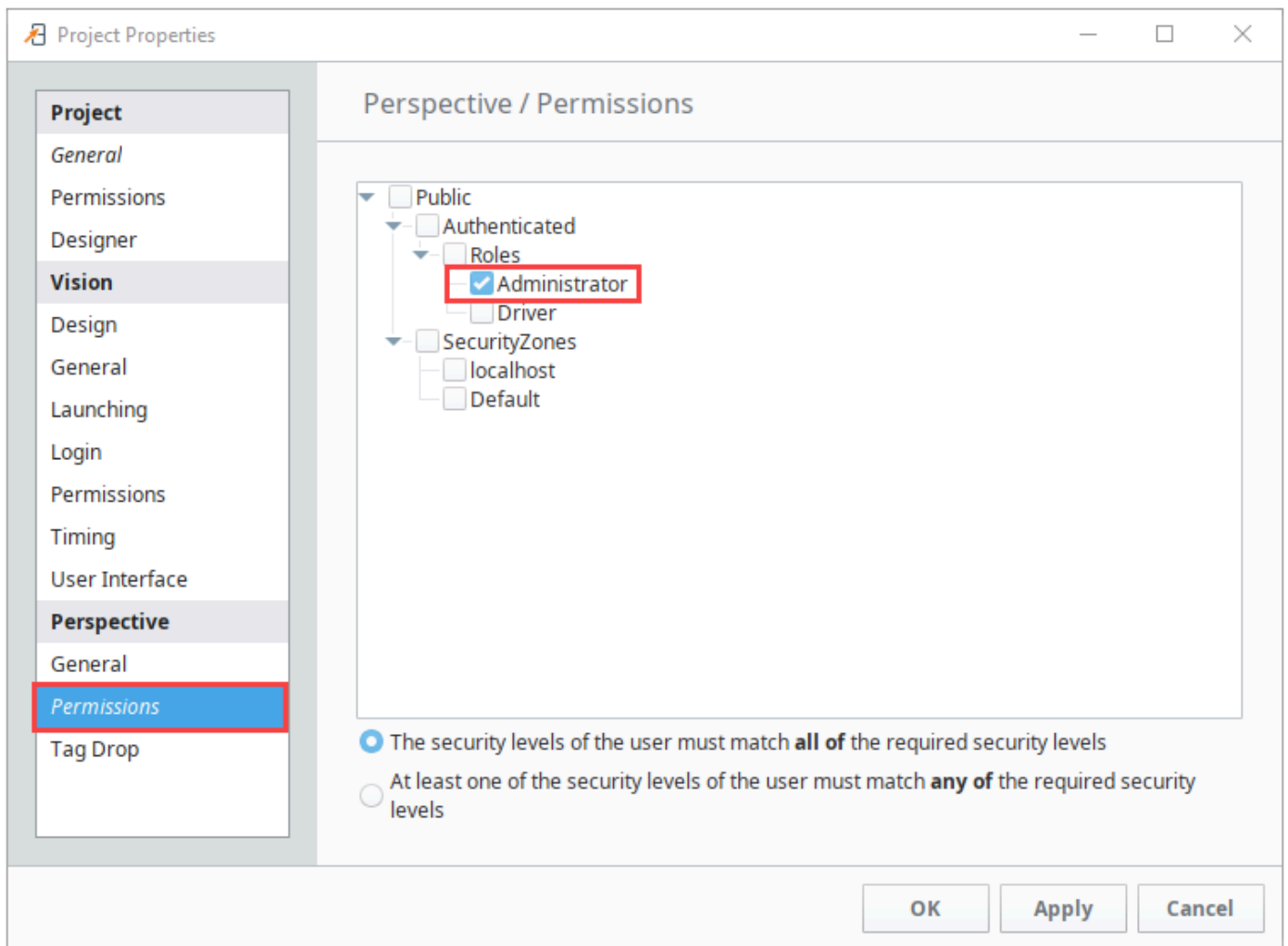
[Watch the video](#)

For each Project, you can set the security for an associated Perspective Session. When you select the security 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.
2. Select **Project > General**.
3. In the **Identity Provider** field, use the dropdown to select the IdP you want to use or to select the default user source.



4. Scroll down and select **Perspective > Permissions**.
5. Expand the tree to view the available security levels.
6. Click the check box next to each of the security levels you want to grant access to for this project in a Perspective Session.



7. Click **OK** to save all Project Properties changes.

Perspective Views Security



INDUCTIVE
UNIVERSITY

View Security

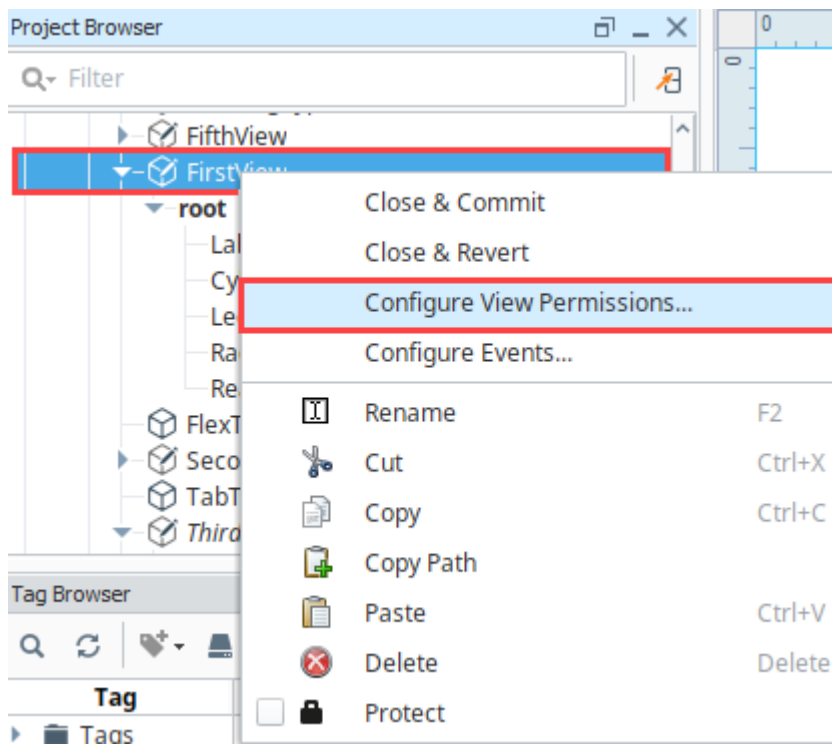
[Watch the video](#)

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 View for that project.

i NOTE

Note that you must have the IdP selected in **Project Properties** > **Project General** before any additional security can be applied to a View.

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. Click the check box next to the Security Zones that will be able to access this View.
4. Select one of the radio buttons at the bottom of the screen to indicate whether the user must 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



INDUCTIVE
UNIVERSITY

Script Action Security

[Watch the video](#)

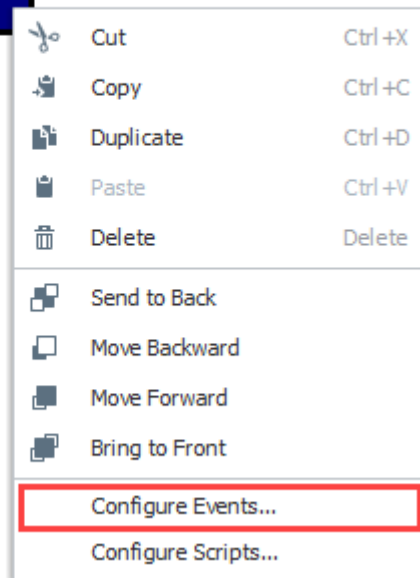
All Perspective components can have event scripts. These are scripts that run on an action, such as when the user clicks with a mouse on a component. For more information about event scripts see [Perspective scripting](#). Security can be configured on events. In the following example, we will set security for the action of clicking on a Button component in a Perspective View.

i NOTE

Before you begin, make sure you have an IdP selected in **Project Properties > Project General**.

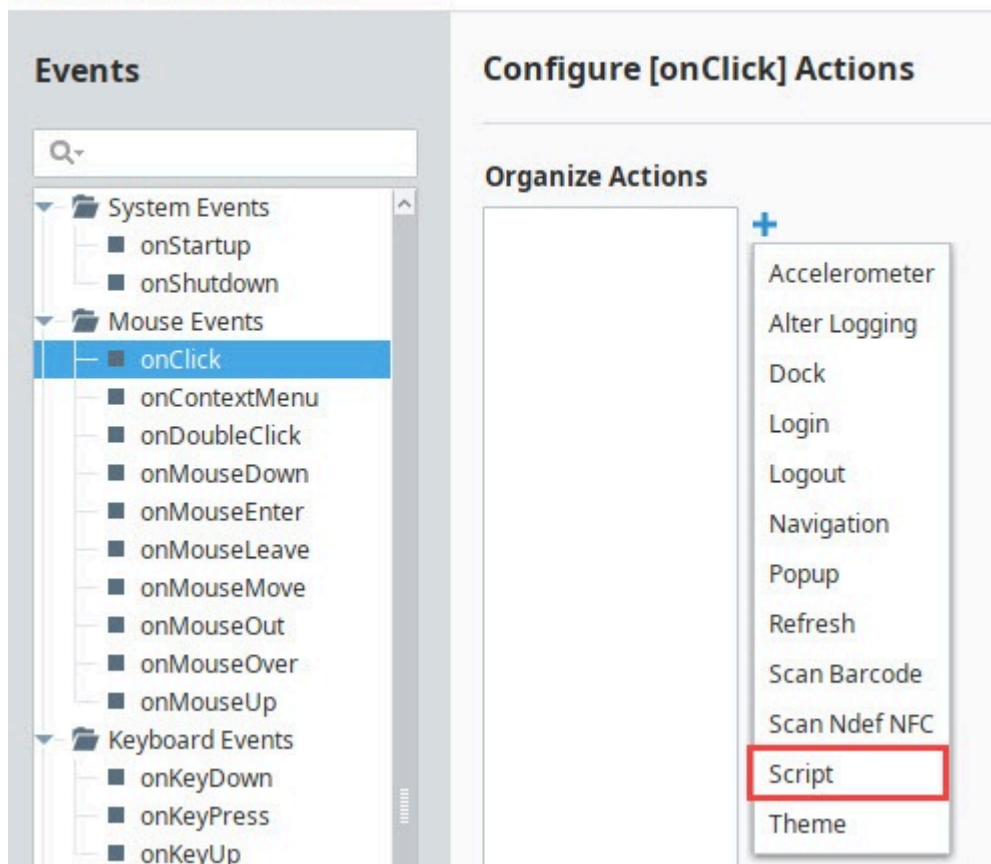
1. To add security to an event on a component, right-click on the Button component and select **Configure Events...**

Update



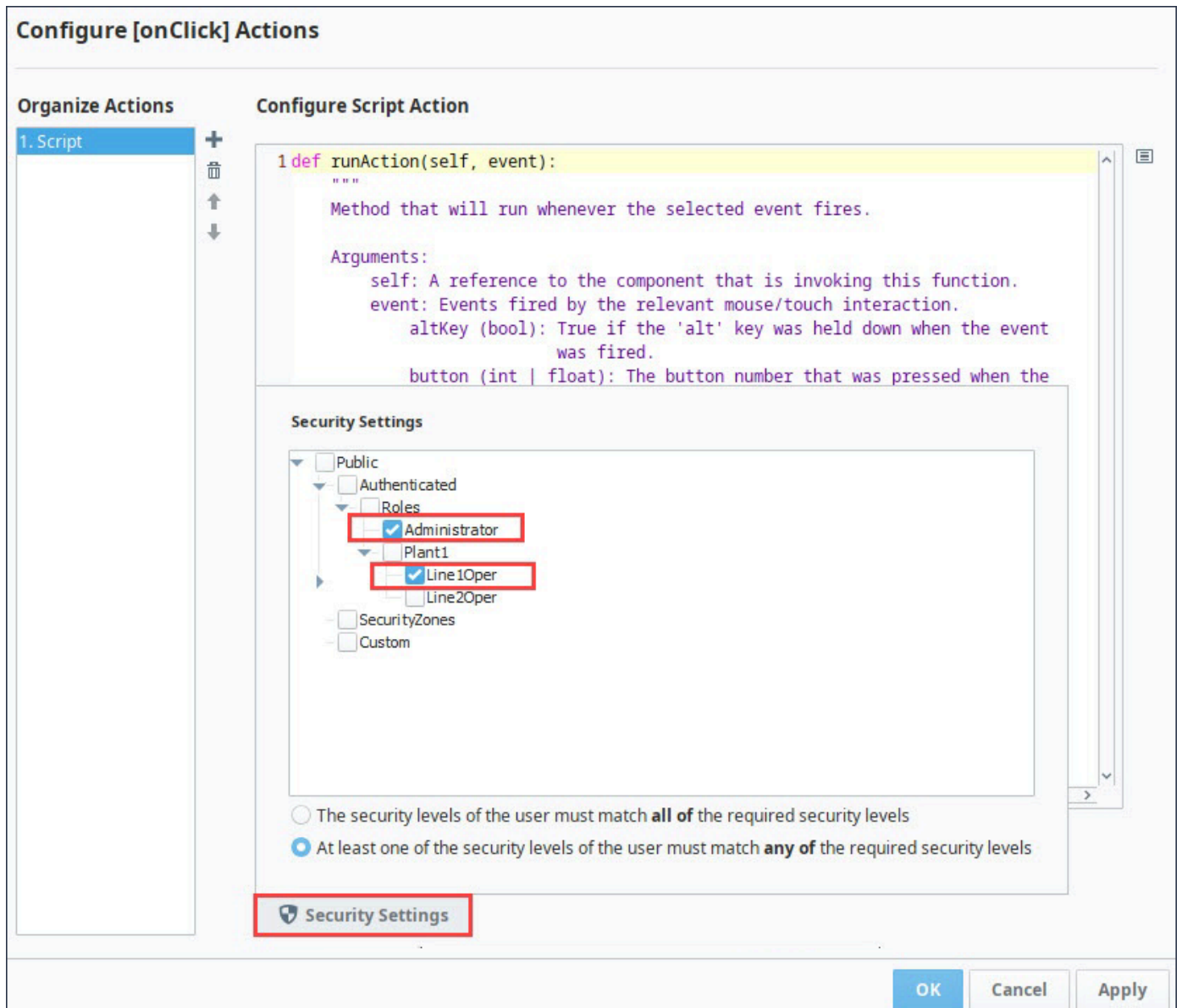
2. Choose **Mouse Events** > **onClick** in the Events Configuration screen.
3. Under **Organize Actions**, click the **Add +** icon, then select **Script** from the list.

✓ Event Configuration on Icon



4. Click  **Security Settings** 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.



6. Click  **Security Settings** again to close the window, then click **OK** to save changes.

Using Component Bindings to Indicate Authorization

Once component security settings have been configured, component property bindings can also be used to signal to users if they have access to a certain component. For example, if a component has an "Enabled" property that you want to toggle based on security levels, then you can apply one of the following approaches.

NOTE

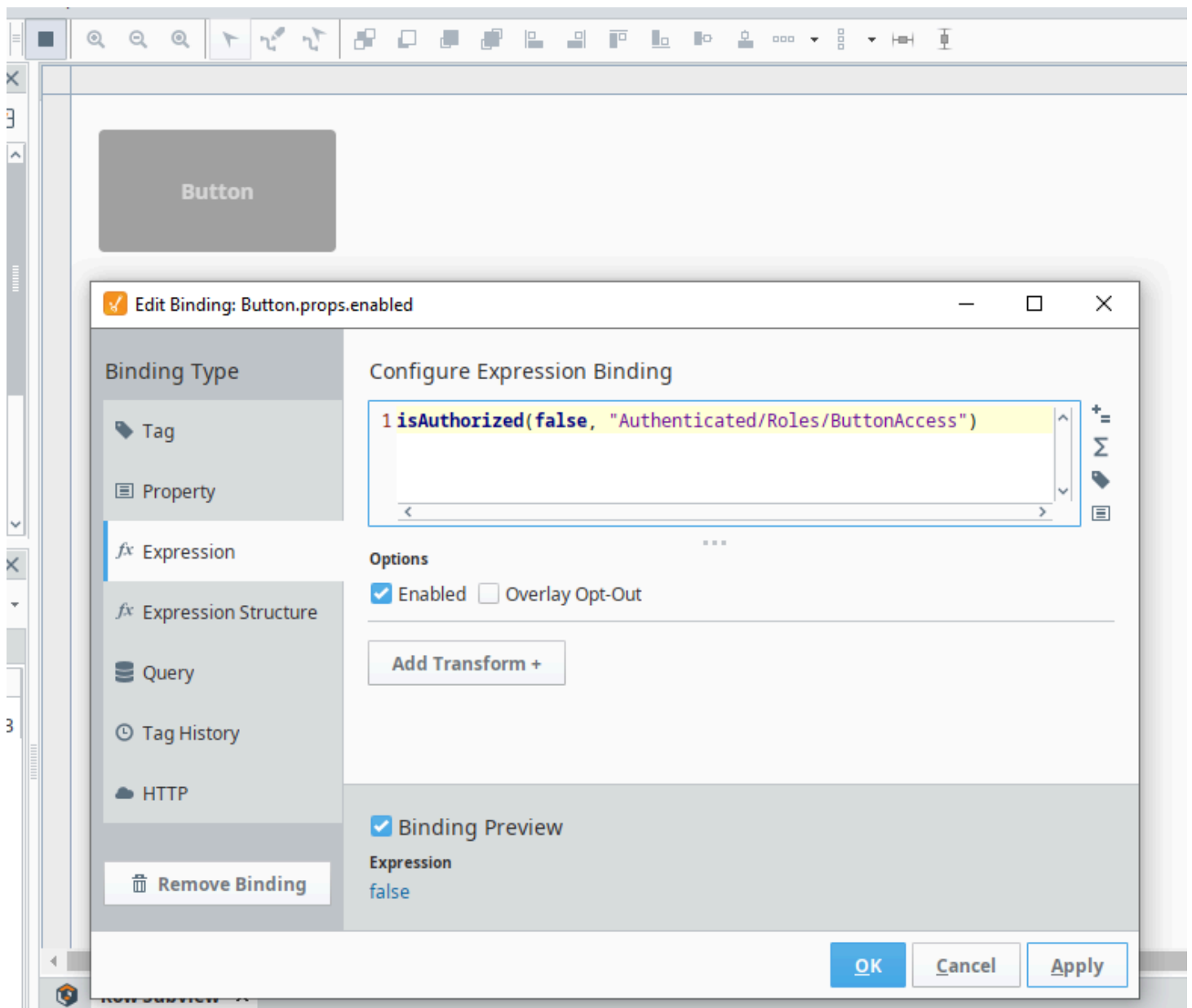
The resulting changes from these bindings serve only as visual indicators of the security settings. They do not impact the security that was already configured on the Gateway.

Using isAuthorized in an Expression

The simplest method involves configuring an expression binding that uses the isAuthorized function.

```
isAuthorized(false, "Authenticated/Roles/YourRoleGoesHere")
```

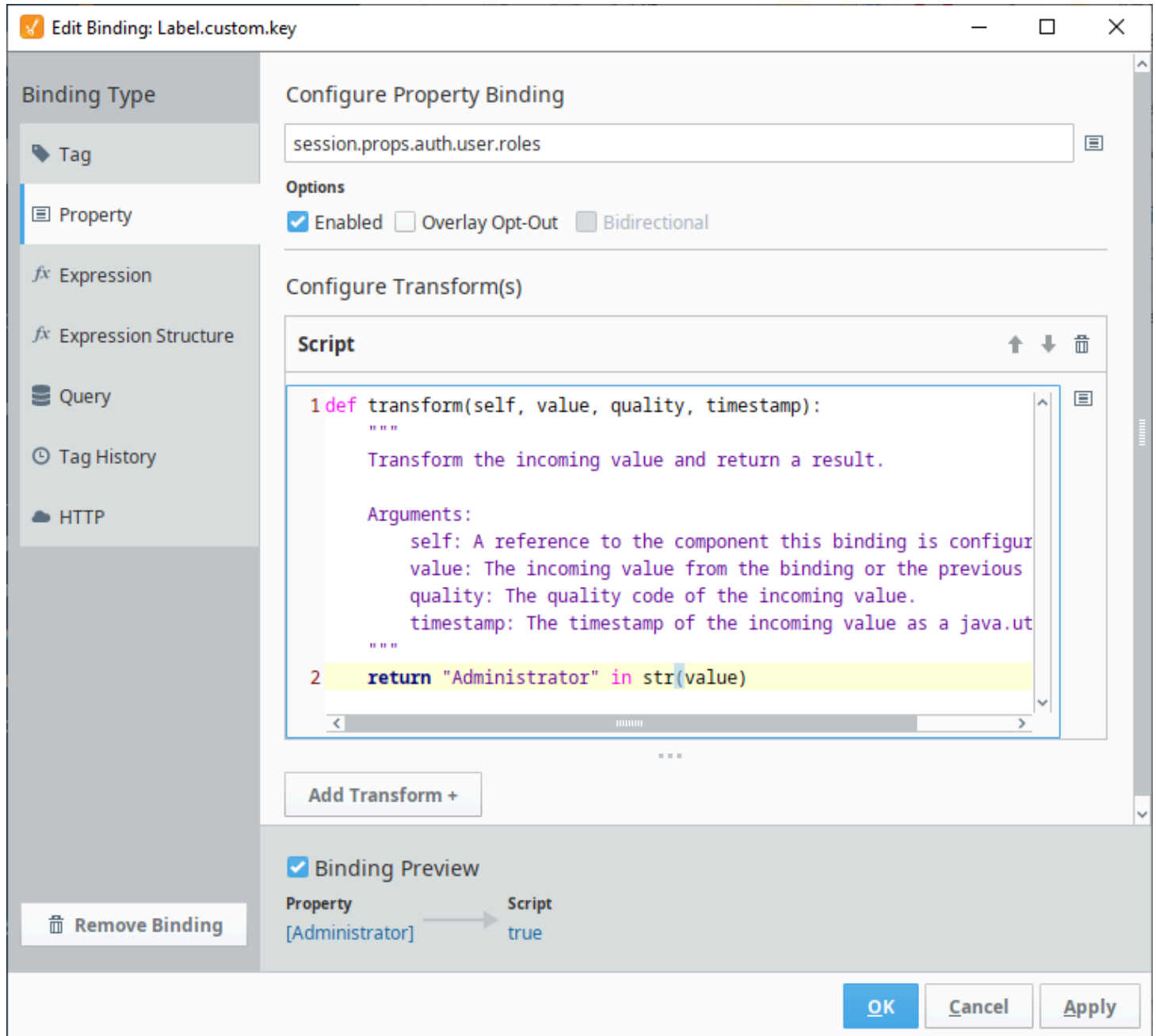
In the image below, an isAuthorized function Expression binding is configured on a Button's `enabled` property, which causes the property to change to **false** if a user doesn't have the specified security level. This has the additional benefit of preventing the `onClick` or `onActionPerformed` scripting event from triggering in cases where a user doesn't have the security level.



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)
```



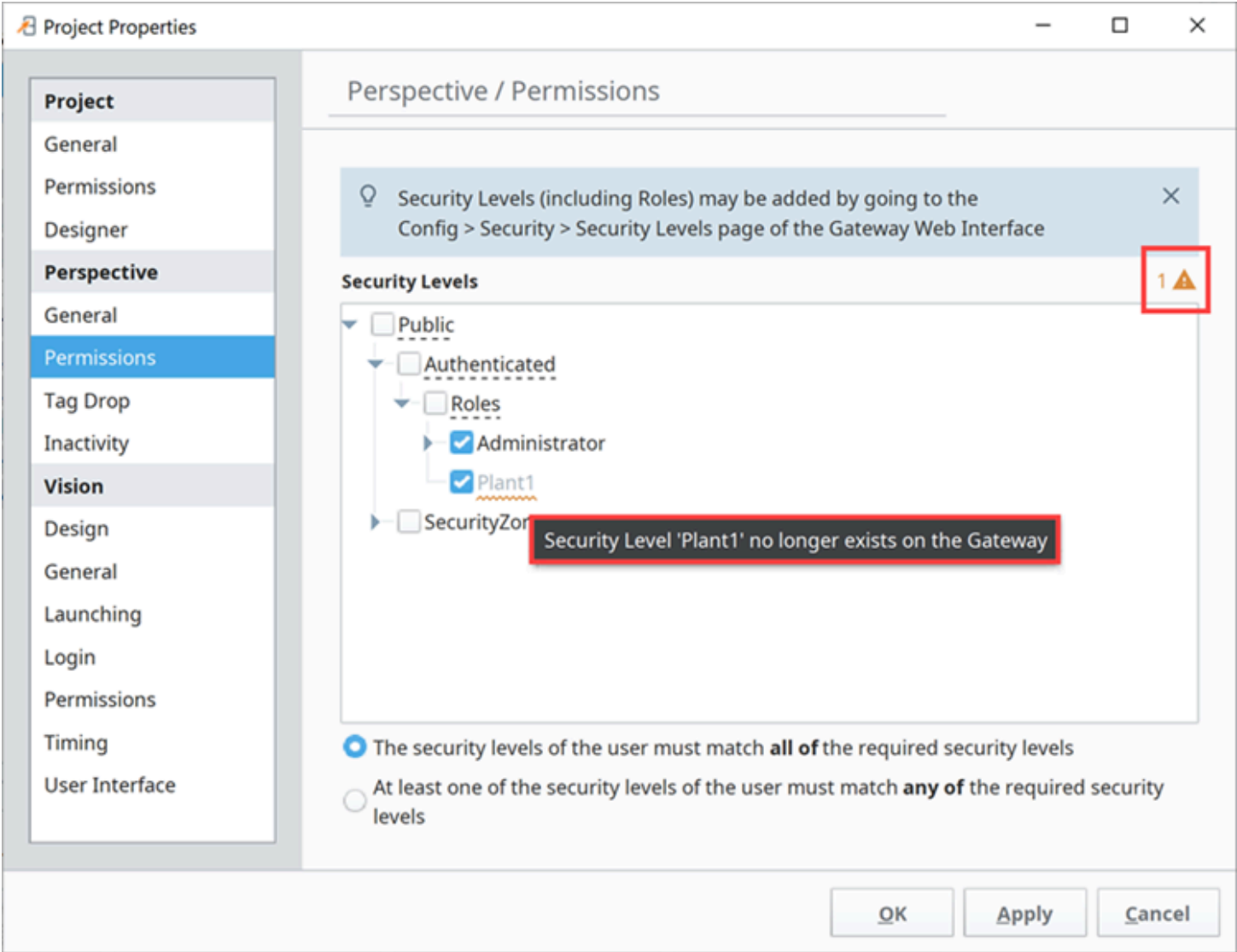
Deleted Security Level Indicators

Since security levels are set through the Gateway, it may not be immediately obvious in the Designer that a security level has been deleted. This can be checked by accessing the **Project Properties** window 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 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.



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).

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 your visualization system. Note that this assumes you already have your Tags, database, and other Gateway items taken care of.

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



INDUCTIVE
UNIVERSITY

Docked Views in Perspective

[Watch the video](#)

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.

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

Perspective Concept	Similar Vision Concept	Perspective Description
		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

Perspective Concept	Similar Vision Concept	Perspective Description
		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 <code>system.perspective.sendMessage</code> 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

Perspective Co-Branding allows users to customize the look of their Perspective Sessions with logos, pictures, and colors. Various aspects of Perspective can also be customized, such as the login screen, the desktop icon, and more.

Brand Customization

Brand properties can be configured on the Gateway by navigating to **Home > Perspective > Brand Customization**. This will open the Perspective Brand Settings page. The table below lists the elements that can be modified. These settings will not affect the internal contents or styles of the Perspective Session.

General

Property	Description
Custom Branding	Choose to toggle custom branding on or off. Default is off.

Colors

Property	Description
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.

Graphics

Property	Description
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

Property	Description
Loading Screen	Shows a preview of the loading screen with custom branding applied.
Login Screen	Shows a preview of the login screen 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	
About Title	

Best Practices

There are some considerations to take when selecting or designing a logo to use as a favicon or app icon. While some of 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.

Design

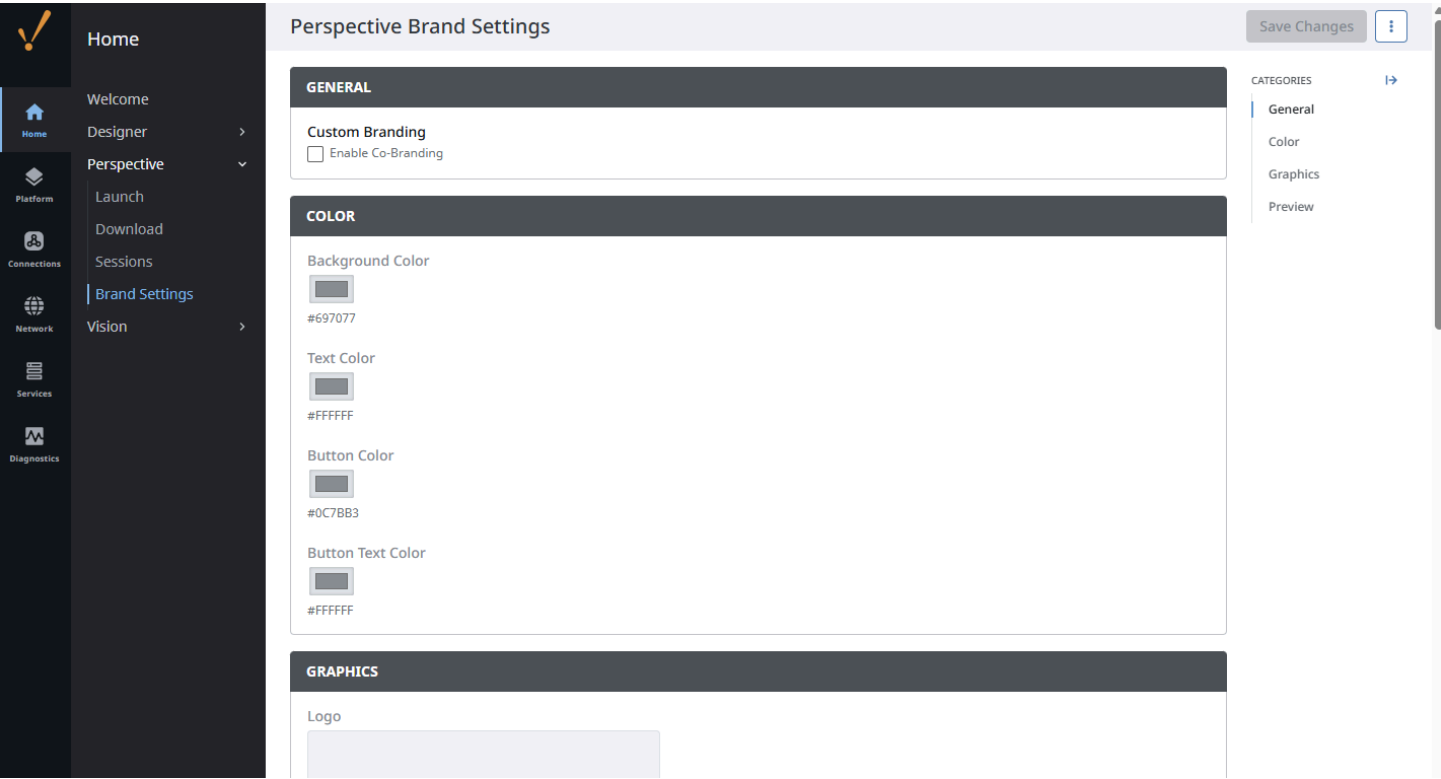
Listed below are a few basic tips when designing your Perspective login, loading, and terminal screens. These tips are not mandatory, as styles, personalization, and customization preferences differ from person to person and organization to organization. Additionally, these tips may extend to apply during 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

Login and Terminal Screens

The settings below will modify the look of the Perspective login, loading, and terminal screens:



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

Powered by Ignition!



test

You must log in to continue.

CONTINUE TO LOG IN

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!

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:

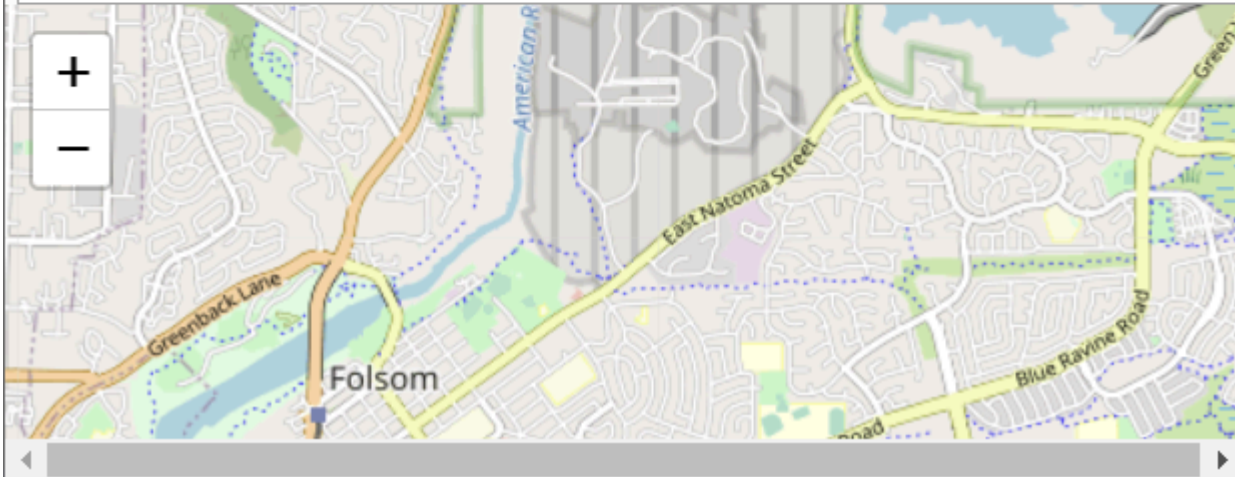


The following is the rendered output affected by the path and title properties:

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.



Time Zones in Perspective

A Perspective session can run with multiple different time zones simultaneously between the Gateway, Perspective session, and browser settings. This page provides guidance on how to incorporate multiple time zones in Ignition 8.1.3+ when using Perspective.

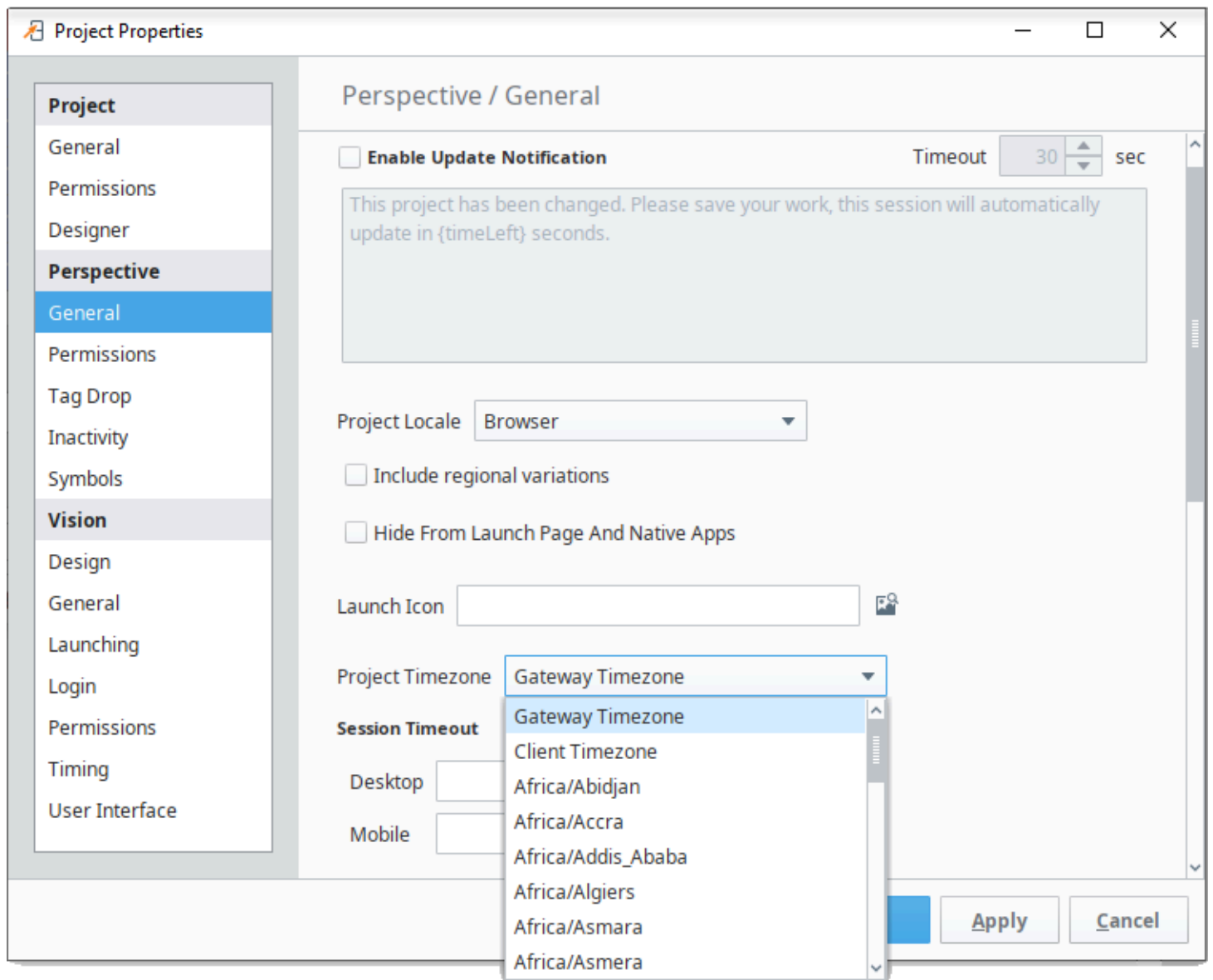
The multiple settings and methods available ensure the correct time zone is applied and visible to the user, while reducing complexity and the potential for confusion. For example, various components that display a formatted date and time based on the time zone of the user's browser can be used along with a script that applies custom formatting to display the date and time to the user in the Gateway's time zone.

Time Zone Settings

The time zone can be set with existing properties through either the Project Timezone setting, found in **Project Properties > Perspective > General**, or the selected component `timeZoneId` session property.

Project Properties - Project Timezone

Perspective includes a Project Timezone setting in the **Project Properties** that allows the developer to control what time zone is used for a Perspective session.



Beyond standard regional time zones, there are three additional options available for custom configuration: Gateway, Client, and Specific.

Gateway Timezone

This option uses the same time zone as the Ignition Gateway. This can help keep the date and times that are displayed by most components and scripts consistent between users located in different time zones.

Client Timezone

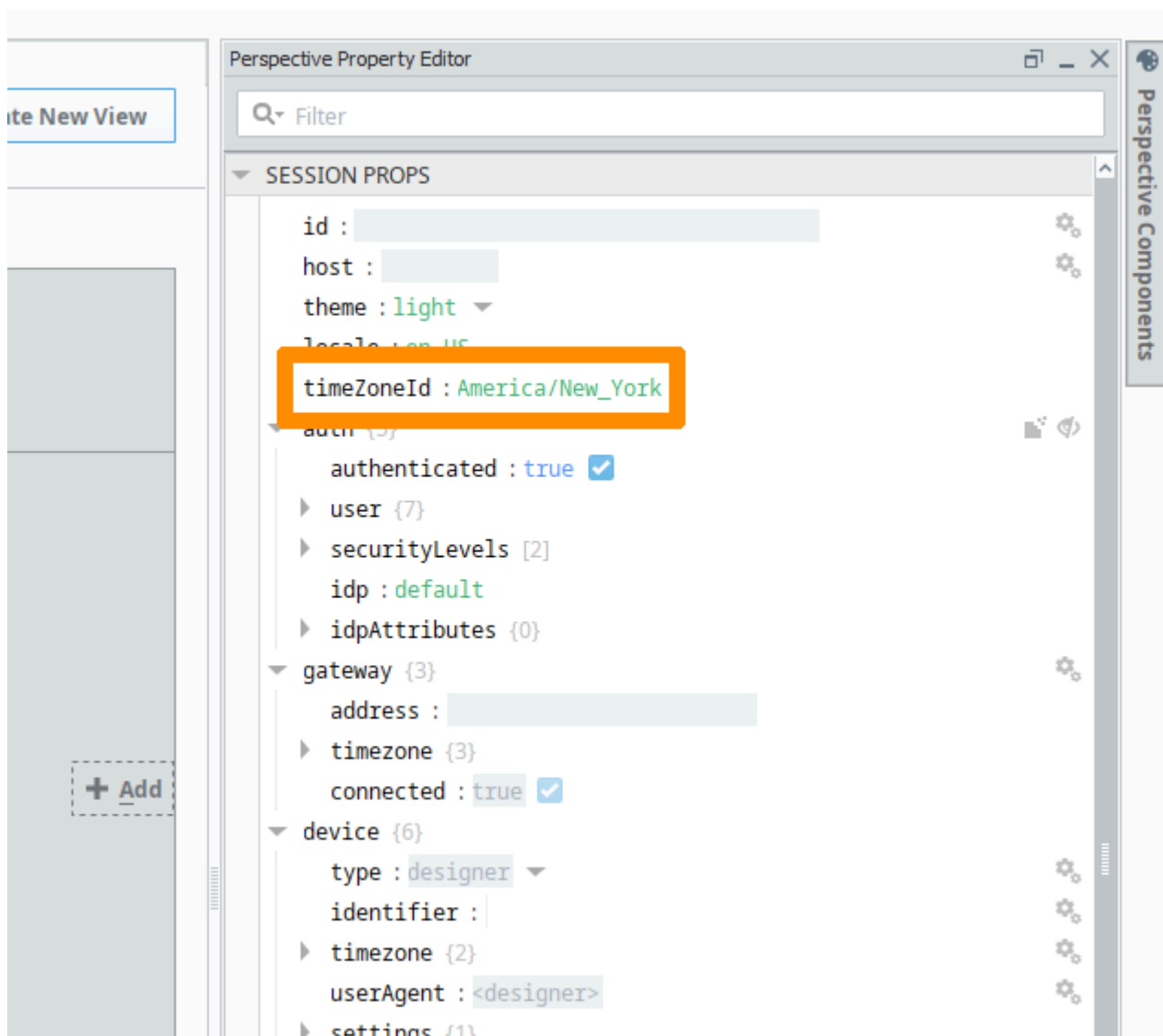
This option uses the browser's time zone as the Perspective session time zone. This can help maintain consistency between different components and displays for a given user. However, this option could result in date and time components displaying values outside of the user's time zone. If so, it should be made clear to the user the given date and time is in a different time zone.

Specific Timezone

This option allows the developer to specify a certain time zone for all users. This is another way to maintain consistency between users. However, certain components and custom formatted date and time values could still be displayed in another time zone, so the end user should be made aware of the difference in those cases.

Component Session Property - Time Zone ID

The component **session properties** include `timeZoneId` for the Perspective session's current time zone. This value is the name of the time zone. For example, **America/New_York** can be used. If no value is entered into the `timeZoneId`, the selected Project Properties Project Timezone is used.

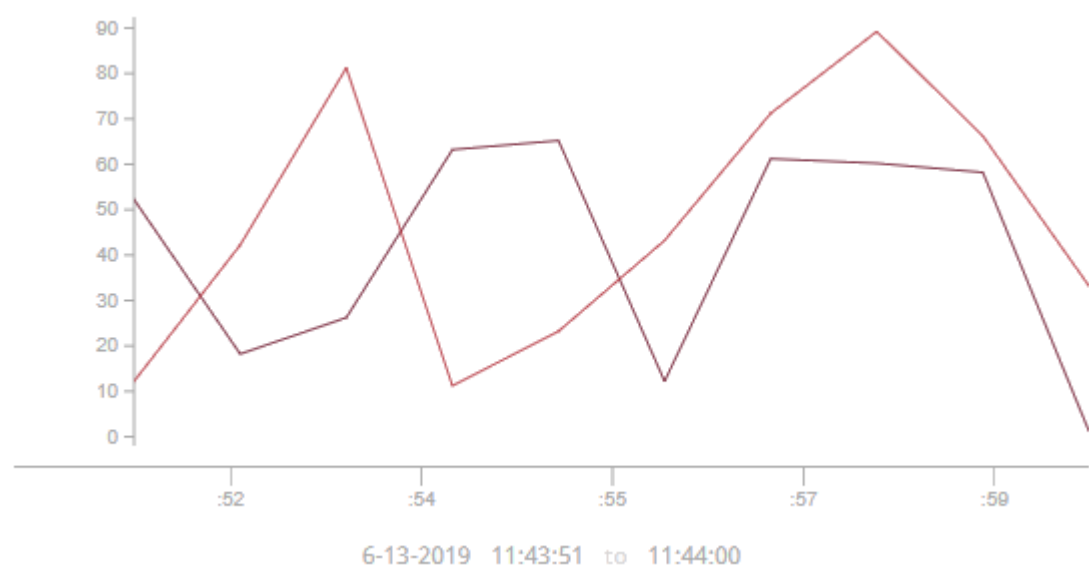


Click [here](#) for a full list of time zones.

Components

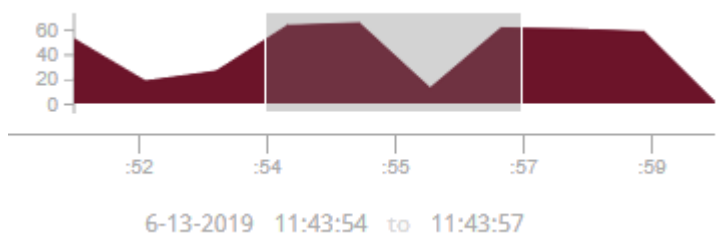
Although components display date and time based on the configured Perspective session time zone by default, a script transform can be used to further change the date and time format.

Time Series Chart



To change the time zone dynamically, or allow the user to select their own time zone, the `timeZoneId` session property can be bound to an input component. A Dropdown component could then be used to give the user an easy way to select from a list of values. After changing the time zone, historical queries can be refreshed to see the results with the timestamp in the new time zone.

Chart Range Selector



To change the time zone dynamically, the same approach to changing the `timeZoneId` session prop can be used, as described for the Time Series Chart component.

Table

t_stamp	Sine0	Sine2
1,566,328,579,173	87.67	47.31
1,566,328,579,773	86.16	46.20
1,566,328,585,173	42.79	0.96

1

2

3

4

By default, the dataset that is returned from a Tag History Query binding will display the timestamp as a long integer. There is a feature available that can automatically format this into a human readable date and time string. This can be enabled by changing the column's render type to **date**. After enabling this render type, the date and time will be formatted according to the browser's time zone. The current time zone can be added to the displayed date and time by adjusting the `dateFormat` property, such as entering MM/DD/YYYY HH:mm:ss z.

To format the values in a different time zone, a script transform can be applied to a Tag History binding under `props.data`.

Time Zone Script Transform Example

```
convertedTimestamps = []
from java.text import SimpleDateFormat
from java.util import TimeZone

dateFormatIn = SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS")
dateFormatIn.setTimeZone(TimeZone.getTimeZone("UTC"))

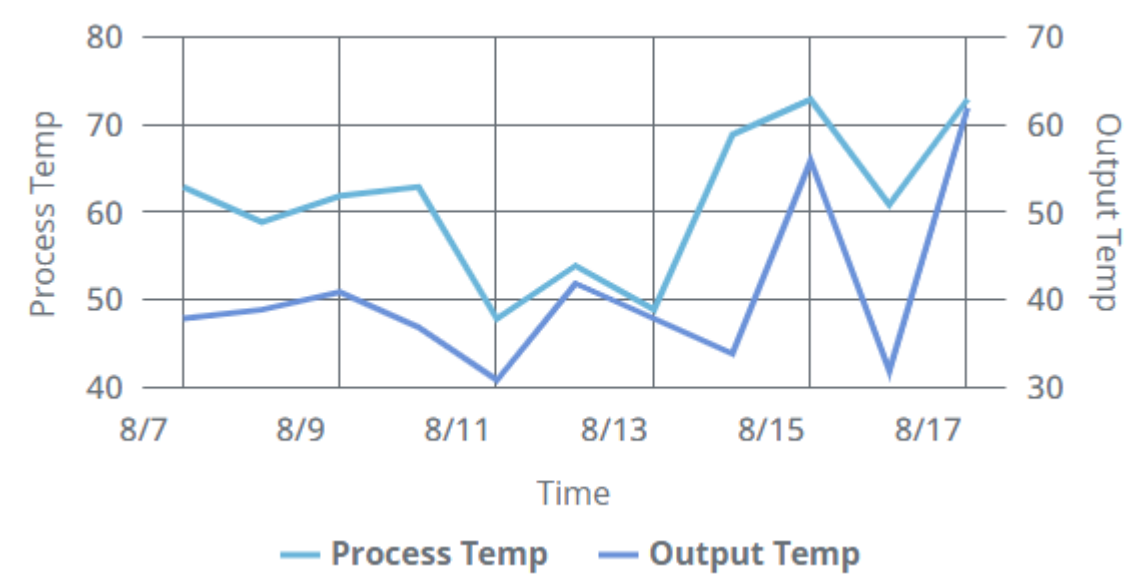
dateFormatOut = SimpleDateFormat("yyyy-MM-dd hh:mm:ss.SSS Z")
dateFormatOut.setTimeZone(TimeZone.getTimeZone(self.session.props.timeZoneId))

for row in range(value.getRowCount()):
    # Start with UTC timestamp
    date = dateFormatIn.parse(str(value.getValueAt(row, 't_stamp')).toInstant())

    # Then format with the Session Time Zone and add to the new column list
    convertedTimestamps.append(str(dateFormatOut.format(date)))

return system.dataset.addColumn(value, convertedTimestamps, 'formatted_timestamp',
str)
```

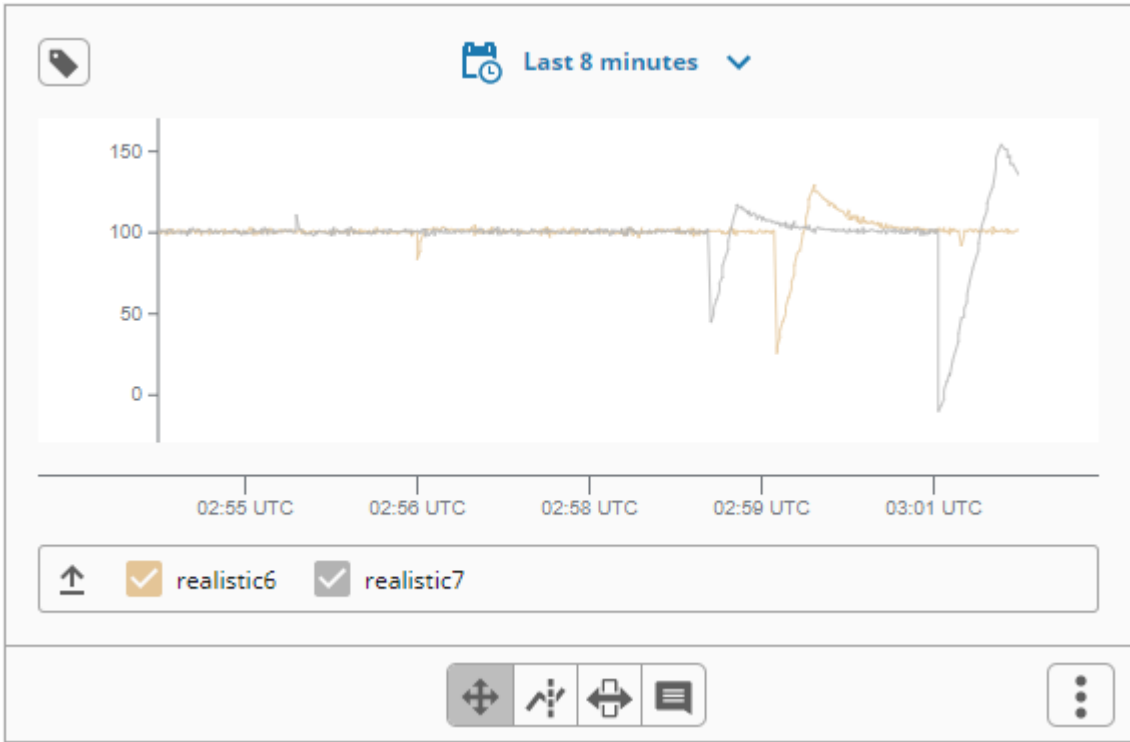
XY Chart



The XY Chart will display the formatted date and time value on the x-axis using the browser's time zone. The current time zone can be included in the date and time string by updating the date `inputFormat` and `format` properties of the x-axis.

If a category type of chart is used, then a similar script to the [Table example](#) can be included to convert a timestamp from one time zone to another.

Power Chart



The current time zone can be added to the display by adjusting the `timeAxis.tick.label.format` property, such as entering `h:mm z` to display the hour, minute, and time zone of the value.

When exporting a CSV, the time zone for the selected date range in historical mode can also be displayed by updating the `config.export.timeFormat` property.

Date Time Input

May 31, 2023 5:00 AM

< May 2023 >

Su	Mo	Tu	We	Th	Fr	Sa
30	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

05 : 00

AM

The Date Time Input will use the browser’s time zone as the offset for the actual date and time value selected. If a different time zone is needed, then a script similar to the [Time Zone Script Transform Example](#) can be used to create an adjusted value property. Now instead of referencing the component’s built-in value property, the custom value property can be used.

Date Time Picker

< May 2023 >

Su Mo Tu We Th Fr Sa

30 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

12 : 00

AM

The Date Time Picker component works in the same way as the Date Time Input.

Alarm Status Table

1 ACTIVE

0 SHELVED

SEARCH

LIST

SETTINGS

FILTERS (7):

Active, Unacknowledged

Active, Acknowledged

Cleared, Unacknowledged

Priority: Low

Priority: Medium

Remove All

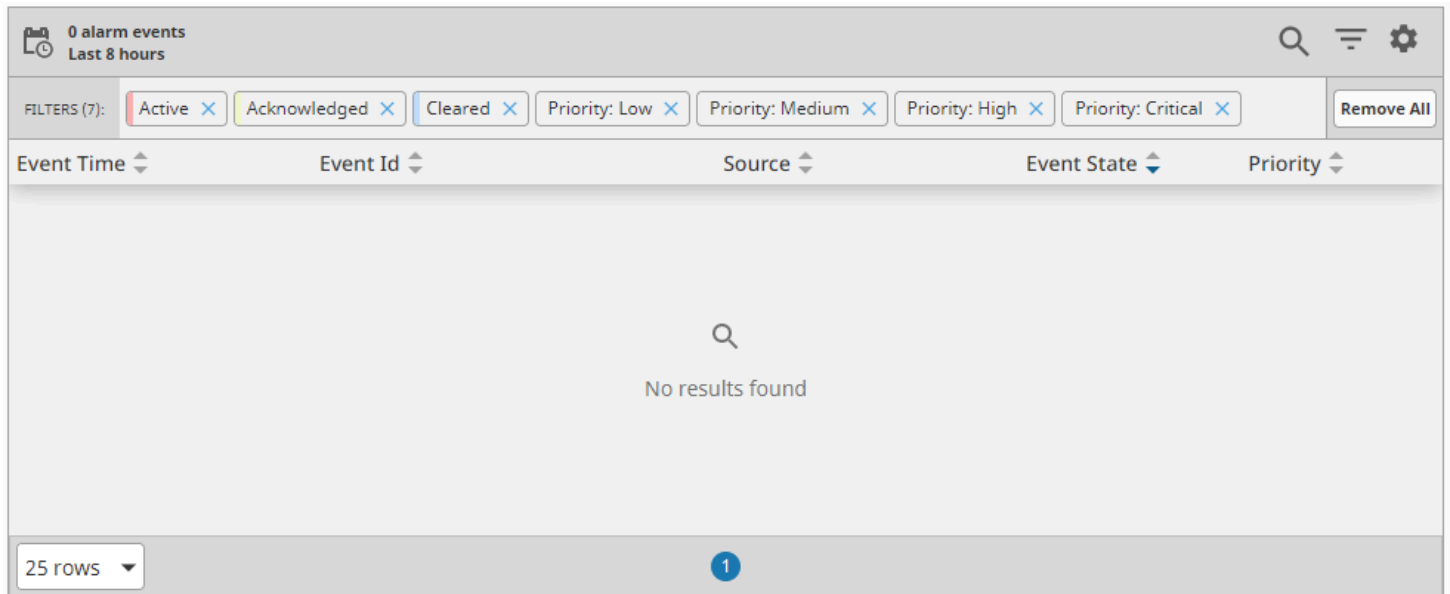
☐ Active Time	Display Path	Priority	State	Source	Name
☒ 05/16/2023 13:44:43	Level Lo Alarm	Medium	Active, Unac...	prov:Sample_Tags:/tag:Writ...	Lo
☐ 05/30/2023 14:59:22	Ramp High Alarm	Low	Cleared, Un...	prov:Sample_Tags:/tag:Ram...	High Alarm
☐ 05/30/2023 15:19:21	Ramp High Alarm	Low	Cleared, Un...	prov:Sample_Tags:/tag:Ram...	High Alarm
☐ 05/30/2023 15:14:21	Ramp High Alarm	Low	Cleared, Un...	prov:Sample_Tags:/tag:Ram...	High Alarm
☐ 05/30/2023 15:09:21	Ramp High Alarm	Low	Cleared, Un...	prov:Sample_Tags:/tag:Ram...	High Alarm
☐ 05/30/2023 15:04:22	Ramp High Alarm	Low	Cleared, Un...	prov:Sample_Tags:/tag:Ram...	High Alarm

25 rows

1

To adjust the time zone for the Alarm Status Table, the `timeZoneId` session property can be bound to an input component. The formatted date and time can also include the time zone in use by updating the `dateFormat` property to something like `MM/DD/YYYY HH:mm:ss z`.

Alarm Journal Table



Event Time	Event Id	Source	Event State	Priority
No results found				

The Alarm Journal Table works in the same way as the Alarm Status Table. It also includes a `dateFormat` property that can be used to include the current time zone.

Bindings, Transforms, and Scripts

Tag

Tag bindings that reference a `DateTime` data type will return a `java.util.Date` object, which is based on the Unix epoch and in UTC. Depending on where the binding is used, the resulting formatted date and time will be displayed according to that component or property's preference. For example, a `Label` component will display the value using ISO 8601 format in UTC, down to milliseconds: `2021-04-06T22:08:29.177Z`. Using the same tag binding in a `Table` on a cell with the render type set to **date** will result in the value being displayed in the browser's time zone, like with the Tag History Query.

Expression

An Expression binding or transform supports multiple functions that work with a `Date` object, including a `DateTime` tag. The value is based on the Unix epoch, but the end result may be formatted using different time zones according to the component or property's preference and the scope involved. For example, a `String` tag that converts a `Date` object to a string will format it according to the Gateway's time zone. The Gateway's time zone will also be applied to the same Expression on a property in `Perspective`, due to the way that bindings in `Perspective` are executed in the Gateway scope.

Property

A Property binding that returns a Date object will function similarly to the Expression binding. The resulting display will depend on what component is using the date and time value from the binding.

Format Transform

The Format transform includes an option to automatically format date and time values. This option includes the ability to select a specific time zone to use or to use the auto selection and apply the Perspective session's time zone.

Scripting

Since scripting in Perspective is executed in the Gateway scope, any date and time object that is formatted for display will be in the Gateway's time zone. A specific time zone can be applied using a script that is similar to the **one mentioned for the Table** component. If the Date object itself is returned and displayed using a component, then that date and time will be formatted according to the preference for that component.