1. Perspective . . . 2
1.1 Perspective Sessions . . . 7
1.1.1 Ignition Perspective App . . . 18
1.1.2 Session Properties . . . 32
1.2 Perspective Design Principles . . . 39
1.2.1 Navigation Strategies in Perspective . . . 42
1.3 Pages in Perspective . . . 52
1.4 Views and Containers in Perspective . . . 62
1.4.1 Coordinate Containers . . . 69
1.4.2 Column Containers . . . 77
1.4.3 Tab Containers . . . 87
1.4.4 Breakpoint Containers . . . 93
1.4.5 Flex Containers . . . 99
1.4.6 Test Your Responsive Design Using Chrome's Developer Tools . . . 111
1.5 Perspective Designer Interface . . . 114
1.6 Working with Perspective Components . . . 126
1.6.1 Perspective Component Properties . . . 140
1.6.1.1 Ignition Server Migration Best Practices . . . 160
1.6.2 Images, SVGs, and Icons in Perspective . . . 161
1.6.3 Localization in Perspective . . . 172
1.6.4 Component Events and Actions . . . 176
1.7 Perspective Project Properties . . . 185
1.8 Styles . . . 193
1.8.1 Copy of Perspective Themes . . . 199
1.8.2 Perspective Themes . . . 215
1.8.3 Style Classes . . . 229
1.9 Bindings in Perspective . . . 246
1.9.1 Tag Bindings in Perspective . . . 260
1.9.2 Property Bindings in Perspective . . . 272
1.9.3 Expression Bindings in Perspective . . . 279
1.9.4 Expression Structure Bindings in Perspective . . . 284
1.9.5 Query Bindings in Perspective . . . 289
1.9.6 Tag History Bindings in Perspective . . . 291
1.9.7 HTTP Bindings in Perspective . . . 298
1.9.8 Transforms . . . 300
1.9.8.1 Map Transform . . . 301
1.9.8.2 Format Transform . . . 306
1.9.8.3 Script Transform . . . 309
1.9.8.4 Expression Transform . . . 312
1.10 Scripting in Perspective . . . 313
1.10.1 Perspective Component Methods . . . 315
1.10.2 Component Message Handlers . . . 325
1.10.3 Perspective Property Change Scripts . . . 330
1.10.4 Perspective Session Event Scripts . . . 334
1.11 Security in Perspective . . . 349
1.12 Alarming in Perspective . . . 354
1.12.1 Perspective Alarm Status Table - Common Tasks . . . 356
1.12.1.1 Perspective Alarm Status - User Interaction . . . 357
1.12.1.2 Perspective Alarm Status - Configuring Properties in Designer . . . 360
1.12.1.3 Perspective Alarm Status - Filtering . . . 367
1.12.1.4 Perspective Alarm Status - Acknowledgement . . . 371
1.12.1.5 Perspective Alarm Status - Shelving . . . 376
1.12.1.6 Perspective Alarm Status - Row Styles . . . 381
1.12.2 Perspective Alarm Journal Table - Common Tasks . . . 390
1.12.2.1 Perspective Alarm Journal - User Interaction . . . 392
1.12.2.2 Perspective Alarm Journal - Configuring Properties in Designer . . . 394
1.12.2.3 Perspective Alarm Journal - Filtering . . . 399
1.12.2.4 Perspective Alarm Journal - Row Styles . . . 407
1.13 Reporting in Perspective . . . 413
1.14 Common Tasks in Perspective . . . 418
1.14.1 Popup Views . . . 420
1.14.2 Navigating with the Horizontal Menu Component . . . 426
1.14.3 Displaying a SubView in a Table . . . 435
1.14.4 Self-Hiding Navigation Drawer . . . 445
1.14.5 Configuring a Dashboard . . . 455
1.14.6 Download and Upload Files . . . 465
1.14.7 Table Column Configurations . . . 469
1.14.8 Carousel Component Example . . . 476

Perspective

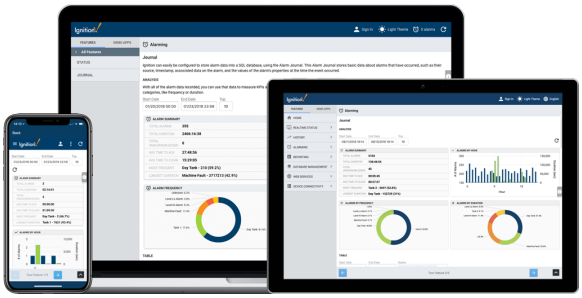
Overview

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 Modul e 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 cameras and GPS information that are natively provided by mobile devices.
- **Cascading Style Sheets HTML5/CSS3 technology** - Enables users to control their application on any device type such as smartphones, tablets, touchscreens, laptops, and desktop computers.
- **Transforms** - Let you easily take the value coming into a binding, manipulate it even further, and then *transform* it to the output of your choice.



On this page

...

- Overview
- Views and Containers
- Components
- Bindings
- Styles and Style Classes
- Security in Perspective

INDUCTIVE UNIVERSITY

Perspective Project Elements

Watch the Video

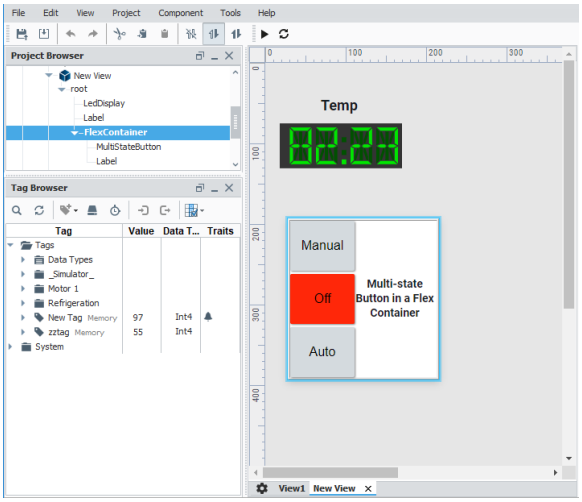
Views and Containers

Views and Containers are an integral part of the Perspective design experience because they work together to create your HMI screens - 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 withi n a View.

INDUCTIVE UNIVERSITY

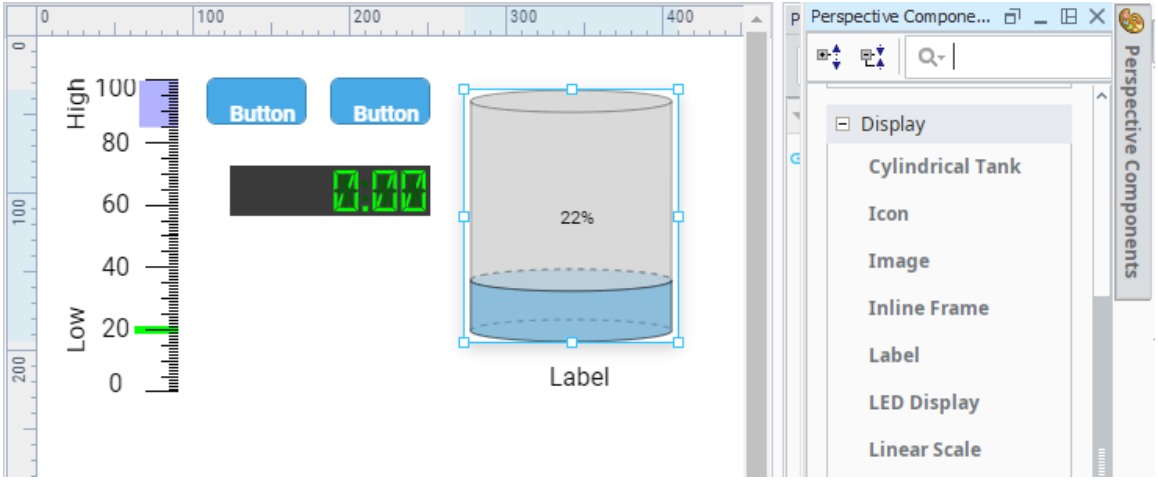
Anatomy of a View

Watch the Video



Components

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

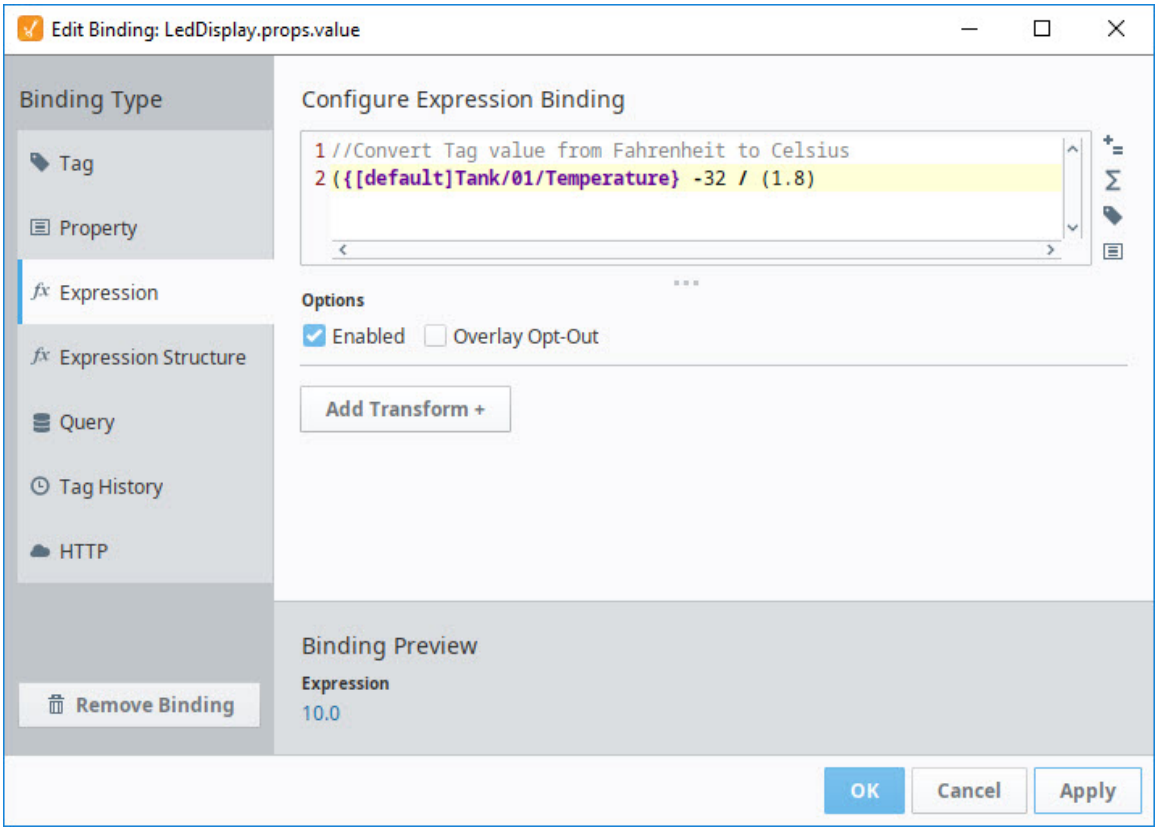


Bindings

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

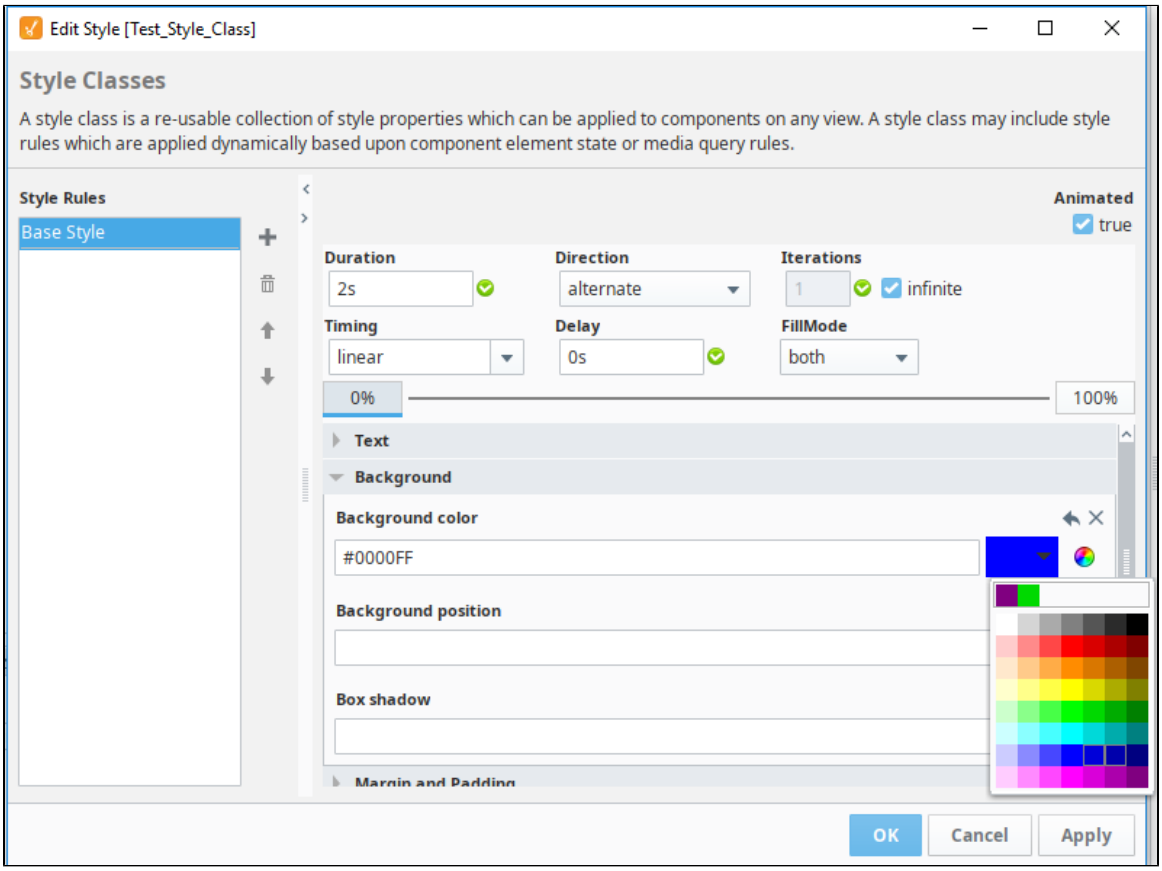
Click on the following links for complete information about binding types:

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



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.

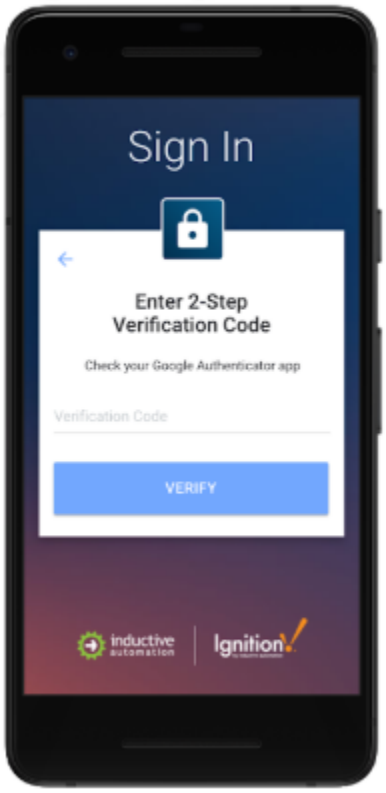


Security in Perspective

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.



Related Topics ...

- Working with Perspective Components
- Views and Containers in Perspective
- Pages in Perspective
- Quick Start - Perspective Session

In This Section ...

Perspective Sessions

Session Overview

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.

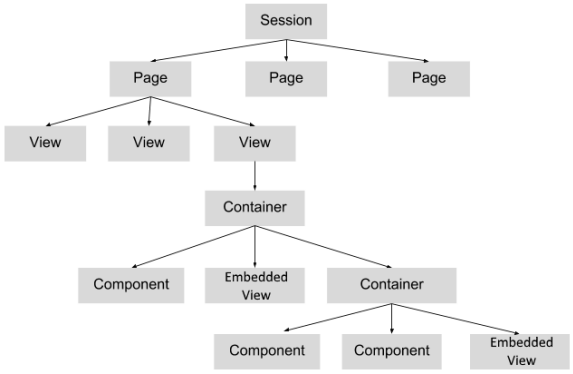
Now that you know that a session is effectively running an instance of the application, it can have multiple **Pages** in the same web browser. Perspective Sessions work the same way as web pages do on your favorite website on the internet. You can log into your favorite shopping website and then open 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.

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.



On this page

...

- Session Overview
- Session Anatomy
  - Component
  - Container
  - View
  - Page
  - Properties
  - A Simple Example
- Launching a Perspective Session
  - Launching a Session from the Designer
  - Launching a Session from the Gateway Webpage
  - Launching a Session using the Web Address
- Session App Bar
- Project Updates
- Page URLs
- Browser Caching
  - Cache-Control
- Browser Version Requirements

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 Ign ition Designer's Project Browser tree. These folders/paths are important not only for organization and referencing, but also because these paths uniquely identify each view, and are used in the session (runtime) for navigation. Each View has a container type that decides how the components inside it will behave. Multiple Views can be present within the same page, and views can also be nested. Parameters can be passed into a view from an external source.

Page

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

Properties

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

A Simple Example

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

Launching a Perspective Session

There are three ways to launch a Perspective Session:

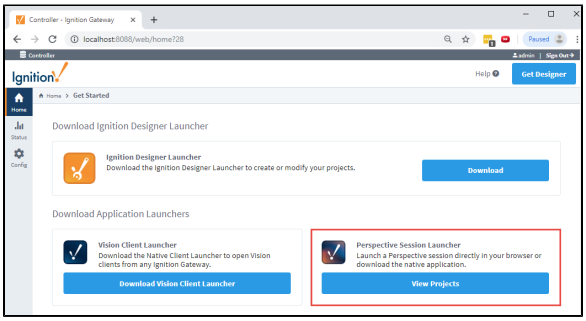
- From the Designer
- From the Gateway Webpage
- By entering the web address of the project in your web browser

Launching a Session from the Designer

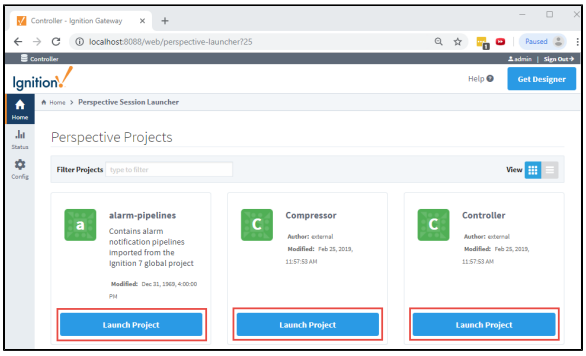
Launching a Perspective Session from the Designer is a great method if you're testing your changes. When you're in the Designer, simply click the **Tools > Launch Perspective > Launch Session** from the top menubar. This opens a session for the project you are working on directly from the Designer.

Launching a Session from the Gateway Webpage

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



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

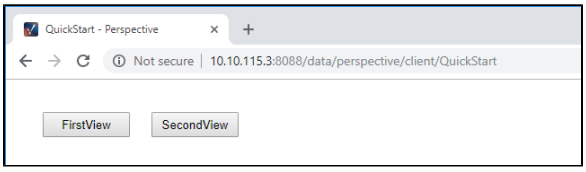


Launching a Session using the Web Address

In your web browser, enter the web address of your project. This address follows a special pattern and can either take an IPAddress:port, or if you have redirection set up on your network, it could be very simple. If you are using a local Ignition instance, you can use localhost:8088. This will take you to the Ignition installed on your computer.

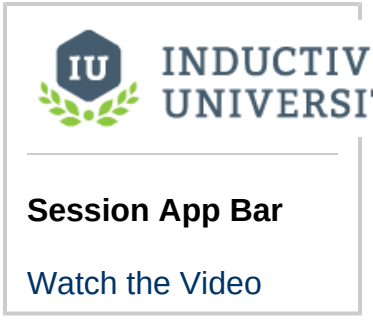
Possible Project Links

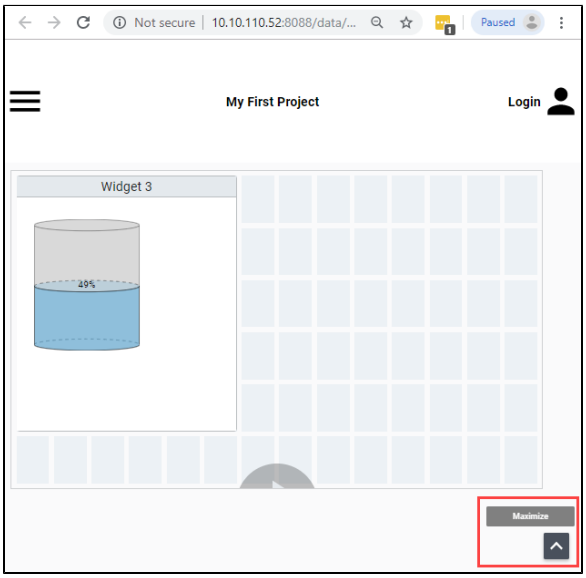
```
Generic Project Link
http://<IP Address>:<port>/data/perspective/client/<Project Name>

IP Address Project Link
http://10.20.30.40:8088/data/perspective/client/QuickStart

Redirection Project Link
http://demo.ia.io


Local Project Link
http://Localhost:8088/data/perspective/client/QuickStart
```



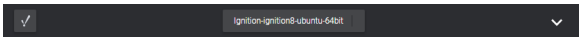
Session App Bar

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



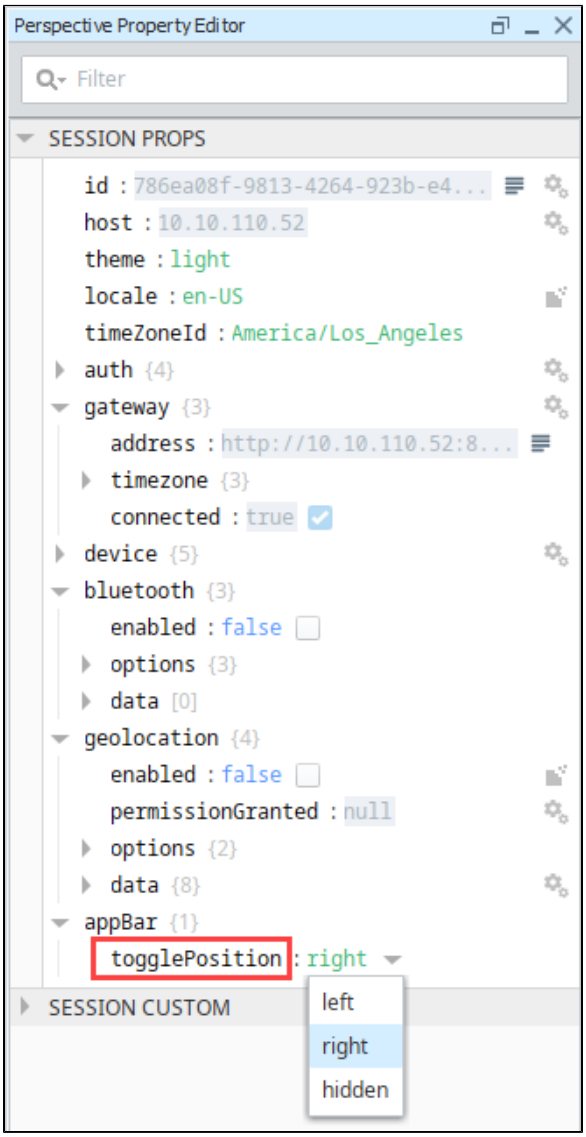


This brings up the app bar at the bottom of the window.

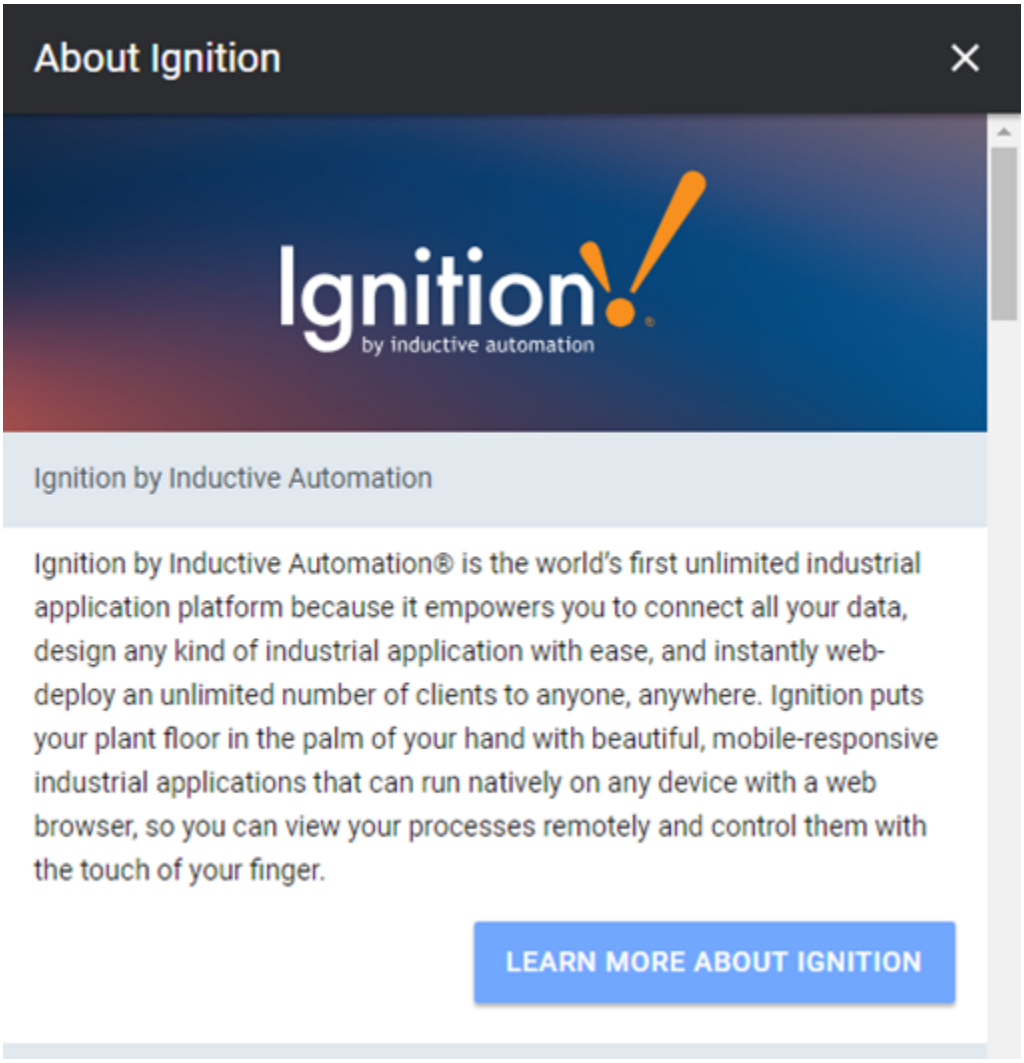


> The following feature is new in Ignition version **8.0.5**
> Click here to check out the other new features

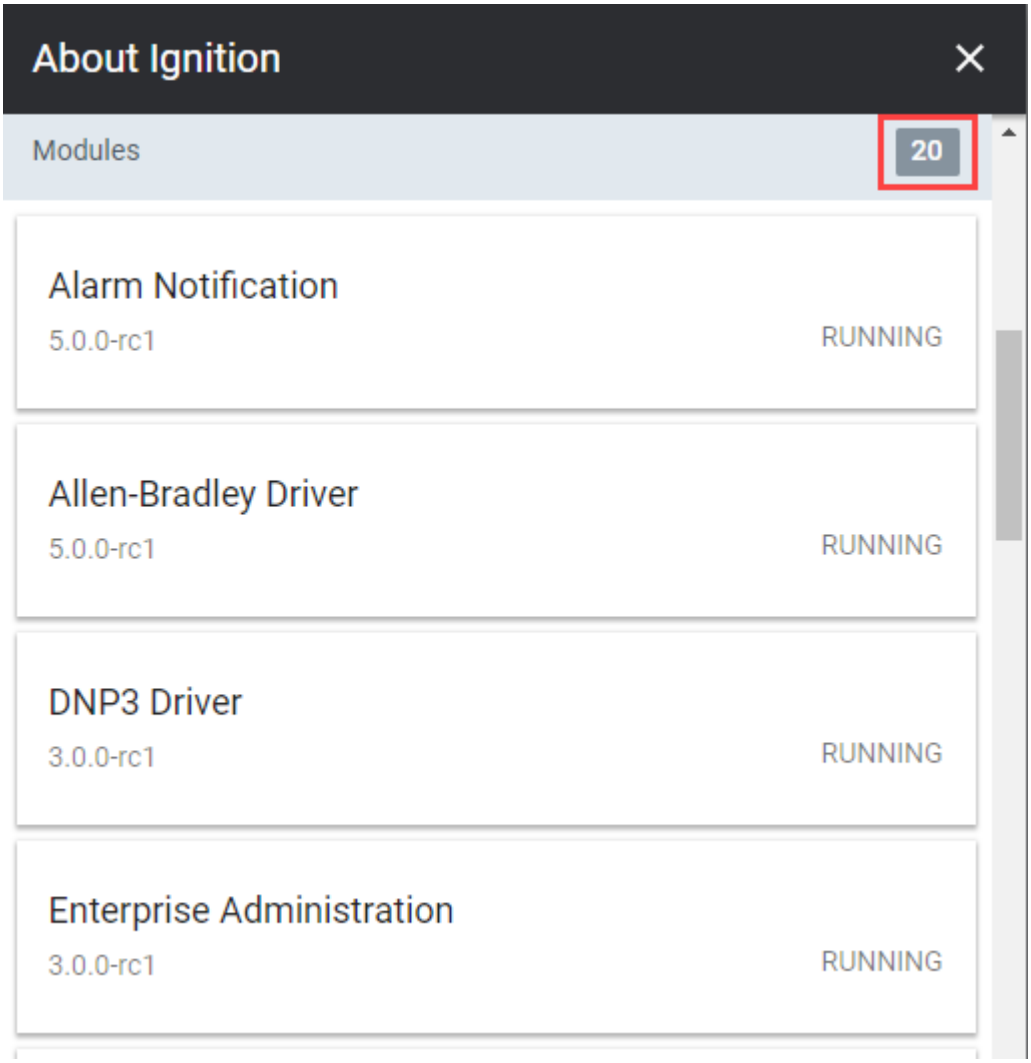
You can change the location of the Maximize/Minimize icon on the app bar in the Property Editor. Scroll down the list of properties and expand the **appBar** property. Right click on the **togglePosition** property and select one of the three options: **left**, **right** or **hidden**.



On the lower left of the app bar, click the Ignition  icon to open the About Ignition information pop up. 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.

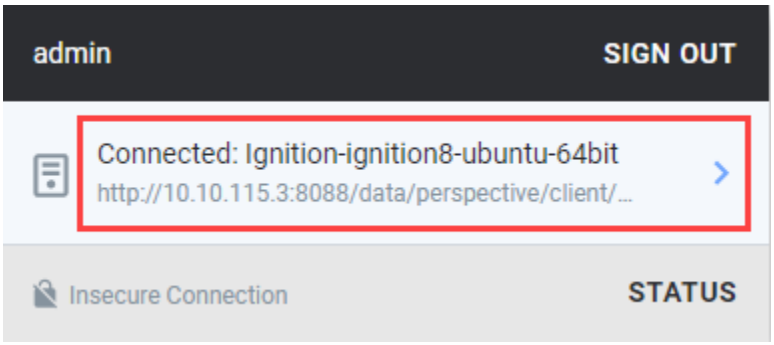


Here you'll see the number of modules installed on the Gateway, along with a list of those a modules, their version numbers, and whether they are currently running.

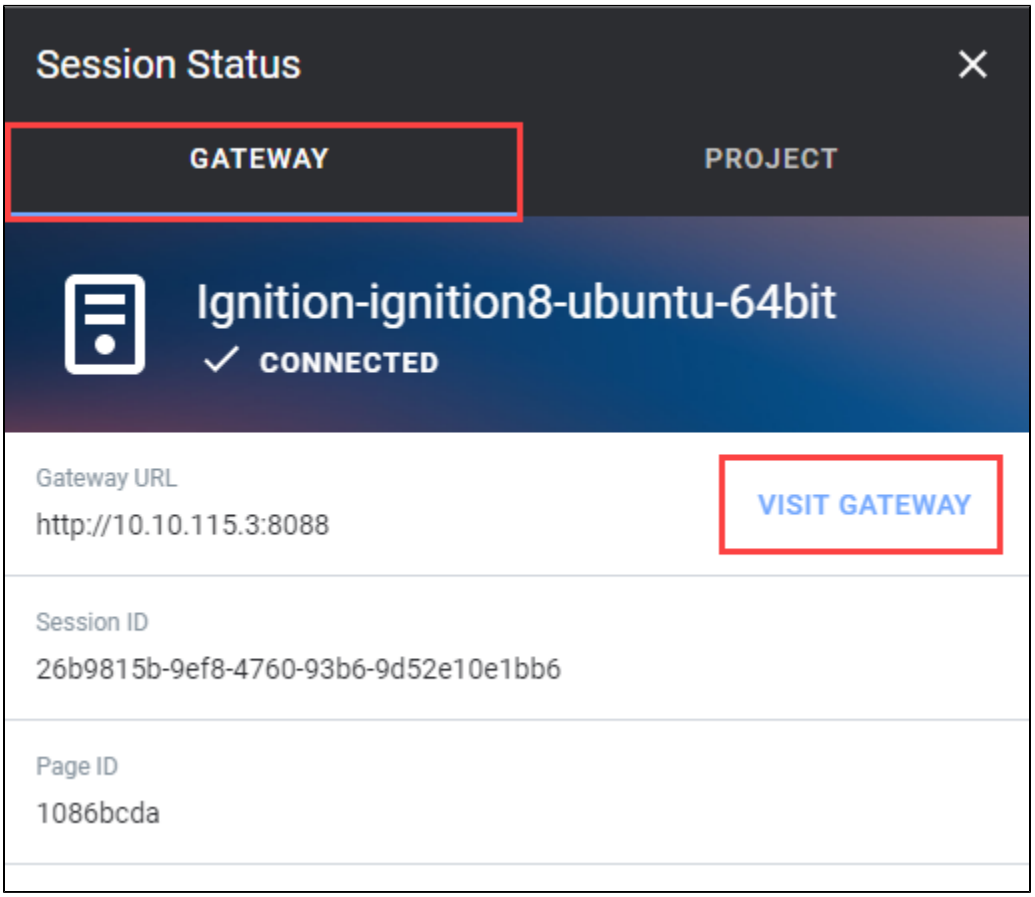


In the middle of the app bar at the bottom of the window, the current Gateway ID is displayed. Click on this for additional information. The popup screen shows the current user and connected Gateway. If no user is signed in, a Sign In button is displayed in the upper right corner.

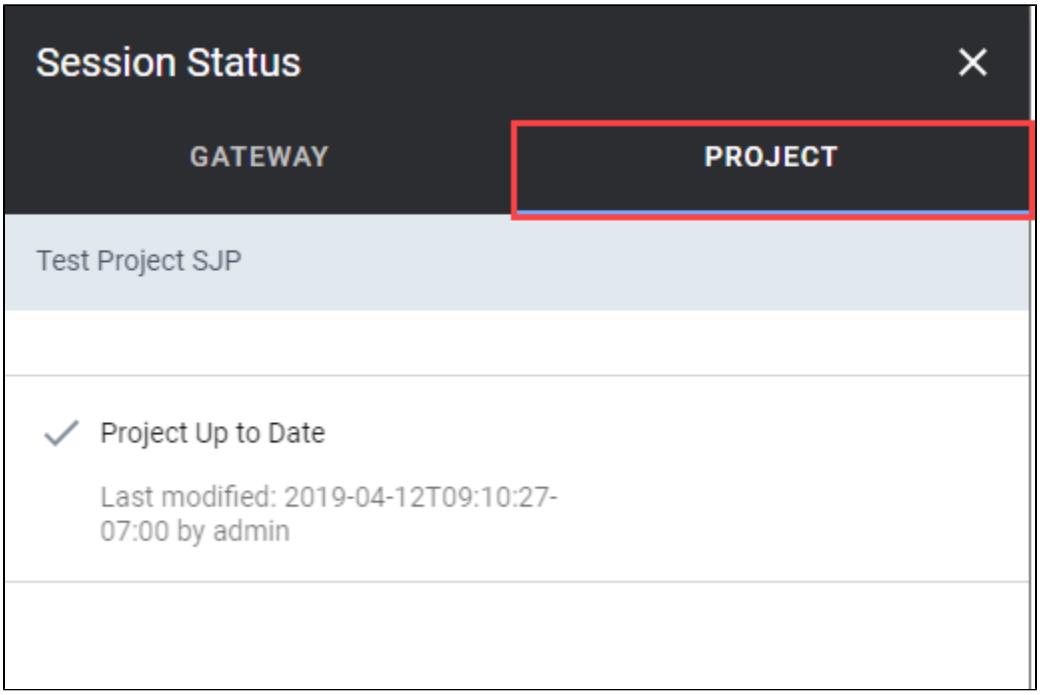
Click on the right arrow or anywhere on the Gateway name to open the Session Status popup.



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 ID. The Visit Gateway button is a shortcut to the Gateway Homepage.

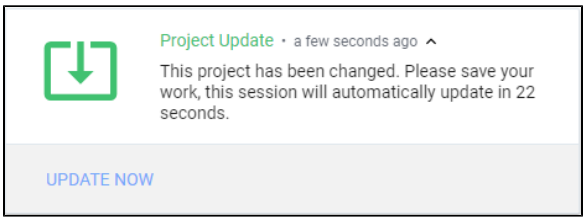


Under the Project tab, you see the name of the project, the last modified date for the project, and whether the project is up to date.



Project Updates

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

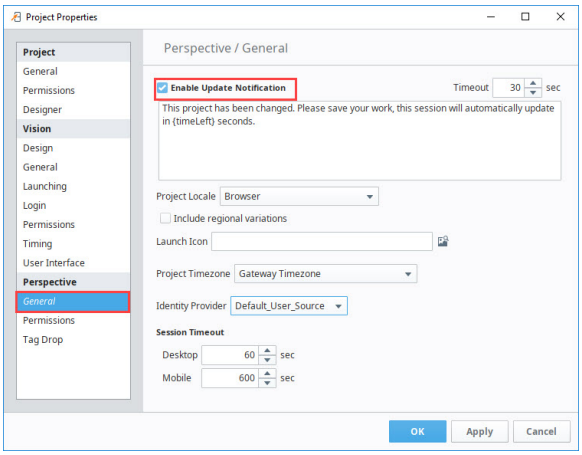


INDUCTIVE UNIVERSITY

Session Update Notifications

Watch the Video

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.

The following feature is new in Ignition version **8.0.3**
Click here to check out the other new features

Browser Caching

Perspective Sessions cache assets (our CSS, JS, fonts, and images) to reduce load times after the first time opening the project. Sessions will cache assets, but some browsers will attempt to re-download assets on refresh. This is mostly controlled by the browser instead of Ignition so there are some important differences between browsers.

Chrome and Safari handle browser cache differently than Edge and Firefox when it comes to hitting the ‘refresh’ or ‘reload’ button. Edge and Firefox both seem to send a request for new copies of the resources with header of max-age=0 instead of pulling from the cache. Navigating via a 'back' button or entering in the URL bar and pressing ‘enter’ will load from cache. This seems to simply be a design difference by these two browsers, in which both are working as intended.

Cache-Control

Caching can be disabled via a flag in the ignition.conf file by using a jvm parameter:

```
Dperspective.gateway.disableClientCaching=true
```

Users can also control what gets cached via the `Cache-Control` flag. Response header's `Cache-Control` value defaults to:

```
public, max-age=2419400, no-transform
```

You can set your own cache-control header via another env var:

```
-Dperspective.gateway.cacheControl=<some alternative cache control settings>
```

You can learn more about the Cache-Control flag through online resources like the MDN Web Docs.

Browser Version Requirements

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

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

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

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

Basic Session Functionality Requirements for Ignition 8.0.0	
Browser	**Minimum Version Required**
Chrome	57
Firefox	52
Safari	11
Edge	16

Related Topics ...

- Pages in Perspective

In This Section ...

Ignition Perspective App

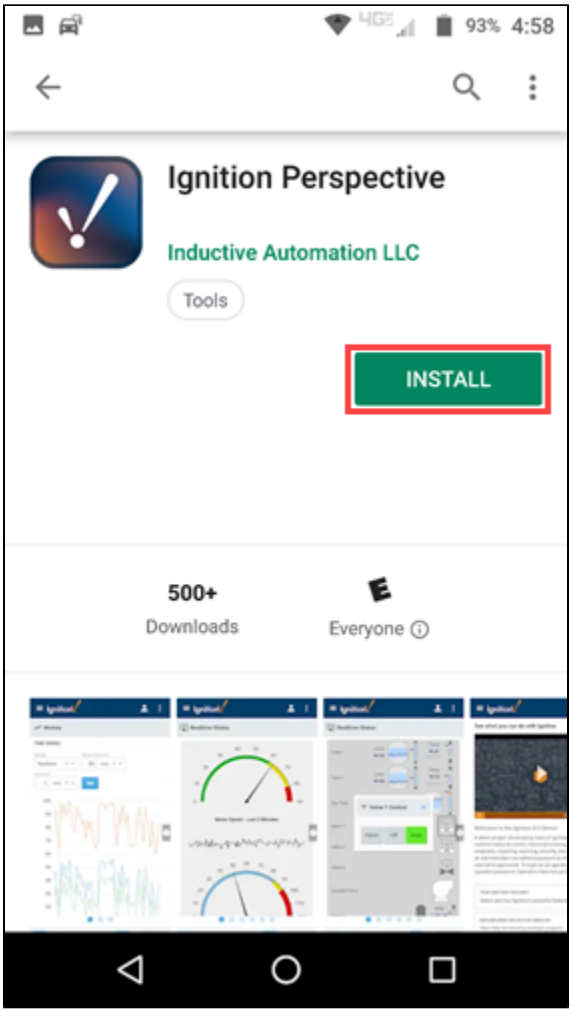
Overview

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

Download the App

You can download the Ignition Perspective App from a local online app stores, 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.

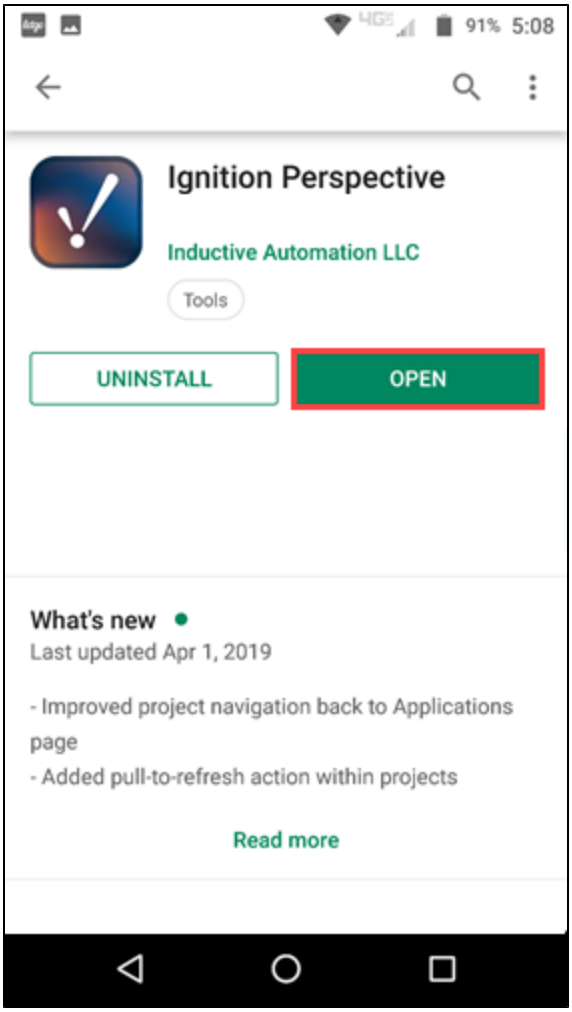


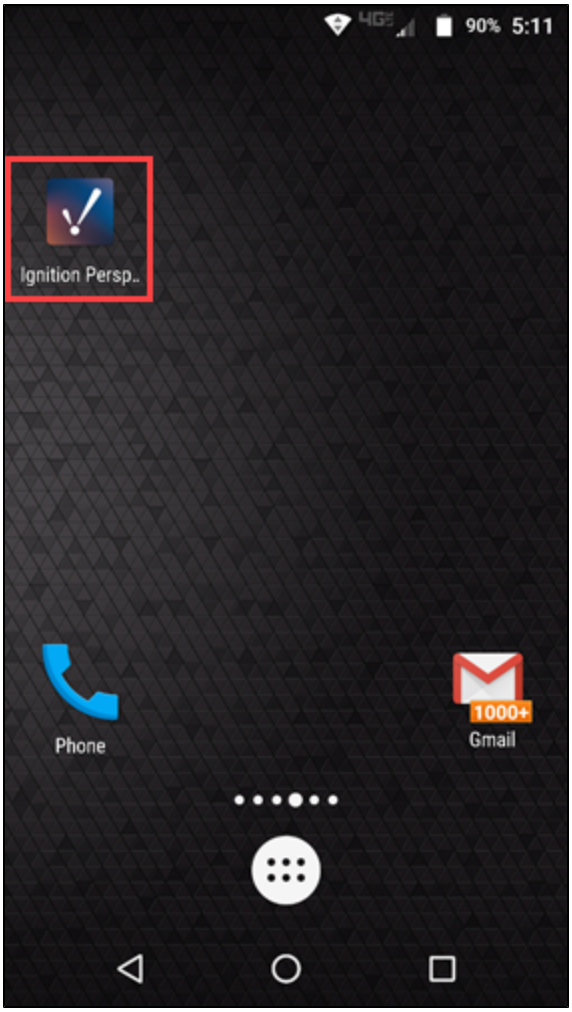
3. Once installation is complete, click the **Open** button, or click on the **Perspective** icon on your device's home screen.

On this page

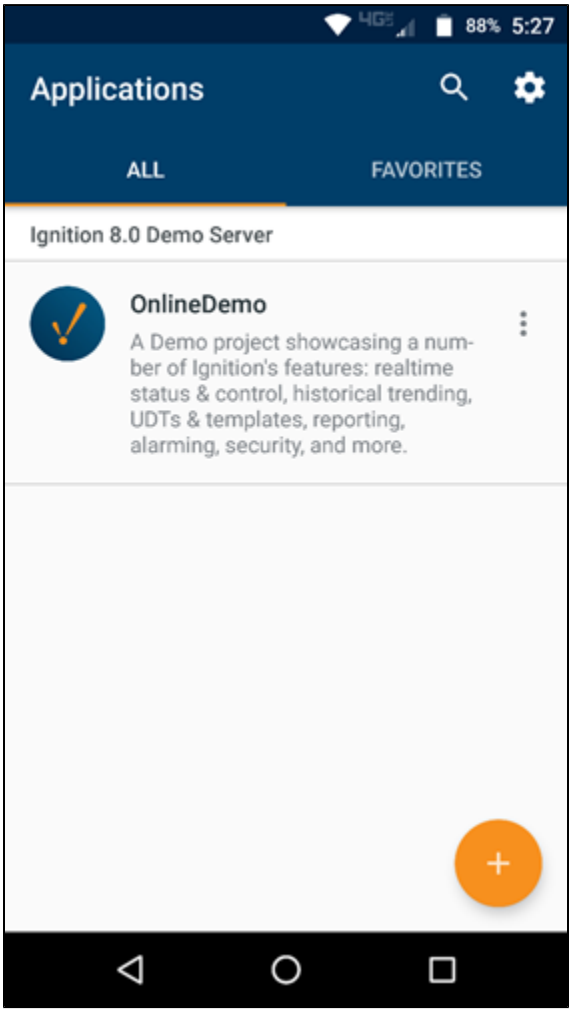
...

- Overview
- Download the App
- Launch a Perspective Session
  - Launch the Online Demo
  - Launch a Project
  - Switch to a Different Application
- Additional Options
- Managed Configurations
  - Sample PLIST Template


91% 5:08
Ignition Perspective
Inductive Automation LLC
Tools
UNINSTALL
OPEN
What's new
Last updated Apr 1, 2019
- Improved project navigation back to Applications page
- Added pull-to-refresh action within projects
Read more


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

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

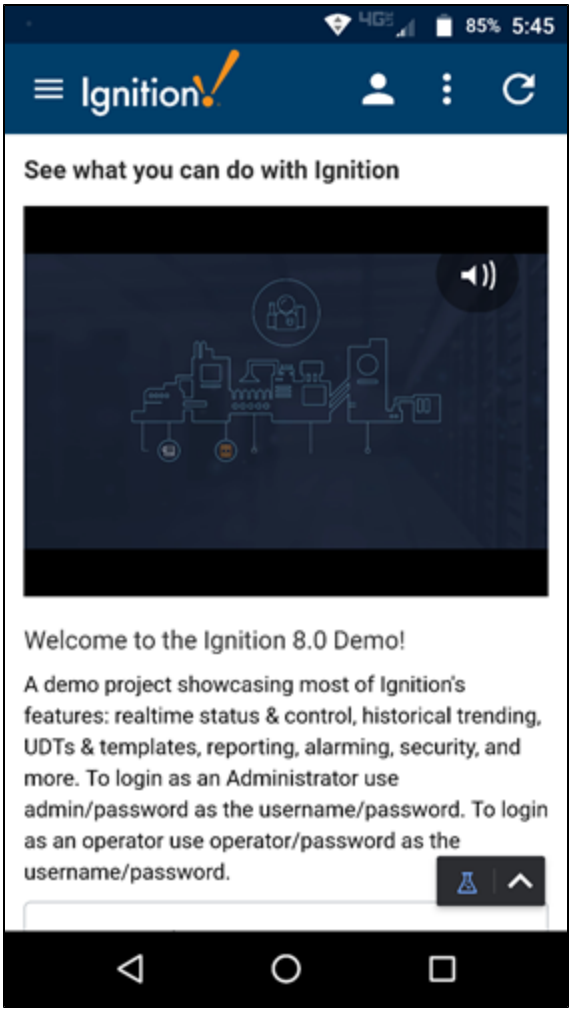


Launch a Perspective Session

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

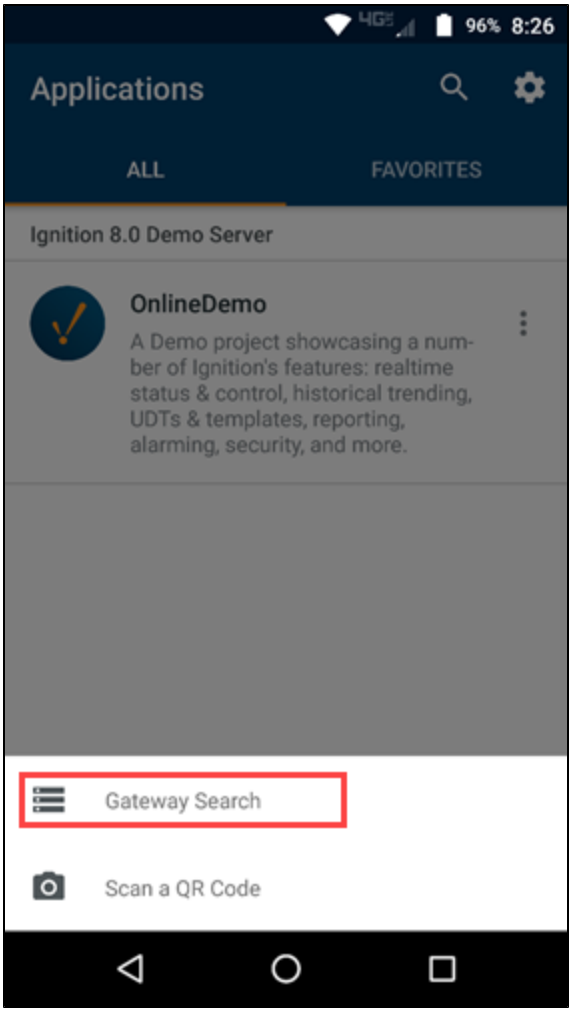
Launch the Online Demo

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



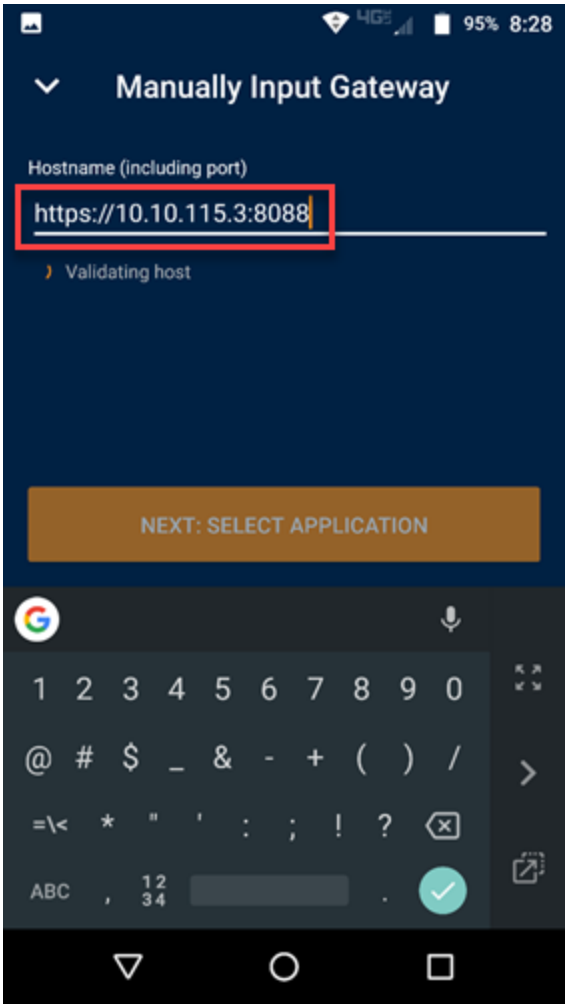
Launch a Project

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**.

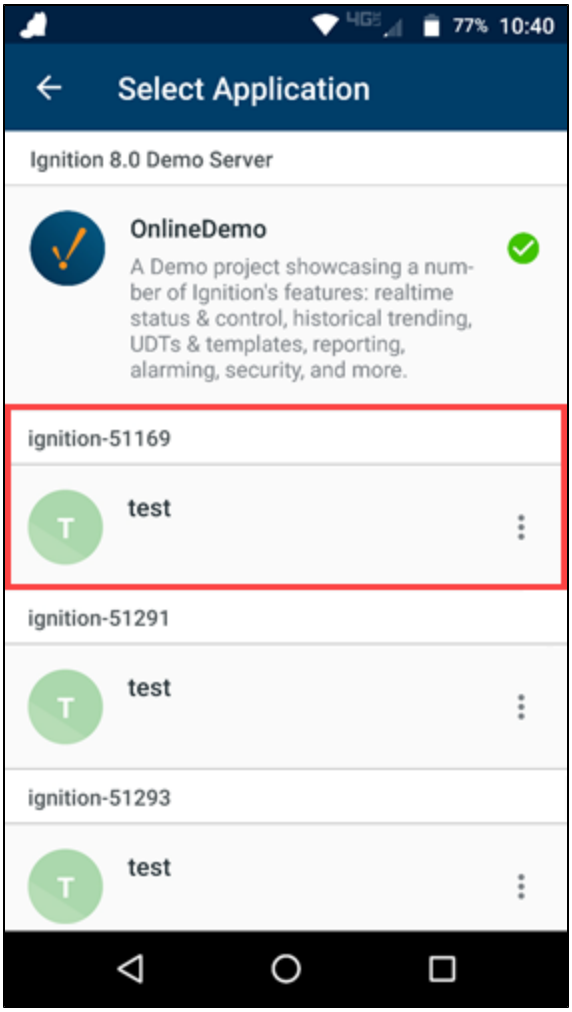


4. Select the Gateway you want to connect to.

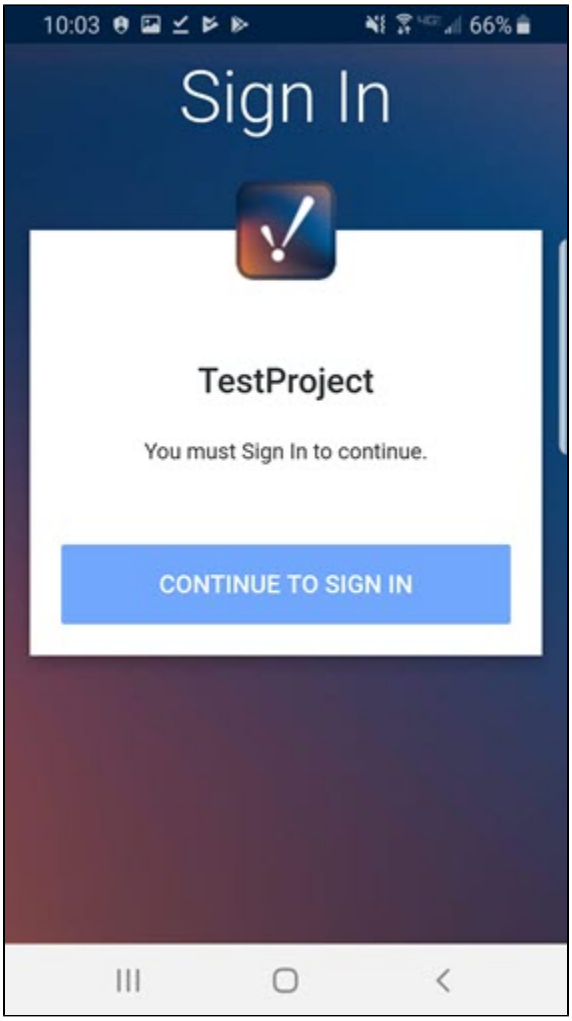
a. If the Perspective App doesn't find the Gateway automatically, you can manually input a Gateway host name and port. Click on **Manually Input Gateway**.
b. Type in the host name including the port, for example: https://10.10.115.3.8088.



5. Click on the project you want to open.


77% 10:40
Select Application
Ignition 8.0 Demo Server
OnlineDemo
A Demo project showcasing a number of Ignition's features: realtime status & control, historical trending, UDTs & templates, reporting, alarming, security, and more.
ignition-51169
T
test
ignition-51291
T
test
ignition-51293
T
test

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

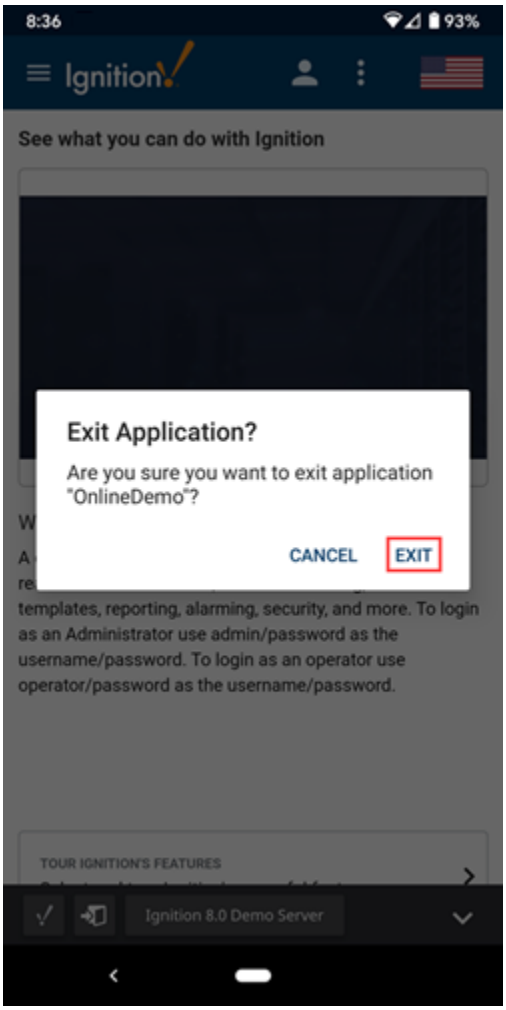


Switch to a Different Application

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 (down arrow in the lower right corner.
2. Click the **Exit** icon.

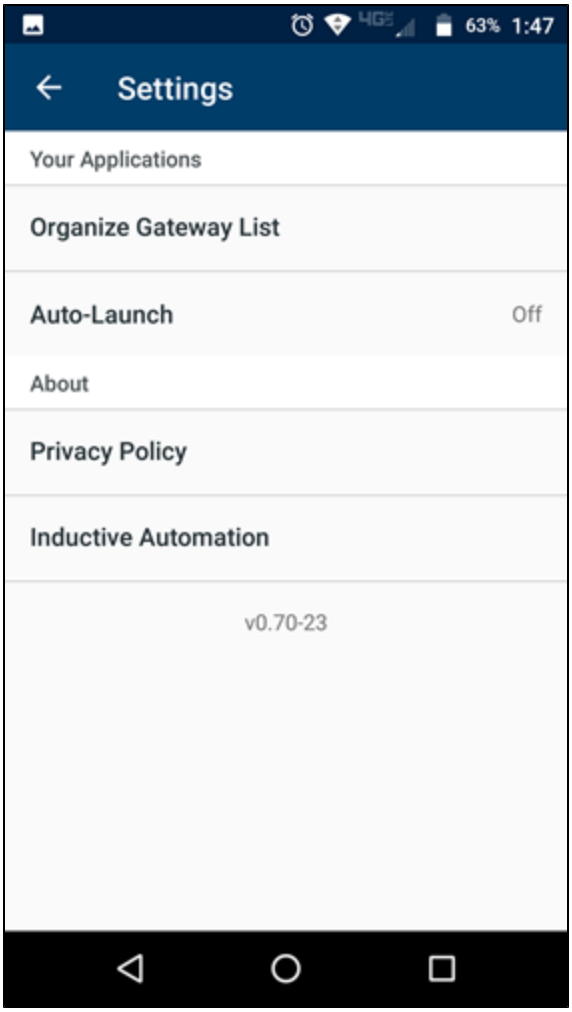
3. At the confirmation prompt, click **Exit**.



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 Perspective app. Here you can organize your list of available Gateways. You can also turn the Auto-Launch option on and select a project that will be automatically launched when the Perspective App is opened on this device.



Managed Configurations

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

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

For iOS devices, you will need to provide the EMM software with a sample PLIST. See Sample PLIST Template further down.

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. Contains the following keys: **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 project specified in the URL will automatically launch upon startup of the app.

<table>
<tr><td></td><td></td></tr>
<tr><td>hide_demo</td><td>If true, the built-in demo project will become hidden.</td></tr>
<tr><td>initial_applications</td><td>Allows you to add projects to the main app page. Each application has the following keys:
<table>
<tr><th>Key</th><th>Description</th></tr>
<tr><td>project_url</td><td>A URL leading directly to the application, for example:
<pre>http://www.someurl.com:8088/data/perspective/client/myproj

OR

perspective://www.someurl.com:8088/myproj</pre></td></tr>
<tr><td>alias</td><td>The name for the application, as it will appear in the project list.</td></tr>
<tr><td>is_favorite</td><td>Sets this project as a favorite.</td></tr>
</table></td></tr>
<tr><td>initial_gateways</td><td>Allows you to add Gateways to the Recent Gateways list. Each gateway has the following key:
<table>
<tr><th>Key</th><th>Description</th></tr>
<tr><td>gateway_url</td><td>A URL to the Gateway you'd like to add to the list, for example:
<pre>http://www.someurl.com:8088</pre></td></tr>
</table></td></tr>
</table>

Sample PLIST Template

Below is a sample PLIST template that can be used when applying managed configurations for iOS.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
    <dict>
        <!-- Launches the project immediately upon starting the app -->
        <key>auto_launch</key>
        <dict>
            <!-- URL of project to Auto Launch -->
            <key>auto_launch_url</key>
            <string>https://demo.ia.io:443/data/perspective/client/OnlineDemo</string>
            <!-- Prevents user from changing auto launch setting -->
            <key>auto_launch_locked</key>
            <true/>
            <!-- Prevent user from leaving auto launch project to use rest of app -->
            <key>prevent_exit</key>
            <false/>
        </dict>
        <key>hide_demo</key>
        <true/>
        <!-- Applications that will be available to user on start of app -->
        <key>initial_applications</key>
        <array>
            <dict>
                <!-- URL of project -->
                <key>project_url</key>
                <string>https://demo.ia.io:443/data/perspective/client/OnlineDemo</string>
                <!-- Add the project to favorites -->
                <key>is_favorite</key>
                <true/>
                <!-- Provide an alternative name user will see when viewing project, in this
                case project name will be 'Perspective Demo' instead of 'OnlineDemo' -->
                <key>alias</key>
                <string>Perspective Demo</string>
            </dict>
        </array>
```

```
        <!-- Gateways that will be available to user on start of app. User's still need to
         add individual projects from each gateway. Gateways from projects included in
         initial_applications will already be included and don't need to specified again here. -->
        <key>initial_gateways</key>
        <array>
            <dict>
                <!-- URL of gateway -->
                <key>gateway_url</key>
                <string>https://demo.ia.io:443</string>
            </dict>
        </array>
    </dict>
</plist>
```

Session Properties

Session Props

Session Properties, as the name implies, 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 which can be used in bindings and scripts, and are also important for passing parameters from one view to another.

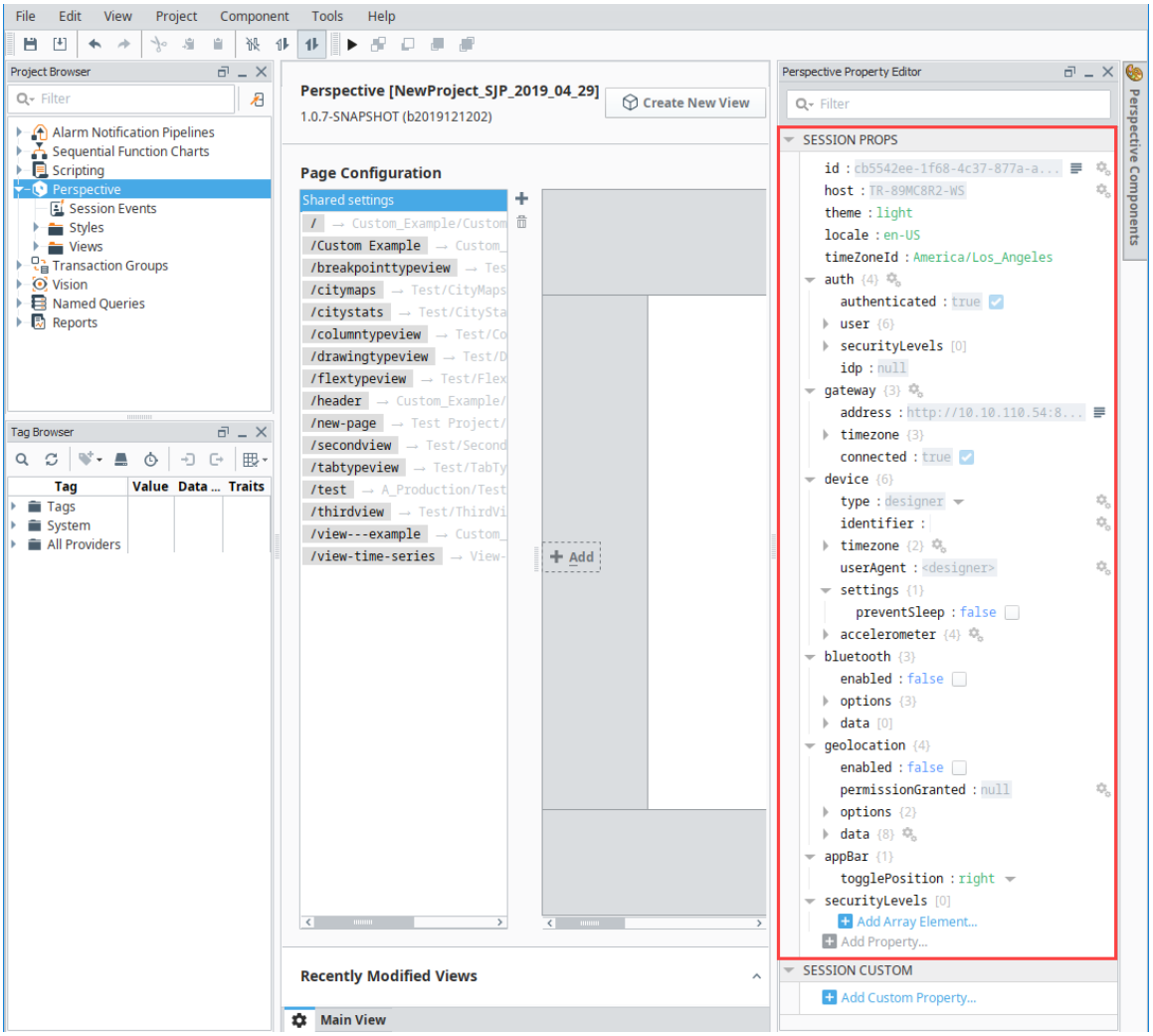
System Properties

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 as you can see in the image below.

On this page

...

- Session Props
  - System Properties
- Session Properties Table



Session Properties Table

Name	Description
id	Unique session identifier.
host	The following feature is new in Ignition version **8.0.1** Click here to check out the other new features Reflects the connecting system's IP address or hostname.
theme	The theme to use in the session. The default theme is `light`. Writing a theme name to this property will change the theme for the ses
locale	The current locale of this session.
timeZoneId	Timezone identification code, for example America/Los_Angeles.
auth	Represents the user's authentication and authorization for this session.

Name	Description

<table>
<tr><td>authenticated</td><td>True if the user is authenticated. False if the user is unauthenticated. Null if the user's authentication status is unknown.</td></tr>
<tr><td>user</td><td>Contains information about the user, if they are authenticated.
<table>
<tr><th>Name</th><th>Description</th><th>Prope
Type</th></tr>
<tr><td>id</td><td>The IdP's unique identifier for this user. Null if the user is not authenticated.</td><td>value:
string</td></tr>
<tr><td>user
Name</td><td>The user's username. Null if the user is not authenticated.</td><td>value:
string</td></tr>
<tr><td>firstN
ame</td><td>The user's first name. Null if the user is not authenticated, if the IdP did not provide this attribute, or if no mapping was configured for this attribute.</td><td>value:
string</td></tr>
<tr><td>lastN
ame</td><td>The user's last name. Null if the user is not authenticated, if the IdP did not provide this attribute, or if no mapping was configured for this attribute.</td><td>value:
string</td></tr>
<tr><td>email</td><td>The user's email address. Null if the user is not authenticated, if the IdP did not provide this attribute, or if no mapping was configured for this attribute.</td><td>value:
string</td></tr>
<tr><td>roles</td><td>The roles that the IdP assigned this user. Null if the user is not authenticated, if the IdP did not provide this attribute, or if no mapping was configured for this attribute.</td><td>value:
string</td></tr>
<tr><td>times
tamp</td><td>The following feature is new in Ignition version 8.0.14
Click here to check out the other new features

A timestamp representing the last time the current user authenticated against the Identity Provider.</td><td>value:
timesta</td></tr>
</table></td></tr>
<tr><td>securityLevels</td><td>The deepest security levels in the tree granted to the current user, starting with the children of the Public security level. The Public security level is never shown since all sessions include Public.
<table>
<tr><th>Name</th><th>Description</th><th>Property Type</th></tr>
<tr><td>name</td><td>The name for this security level. Must be unique among its siblings.</td><td>value: string</td></tr>
<tr><td>children</td><td>Security levels which descend from this security level.</td><td>array</td></tr>
</table></td></tr>
<tr><td>idpId</td><td>This feature was removed from Ignition in version

8.0.6

The identity provider's ID. Replaced by the <code>idp</code> property</td></tr>
<tr><td>idp</td><td>The name of the Identity Provider configuration set on the project.</td></tr>
<tr><td>idpAttributes</td><td>The following feature is new in Ignition version 8.0.16
Click here to check out the other new features

Represents the JSON object returned by the identity provider after logging in. The structure of this object will match that of th JSON provided by the Test Login Identity Provider page.

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.</td></tr>
</table>

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 b

Name	Description
address	Remote host address of the connected Gateway.
timezone	Document providing time zone information.

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

connected	The following feature is new in Ignition version **8.0.2** Click here to check out the other new features 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.

device

Properties for the device that is running the session.

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

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

userAgent	User agent string of the connected device.
settings	The following feature is new in Ignition version **8.0.7** Click here to check out the other new features Array of settings for the device.

Name	Description
preventSleep	Prevents the device from sleeping while viewing project in the mobile Perspective App. Default is false.

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

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

z	Acceleration force (in m/s^2) along the z axis (including gravity).	value: numeric

bluetooth

> The following feature is new in Ignition version **8.0.5**
> Click here to check out the other new features

Options and data provided by device Bluetooth services.

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

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

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

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

eddystone	Eddystone open beacon format.	object

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

iBeacon	iBeacon format.	object

Name	Description	Property Type
exclusive	Exclude other beacon types.	value: boolean
uuid	16 byte proximity uuid of iBeacon. On iOS this must	value:

		be specified in order to receive iBeacon data.	string

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

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

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

options

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

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

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

Name	Description	Property Type
latitude	A floating point value representing the position's latitude in decimal degrees. Null if location is disabled.	value: float
longitude	A floating point value representing the position's longitude in decimal degrees. Null if location is disabled.	value: float
altitude	A double representing the position's altitude in meters, relative to sea level. This value can be null if the implementation cannot provide the data.	value: double
accuracy	A double representing the accuracy of the latitude and longitude properties, expressed in meters.	value: double
altitudeAccuracy	A double representing the accuracy of the altitude expressed in meters. May be null if device fails to provide or if geolocation is disabled.	value: double
heading	Returns a double representing the direction in which the device is traveling. This value, specified in degrees, indicates how far off the device is from heading true north. 0 degrees represents true north, and the direction is determined clockwise (which means that east is 90 degrees and west is 270 degrees). If speed is 0, heading is NaN. If the device is unable to provide heading information, this value is null.	value: double

speed	Returns a double representing the velocity of the device in meters per second. This value can be null.	value: double
timestamp	Time the last location update was received.	value: string

appBar

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

> The following feature is new in Ignition version **8.0.5**
> Click here to check out the other new features

Name	Description	Property Type
togglePosition	The position of the overlaid toggle button that shows the app bar: **right**, **left** or **hidden**.	value: string

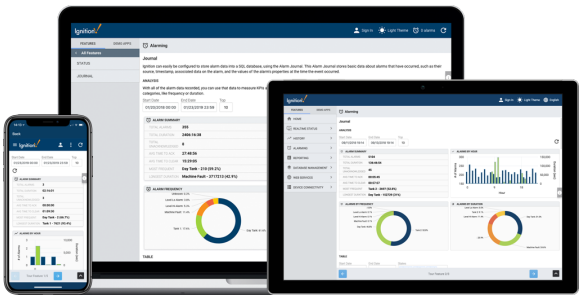
Related Topics ...

- Perspective Component Properties

Perspective Design Principles

Overview

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



On this page

...

- Overview
- Designing for Mobile-Responsive Applications
- Design Principles
  - Touch and Ergonomics
  - Mobile-First Design Approach
  - Fluid Content
  - Content as UI
  - Declutter
- Understanding Views and Containers

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

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

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

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

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

Designing for Mobile-Responsive Applications

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

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

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

Design Principles

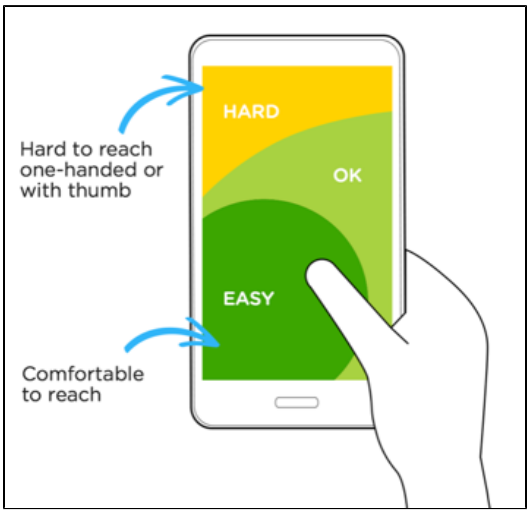
We all know what good design looks like in the desktop world. Now let's expand our knowledge to mobile devices using the following design principles.

Touch and Ergonomics

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

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

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



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

Mobile-First Design Approach

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

Fluid Content

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

Content as UI

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

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

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

Declutter

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

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

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

Understanding Views and Containers

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

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

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

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

Related Topics ...

- Views and Containers in Perspective
- Coordinate Containers
- Column Containers
- Tab Containers
- Breakpoint Containers
- Flex Containers

In This Section ...

Navigation Strategies in Perspective

In addition to traditional navigation, Perspective offers a variety of new approaches, including many that might look familiar to smartphone users. In designing Perspective, we incorporated a few common trends in UI design, particularly those favored in responsive design. The strategies below should help phone-savvy users navigate easily in a Perspective project.

There are many ways to set up your navigation, and the best looking projects will mix several types together. In this section we'll walk through a few examples:

- Simple Buttons
- Drill Down in Views
- Navigation Drawers
- Single-Page Projects
- Back Buttons
- Navigating to External Websites

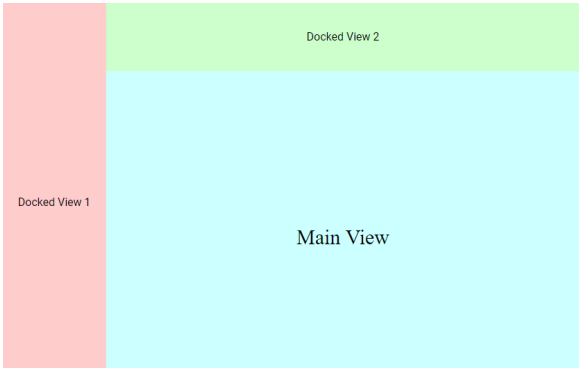
On this page

...

- Configuring a Docked View
- Navigation Actions
  - Other Resources
- Basic Page Navigation
  - Buttons and Clickables
  - Tab Container and Menu Component
  - Navigation Tree
- Drill-Down Navigation
  - Clicking on Components or Containers
- Same-Page Navigation
  - Side Scrolling / Carousel
  - Tab Containers
- Back Buttons
- Navigating to Other Websites
  - Links

Configuring a Docked View

In setting up navigation, most projects will call for one or more **docked views**. These are views that sit flush with one of the four edges of the page.

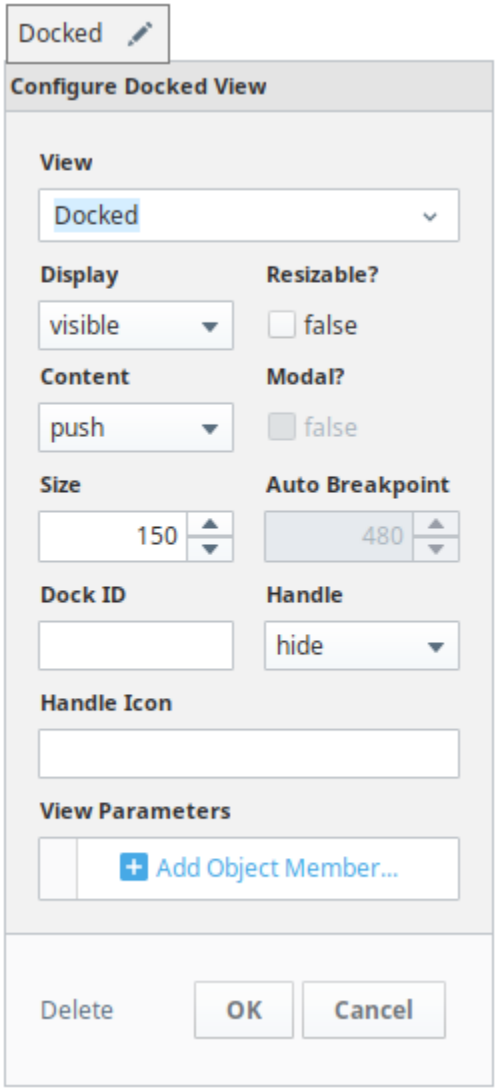


To dock a view you created:

1. Click the Settings icon in the Designer.
2. If you want your docked view to appear on all pages, select the **Shared settings** option in the menu, otherwise select the page you want the view to appear on.
3. Click one of the four **Add** icons, corresponding to where you want your view to be docked.

4. Select the path to your view in the **View** dropdown.
5. You may also want to configure other properties on this dialog. Click on the **Edit** icon next to the docked view name. You have many options, we'll just discuss a few:

a. Pay particular attention to the **Size** property, as this controls how far the view protrudes toward the center of your page.
b. If you want to show or hide the docked view from an event action or script, you'll need to specify a **Dock ID**. This can just be a keyword (like "menu") that you'll provide again later when you want to control this view.
c. **Handle Icon** can be used to provide a small icon for a user to show or hide your docked view at will.



Navigation Actions

Throughout Perspective development, you'll have the opportunity to set **Actions** in response to **Events**. Simply put, an event is something that happens on a component or session, and an action is how we respond to it. Many of the navigation options you'll have in Perspective depend on events and actions. Select the navigation action that works best for your needs:

- **Navigation** is the most self-evident. It allows us to open a new **Page** (either in our current browser tab or a new one), **View** (overriding our current *main* view), or **URL** (again, either in our browser tab or a new one).
- **Popup** allows us to open (or close) a popup view that hovers over our page's existing views. Configurable options include parameters to pass to the view, a title, where to open it and how big to make it, and some behavior options. The behavior options are straightforward apart from two:
    - **Modal** indicates whether the user should be allowed to interact with the page behind the popup. Essentially, should the user be able to move the popup to the side and continue working?
    - **Background dismissible** indicates whether the popup should be automatically closed if you click on the page area behind the popup.

- **Dock** allows you to open or close a docked view. Note that the view needs to be preconfigured as a docked view, complete with a **Dock Id** that the **Dock** action requires.
    - You can configure a **Dock Id** for a view by clicking the **Settings** icon in the Designer, adding the view using one of the four **Add** icons, then selecting the view to configure it.
- **Script** allows you to configure a script, from which you are free to call any of the built-in Perspective navigation functions. See system.perspective in the scripting section of the Appendix for details.

The Component Events and Actions section has more info about configuring events and actions.

Other Resources

The Common Perspective Tasks section provides a couple of examples specific to navigation:

- Creating Popup Views
- Horizontal Menu Component
- Self-Hiding Navigation Drawer

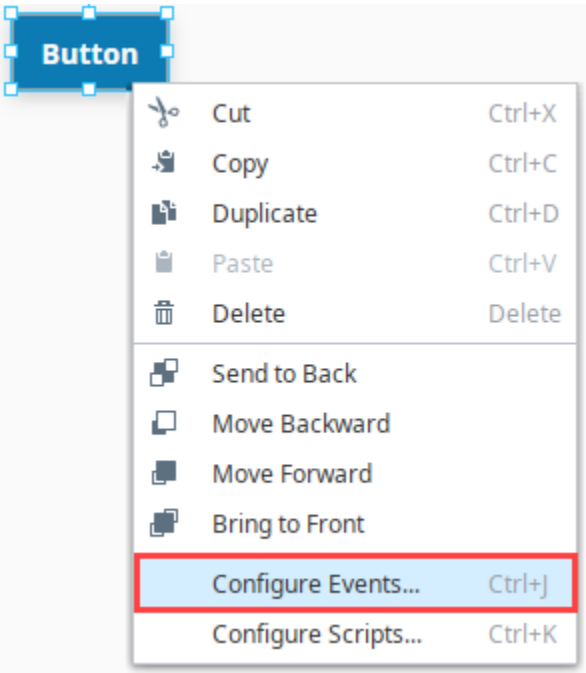
Basic Page Navigation

Buttons and Clickables

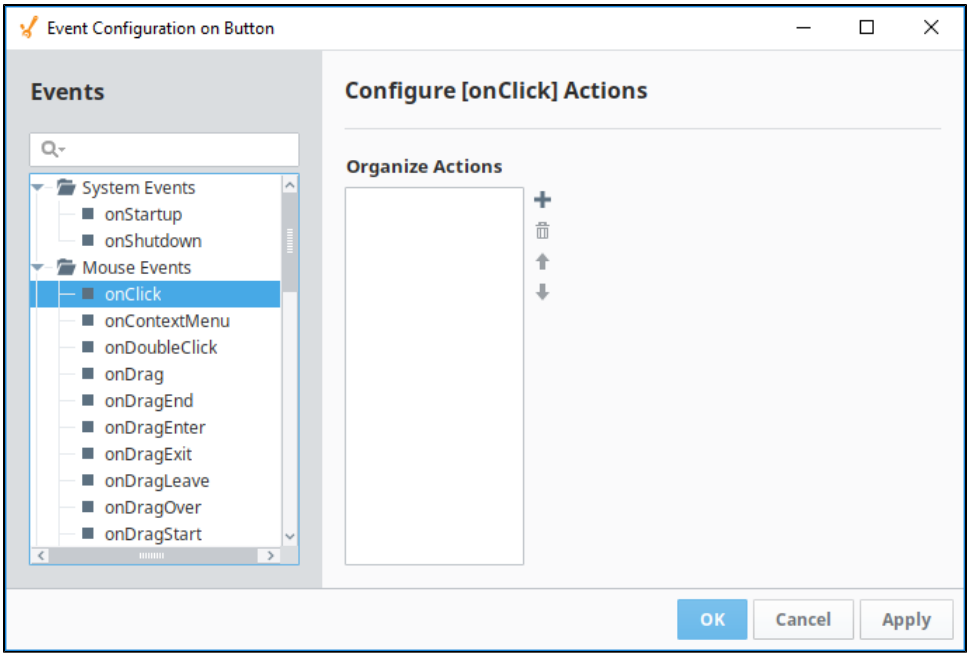
Before we get into fancier strategies, it's worth noting that we can use Button components (or any other components or containers you want users to click) to easily navigate from one page to another.You can use buttons on each main view, or create a docked view with buttons for each your main pages. This is one of the simplest ways to do navigation and will help you get a project started very quickly.

Let's walk through a quick example of how to make a button navigate when clicked:

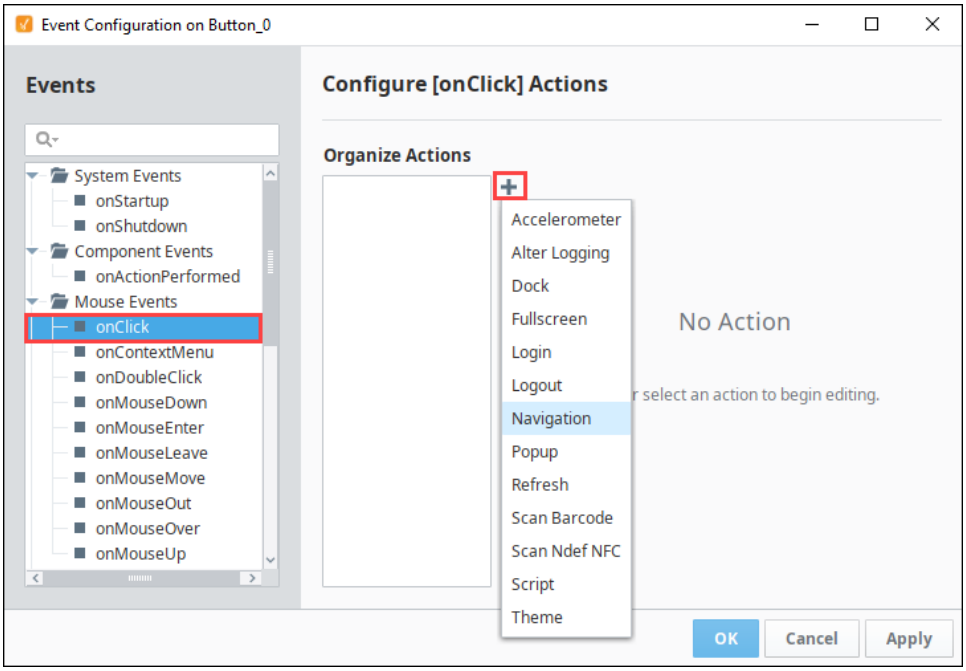
1. For a more consistent and hassle-free project, configure a **docked view** for your buttons. See **Configure a Docked View** above for a walkthrough.
2. Drag a Button component onto the container.
3. Right-click on the component and select **Configure Events...**.



4. Expand **Mouse Events**. As you can see, Perspective provides a multitude of ways to interact with your component. Select **onClick**.



5. Click the **Add** ✚ icon to add a new action to the event. Select an action you'd like to use for the navigation. See the Navigation Actions section above for details about each action.



6. Repeat Steps 2 through 5 for each new Button that you want.

Tab Container and Menu Component

A Tab Strip or Menu is an effective upgrade to the basic button navigation strategy, particularly when you want an indicator that shows which page is selected.

There's a caveat to this approach, however; the tab *container* offered in Perspective cycles between views, not pages. This means that, if you'd like to organize your project into pages (and you probably should), the tab layout doesn't quite do what we need in terms of tabbed top-level navigation. There are of course other options, from using a row of buttons to using a Flex Repeater with a "tab" view.

> The following feature is new in Ignition version **8.0.3**
> Click here to check out the other new features

The Horizontal Menu component enables you to build a menu structure by setting up multiple links to different page URLs from the component. An example is provided in Common Tasks in Perspective.

Navigation Tree

If you're looking for a pre-configured navigation option that is sleek and customizeable, the Menu Tree component allows you to create a menu that has expandable sections with buttons that users can click through. The Menu Tree works best inside a docked view (see **Configuring a Docked View** above), or, if you're feeling fancy, within a Navigation Drawer.

Only one property on a menu tree, the **items** property, controls the *structure* of the tree, all others control its look and feel. The **items** property can become fairly complex, but fundamentally each element in **items** has five sub-properties:

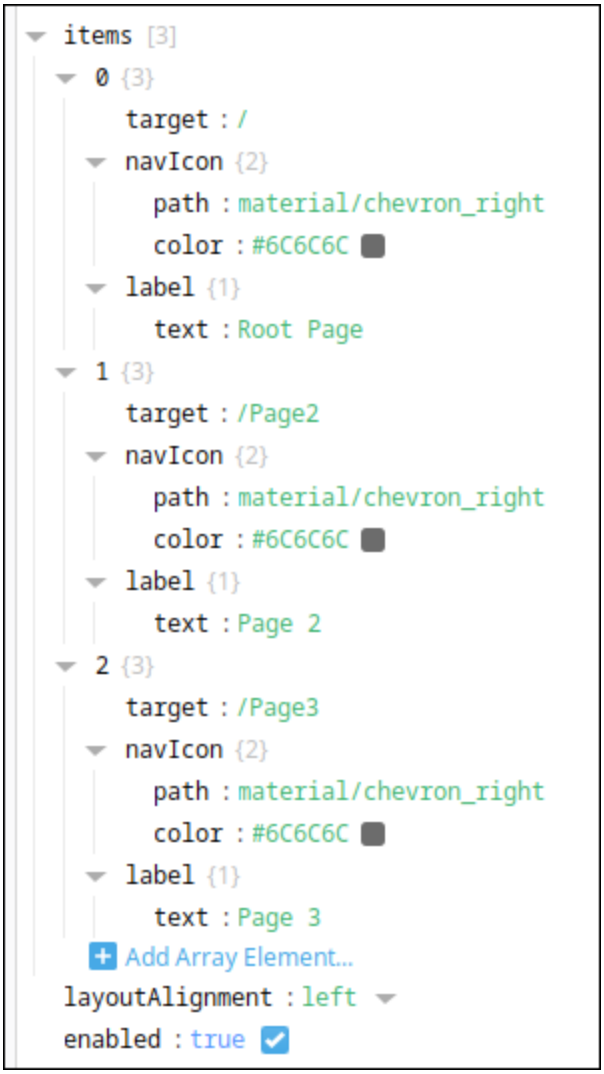
- **target** is where clicking on the item in the menu should take you. It can be a page URL, or an external (non-project) URL.
- **items** provides an opportunity to *show another menu* (instead of navigating) when this option in the menu is clicked. A *one-menu* configuration will ignore this sub-property.
- **navIcon** and **label** control the content of the menu item.
- **showHeader** controls what is at the top of a submenu when it is shown.

Simple Menu

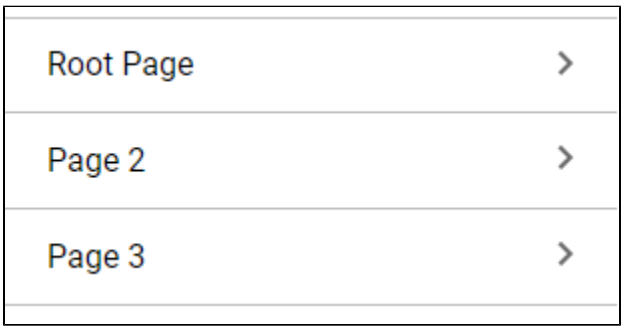
In a simple menu, you have a list of pages you'd like to navigate to, and you want them displayed in a column. The Menu Tree component can accommodate this without much fuss.

1. Drag the Menu Tree component into the view you'd like to use. Again, using a docked view is probably the best approach.
2. Click on the menu tree. We'll be walking through its properties in the Property Editor.
3. The **items** property on our menu tree will contain an entry for each page we'd like to navigate to. We won't walk through configuring those pages; here we'll just use the page URLs **/**, **/Page2**, and **/Page3**.
    a. The default menu tree has two entries, add a third one by right clicking on **0** or **1** (*not* **items**).
    b. Configure the **target** properties on the three entries to point to the appropriate page URLs.
    c. Configure the **label** property on each entry to show desired text.
        - You can customize the icon accompanying the label by changing the **icon** property (not to be confused with **navIcon**).
        - You can also delete the icon property if you'd prefer not to have one.
    d. The **navIcon** property can be customized or removed. It controls the right-most icon in the menu item.

    Here's how the final **items** structure should look. We deleted unnecessary properties for clarity.



Here's our new menu tree:



Editor notes are only visible to logged in users
Insert Bottom Navigation Bar section here - I have to shelve this one, because it isn't possible right now to make a docked view show up only on small sessions. FB 12876

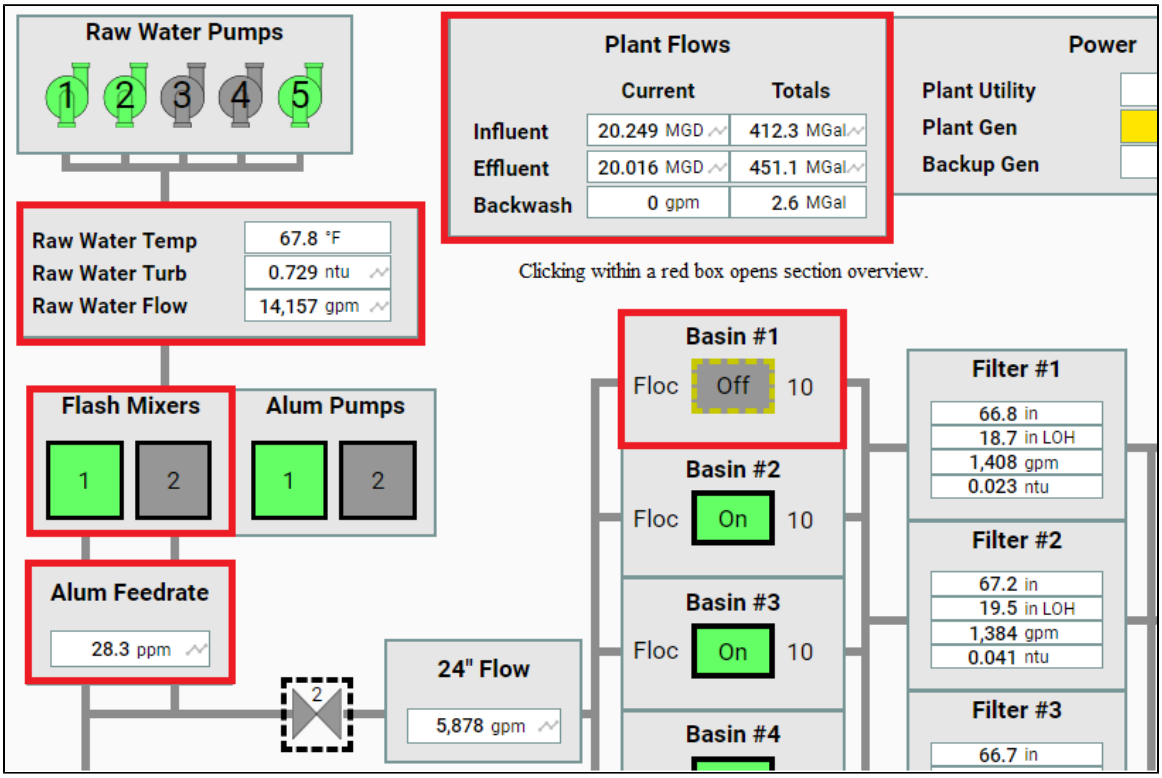
Drill-Down Navigation

"Drill down" (also known as *forward*) navigation strategies follow their namesake: they allow you to "drill" into a project page or view from an overview page. They also allow you to proceed to the *next* option in a chain. Most of the time, these strategies will feel quite natural: for

example, an operator clicks on a line or machine on an overview screen, or they have finished scanning a barcode and want to record the data using a "Save" button. You could use a button press to complete tasks like this, but any component or container can be configured to handle mouse (and touch) events. In fact, you may wish to create a custom view and/or style for this purpose, since the way that you navigate will be closely tied in with the needs of your project.

Clicking on Components or Containers

Very similar to adding navigation to a Button component, simple navigation can be added upon clicking (or touching) any component:



1. Drag in an instance of a desired component, view, or container.
2. Right-click on the component and select **Configure Events...**. Expand **Mouse Events**. Select **onClick**.
3. Click the **Add** ✚ icon to add a new action to the event. Select the navigation option that works best for your needs. See Navigation Actions above for details.
4. Add your script to navigate to the new page. You will often be passing a value into a re-usable view that uses indirection to show a specific tank, motor, etc.

Editor notes are only visible to logged in users
Once we have table extension functions we can give them a brief mention them here. IE: clicking on a row in the table to open a screen with details about that record.

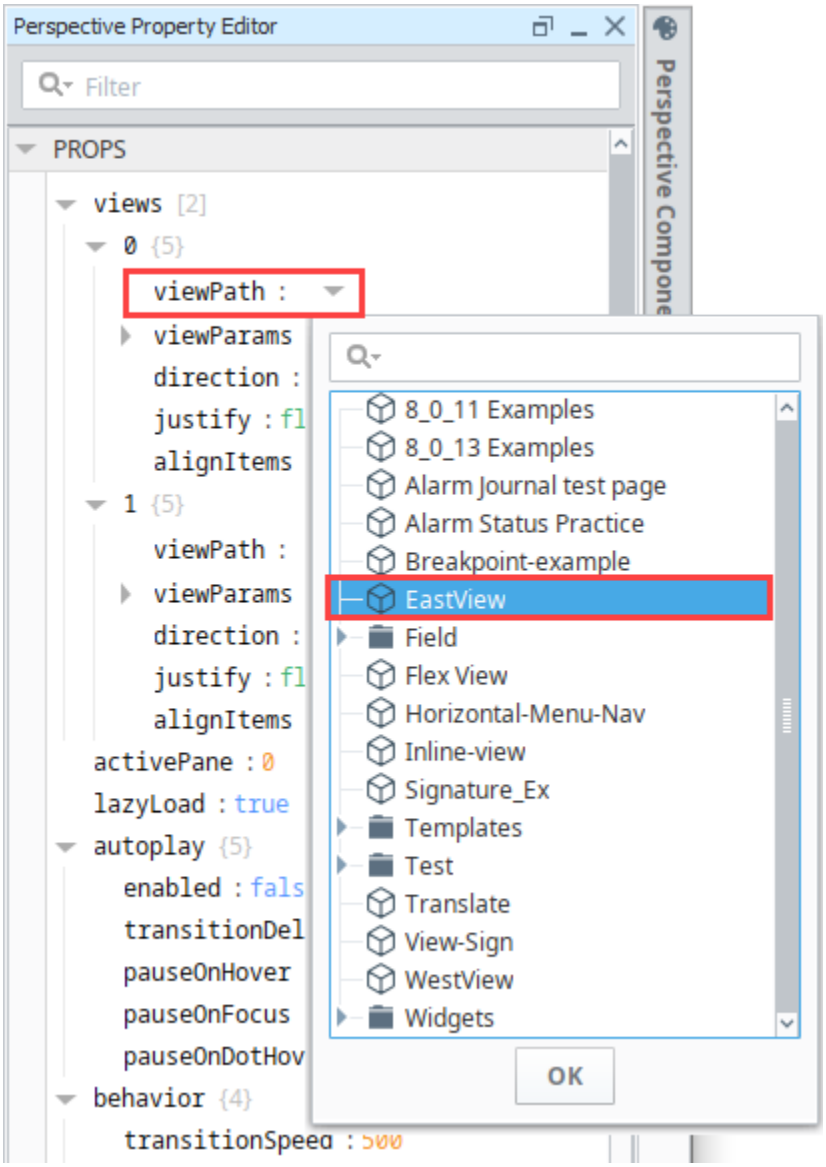
Same-Page Navigation

In this section we'll talk about a couple navigation strategies that explicitly *can't* take you to a new page of your project. They are good for navigating between views on the same page.

Side Scrolling / Carousel

Perspective offers strategies for dragging (or swiping) left and right as a way of navigating between views. The **Carousel** component is specifically for this strategy, which is perfect when working with several instances of the same view (including a dynamic number of them). Maybe your managers make a daily To Do list, and you'd like to scroll between them for different dates. Or you're looking for a way of collapsing a lot of content onto a small screen, and need a way of scrolling through it. Configuring side scrolling through a carousel is pretty straightforward:

1. Create the view you'd like to embed in a Carousel, and configure it as you'd like. It's probably a good idea to put something on the view that will distinguish it from other views on the Carousel (like a creation date, or distinct title).
2. Drag in a Carousel component, and position it.
3. Select the Carousel component. To add views to the carousel, click the **Add Array Element...** icon below the **views** property on the Carousel. Each created object has five properties, the most important of which are **viewPath** and **viewParams**. Select a view from the dropdown, and configure any necessary view parameters in **viewParams**.

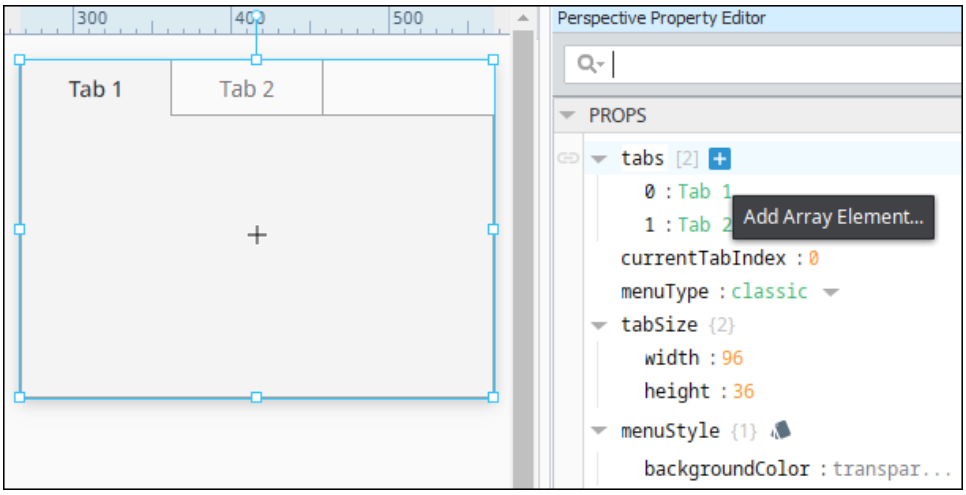


Tab Containers

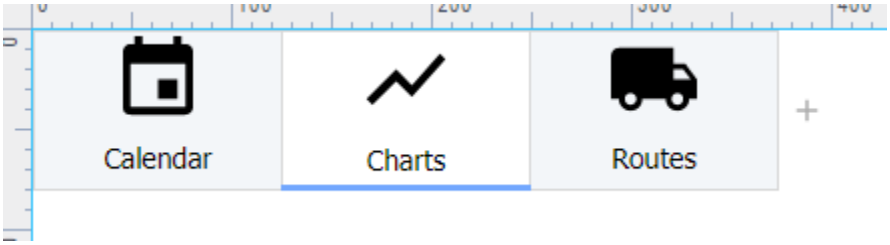
Tabs are an effective primary navigation strategy, particularly when you don't have many items to choose from. There's a caveat to this approach, however; the tab layout cycles between views, not pages. This means that, if you'd like to organize your project into pages (and you probably should), the tab layout doesn't quite do what we need in terms of tabbed top-level navigation.

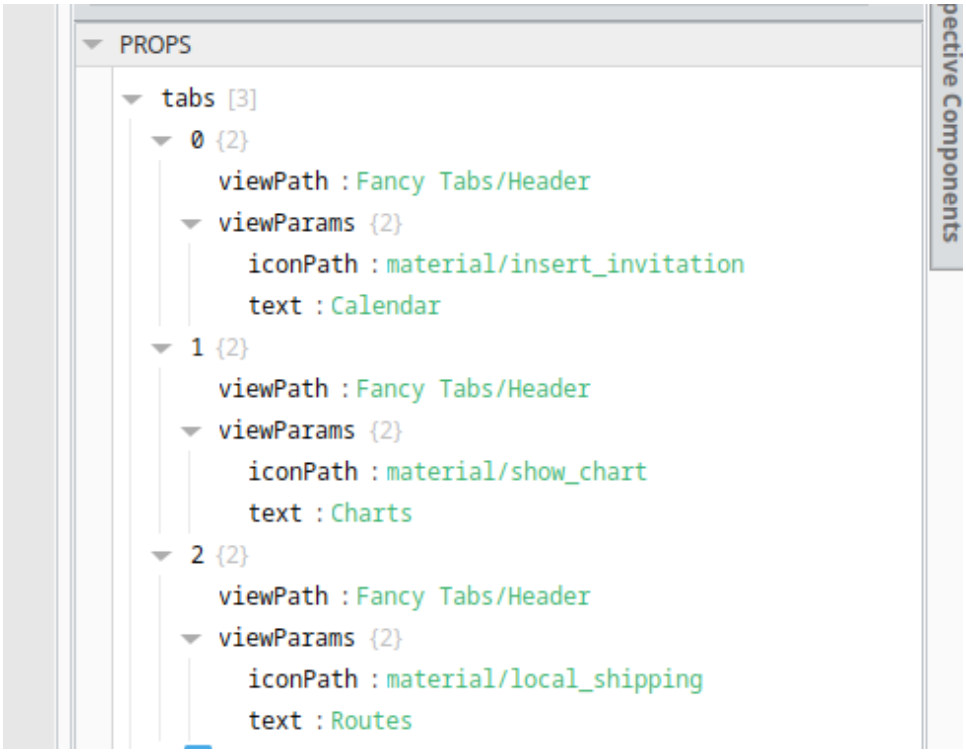
Tab Containers can be effective tools in designing complex single pages. In Perspective, they're easy to configure - you can use the Tab Container for simple drag-and-drop configuration; you don't even need to set up a docked view to control it. In these settings, the tab layout is perfect because it swaps between any views, containers, or components you provide. It's also easy to set up:

1. Drag a Tab Container into an existing view, and position it how you'd like.
2. Select the Tab Container. Add or remove tabs as needed by adjusting the **tabs** array. Note that if you're adding a tab, you'll want to click the **Add Array Element...** icon. From there, you have two options:
    a. Choose **Value** as the type. The string you enter will be displayed on the tab header.



b. If you feel like being extra fancy, you can nest a view of your choosing in the tab *header* (not to be confused with displaying a view in the tab itself). Choose **Object** as the type, then add a **Value** element to the new object. The element's key should be **viewPath**, and the value should be a path to a view to render in place of the typical tab header. You can include an additional **viewParams** object if parameters are needed. In the example below, we've added icons with the iconPath property.





3. Click on a Tab Header and drag in a view, container, or component to connect it to that tab.

Back Buttons

A Back button or reverse navigation generally refers to how a user might retrace their steps in an application, or move to a higher level page in the session. In Perspective, you will probably include "back" and "cancel" buttons and links in contexts where they seem appropriate (like navigating out of a view designed for a specific task). Since your project is being run in a browser, users will likely have access to the browser's back button, or a hardware back button on a phone. These buttons will typically navigate to the most recently visited *page* of your application. With this in mind, there are some good rules of thumb for developing browser-ready applications:

- Don't *depend* on the browser or phone's back buttons for project navigation. Many people aren't in the habit of using it.
- Don't assume your users *won't* use a back button. The back button is a valuable resource, and in most use cases, your project should gracefully handle navigation to any of its page URLs at any time.
- Break up your content into different pages when called for. If your entire project exists at the root page URL, an unfortunate use of the back button will leave a user outside their Perspective session, when maybe all they wanted was to return to a previous dialog.
- Don't make something a page if you don't want the back button to land you on it. For instance, if you have 50 PLCs and an operator needs to click through a page on each one, it may prove frustrating and disorienting to have to use the back button 50 times to navigate out of the list.

Navigating to Other Websites

Links

The **Link** component provides an easy navigation option when you want to invite users to view another network or internet resource from your project. The **target** property on the component dictates whether the page will open in the current tab or a new one. Tabs within the Horizontal Menu Component can also be set to navigate to a different website.

Related Topics ...

- Perspective Components

Pages in Perspective

Pages are the main method of navigation in Perspective. A 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). Each configured page specifies a primary view, which is the view which will fill the available space of the page. Other views may be configured to be displayed “docked” to the sides of the page, or “popups” floating on top of the other views. Each page has a URL associated with it, which is how the browser navigates to and displays that specific page.

On this page

...

- Page Layout
- Page Configuration
  - Corner Priority
  - Configure Docked View
  - Configure Docked View Parameters
  - Docked View Parameter Example
- Page URLs
  - Configuring a View to a Page URL
- Configuring Docked Views
  - Shared Views
  - Page-Specific Docked Views
- Passing Parameters (URL Parameters)

Creating Pages

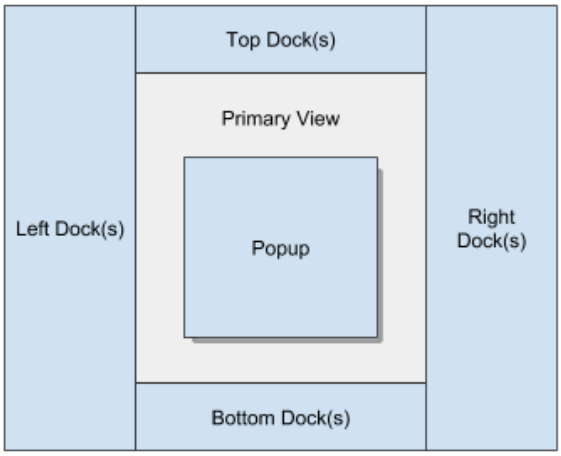
Watch the Video

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, on the other hand, can have multiple views on each side of the page or no views at all, but only one view may be shown at a time for a given dock position. In a session, docked views can appear permanently along the edges or the browser, as small handles that look like tabs on the edges of the browser, or the can auto auto-hide themselves.

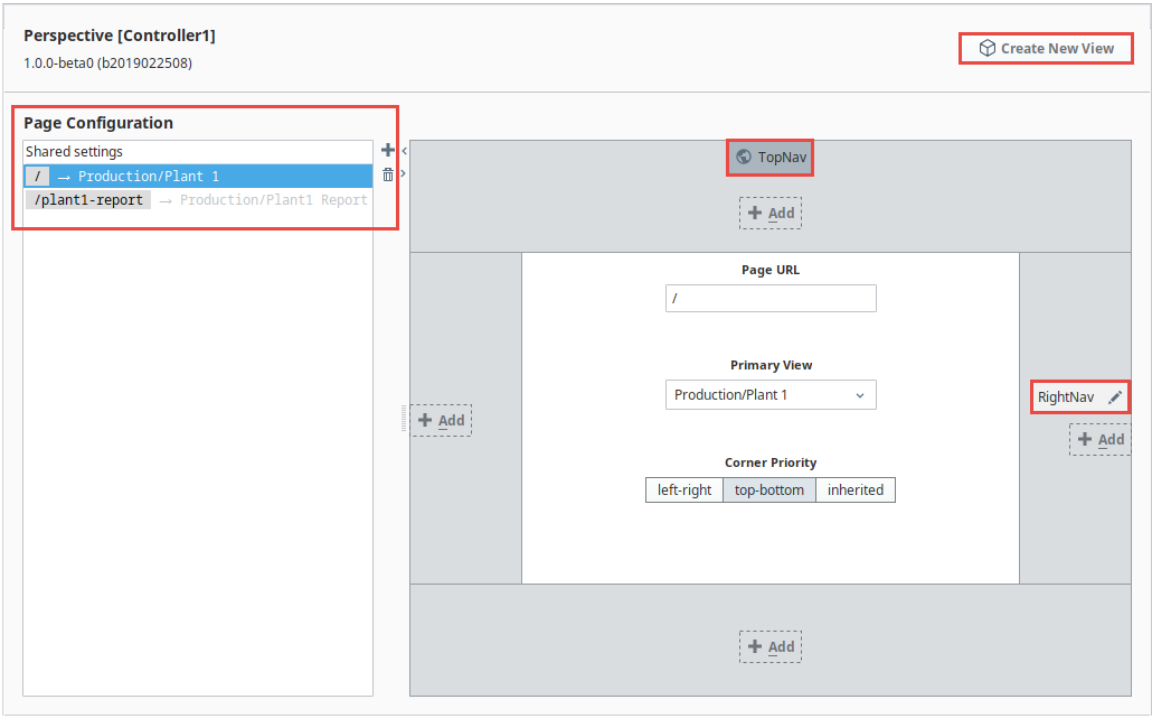


Page Configuration

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, such as docked views and corner priority settings. This is intended to provide an easy way to configure a project such as using a docked header across all pages in the project without having to configure a header on each page individually.

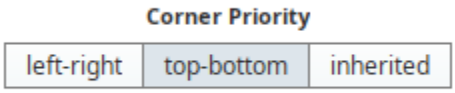
You can open the **Page Configuration** from any view in your project by clicking on the Settings icon at the bottom left of the Designer window. Here you can assign the Page URLs to your primary views and add docked views to your pages. You can even create a new view without going to the Project Browser.

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.



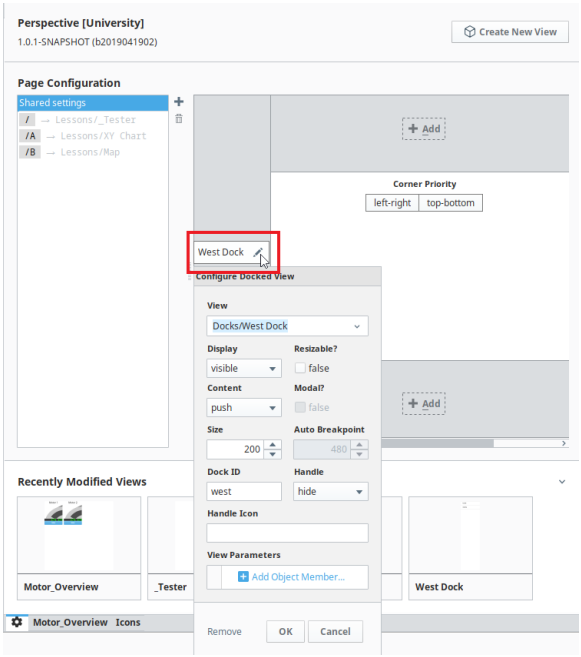
Corner Priority

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.



Configure 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 pencil icon next to the name (i.e., **RightNav**).



Configure 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. Options are: **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** – The following feature is new in Ignition version **8.0.6** Click here to check out the other new features 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.

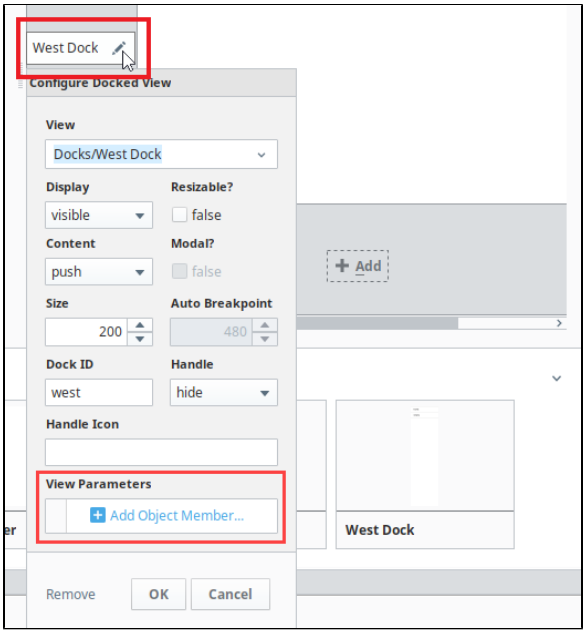
	<table><tr><th>Option</th><th>Description</th></tr><tr><td>fixed</td><td>The docked view will remain in a fixed position, relative to the page. Useful when a north-docked view should stay at the top of a page. Select this option if a docked view is acting as a header that should always be present.</td></tr><tr><td>scrollable</td><td>The docked view will not stay in a fixed position as the user scrolls down in the page. Select this option if the north dock should move along with the page as the user scroll down.</td></tr></table>
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. <table><tr><th>Option</th><th>Description</th></tr><tr><td>Show</td><td>Show handle at all times.</td></tr><tr><td>Hide</td><td>Hide handle at all times.</td></tr><tr><td>AutoHide</td><td>Hide handle when page is not active.</td></tr></table>
Handle Icon	Path to an icon used to identify the view when the view is hidden.
View Parameters	Allows specific parameters to be passed to the view when the page is called.
Remove	Deletes the view from the page.

Configure Docked View Parameters

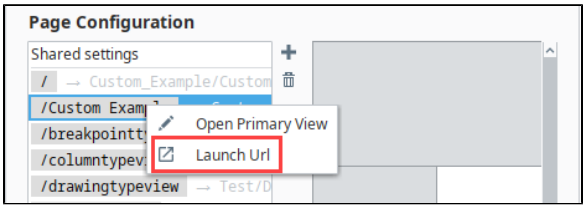
In Perspective, you can pass parameters to Docked Views. This basically enables you to specify one or more Params properties on the View that's docked, and pass a view to the docked view when the page is accessed. This is mostly used to modify content in a docked view based on the page it is being accessed from.

Docked View Parameter Example

1. Click on Perspective node in the Project Browser to view the Page configuration.
2. Click on the **Edit** icon next to the docked view name (in our example it is named West Dock). If your project does not contain any docked views, refer to the corresponding section on this page.
3. Under View Parameters, click Add Object Member.



4. You will see a popup allowing you to select the type of the property. Ideally, this should match the property type on the view. Additionally, the property should be located under the Params category.
5. Enter the name of the view parameter.
6. Enter a value, for example, "one." Then click OK.
7. Save the Project.
8. Under Page Configuration, right click on the Page and click on Launch Url .



9. When the page opens, any components bound to the view parameter will now use the value "one".

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 **"/"**. The exception to this being cases where the user manually enters a different Page's URL into the address bar of their browser.

The primary view on the Page Configuration screen denotes that it's the main view attached to the Page URL. Everything else surrounding the primary view are docks if they are set. If you have Shared settings selected, these settings are shared for all pages. If you have a particular page selected, these settings are page-specific. In the following section, you'll see how to configure a view to a Page URL and create docked views.

> The following feature is new in Ignition version **8.0.4**
> Click here to check out the other new features

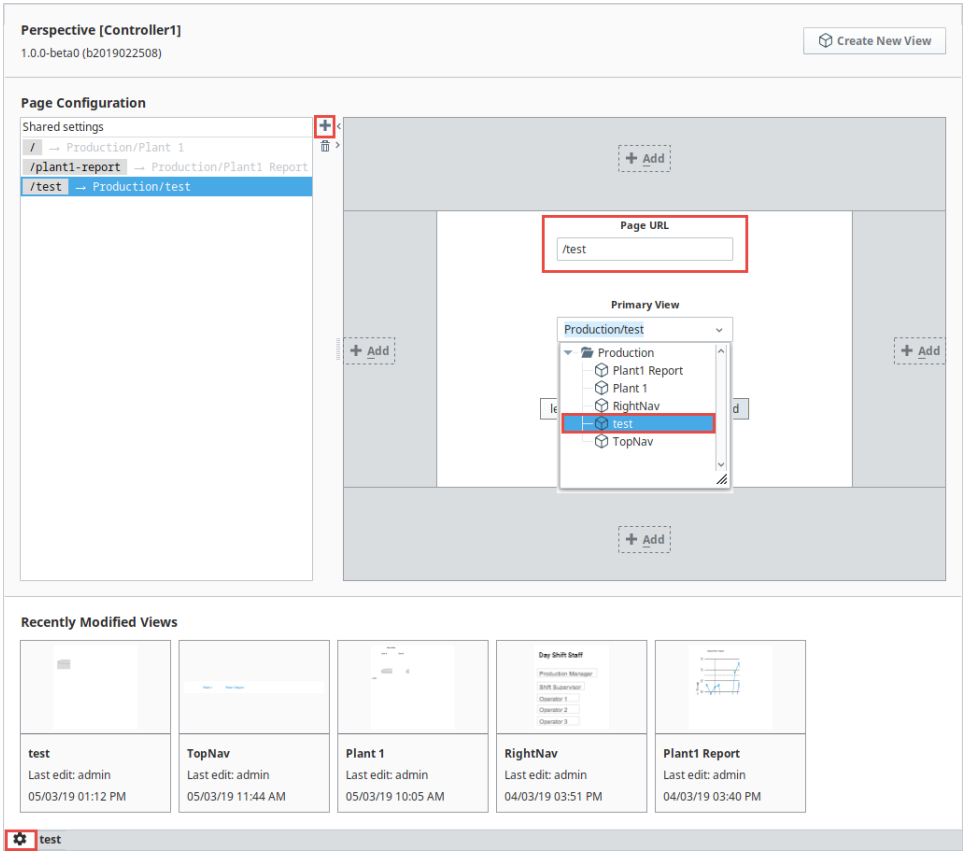
In version 8.0.4 a page.props.title property has been added. You can use the pages.props.title in a script to set a different title for the page. By default, the browser tab displays the project title (or the project name if the title is blank). You can also create bindings to the page title by selecting it as a Property Binding under the page properties on the Edit Binding screen.

Configuring a View to a Page URL

To configure a view to a Page URL, follow these simple steps. Note that this example requires that you have a view created.

> **Note**
>
> If you configured a Page URL at the time you created your view, you're done. The Page URL is already configured for your view.

1. Open the Page Configuration by clicking the **Settings** icon at the bottom of your Designer window.
2. Under the Page Configuration column, click the **Add** icon to add a new page.
3. In the Page URL field, enter the name of your Page URL (i.e., **test**).
4. Select the **Primary View** from the dropdown list (i.e., **Production/test**).



Configuring Docked Views

You have the option of making a docked view shared across all pages of your project or only on specific pages.

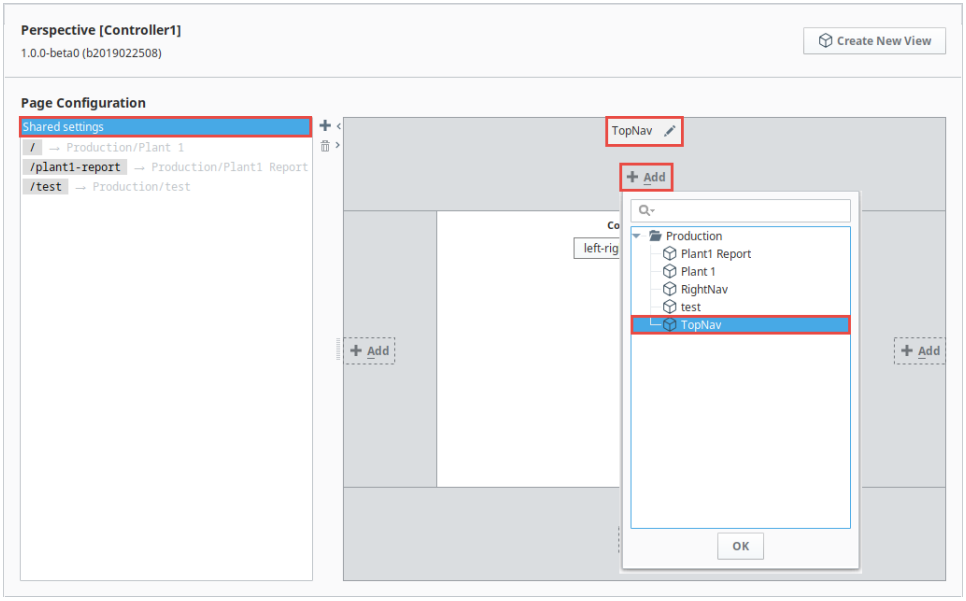
Shared Views

Let's configure a docked view under **Shared Settings** so it is shared across all pages of your project.

> This example requires that you have one or more primary views created with an assigned Page URL(s), and a docked view.

1. Open your Page Configuration if it's not already open.
2. Under the Page Configuration area, select **Shared Settings**.
3. Click the **Add** icon on the Top Dock. From the dropdown, select the view you want to be displayed on all pages of your project (i.e., **TopNav**).
4. Click **OK**.

5. If you want to customize your view, click on the TopNav pencil to open the **Configure Dock View** window.



Page-Specific Docked Views

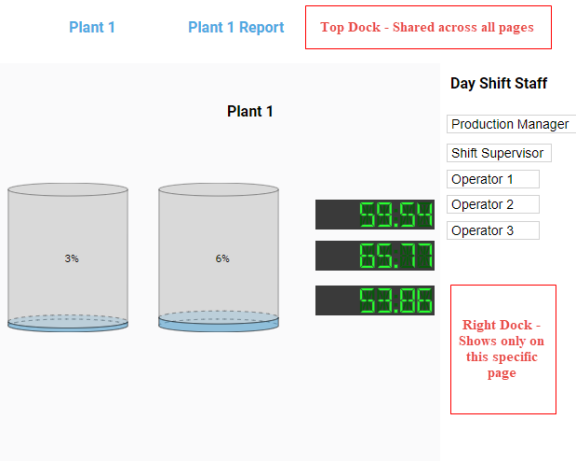
You can choose to dock a view on a specific page URL with a set primary view. Let's add a view to the right dock.

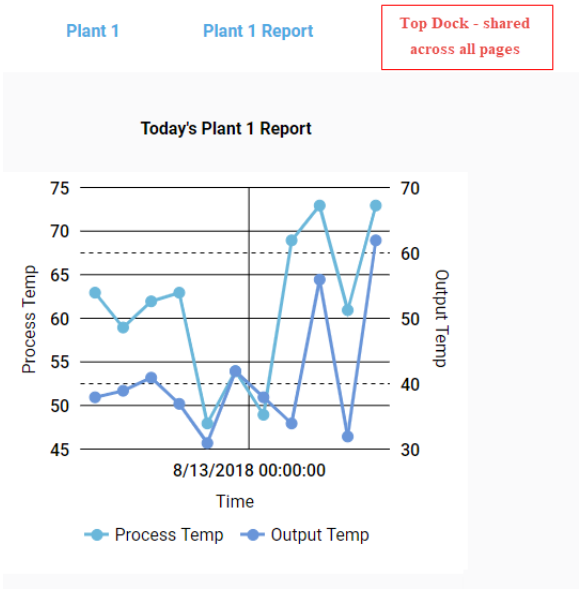
> ⚠ This example requires that you have one or more primary views created with assigned Page URL(s), and a docked view.

1. Open your **Page Configuration** if it's not already open.
2. Select a primary page from the Page Configuration column to add a docked view (i.e., **/Plant 1**).
3. Click the **Add** ✚ icon on the Right Dock. From the dropdown, select the view you want to be displayed on the specific page of your project.(i.e., **RightNav**).
4. Click **OK**.
5. If you want to customize your view, click on the **Edit** icon next to TopNav to open the Configure Dock View window.

Here is an example of Docked Views. Plant 1 and Plant 1 Report are two different pages displayed in a browser. The Top Dock view is shared across both pages. The Right Dock view shows only on the Plant 1 page because it is specific to this one page.

Plant 1 **Plant 1 Report**

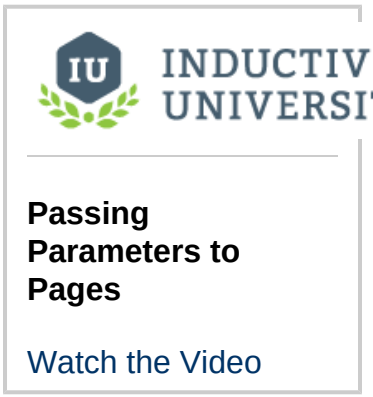


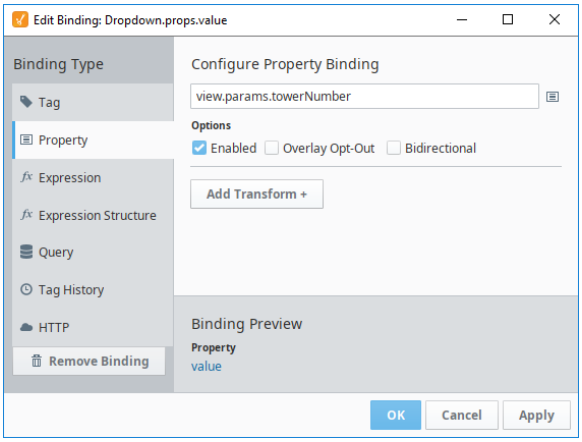


Passing Parameters (URL Parameters)

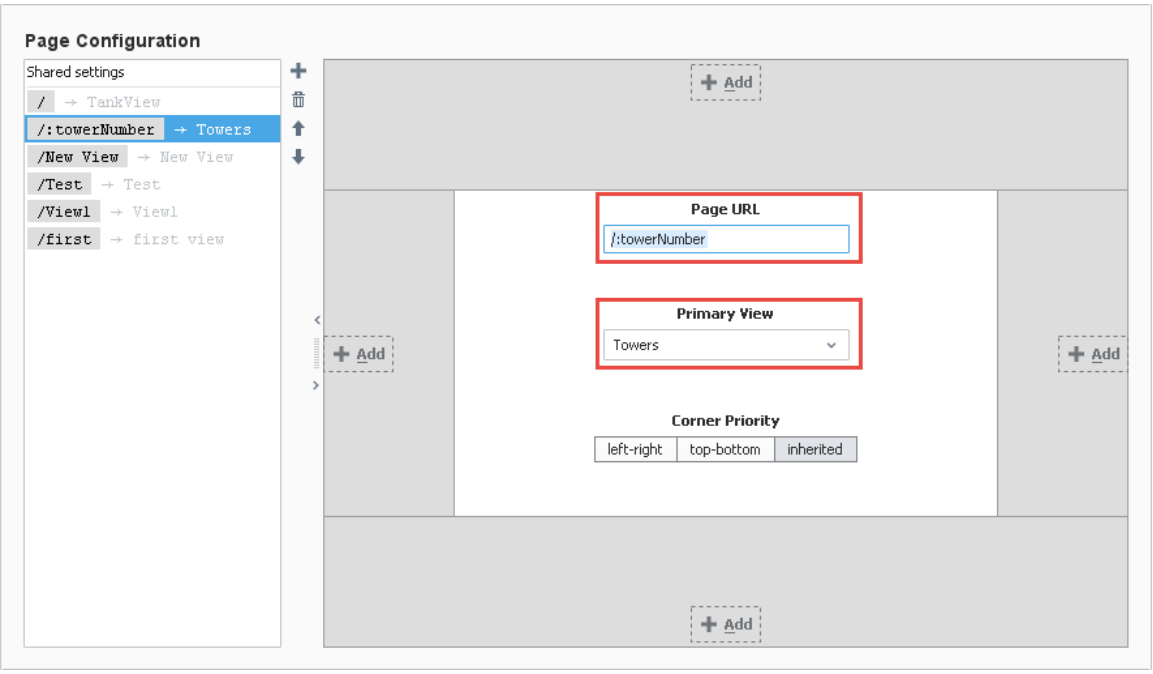
Pages can be mounted at URLs that include parameters. These parameters are used to allow a page to be mounted at a dynamic URL, allowing information in the URL to be interpreted as input parameters to the page's primary view. For example, suppose we had a page that displayed information about a Tower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

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

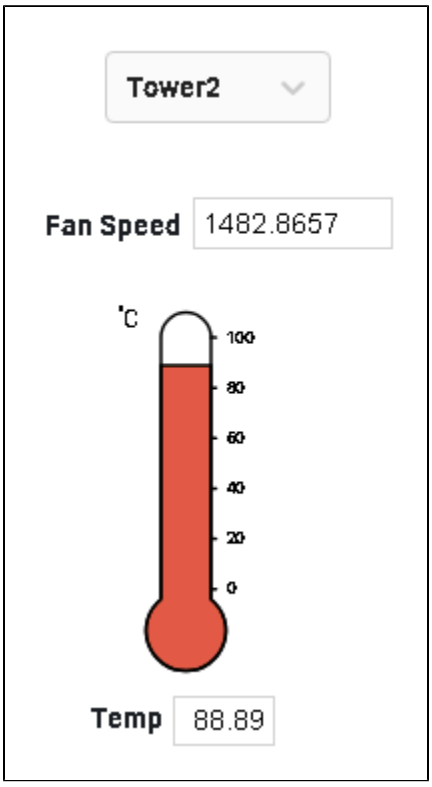




We configured our project to have a page, and set the page's primary view to **Towers**. We now need to configure the page's URL so that a user can navigate to any Tower using their browser's URL bar. To do this, we mount the page at a special URL using a parameter replacement syntax as follows: <page>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



Perspective understands this URL format. Therefore if the user were to navigate to the <URL> **/Tower2**, Perspective would display our page and render "**/"** as the primary view with an input parameter mapping of **towerNumber = Tower2** .
Using the example above, the primary view contained a parameter named **towerNumber**, and the value in the URL (i.e., **Tower2**) was passed to the parameter. Now, all the information about Tower2 is shown in the image below.



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

Original Page URL Property	Parameter Name	New Page URL	Sample URL in Browser
/	`param`	`/:param`	http://localhost:8088/data/perspective/client /ProjectName/value

/new-page	`param`	`/new-page/:param`	http://localhost:8088/data/perspective/client /ProjectName/new-page/value1
/new-page	areaNumber and line Number	`/new-page/:areaNumber/:lineNumber`	http://localhost:8088/data/perspective/client /ProjectName/new-page/100/101

Related Topics ...

- Views and Containers in Perspective
- Navigation Strategies in Perspective

Views and Containers in Perspective

Views and Containers are an integral part of the Perspective design experience because they work together to create your HMI screens - the windows into your application. The view is the primary unit of design and the Container provides a way of laying out and organizing child components within a View. Let's see how views and containers work.

Views

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.

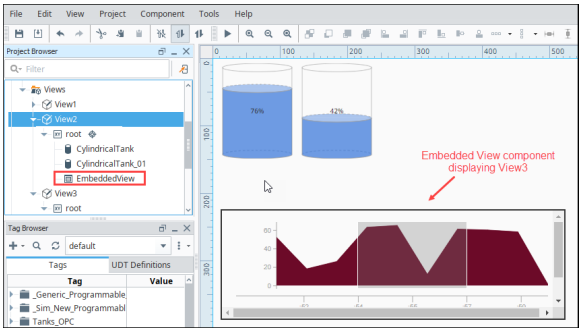
While views themselves are not strictly components, there are components that can display an instance of a View, such as nesting views inside of other views. There are also repeater components that may dynamically create view instances at runtime.

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.

Embedded Views

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

- **path** is the path to the view you wish to embed.
- **params** is an object in which you'll put any parameters you wish to pass through to the embedded view
- **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.



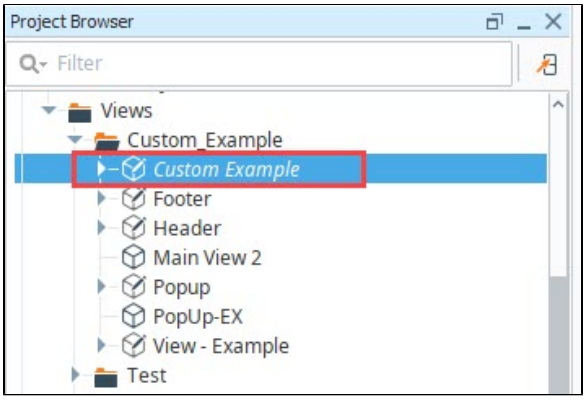
When designing your project, views need not be specified as embedded. It is possible that a single view may end up being instantiated as a regular view and an embedded view, even simultaneously in the same session. For example, you may have a view that shows 10 tanks using a view repeater as well as 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.

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

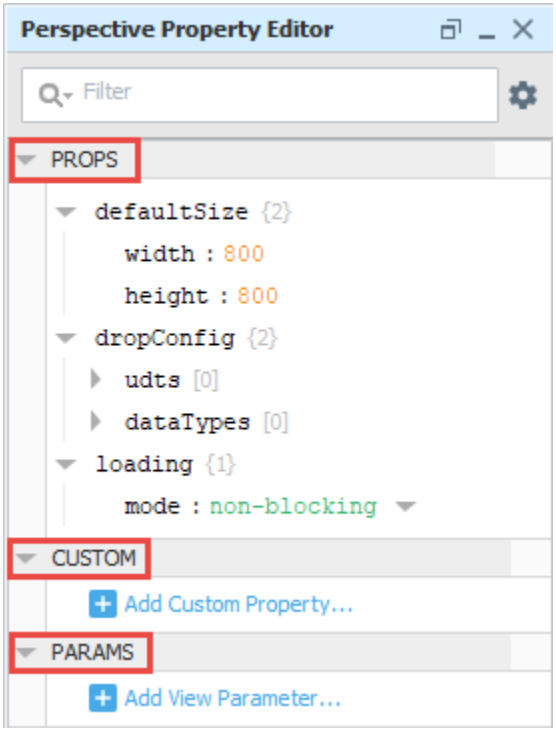
On this page

...

- Views
  - Embedded Views
  - View Properties
  - Input/Output Parameters
  - Popup Views
- Configuring Views
- Containers
  - Breakpoints
- Nesting Layouts Using Containers



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



Root level containers have properties are unique to the container type. For more information, see Root Container Properties .

View Properties

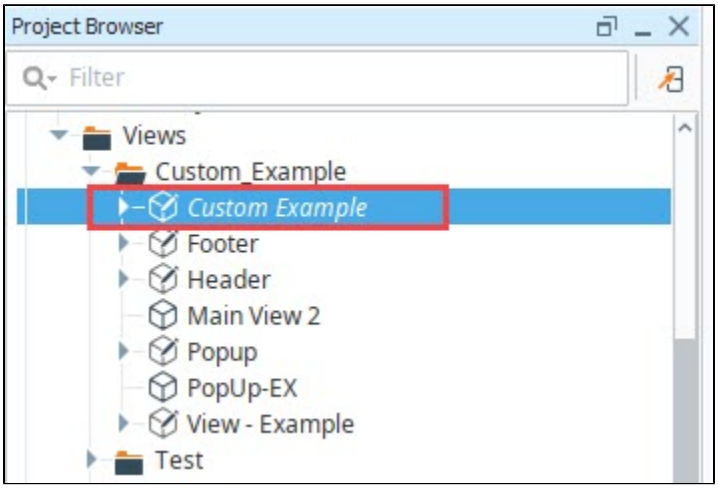
Views, like components, have properties. They are organized into a few types: **props** , **params** , and **custom** . Custom properties can be defined for views. They act just like custom properties of a component and are internal to the View, and can be referenced by all child components and containers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

Each view contains exactly one “root” level container, which may be any of the available container types: Column , Coordinate , Tab , Breakpoint , or Flex . Therefore, the design experience of a View is simply the design of the selected container type.

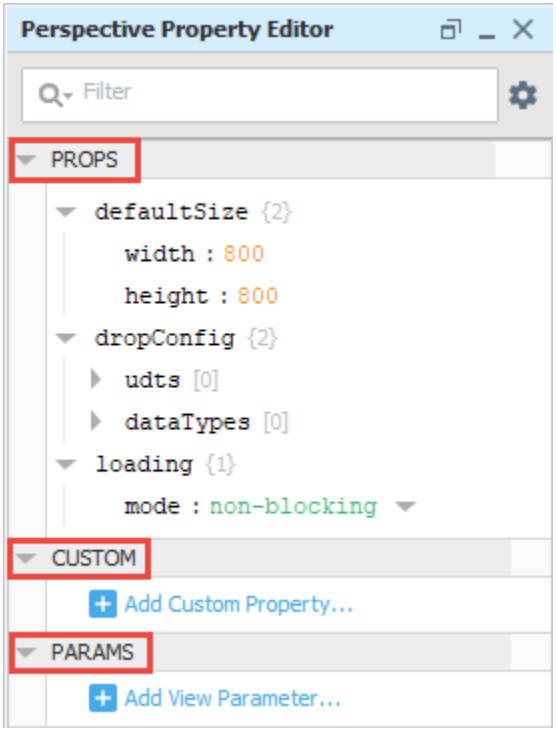
The view properties have three categories:

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

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



The property types will 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.

Popup Views

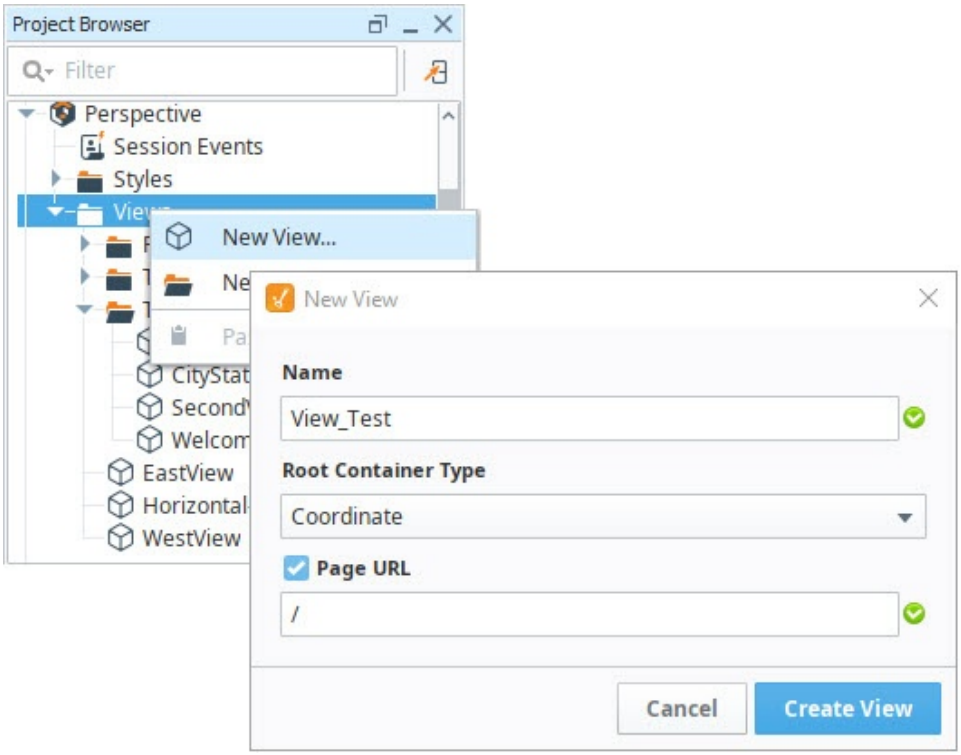
Popup Views float on top of a primary view and docked views. They are not part of a page's initial configuration, but may be opened using a popup action or scripting call. An easy way to create a popup view is by using a Button component to open and close the popup view in the primary view. A popup view can be opened and closed at the user's discretion in a Session. To learn more, go to Popup Views .

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.

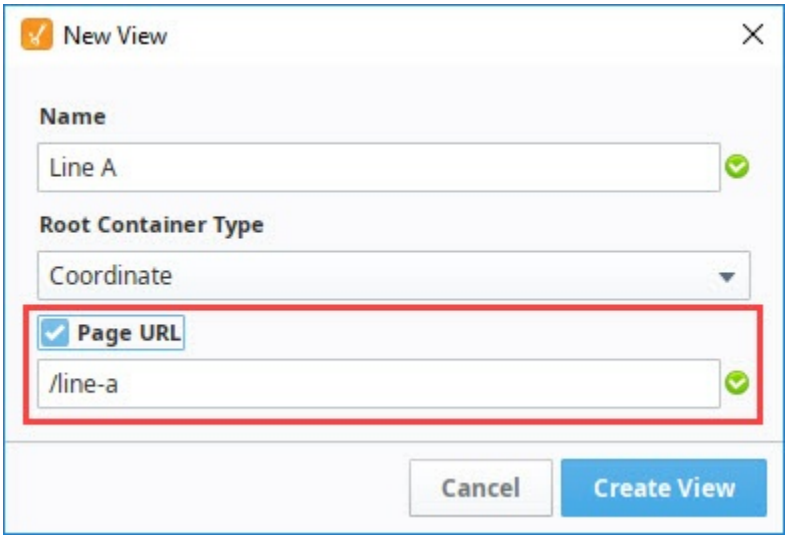
When you start designing your project in Perspective, the first thing you'll do is create a view and how you want to configure it. You can configure it as a primary view attached to a Page URL, or as a docked view on a specific Page URL with a primary view, or you can choose your docked view to be shared 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 select **New View.**
    a. In the New View window, give your view a name.
    b. Select a root container type.
    c. Make sure to check the **Page URL** checkbox if you want your view attached to the Page URL. Perspective will match the **Page URL** with the View's folder structure upon creation.
    d. Click **Create View**.

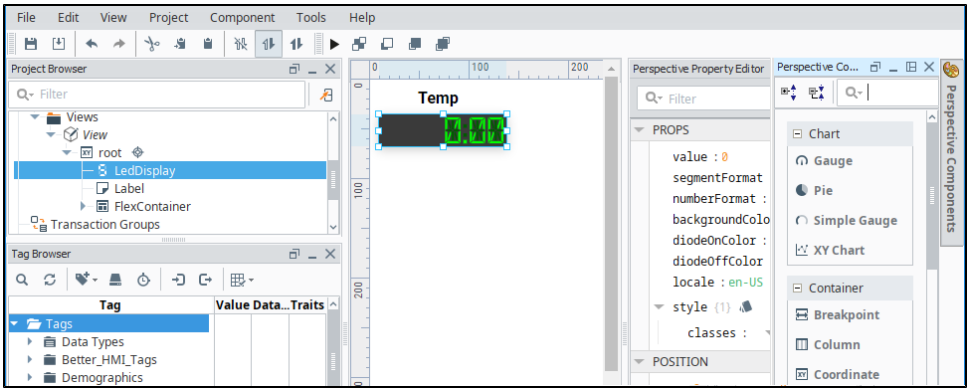


In release 8.0.8 the New View screen was updated to clarify the selection of container types.

When creating additional views, the Page URL will automatically get filled out for you 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 as shown in the image below.

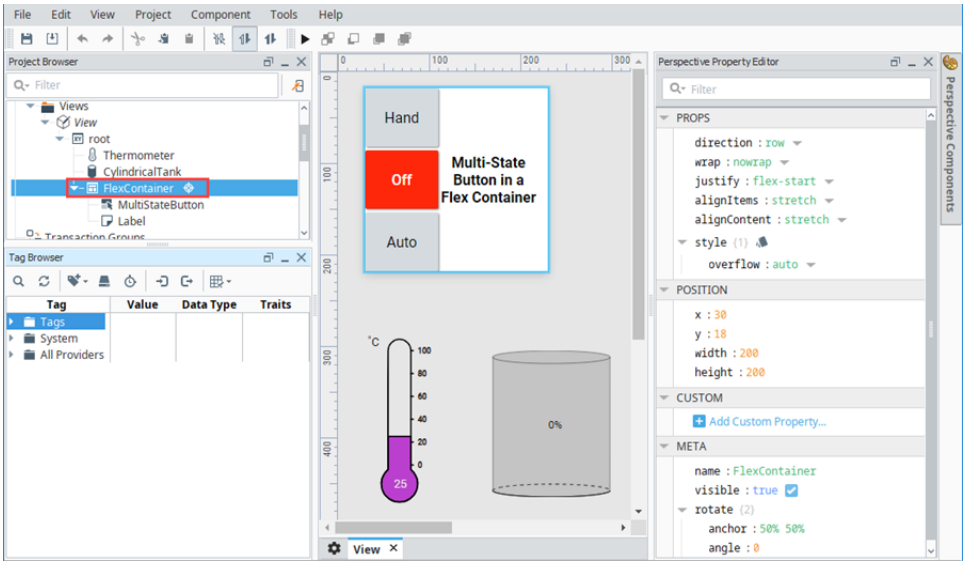


2. Once you create your view, the view will open in the Designer workspace, and you can begin adding and configuring components.



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.

> ⚠ To add components to this container, double click on the container to deep select it. Then you can 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.

Containers

The **Container** is a special kind of component. True to its namesake, a container is a component that contains other components. Within every created view is a **root** container that cannot be removed or renamed. The dimensions of this container will span the entirety of the view. If you wish to change the container type for a view, you'll need to create a new view. In the Project Browser, next to the root there will be an icon representing the type of container it is. There are types used in Perspective:

In Perspective containers enable you to have a *layout strategy* to control the size and shape of any housed components. The layout strategy defines how the container displays each of its child components in the view. There are a variety of container types that support different layout strategies.

Root Container Type	Icon	Description
Coordinate		This is the most simple container type. Components inside this type use basic X, Y, Width, and Height position properties. They can be fixed in size, or be configured to grow and shrink proportionally when the view is stretched. To see the properties and scripting functions available, got to Perspective - Coordinate Container and Coordinate Containers.
Column		The Column type shows a column-based layout and has three breakpoints. Breakpoints help manage a view when the devices using your session are very different sizes. Components inside this container type can span one or more columns, and you can add as many rows as you want. Row heights are determined by the tallest component in that row. To see the properties and scripting functions available, got to Perspective - Column Container and Column Containers.
Tab		The Tab type is used to show multiple views in the same page, while cycling through them all by clicking on each tab. You can add a single component, container, or view to each tab space. Each child element uses a tabIndex property for its position. To see the properties and scripting functions available, got to Perspective - Coordinate Container and Tab Containers.
Breakpoint		A Breakpoint container type allows you to create multiple versions of a single container using a pixel width as the cutoff. This allows you to completely change the look or structure for displays on mobile versus desktop sessions. To see the properties and scripting functions available, got to Perspective - Breakpoint Container and Breakpoint Containers.
Flex		The Flex type shows a single row of components spanning the height of the container. It allows you to set each component to either stretch to fill the available width, or to use a static width. The components in this view type will reposition and realign based on the size of the view. To see the properties and scripting functions available, got to Perspective - Flex Container and Flex Containers.

Breakpoints

A simple strategy to accommodate radically different screen sizes is to design a view or container in several different ways, with each design tailored to different screen sizes. The column and breakpoint types accommodate this design strategy, by picking one of two or three different containers to open based on a specific range of pixel widths for the session. The term "Breakpoint" is a common piece of terminology used to describe this design - it is the width in pixels at which the Perspective session decides to use one container over another.

Using Column Containers

The Column container is a good option when you want to have the same components and views for every screen size, and simply want to lay them out differently at smaller or larger sizes.

Selecting one of the sizes will give you the opportunity to configure components' layouts for that size. For instance, selecting **Small** will allow you to position the components, embedded views, and containers in your view in a way that makes sense for mobile devices, since most mobile phones will sport a session width less than 480 pixels. Rearranging the same components in the **Medium** container would provide a nice design for larger smartphones, many tablets, and some desktops, while the **Large** configuration will yield a container that works for high-resolution monitors and televisions.

Using Breakpoint Containers

The Column container is an easy-to-use option for simple layouts, but more involved ones might require more customization. Perhaps you want desktop, tablet, and mobile sessions to look and feel quite different, or you want to use one of Perspective's new navigation components , but only for mobile devices. Even in these circumstances, you don't have to design several sets of pages, or even several projects, to all implement the same task.

Breakpoint containers enable you to design views tuned to different screen widths, but unlike a column container, where a single view can take on three forms, in a breakpoint container you actually specify two distinct views, each with their own components and functionality. Breakpoint containers are thus the more flexible option; as long as your "mobile" views share an essentially similar framework with your "larger" views, you will be able to develop the views concurrently, and the Perspective session will decide at runtime which set of views to use. You can think of a breakpoint container as bundling together two views that serve a similar purpose, but are tailored to different screen sizes.

Nesting Layouts Using Containers

Sometimes you may want to compartmentalize a view or container in a particular way, either to modify the component layout or organize the structure of your components in the Project Browser. Containers are a special kind of component that can contain children and can also be nested, which supports multiple layout strategies. This means that application designers can nest containers inside of other containers to compose layout strategies that are more complex and capable than any one container can provide.

You might want to have nested containers to create a more complex layout. For example, you could have a Tab container that has several tabs with very different information in each tab. In one tab, you might have a table that looks great on a desktop but terrible on a phone. Using a Breakpoint container inside of only one tab in a Tab container will allow you to make a more useful mobile experience while still showing a table on a desktop computer.

Single-Component Containers

Sometimes you'll want to make a very simple view for use inside a higher-level setting, like a Carousel or a Breakpoint container. If you only need a view to hold a single component that fills the container, you can:

- Use a coordinate container in **percent** mode, and give the component **X** and **Y** of **0**, and **width** and **height** of 1.
- Use a flex container, giving the component a basis of **100%** or **auto**.

Repeated Containers

Sometimes you'll need to repeat the same container a specific number of times in a row or column, or you want to create a grid of child containers. For this purpose, you can:

- Use a Flex Repeater component, in the specific instance that you only need a row of instances of a specific view.
- Use a Flex Repeater component, when you want to use more than one container or view.

Coordinate Containers

A **Coordinate** container creates a traditional yet simple type of layout to use in Perspective. Each element (i.e., component or nested container) inside the layout has the familiar x, y, width, and height **Position** properties for each component. The way these properties work depends on the **mode** property of the layout, specified on the Coordinate container.

On this page

...

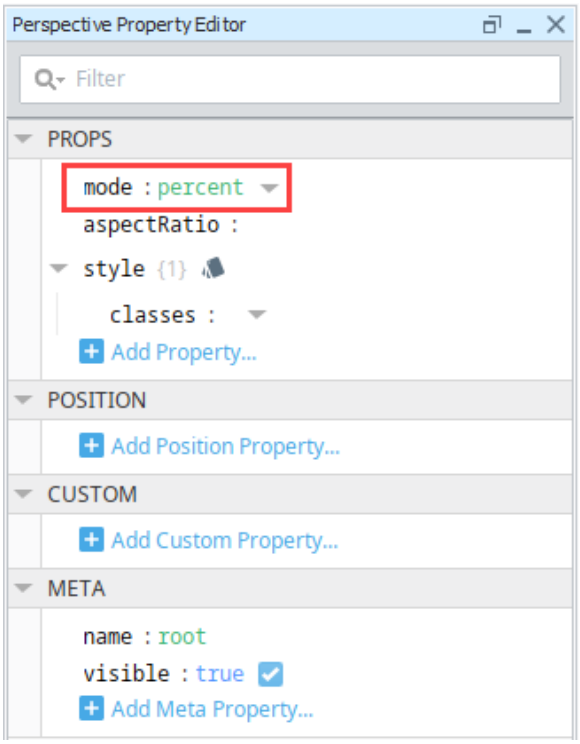
- Position Properties
- Root Properties
- Configuring a Coordinate Layout
  - Fixed Mode (No Stretching)
  - Percent Mode (Simple Stretching)
  - Using the Coordinate Container as a Component

Position Properties

The position properties are always named the same, but have two different modes that drive how they work: **Fixed** and **Percent**. These two modes change how the width and height values work in respect to the view size.

- **Fixed mode** - The width and height are a fixed number of pixels. The component **will not** stretch or shrink as the view size changes.
  - **X** and **Y** specify the offset, in pixels, of the top left corner of the element, relative to the top left corner of the coordinate container. A component flush with the top and left sides of its parent will have coordinates x = 0, y = 0.
  - **width** and **height** specify the size of the element in pixels.
- **Percent mode** - The width and height are a variable percentage of the width and height of the view. The component **will** stretch or shrink as the view size changes.
  - **X** and **Y** specify the offset of the top left corner as a ratio of the width and height of the coordinate container. For example, x=0.1, y=0.1 will set the top left corner of the element at 10% across, 10% down.
  - **width** and **height** specify the width and height as a ratio of the width and height of the coordinate container.

To change the Mode property, select the **Root** object under the View in the Project Browser. The Mode property is the first in the list.



The position properties control the location of a component on the view and the width and height of the component. The rotate property sets the angle of rotation for a component and the point (anchor) around which it should be rotated. For more information on rotating components, see Working with Perspective components.

Mode: Fixed		
Position Property	**Description**	**Data Type**

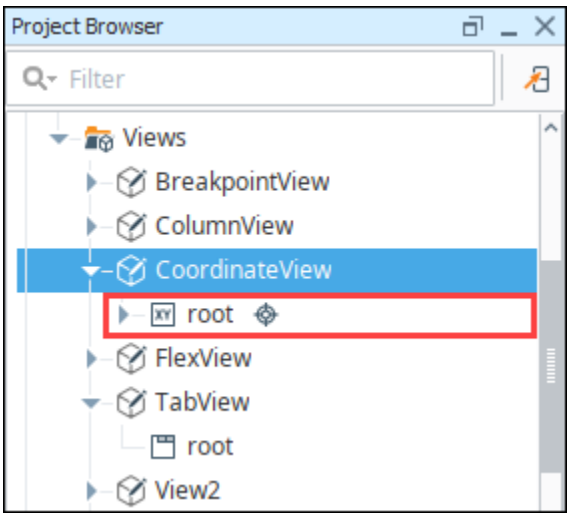
x	Specifies the horizontal positioning of a component in pixels.	value: numeric
y	Specifies the vertical positioning of a component in pixels.	value: numeric
width	Specifies the width of a component in pixels.	value: numeric
height	Specifies the height of a component in pixels.	value: numeric
rotate	Settings that set the anchor and angle of rotation. **Position Property** / **Description** / **Data Type** anchor: The point around which the rotation happens. Either as an {x:number, y:number} object where x and y represent percentages such that {x:0, y:0} represents the (0%, 0%) or top-left corner of the component, or as a valid CSS transform-origin string. — value: string, or object of numbers angle: How much to rotate the component. Valid values include numbers (as degrees), and valid CSS angle strings such as '45deg', '2rad', '0.5turn', etc. — value: numeric or string	object

Mode: Percent

Position Property	Description	Data Type
x	Specifies the horizontal positioning of a component in pixels.	value: numeric
y	Specifies the vertical positioning of a component in pixels.	value: numeric
width	Specifies the width of a component as a percent from 0 to 1 where 1.0 means 100% of the view width.	value: numeric
height	Specifies the width of a component as percent from 0 to 1 where 1.0 means 100% of the view height.	value: numeric
rotate	Settings that set the anchor and angle of rotation. anchor: The point around which the rotation happens. Either as an {x:number, y:number} object where x and y represent percentages such that {x:0, y:0} represents the (0%, 0%) or top-left corner of the component, or as a valid CSS transform-origin string. — value: string, or object of numbers angle: How much to rotate the component. Valid values include numbers (as degrees), and valid CSS angle strings such as '45deg', '2rad', '0.5turn', etc. — value: numeric or string	object

Root Properties

Root properties are accessed by selecting the **root** folder for a Perspective View on the Project Browser tree.



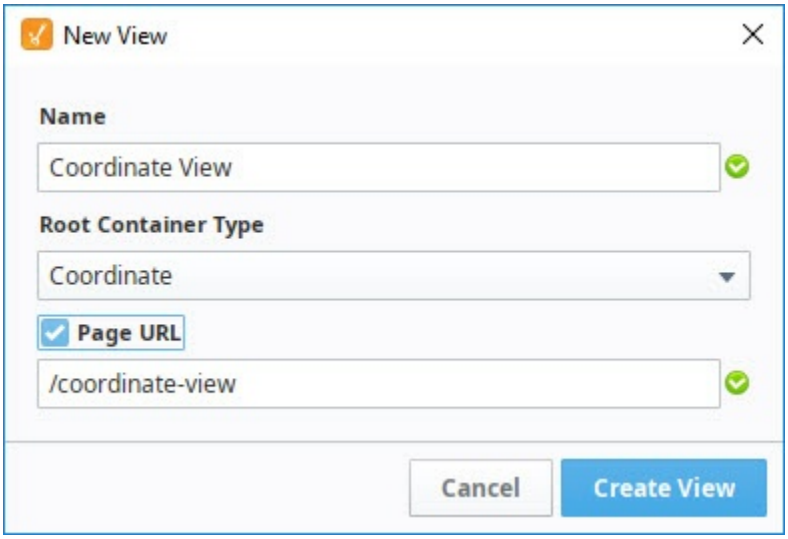
Root Property	Description	Data Type
mode	Whether child layouts should always be in fixed coordinate space, or should stretch relative to different container sizes. Options are fixed or percent.	value: string
aspectRatio	Only applies in percent mode. Optional dimension, in x:y format, to apply to maintain container aspect ratio for different sizes. Empty string (or non x:y input) will disable this mode.	value: string
style	Sets a style for this view. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a style class.	object

Configuring a Coordinate Layout

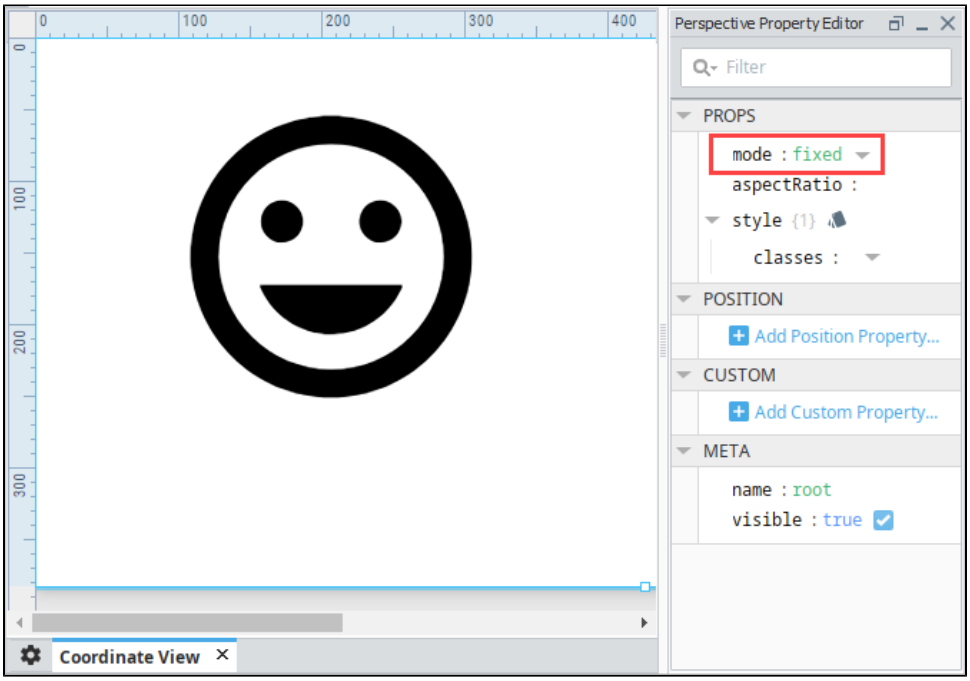
Fixed Mode (No Stretching)

The default behavior of all new Coordinate Layouts is Fixed mode (no stretching). In the **Fixed** mode components are given x, y, width, and height values in pixels and will strictly stick to those. This mode will not adjust its size or shape regardless of screen size. If you want to plan out your views to the pixel, coordinate layouts serve this purpose well. If you plan to launch your Perspective Sessions on a variety of different screen sizes, the Coordinate Layout is not very flexible.

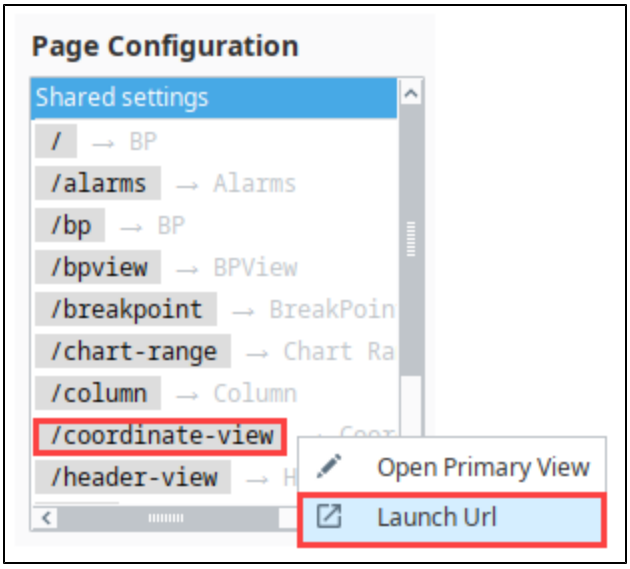
1. Let's create a new view using the Coordinate layout with a Page URL.



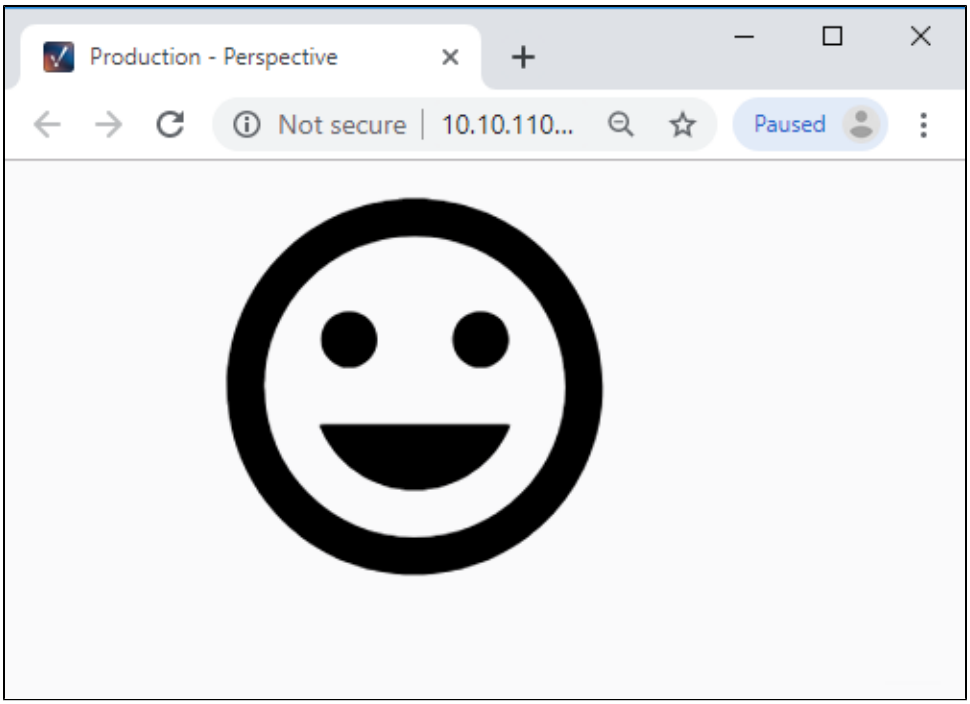
2. Drag a component into your view. In this example, we used the Icon component. You'll notice the Coordinate container's **mode** property defaults to **fixed**. Place your components and sub-containers however you see fit. Use the edge handles of the component or position properties to change the width and height.



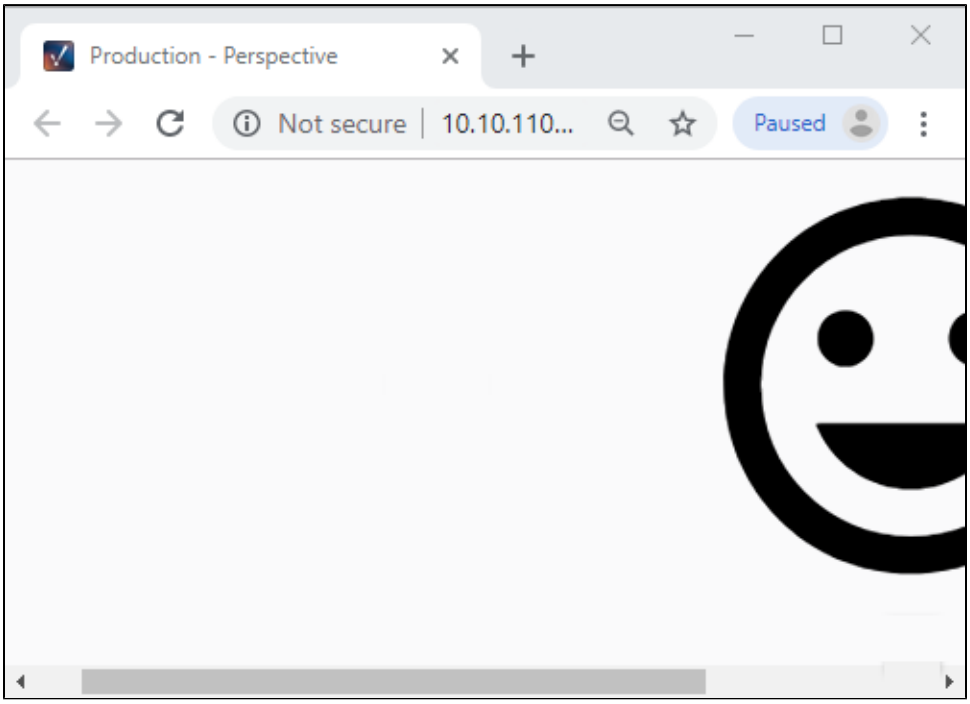
3. Let's open a Perspective Session by clicking on the **Page Configuration** icon. Right click on your Page URL for your Coordinate View to launch a Perspective Session.



4. A Perspective Session will never move, stretch, or shrink the components on the screen.



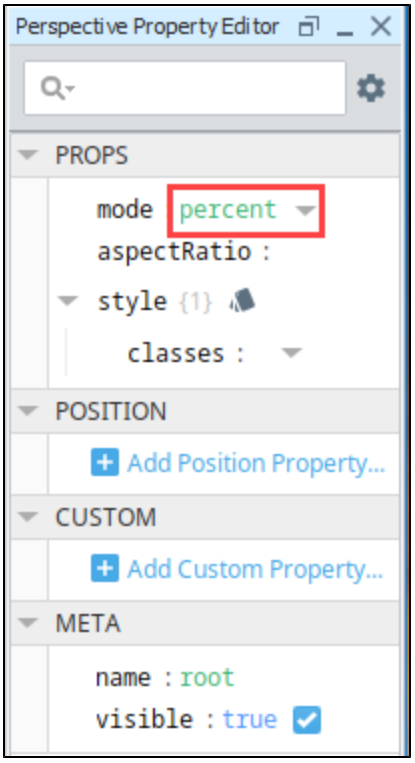
5. If you resize the session to make the screen narrower, you'll notice that the components will not resize, and you'll get scroll bars at the bottom of the screen.



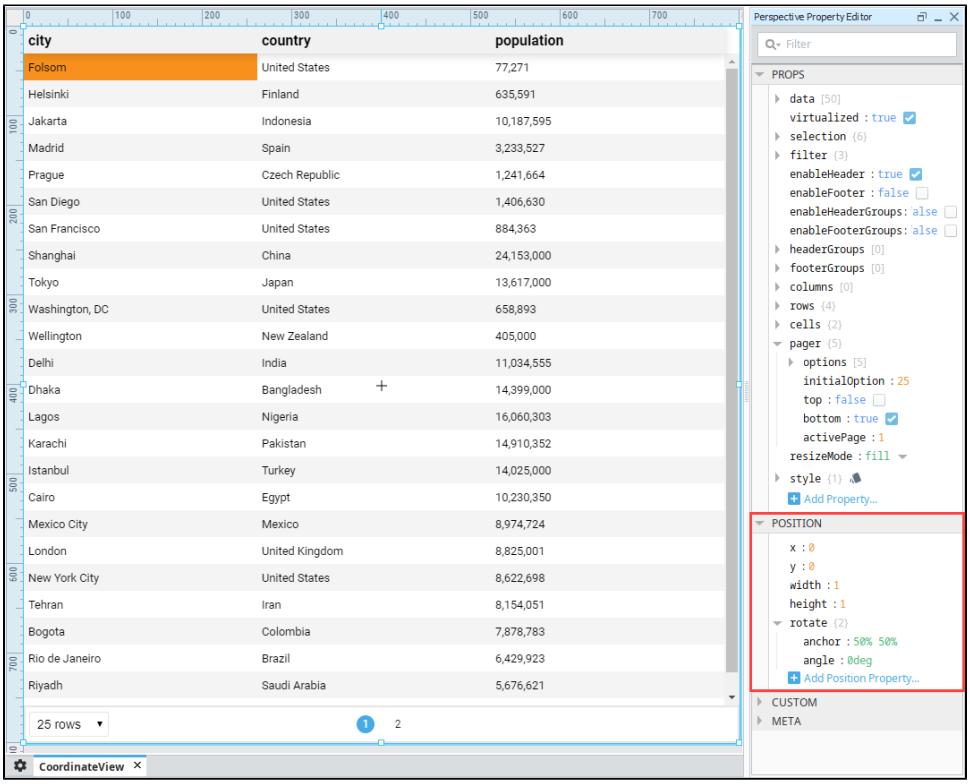
Percent Mode (Simple Stretching)

The **Percent** mode on coordinate layouts provides a simple and intuitive scaling option for screens of different sizes. As a special case, you'll often want to make a component or container fill its parent container entirely, regardless of the final dimensions of the parent. A Coordinate Layout is one way to accomplish this:

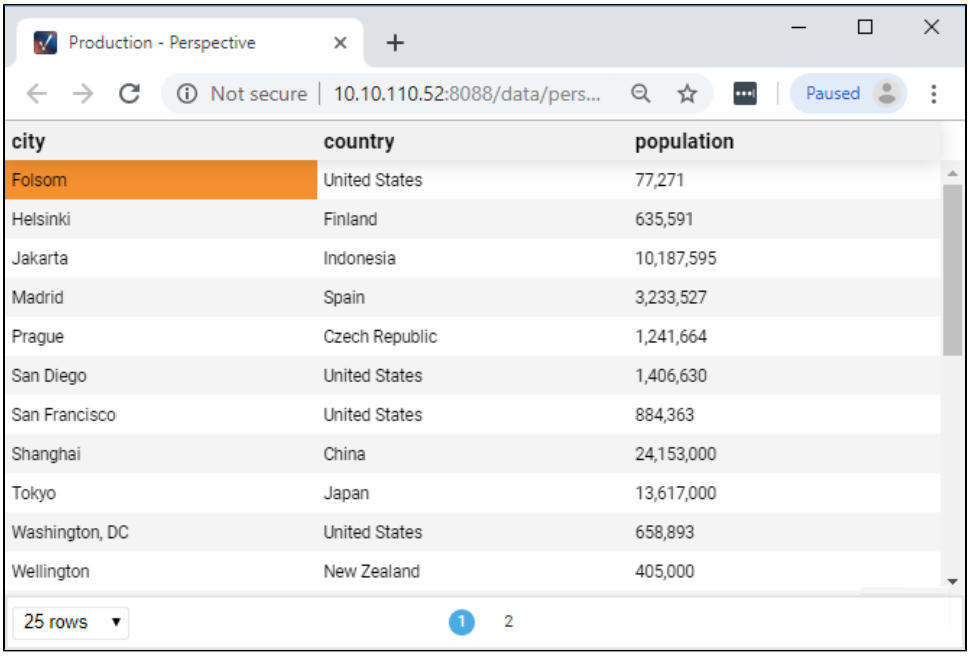
1. If working in a Coordinate View, select the view's root container, otherwise select the Coordinate Container component.
2. In the Perspective Property Editor, set the container's **mode** property to **percent**.



3. From the Component Palette, drag a new component to the top left corner of the parent of your Designer workspace. For this example, we used a Table component.
4. Set the the **x** and **y** position properties to 0.
5. Set the **width** and **height** properties to 1.



6. Percent mode allows your components to resize and completely fill the space as you can see in this Perspective Session.



Using the Coordinate Container as a Component

The Coordinate container also works well as a separate component. You can use it as a small container to hold components that you don't want to move around, and place it inside a more responsive container such as a Flex container or Column container. This works extremely well if you are looking for a more responsive design for your project.

Related Topics ...

- Pages in Perspective
- Perspective Coordinate Container

Column Containers

In a Column Layout, the view or container is divided into consistently-sized and spaced columns. By default, there are 12 columns. Each component's width is described only by how many of these columns it spans. Since these columns will grow wider or narrower to accommodate changes in the width of the parent container, elements in a column layout can stretch horizontally. To this end, each component has two properties that control its size and position:

- **height** for components in a column layout does not scale with the height of the layout itself. It is specified in pixels, and unlike width will not stretch or shrink based on the parent container. A height of **auto** will tell a component to keep the same vertical size as other elements in its row.
- **breakpoints** contains three sets of information about the component's positioning. Which set is used is based on which **breakpoint** is active (more on breakpoints follows). Each set consists of:
    - **name** is "sm", "md" or "lg". The name simply corresponds the the breakpoint that the particular set of positioning data refers to.
    - **span** refers to how many columns wide the entity should be in the current breakpoint.
    - **rowIndex** and **colIndex** refer to the X and Y coordinates of the leftmost cell that the element occupies in the given breakpoint.
    - **order** refers to the order of the element in its row. This property will be set and updated automatically.

On this page

...

- Easy Screen Flexibility
- Column Container Position and Root Properties
    - Position Properties
    - Root Properties
- Configure a Column Container

Easy Screen Flexibility

A Column Layout is a simple approach to what could be termed responsive design - the process of developing the same view, or similar ones, in different ways to seamlessly accommodate everything from phone screens to 4K televisions in a single project and set of pages.

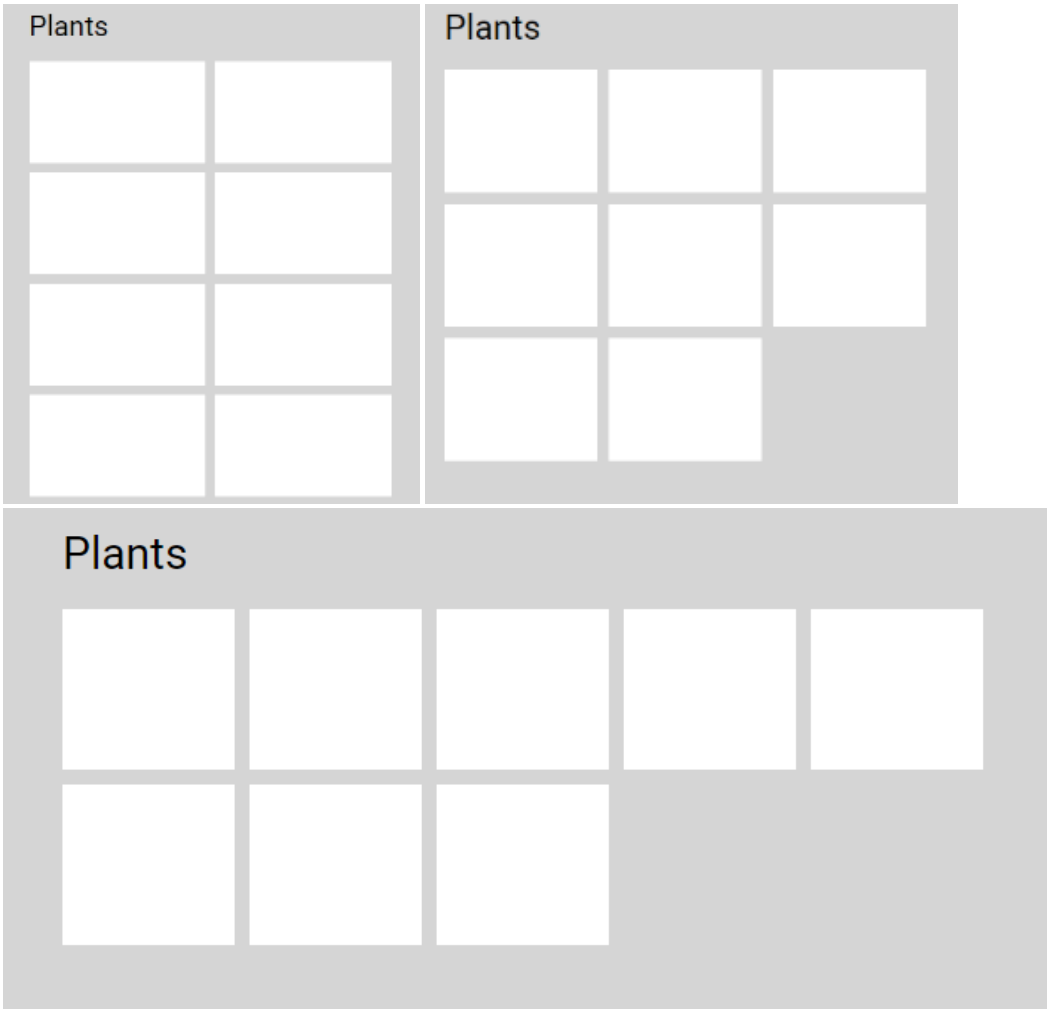
Breakpoints encapsulate this strategy in Perspective. The column layout offers the opportunity to establish three different versions of your view, which will then be used at different ranges of screen widths. You can find and configure these versions from the top of the Perspective Property Editor for a Column container; they are:

- **Small** (default 0 px)
- **Medium** (default 480 px)
- **Large** (default 996 px)

The px values denote at what minimum screen width (in pixels) the view or container should use a given configuration. Selecting one of the sizes will give you the opportunity to configure component positions and spans for that size. This gives more refined control over how a given view or container scales. A column layout is appropriate when your view or container meets some specific criteria:

- You don't need your components to scale vertically, outside of the three provided breakpoints.
- You want to include the same components in the view regardless of screen size.
- You want the components to behave the same way at any size (in terms of scripting, navigation, bindings, etc.)
- You don't need screens at different sizes to use different views.

Here is an example of a column view, configured with small, medium, and large breakpoints.



Column Container Position and Root Properties

Position Properties

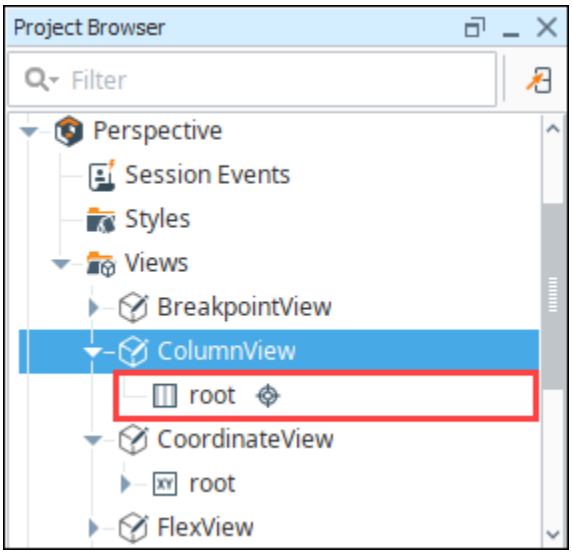
Position properties in for a Column container indicate the height of the component placed in each column and the name and order of the breakpoints. For more information, see Perspective - Column Container.

Position Property	Description	Data Type
height	Height, in pixels of the component placed in the column.	int
breakpoints	<table><tr><th>Position Property</th><th>Description</th><th>Data Type</th></tr><tr><td>name</td><td>Name of a breakpoint defined in container. If this matches the currently applied breakpoint, these rules determine child layout. Options are sm (small), md (medium), or lg (large).</td><td>value: string</td></tr><tr><td>span</td><td>Number of columns the child's width will span.</td><td>value: numeric</td></tr><tr><td>rowIndex</td><td>Row index (starting from 0) in which to place child. Children may wrap lines within a row. Children in separate rows don't affect each other's layout.</td><td>value: numeric</td></tr><tr><td>colIndex</td><td>Column number upon which the child's span should begin unless forced to wrap.</td><td>value:</td></tr></table>	array

		numeric
order	Where component is places among its siblings within its row. Ordering is independent per row.	value: numeric

Root Properties

Root properties are accessed by selecting the **root** folder for a Column view on the Project Browser tree.

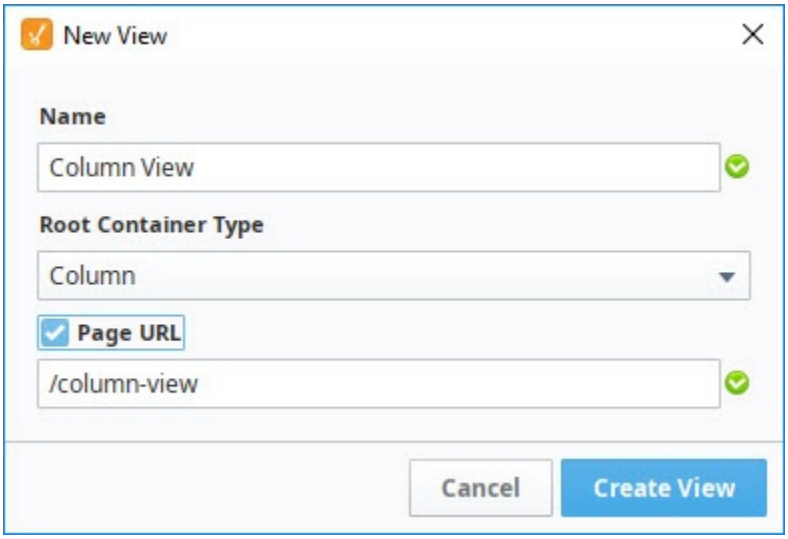


Root Property	Description	Data Type
columns	Number of column spacings the container is divided into. Children plot layouts across groups of columns, or wrap to the next line when space does not allow. Default is 12.	value: numeric
breakpoints	Width breakpoints declarations for child layouts. When container is sized below minWidth, child position rules will fall back to the next set breakpoint rules. **Position Property** / **Description** / **Data Type** name / Name of the breakpoint. Options are sm (small), md (medium), or lg (large). / value: string minWidth / Minimum width of container, in pixels. / value: numeric	array
gutters	Amount of space, in pixels, to place between child components. **Position Property** / **Description** / **Data Type** vertical / Vertical space, in pixels, to place between child components. / value: numeric horizontal / Horizontal space, in pixels, to place between child components. / value: numeric	object
style	Sets a style for this view. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a style class.	object

Configure a Column Container

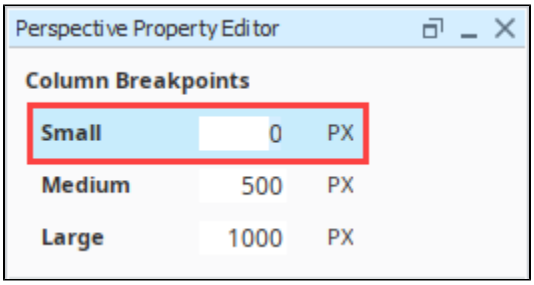
The Column container allows you to switch between three different views depending on the size of your Perspective Session. In this example, let's create a data entry form for the company's client list. We'll create a view and add components, then we'll switch between the views to arrange the components for each layout size.

1. Create a new view using a Column container.



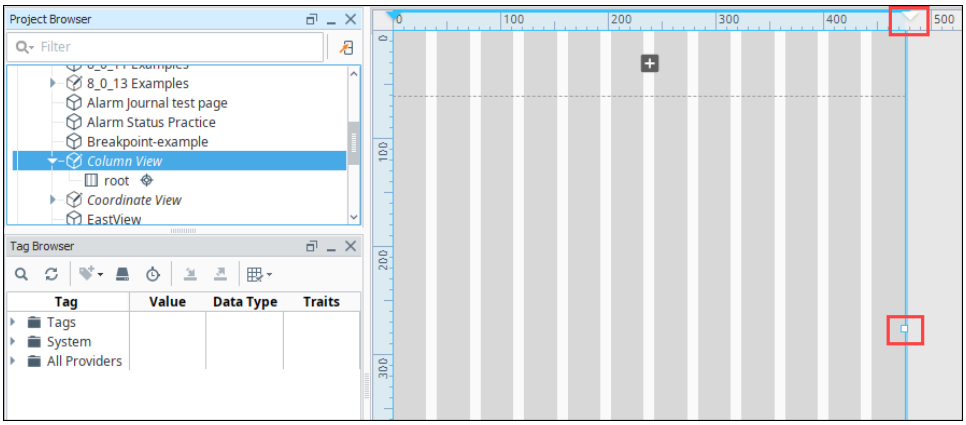
2. With the new Column container selected, you'll see the column breakpoints at the top of the Property Editor. Set each of the breakpoints to the following:
    a. Small breakpoint - **0**
    b. Medium breakpoint - **500**
    c. Large breakpoint - **1000**

3. Select the **Small** breakpoint in the the Property Editor.



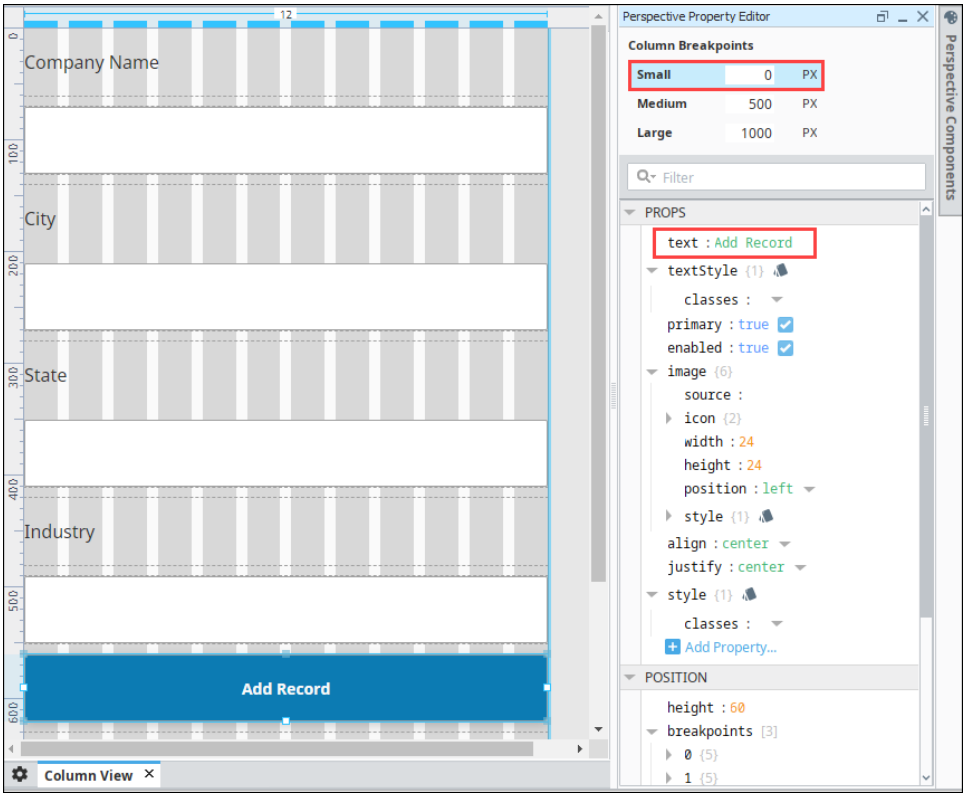
> **Selecting Breakpoints**
>
> It's important to pay attention to which Breakpoint is selected. Any changes you make will be applied only to the *currently* selected breakpoint.

4. Click and drag the square handle on the right side of the view to shrink the width of the container to just under 500 px wide. Arrow guidelines along the top of the component show where the small breakpoint ends and medium breakpoint begins at 500 px.

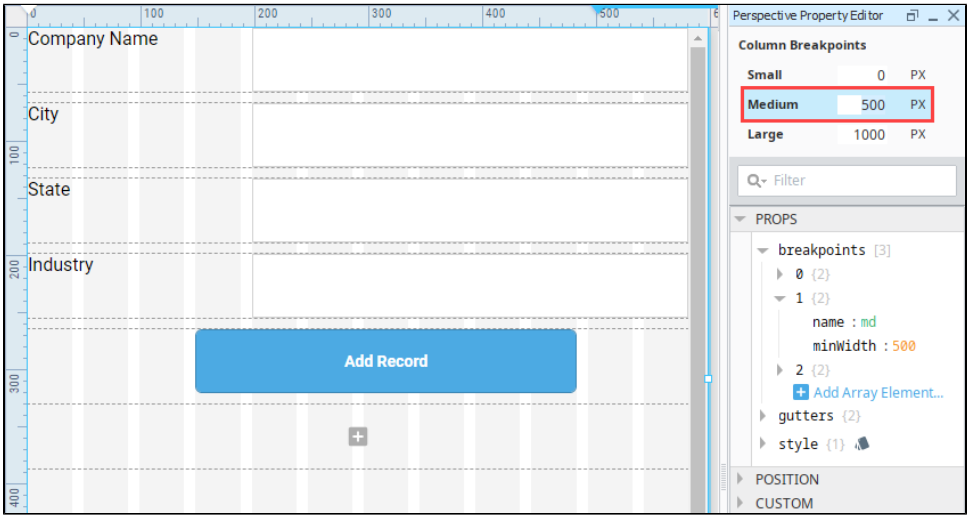


5. Let's add some components to the Column container.
    a. Drag a Label component on to the container and expand it to fill all 12 columns in the first row.
    b. Drag a Text Field component on to the container and expand it to fill all 12 columns of the second row.
    c. Repeat Steps a and b until you have four Labels and four Text Fields.
    d. Add a Button component at the bottom of the screen and expand it to fill all 12 columns.

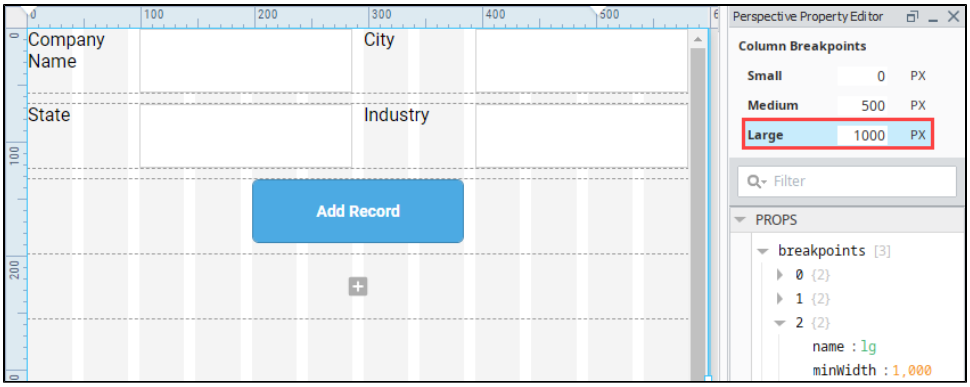
6. In the Property Editor, update the **text** property of the Label and Button components and assign them a useful description:
    a. First Label: **Company Name**
    b. Second Label: **City**
    c. Third Label: **State**
    d. Fourth Label: **Industry**
    e. Button: **Add Record**



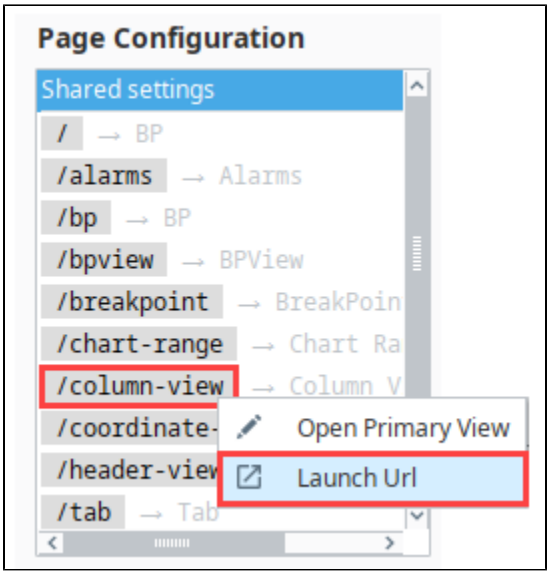
7. This is a good time to **Save** your project before continuing.
8. Now, let's switch to the Medium breakpoint. Select the **Medium** breakpoint at the top of the Property Editor. Notice that all of the components are still showing, but they are now unorganized.
9. Using the square handle on the right side of the view, click and drag it to expand the width of the container to approximately 500 px wide. This will give you a better idea of what the view will look like in the browser.
10. Move the components around so each row has Label and one Text Field.
11. Set each Label to the first four columns.
12. Set each Text Field to be the last eight columns.
13. Keep the Button component in its own row at the bottom and shrink it to be centered with eight columns.



14. **Save** your project.
15. Now, select the **Large** breakpoint in the Property Editor. Once again, you'll notice that the all the components are showing, but they are unorganized.
16. Using the square handle on the right side of the view, click and drag it to expand the width of the container. This will give you a better idea of what the Large view will look like in the browser.
17. Move the components around so each row has two Labels and two Text Fields on each row.
18. Set each Label to be two columns wide and each Text Field to be four columns wide.
19. Keep the **Button** component on its own row at the bottom and shrink it to be centered with four columns.



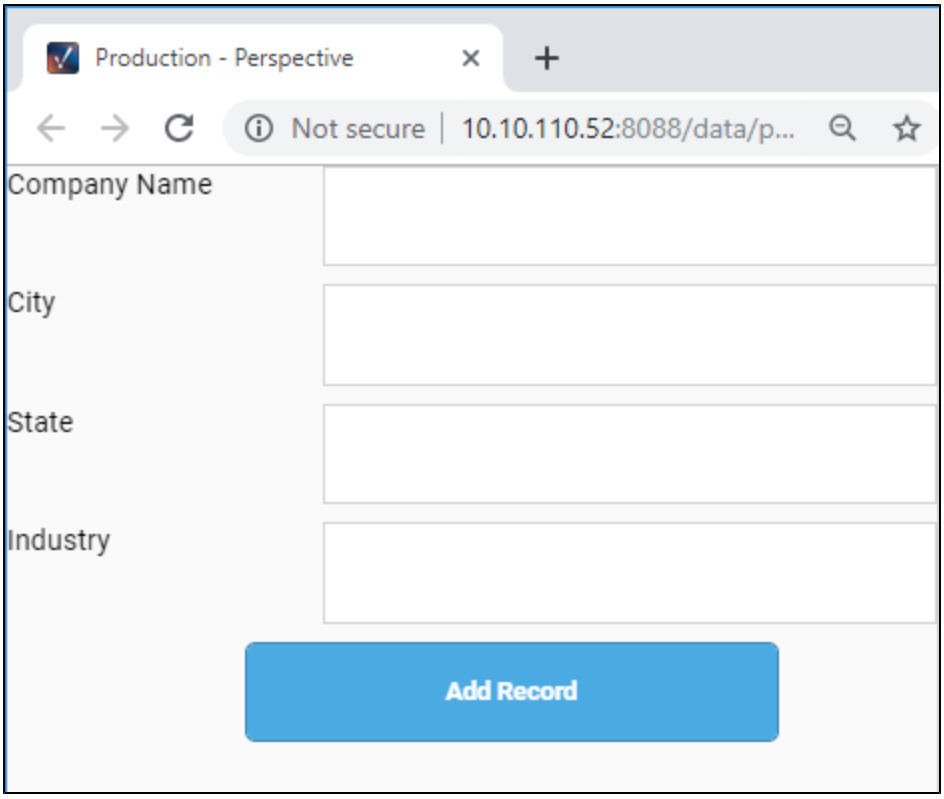
20. **Save** your project.
21. Let's launch a Perspective Session to test our breakpoint views. Click on the **Page Configuration** icon at the bottom of the Designer window.
22. Right click on your column container and select **Launch Url**.



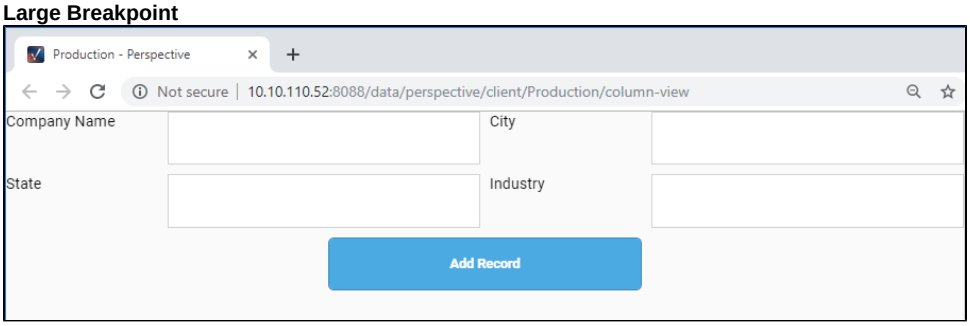
23. With your Perspective Session open, stretch and shrink your browser window and you'll see the views change depending on the width of your browser.

Small Breakpoint

Medium Breakpoint



Large Breakpoint



Related Topics ...

- Pages in Perspective
- Perspective Column Container

Tab Containers

A Tab container can be used for a relatively simple layout type. Tabs run along the top of the layout. You can configure and select a distinct embedded view, container, or component in each tab. The tab headers themselves can be configured to show text or even embedded views. A **menu Type** property on the tab container offers two basic appearances for the tabs: **classic** and **modern** .

Critically, each element in a Tab container has only a single position property, **tabIndex**. The edges of the enclosed element will be stretched to fill the entire tab, allowing easy dragging and dropping of different subviews. Of course, you can nest containers inside of a tab layout to provide whatever functionality you'd like.

For details on the properties in a Tab component, see Perspective - Tab Container.

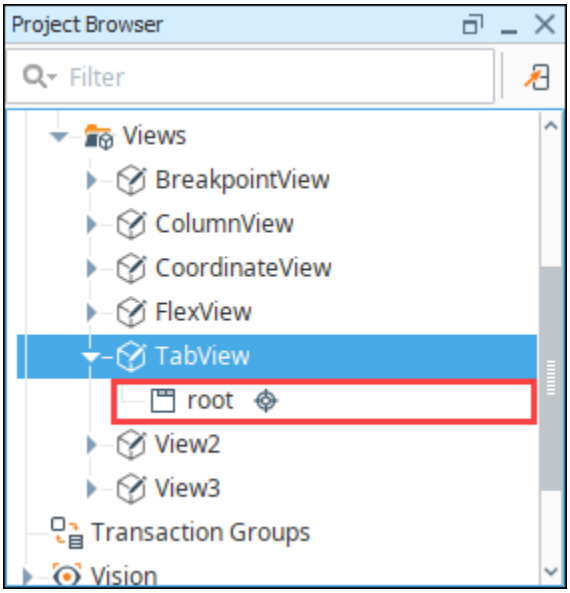
On this page

...

- Tab Container Root Properties
- Tab Navigation
- Nested Tabs

Tab Container Root Properties

Root properties are accessed by selecting the **root** for a View with a Tab container on the Project Browser tree.



Root Property	Description	Data Type
tabs	Content to display in the menu as tabs. Each tab in this array may be either a string to display as the tab text or an object with viewPath and optionally viewParams. If the latter, a view will render as the tab in place of text.	array
currentTableIndex	Which index in the tabs property is currently active.	value: numeric
menuTy	If the type is 'classic', a traditional menu with boxed tabs is shown. The 'modern' type has no borders around each	value:

<table>
<tr><td>pe</td><td>tab and shows selection with an underline. Options are classic or modern.</td><td>string
dropdown</td></tr>
<tr><td>tabSize</td><td>Default size allotted to a single tab. If container width does not allow, tab width will shrink from this size accordingly.
<table><tr><th>Root Property</th><th>Description</th><th>Data Type</th></tr><tr><td>width</td><td>Width of tab, in pixels. Default is 96.</td><td>value: numeric</td></tr><tr><td>height</td><td>Height of tab, in pixels. Default is 36.</td><td>value: numeric</td></tr></table></td><td>object</td></tr>
<tr><td>menuStyle</td><td>Additional styling to apply to the menu.
<table><tr><th>Root Property</th><th>Description</th><th>Data Type</th></tr><tr><td>backgroundColor</td><td>Background color for the menu.</td><td>color</td></tr></table></td><td>object</td></tr>
<tr><td>tabStyle</td><td>Additional styling to apply to all tabs depending on active (selected) or inactive state.
<table><tr><th>Root Property</th><th>Description</th><th>Data Type</th></tr><tr><td>active</td><td>Style for all tabs when active. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous.</td><td>object</td></tr><tr><td>inactive</td><td>Style for all tabs when inactive. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous.</td><td>object</td></tr></table></td><td>object</td></tr>
<tr><td>style</td><td>Sets a style for this view. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a style class.</td><td>value:
numeric</td></tr>
</table>

Tab Navigation

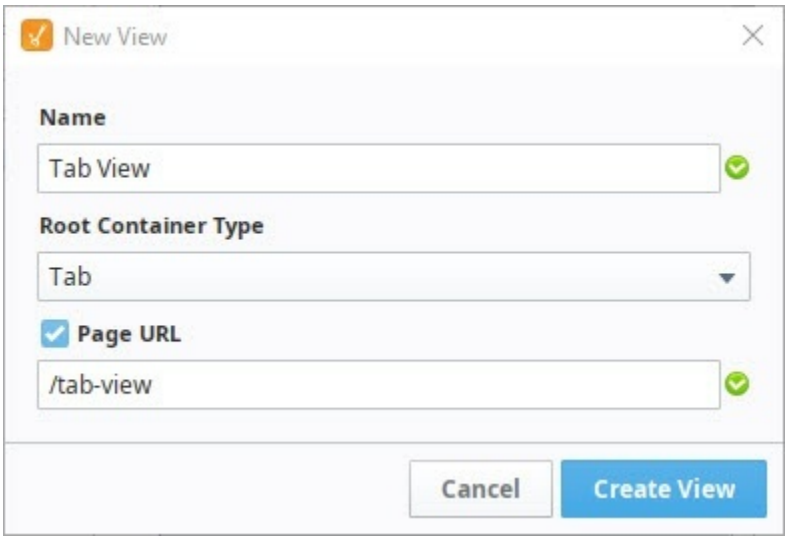
The Tab Layout is a way to allow your users to swap between several views easily. However, this is not a navigation strategy that works for anything other than a very small project. This navigation strategy is intrinsically limited. You cannot use a tab layout to navigate to a new page, only to cycle between different views on the *same* page. As such, the page URL will never change, and the forward and back buttons on your browser will not navigate through the tabs.

For complex pages, this type of layout can work very well. When we talk about using tabs to organize a page, we are talking about a single page that has a lot of information on it. A good example is an customer details screen. You may need to know all about your own customers, location, contacts, order information, and payment. If each of these items is a new tab on a single page, you can show much more information in a smaller space. If this sort of tab structure suits your page's organizational needs, the tab layout can integrate that structure without complicated scripting or bindings.

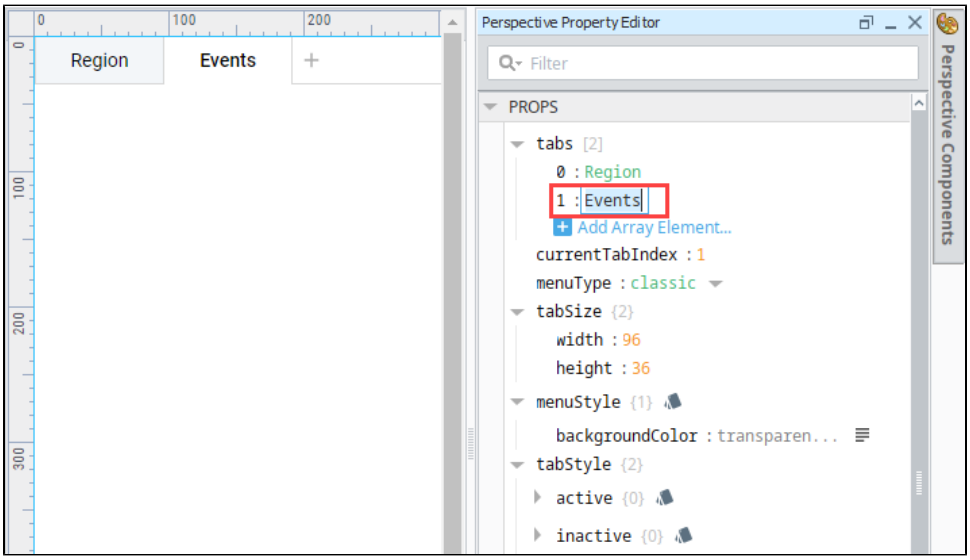
Nested Tabs

If you're after a nested tab structure, all this requires is nesting another tab container inside of each of the tabs in a parent container:

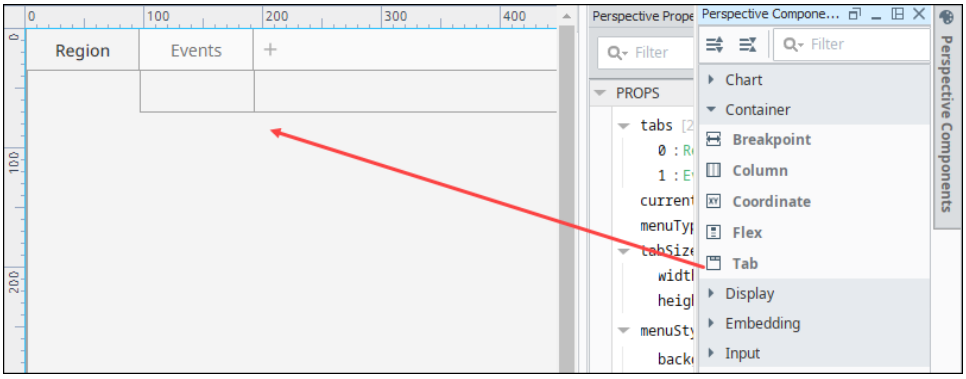
1. Create a new view with a **Tab** container type, and create a **Page URL**.



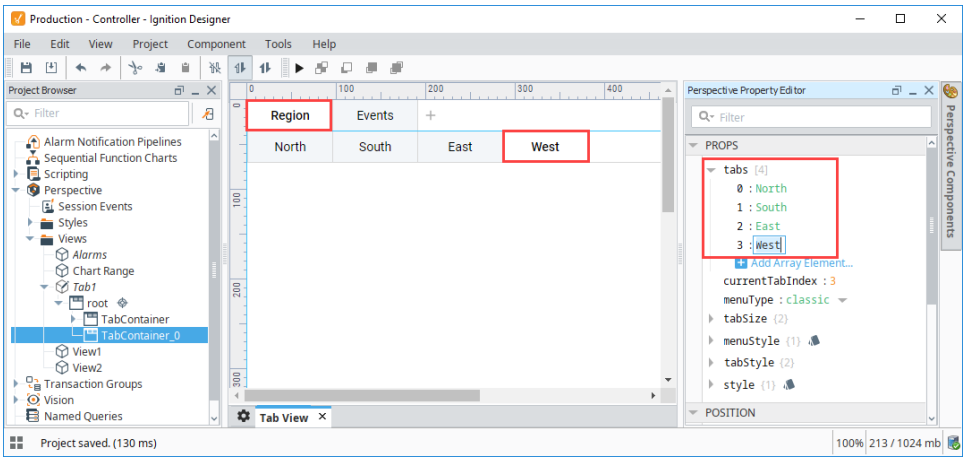
2. Next we added headers to the tabs.
    a. Select the **root** container.
    b. In the Property editor, select the first element in the **tabs** array. Enter the name "Region".
    c. In the second element, enter the name "Events".
    You'll see the names appear on the tabs as you enter them in the Property Editor.



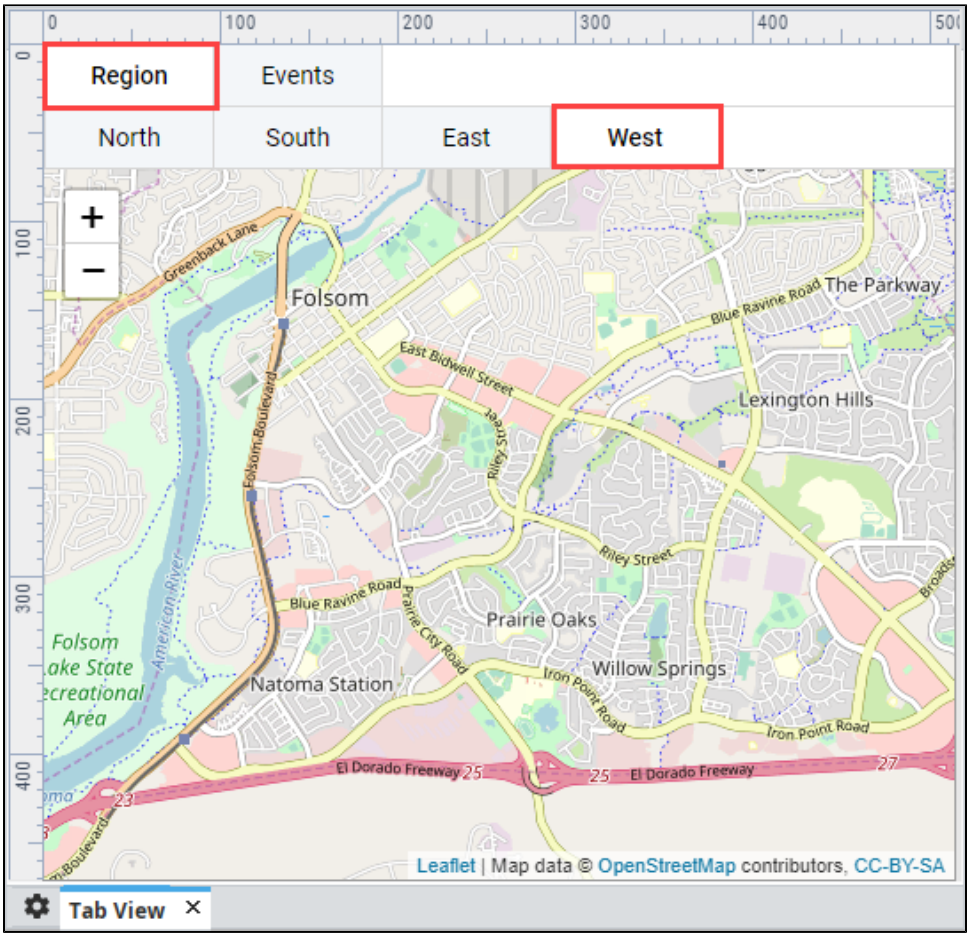
3. Next we'll configure some nested tabs under the Region tab. Double click to deep select the Tab Container component for the nested tabs.
4. From the Perspective component palette, drag a Tab component onto the View.



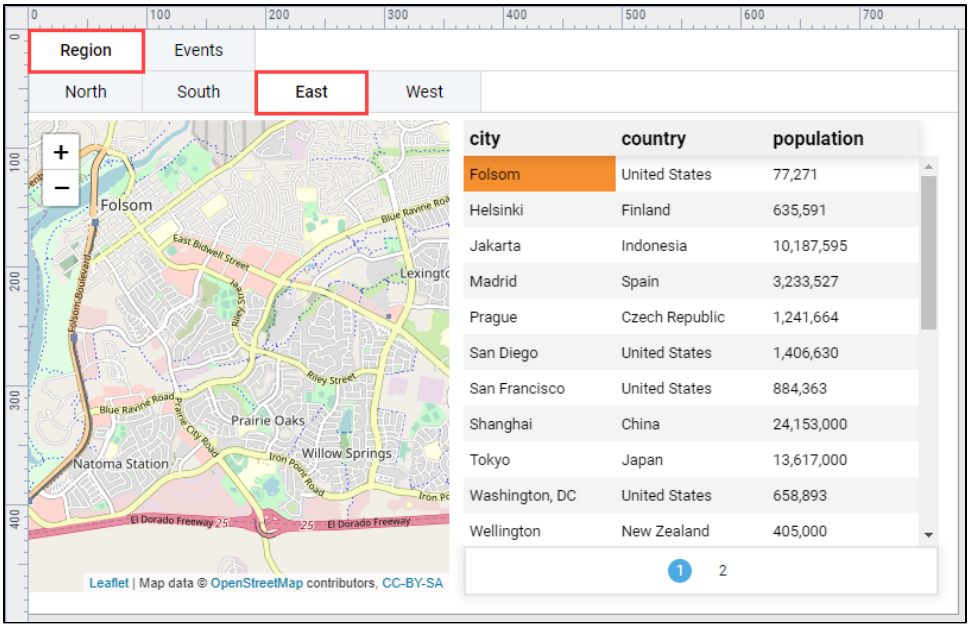
5. In the Property Editor, click the **Add** icon twice to add two more tabs to the component.
6. Name the tabs North, South, East and West.



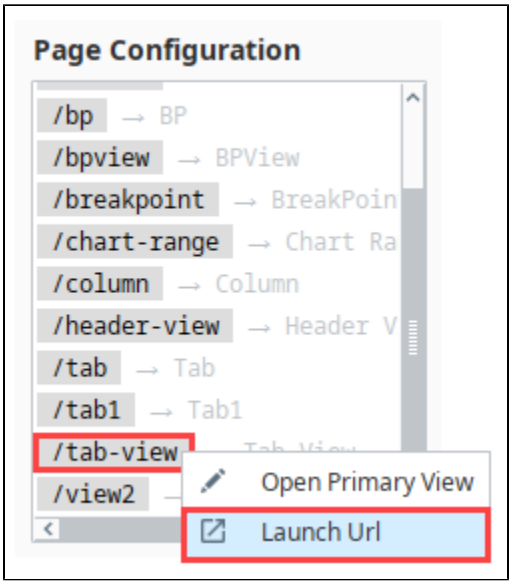
7. Select the tab you want to add a component to (i.e., West).
8. Drag a component onto the tab. In this example, we added a Map component on the tab labeled West. When you drag a component into a tab, the component fills up the entire space. If you want to add multiple components on a sub-container/nested tab, go to Step 10.
9. The example now shows the Map component when the West tab is selected.



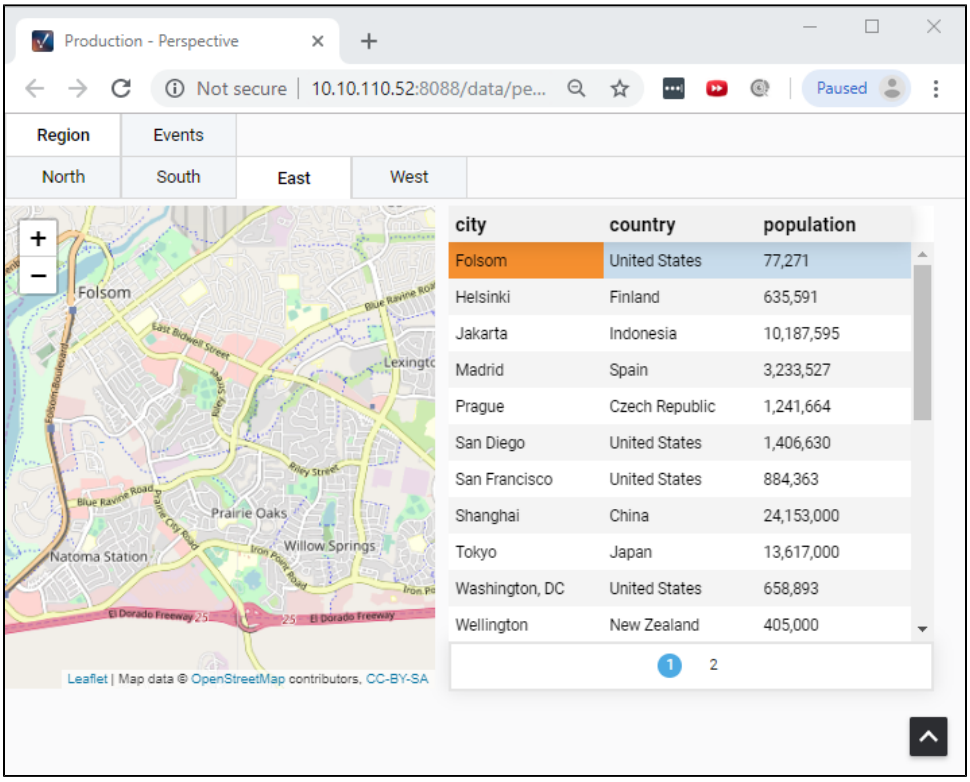
10. To add multiple components on a sub-container, let's use a different tab (i.e., East) from our example above.
11. Double click to deep select the Tab component (i.e., East) and drag a **Coordinate container** from the component palette into the sub-container.
12. Now you can drag multiple components into the container. This example drags in a **Map** and **Table** components. You can resize and organize the components within the tab. Here's what the components look like within the tab.



13. Now, lets open it up in a Perspective Session by clicking on the **Page Configuration** icon. Right click on your Page URL for your Tab View to launch a Perspective Session.



14. Here's what your Tab container layout will look like in a Perspective Session.



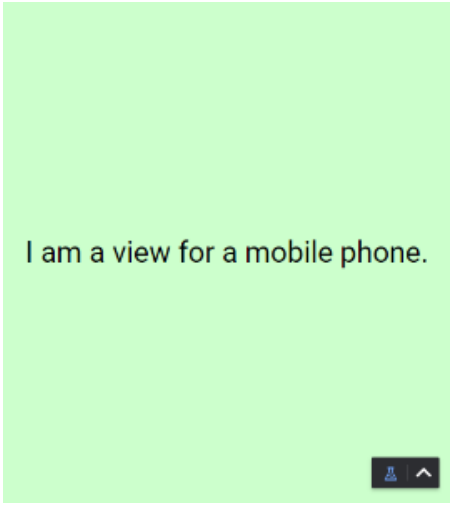
Related Topics ...

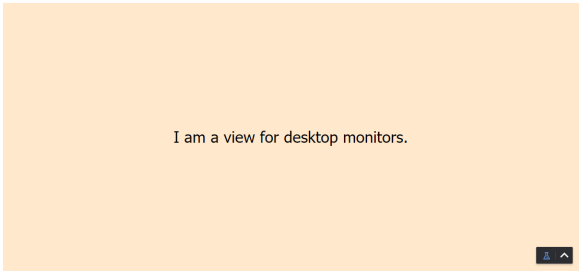
- Pages in Perspective
- Perspective Tab Container

Breakpoint Containers

The Breakpoint container consists of a single *breakpoint*, with two child views. In other words, using a Breakpoint container offers you a layout with the opportunity to create two different views that are shown at two distinct ranges of layout widths.

The breakpoint is configured by selecting the **Small** and **Large** breakpoints at the top of the Perspective Property Editor, then dragging in a subview, container, or component. However, little additional structure is provided; each of the two breakpoint configurations expects a single component, container, or view.





On this page

...

- Flexibility without Limits
- Breakpoint Position and Root Properties
  - Position Properties
  - Root Properties
- Configuring a Breakpoint Container

Flexibility without Limits

The breakpoint layout is the most comprehensive choice when you want to develop a project that works for sessions of many shapes and sizes. Unlike column and flex layouts, which offer ways of controlling how a fixed set of components respond to changes in screen size, a breakpoint layout expects two entirely distinct subviews. This makes the breakpoint layout a powerful, but more involved choice of layout.

There are many circumstances that call for this layer of abstraction. Maybe operators using mobile phones need different control options than desktop computers at a coordinating hub. Or more fundamentally, maybe you can't quite pack as much functionality per view into a mobile screen as you can in its desktop counterpart, and need to add a couple more views to your mobile version to bridge the gap.

A breakpoint layout might be a good idea if you want small and large versions of your session to have:

- Differing components
- Differing properties, bindings, or layout behavior
- Differing navigation options

When designing pages for mobile and desktop applications, it's important to consider not only the type of layout, but if your design fits nicely on a page in a mobile session. Some pages may work for a desktop session, but may not work for a mobile device such as expecting a user to enter data into multiple tables on a mobile session that don't fit seamlessly across a page.

Breakpoint Position and Root Properties

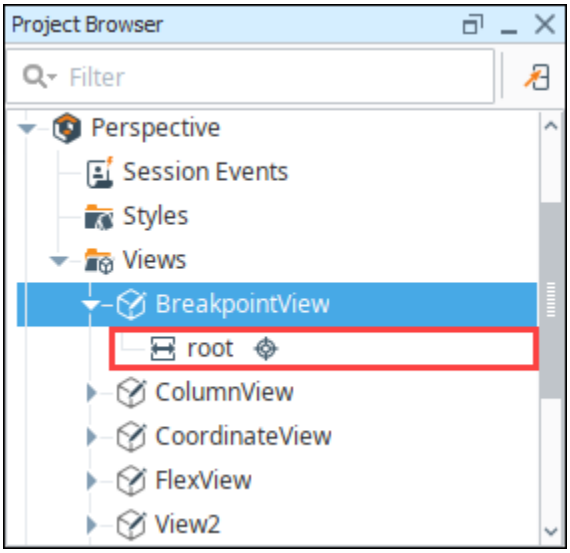
Position Properties

Position properties in a Breakpoint container indicate simply whether the component is in the large version of the container or the small version. For more information, see Perspective - Breakpoint Container.

Position Property	Description	Data Type
size	Indicates small or large.	boolean

Root Properties

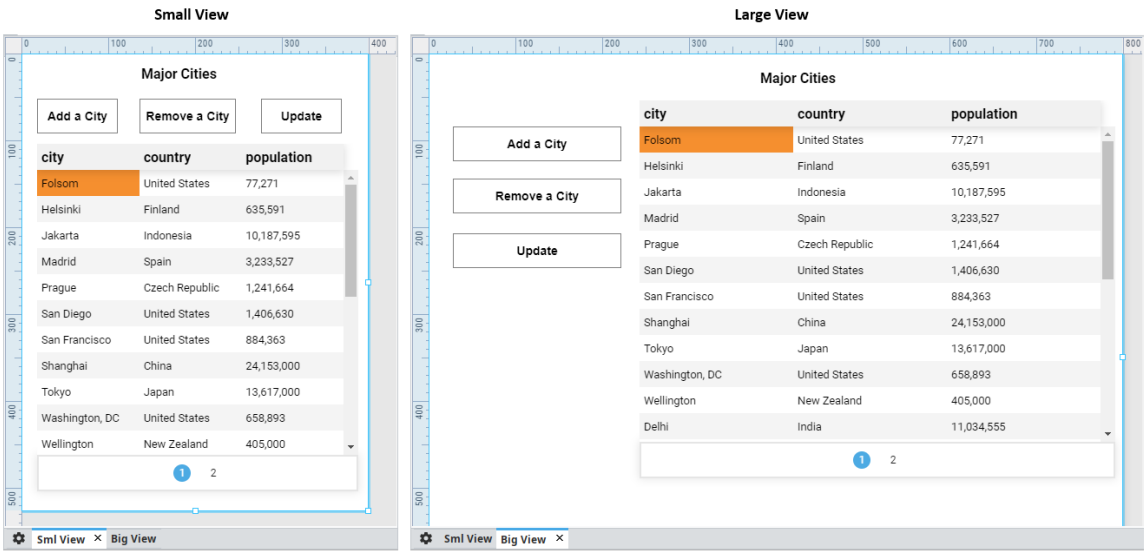
Root properties are accessed by selecting the **root** folder for a Breakpoint View on the Project Browser tree.



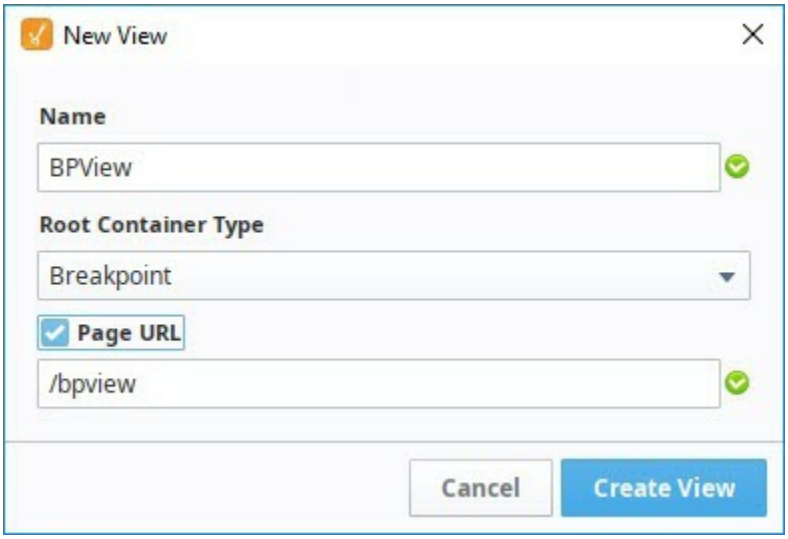
Root Property	Description	Data Type
breakpoint	Width (in pixels) breakpoint declarations for child layouts. When the container is sized below minWidth, child position rules will fall back to the next set breakpoint rules.	value: numeric
style	Sets a style for this view. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a style class.	object

Configuring a Breakpoint Container

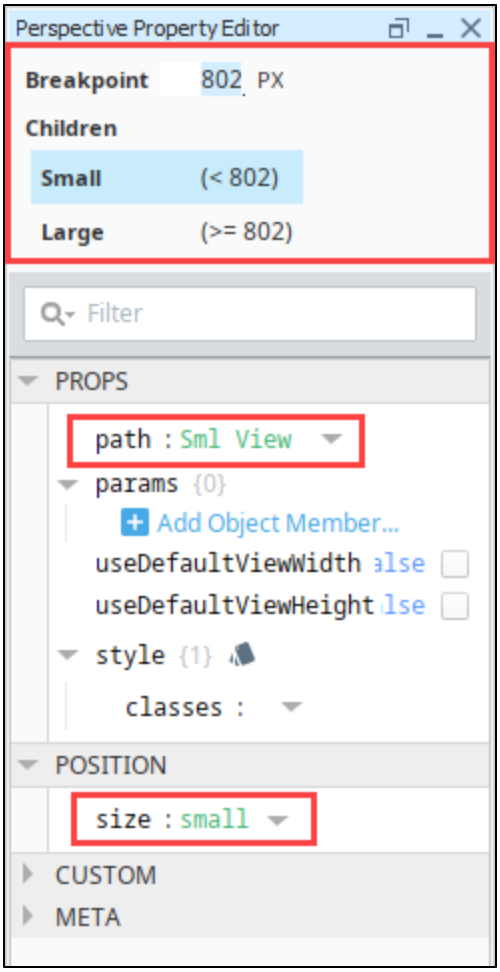
A Breakpoint container can switch between two configurations based on how wide the container is. Here is a simple example of creating a Breakpoint layout using two different views. The components on each view are laid out a little differently because they both represent two different distinct ranges of layout widths: small and large. We are going to use the two views below to add as children to our Breakpoint container. Since you can only add a single component to each child in a Breakpoint container, we'll place each of these views, containing multiple components, in their own Embedded View. This example assumes you have a small view and large view already configured.



1. Create a new view using a Breakpoint container.

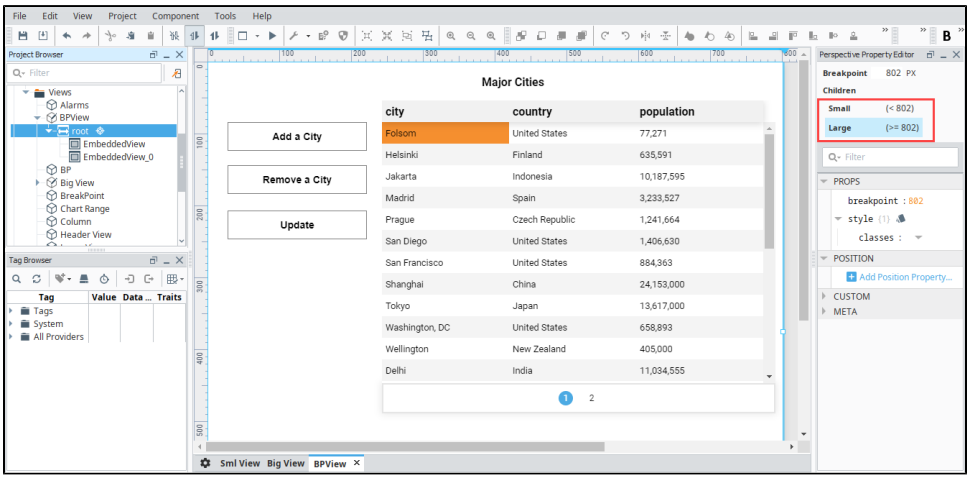


2. Let's configure the Breakpoint container to show the small view when the screen size is narrow, like on your mobile device, and the large view when you're on your desktop.
3. In the Property Editor, the breakpoint is determined by using the Small and Large properties, or the slider at the top of the Designer workspace. In this example, the Breakpoint is set at 800 px. When the Breakpoint container is sized below minimum width, child position rules will fall back to the next set breakpoint rules.
    a. With the Breakpoint container open, click on the **Small** property in the Property Editor.
    b. Drag an Embedded View component onto the Breakpoint container.
    c. Select the **EmbeddedView** in the Project Browser.
    d. Set the **path** property to the name of your small view (i.e., Sml View) from the dropdown.
    e. Set the Position **size** property to small. **Note**: This lets the embedded view component know which child rule it is linked to, Small or Large.



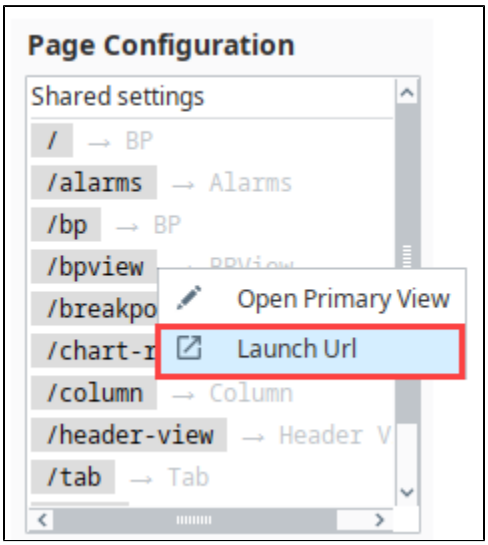
4. Now let's repeat the steps above for the large view.
    a. With the Breakpoint container open, click on the **Large** property in the Property Editor.
    b. Drag an Embedded View component onto the Breakpoint container.
    c. Select the first **EmbeddedView_0** in the Project Browser.
    d. Set the **path** property to the name of your large view (i,e., Big View) from the dropdown.
    e. Set the Position **size** property to large. **Note**: This lets the embedded view component know which child rule it is linked to, Small or Large.

5. Now, when you toggle the Small and Large properties in the Property Editor, you'll see the window switch between the two different Small and Large views.

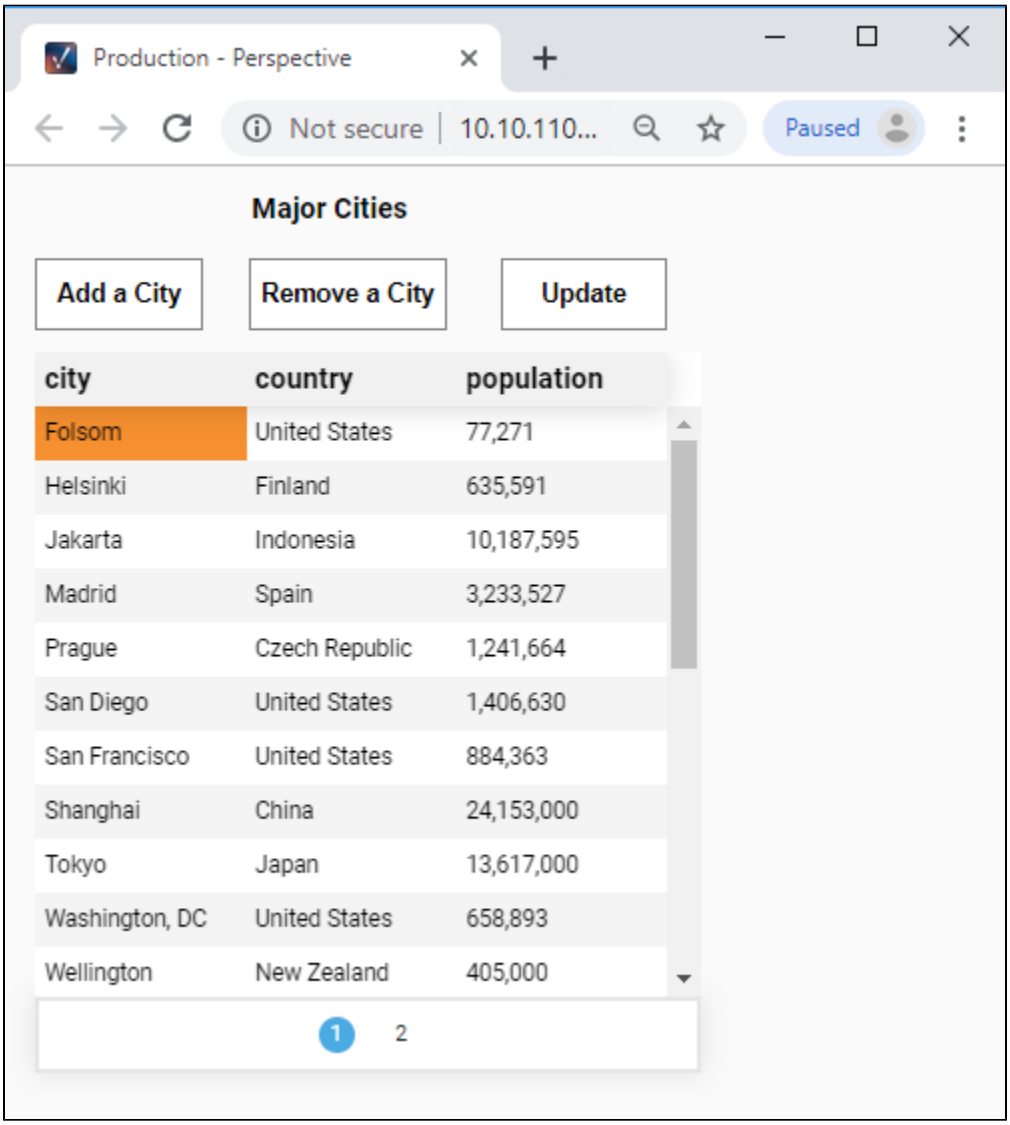


6. Now, let's open a Perspective Session by clicking on the **Page** Configuration icon at the bottom of the Designer window to test your Breakpoint layout.

7. Right click on your Breakpoint container (i.e., bpview), and click on **Launch Url**.



8. With your Perspective Session open, stretch and shrink your browser window. You'll see the views change depending on the width of your browser.



Related Topics ...

- Pages in Perspective
- Perspective Breakpoint Container

Flex Containers

The **flex** layout is a powerful layout, with many configuration options, based on the CSS Flexbox layout. The flex layout consists of a single series of adjacent components or containers, configured as either a column or a row. To this end, the layout has five properties:

- **direction** controls whether the flex layout appears as a row or a column.
- **wrap** controls what happens when the components are too big for the container. For instance, if the **direction** property is set to **row**, but the components in the row are too wide to fit in the flex container, this property controls whether a new row should be created, or scrollbars should be used.
- **justify** controls where extra space is located when the components are not large enough to fill the container along the main axis. i.e. the space at the end or a row, or on the bottom of a column. If the components do not fully fill the space, the remaining gap can be placed at the beginning, the end, evenly on both sides, or evenly distributed between the components.
- **alignItems** controls where extra space is located when the components are not large enough to fill the container along the alternative axis, i.e. the space below a row, or on the right of a column. If the components do not fully fill a row, the remaining gap can be placed above, below, evenly on both sides, or evenly distributed between the components. For a column it is left, right, both, or evenly distributed.
- **alignContent** controls what happens to items that have been wrapped.

In addition, each component in a flex layout contains three properties:

- **basis** controls the default size of a component along the flex container's **direction**. You can enter the value in pixels (e.g., **75px**), as a percentage of the total length of the container (e.g., **50%**), or you can use **auto**. All components configured to **auto** will equally share the available space in the container.
- **grow** and **shrink** control the way that a component responds to changes in the flex container's width (for rows) or height (for columns). **grow** controls what happens when additional space is available, and **shrink** controls what happens when the component does not have enough space to fulfill its basis.

Grow and **shrink** are interpreted relative to other components in the Flex container, and are best understood through some simple principles:

- A **grow** value of 0 will prevent the component from stretching beyond its **basis**. A **shrink** v alue of 0 will prevent the component from shrinking beneath its **basis**.
- If all components in a row have the same (nonzero) **grow** value, they will all grow equally to accommodate more space. If they have the same **shrink** value, they will all shrink evenly.
- Components with higher **grow** and **shrink** values will grow and shrink more. Specifically, a grow or shrink value is evaluated relative to the sum of all grow and shrink values in the container.

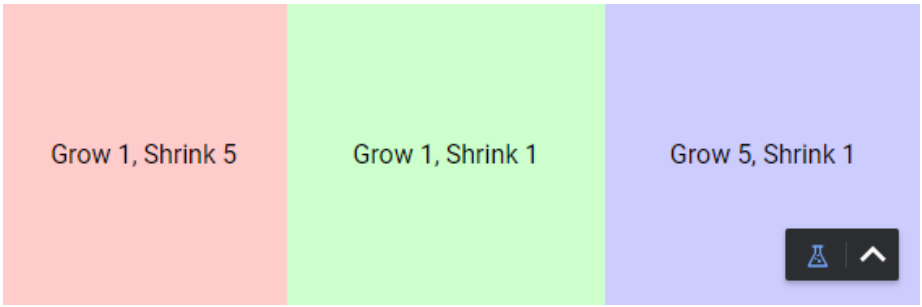
On this page

...

- Growing and Shrinking
  - Static Widths
  - Even Scaling
  - Position Properties
  - Root Properties
- Configuring a Flex Layout
  - Creating a Main View
  - Creating a Header View

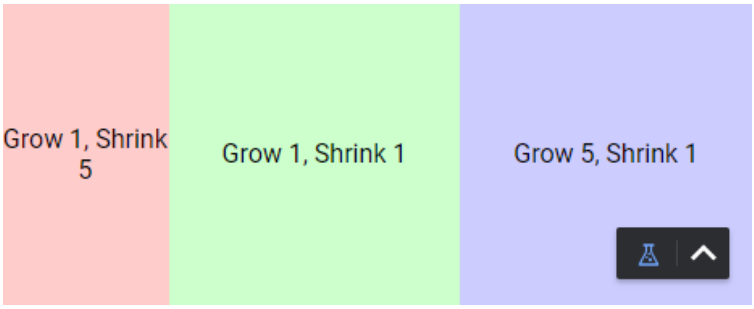
Growing and Shrinking

The flex container's **grow** and **shrink** properties allow a great deal of control over how different components expand and shrink. To demonstrate, here are three components nested inside a flex container, with a direction of **row**:



All three components have the same **basis**, so at this session width, they all have the same size. However, the blue component has a much larger **grow** value than the other two components. So, when we increase the length of the flex container along its **direction** property:

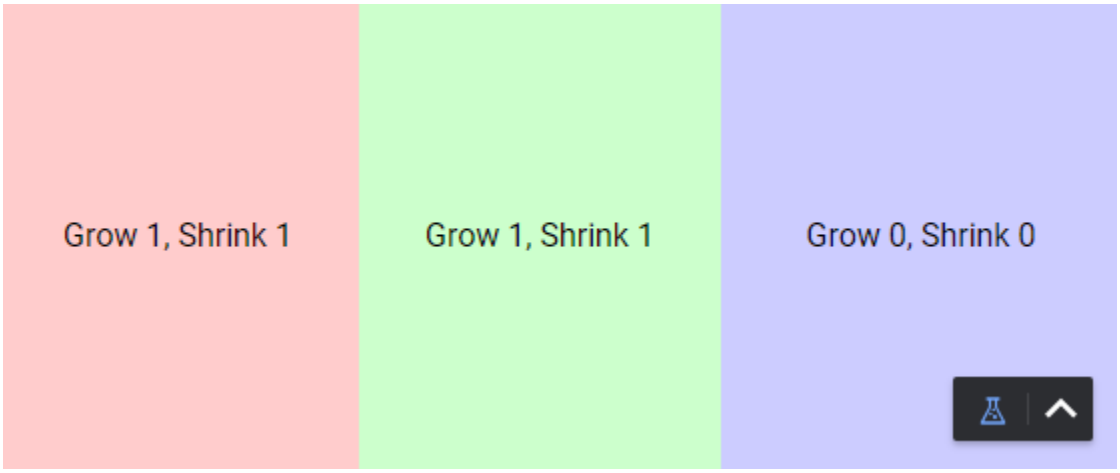
The blue component does most of the stretching. Specifically, since the sum of all grow properties is 1 + 1 + 5 = 7, and the blue component has a grow property of 5, for every 7 pixels the flex container grows, the blue component will grow by 5. Now let's try shrinking the container, noting that the red component has a shrink value of 5:



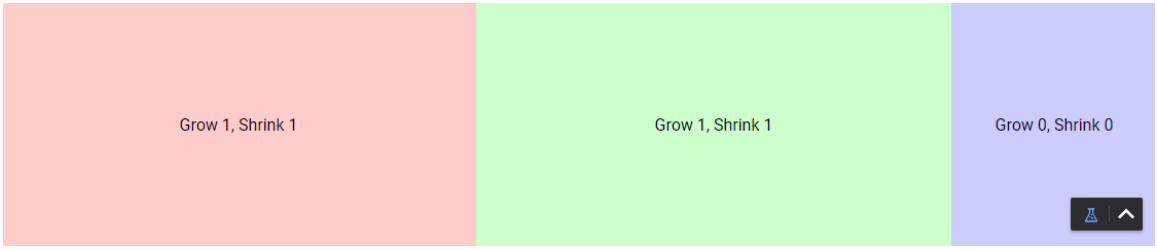
As you can see, a larger shrink value will make the container shrink more. For every 7 pixels the flex container loses, the red component will lose 5.

Static Widths

Now let's try the same example again, but with some different **grow** and **shrink** values:

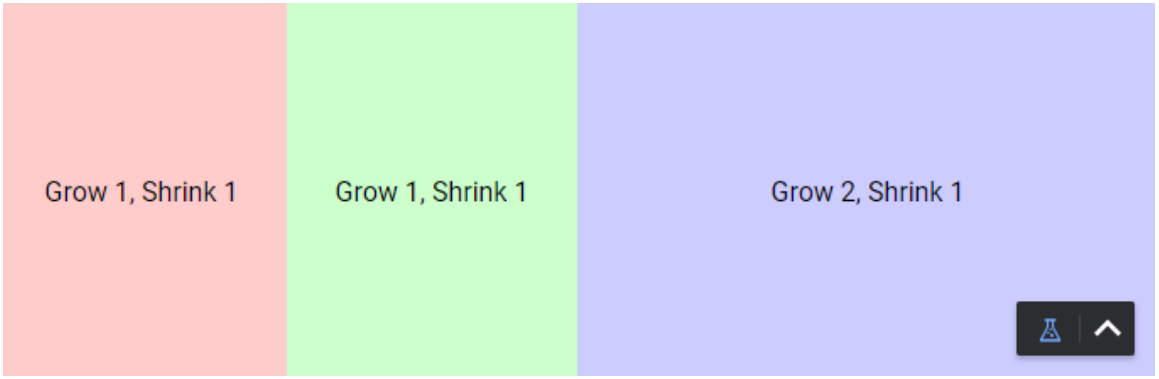


We've given the red and green components identical values, so they should stretch and shrink at the same rate. Meanwhile, the blue component's **grow** and **shrink** values are both 0, so when we make the flex container wider, it stays the same size:

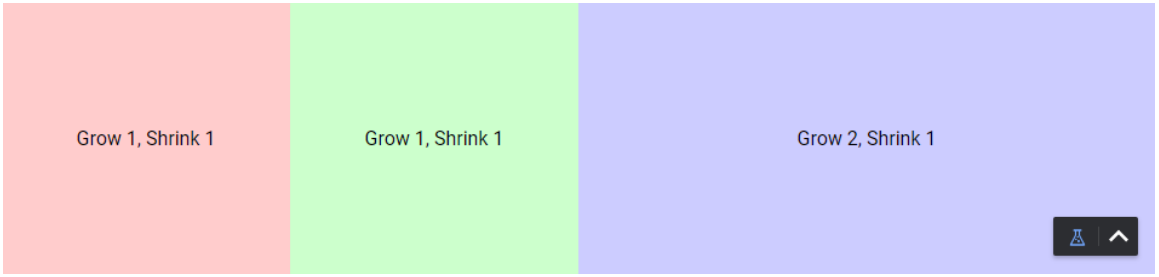


Even Scaling

Now let's say we start off with a blue component twice the **basis** of the others:

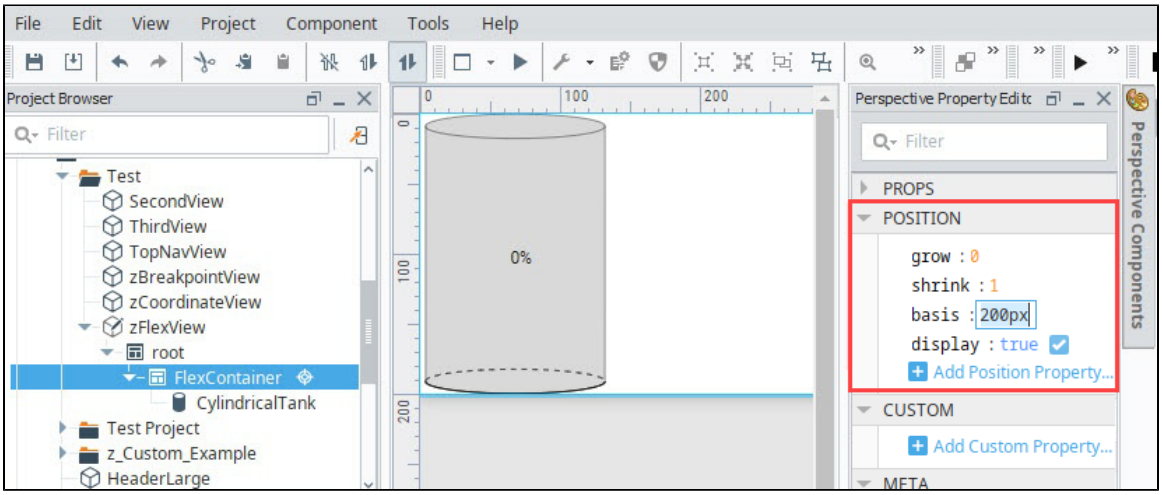


In order to maintain this ratio as the width of the container increases, it must have a **grow** value twice that of the others:



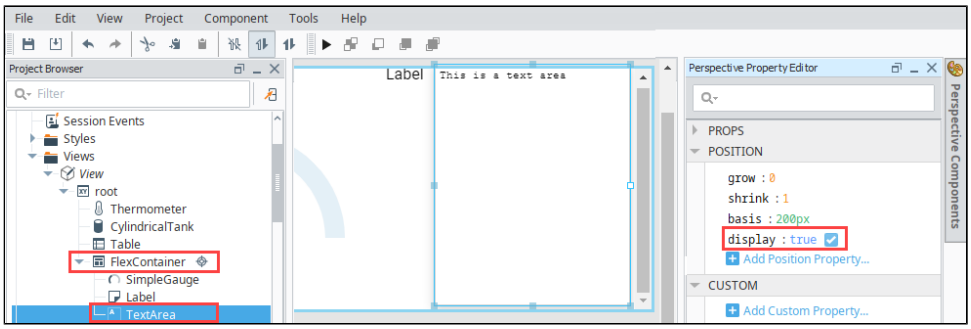
Position Properties

Position properties for a Flex container indicate whether the container is allowed to grow or shrink in relation to its siblings. For more information, see Perspective - Flex Container.



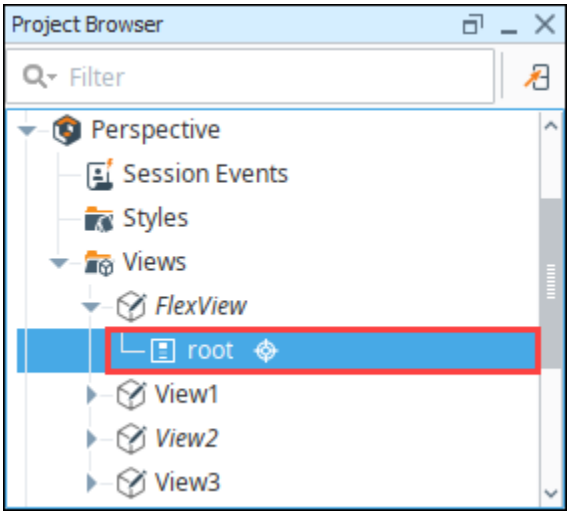
Position Property	Description
grow	Ability to grow in 'direction' dimension as needed, relative to siblings. If space is available and grow is not zero, it may stretch, depe that two components with the same grow value will grow at the same rate.
shrink	Ability to shrink in 'direction' dimension as needed, relative to siblings. If space is available and grow is not zero, it may stretch, dep that two components with the same shrink value will shrink at the same rate.
basis	Space filled by component by default, before 'grow', 'shrink' and sibling considerations are evaluated. This is the component's base base height when the direction property is set to 'column'.

display	Determines if the component will be displayed in the container or not. Components that are not displayed won't just be invisible, but components to fit.



Root Properties

Root properties are accessed by selecting the **root** folder for a Perspective View on the Project Browser tree.



Root Property	Description	Data Type
direction	Direction of the child layout. Options are row, row-reverse, column, column-reverse	value: string dropdown
wrap	Whether the container allows children to wrap to the next line if space has run out. Options are wrap, no-wrap, wrap-reverse.	value: string dropdown
justify	Adjusts placement of children along the main axis hen there is extra space, which may be used to fill areas before, after, or in-between. Options are flex-start, flex-end, center, space-between, space-around, or space evenly.	value: string dropdown
alignItems	Adjusts placement of children along the cross axis when there is extra space. Options are flex-start, flex-end, center, baseline, or stretch.	value: string dropdown
alignContent	Adjusts alignment of wrapped content when there is free space in the cross axis. Options are flex-start, flex-end, center, baseline, or stretch.	value: string dropdown
style	Sets a style for this view. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a style class.	object

Configuring a Flex Layout

The Flex container is ideal when designing mobile responsive applications because it lets you set dynamic sizes to accommodate various screen sizes. By using the Flex container properties, it allows components to grow and shrink dynamically relative to each other. In the following examples, we'll use the Flex Container to create two types of views similar to the descriptions above:

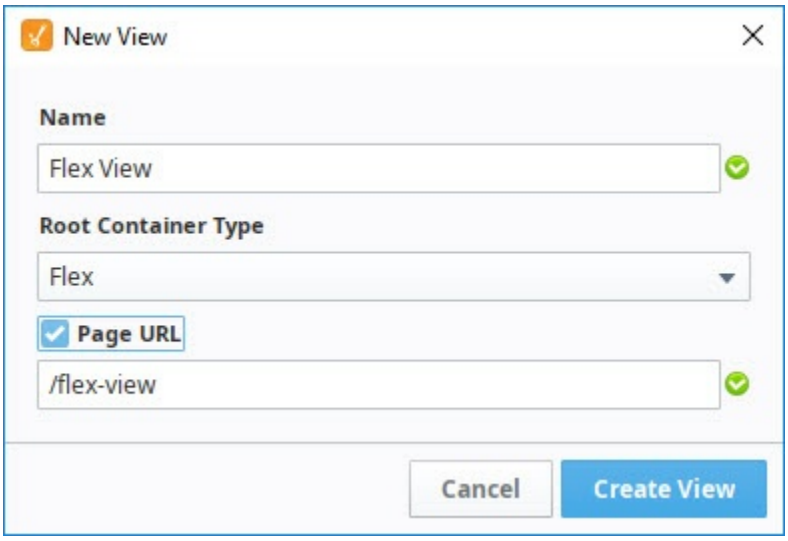
- a main view containing three components where only the bottom two stretch
- a header view that has components pushed to both the left and right of the view with empty space between them

> **Grow and Shrink Properties**
>
> When the grow and shrink properties are set to non-zero numbers, they are allowed to change size. If they are set to zero, they are not allowed to grow or shrink, they will always match their basis value.

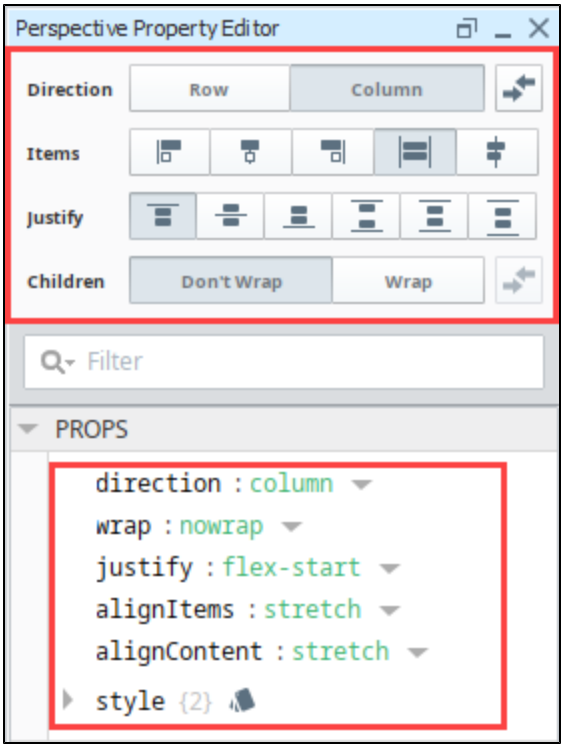
Creating a Main View

In this first example, we are going to configuring three Label components using a Flex container to demonstrate how to freeze the size of the first label and allow the second and third Labels to stretch to fill the remaining space. We can do this by setting the **grow** and **shrink** Position properties to a non zero number. The first Label component will not be allowed to grow or shrink because both properties are set to 0.

1. Create a new view using the Flex container.



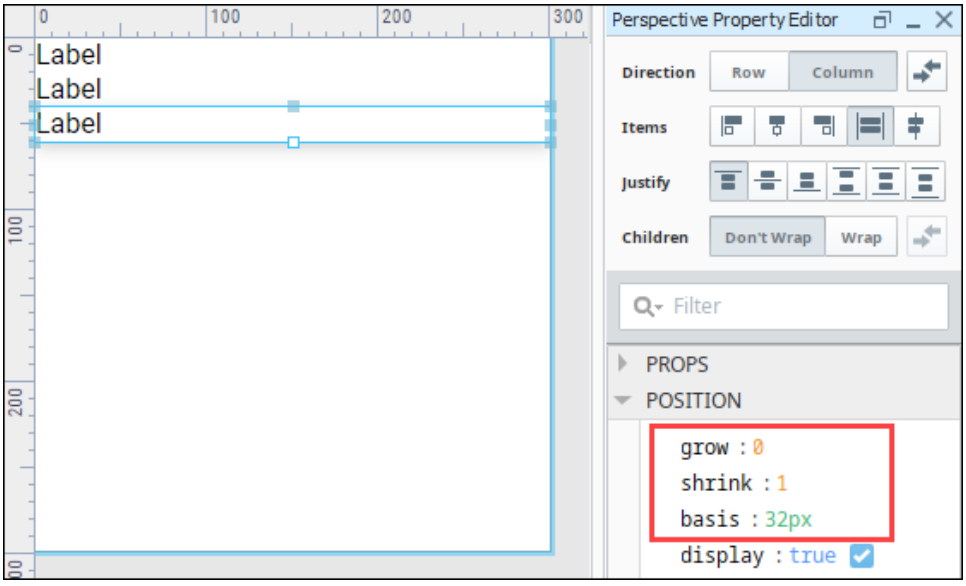
2. The new view will open in the Designer. The Flex view will have some preset properties so you might want to look at the GUI and the PROPS in the Property Editor. These can be modified based on your design requirements.



3. Using the square handles, set your view to **300px** by **300px**.

4. With the root selected, drag three **Label** components onto the view from the Component Palette. Since the direction property is set to column, all three labels are placed in a column starting at the top of the view. The default **Position** property settings for all three Labels are:

    a. **grow**: 0 - the component can not grow
    b. **shrink**: 1 - allows the component to shrink
    c. **basis**: 32px - is the ideal size for the component



5. We want the first label to be a static size and never change. Select the first Label and set the following properties:
    a. grow: **0**
    b. shrink: **0**
    c. basis: **100px**

6. Modify the component style for the Label component any way you want. We used the following settings:
    a. Click the **Modify Style** icon next to the **style** property.
    b. Expand the **Background** category and set the **Background color** to **Orange**.

c. Close the style popup and set the text property of the Label component to **Orange Label**.

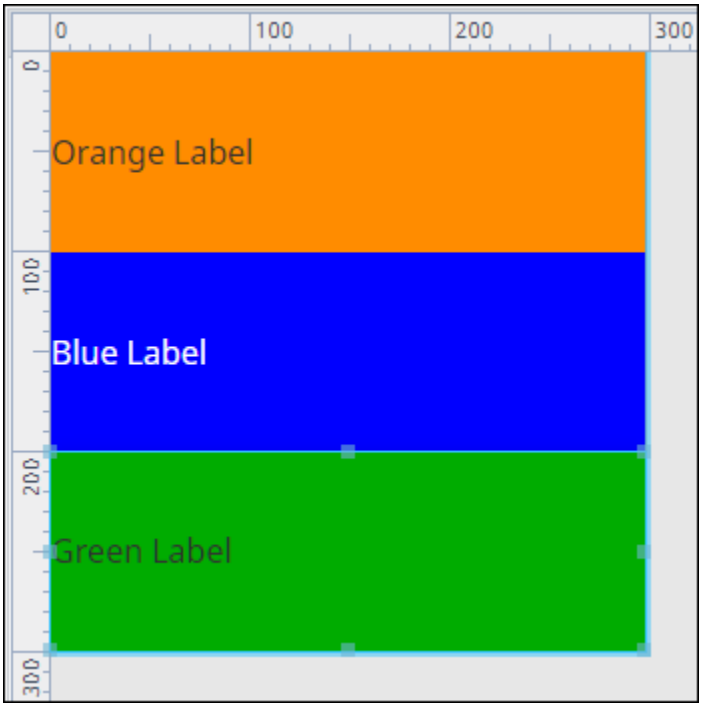
7. Select the second Label component and set the following properties:
    a. grow: **1**
    b. shrink: **1**
    c. basis: **100px**

8. Modify the component style for the Label component. We used the following settings:
    a. Click the **Modify Style** icon next to the **style** property.
    b. Expand the **Text** category and set the **color** to **White**
    c. Expand the **Background** category and set the **Backgroundcolor** to **Blue.**
    d. Close the style popup and set the text property of the Label component to **Blue Label**.

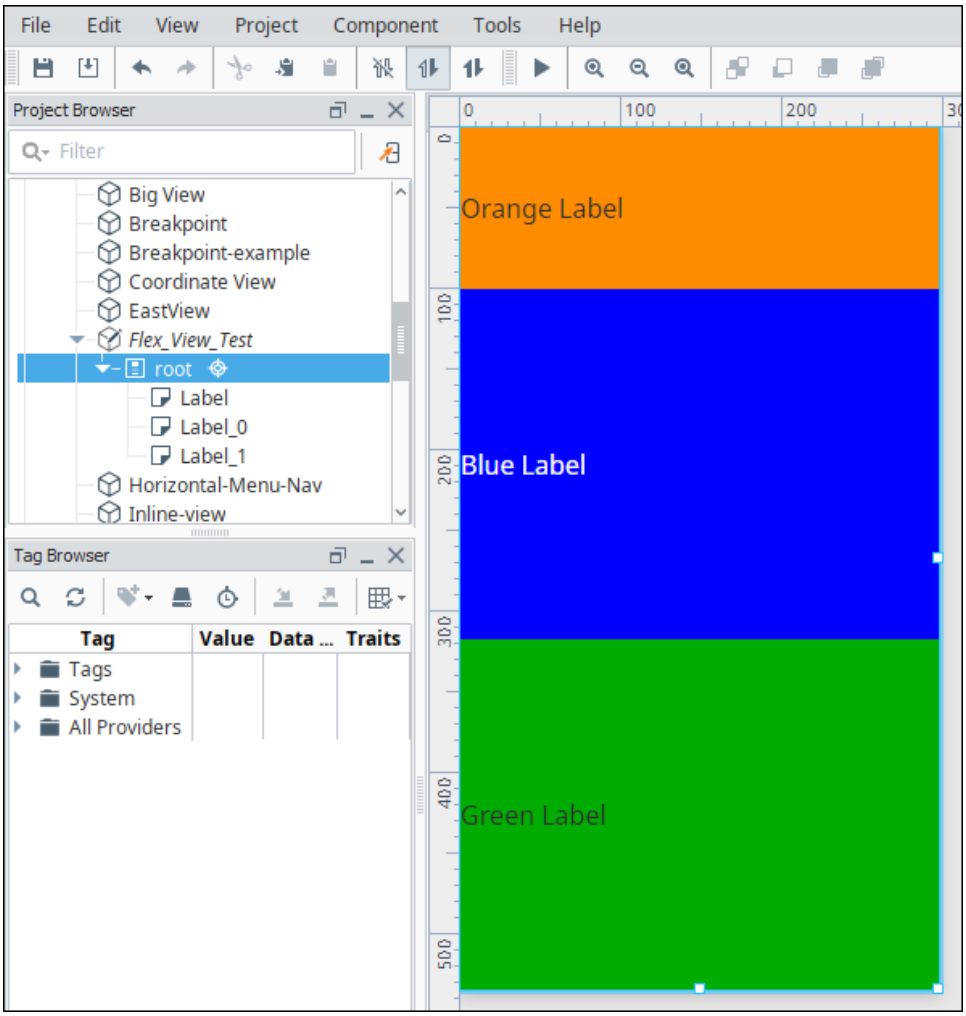
9. Select the third Label and set the following properties:
    a. grow: **1**
    b. shrink: **1**
    c. basis: **100px**

10. Modify the component style for the Label component. We used the following settings:
    a. Click the **Modify Style** icon next to the **style** property.
    b. Expand the **Background** category and set the **Background color** to **Green.**
    c. Close the style popup and set the text property of the Label component to **Green Label**.

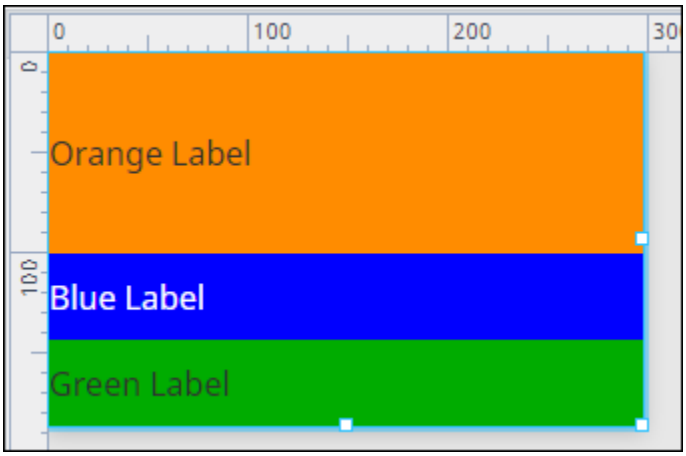
    Once all the Label **Position** properties are set, your view should look like the image below.



11. Now, let's resize the components in the same direction as the container. Select the **root** container in the Project Browser. Using the square handle at the bottom of the view, drag the view down the length of your workspace. You'll notice that the Orange Label component didn't grow, and because the Blue and Green Label components were give the same grow values, they were stretched at the same rate.



12. Select the **root** container in the Property Editor, and grab the square handle at the bottom and shrink your view. As the container gets smaller, the bottom two Labels will start to shrink. As you can see in the image below, because the Orange Label shrink property is set to zero, it didn't shrink.

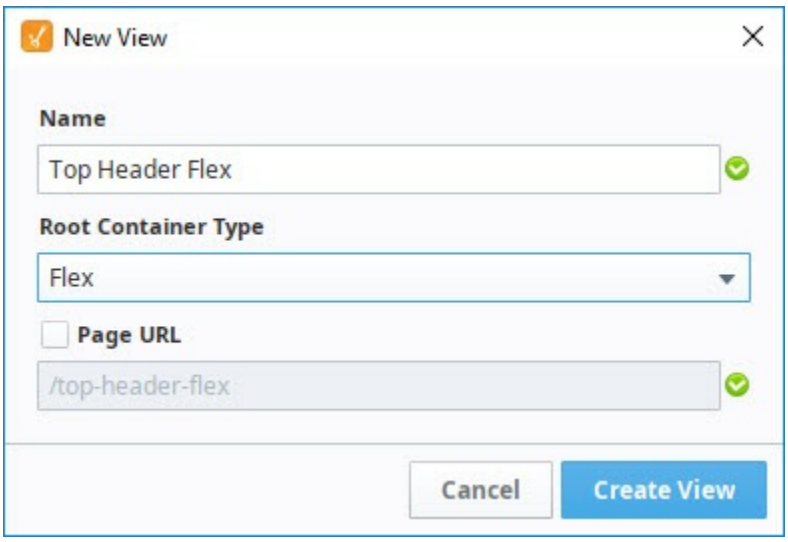


You can use this example to modify the different Properties and Position properties to see how the behavior of the Label components change. If you don't want them to shrink past their basis, change the shrink property to zero.

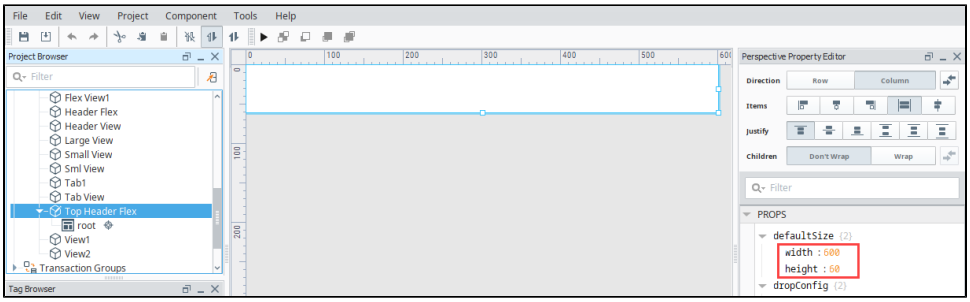
Creating a Header View

In this example, we'll use the Flex Container to create a header view that contains several components pushed to the far left and right of the header. Then we'll show the header view on top of the Flex View (from the previous example) in a Perspective Session.

1. Create a new view using a Flex container.

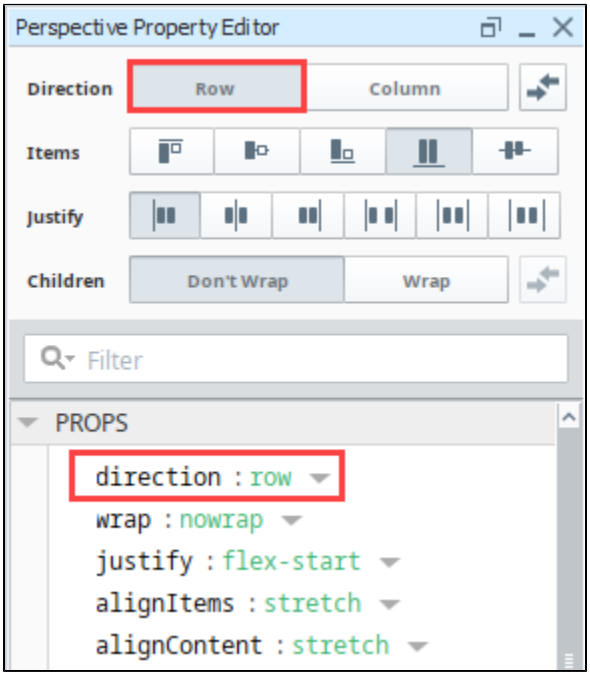


2. The new view will automatically open in the Designer. With the view selected, set the following size properties so it is wide and short:
    a. width: **600**
    b. height: **60**

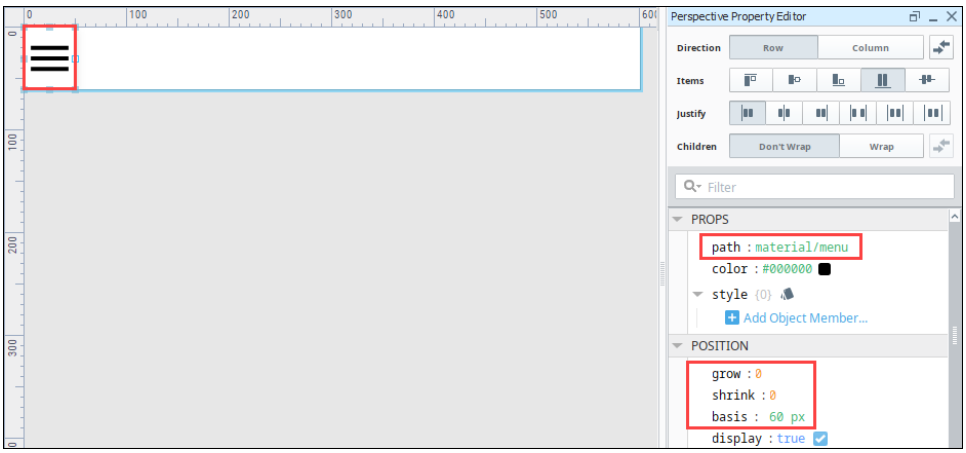


3. Select the **root** container in the Project Browser.

4. In the Property Editor, set the **direction** property to **row**. Notice how the GUI at the top of the Property Editor also changed. You can use either the **GUI** or the **PROPS** to modify the containers properties.



5. From the Component Palette, drag an **Icon** component and place it on the left side of the view.
6. With the Icon selected, set the following **Position** properties:
    a. iconPath: **material/menu**
    b. grow: **0**
    c. shrink: **0**
    d. basis: **60px**



7. Next, add a Label component and set the following **Position** properties:
    a. text: **My First Project**
    b. grow: **1**
    c. shrink: **1**
    d. basis: **150px**

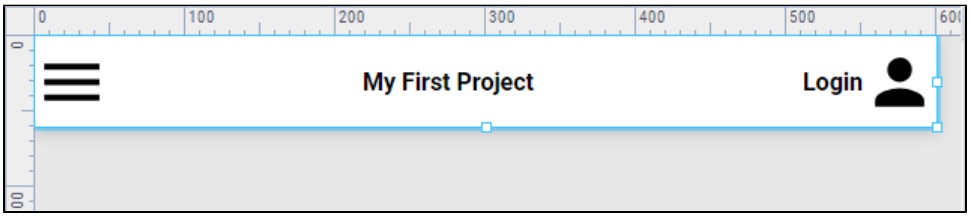
    While we still have the **Label** selected, let's modify the text style:
    a. Click the **Modify Styles** icon next to the **style** property and expand the **Text** category:
    b. Font Weight: **Bold**
    c. Text align: **Center**

8. Add another **Label**, and set the following **Position** properties:
    a. text: **Login**
    b. grow: **0**
    c. shrink: **0**
    d. basis: **50x**

    Let's modify the text style for the Label component.
    a. Click the **Modify Styles** icon next to the **style** property and expand the **Text** category:
    b. Font Weight: **Bold**
    c. Text align: **Right**

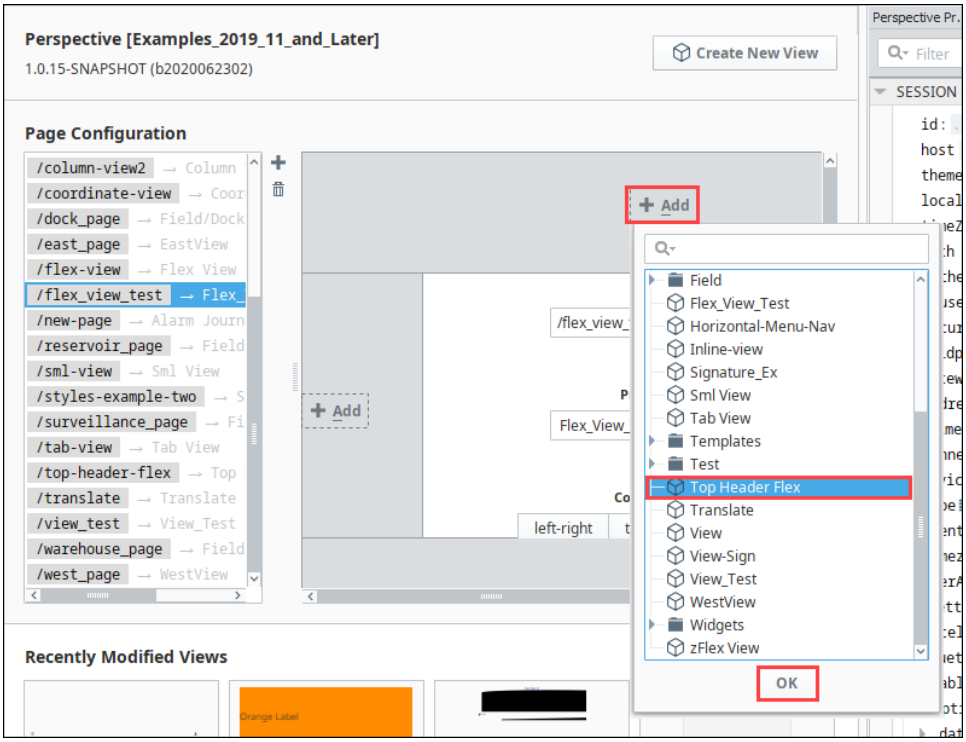
9. Add another **Icon** component to the container on the right and set the following **Position** properties:
    a. iconPath: **material/person**
    b. grow: **0**
    c. shrink: **0**
    d. basis: **60px**

    Here's what your header view should like in the Designer after adding all the components and setting each component's property values.

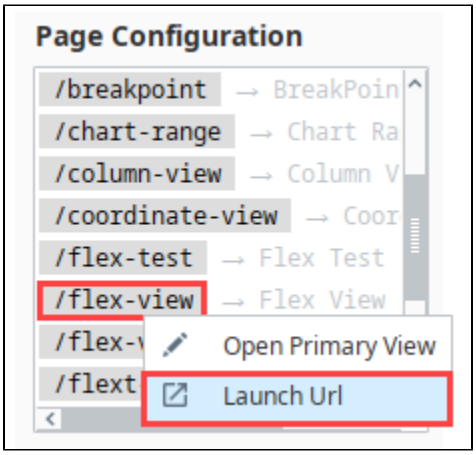


10. **Save** your project.
11. Now let's show the header view on top of the Label page in a Perspective Session. Click on the **Page Configuration** icon at the bottom of the Designer window.
12. Select the /flex-view page.

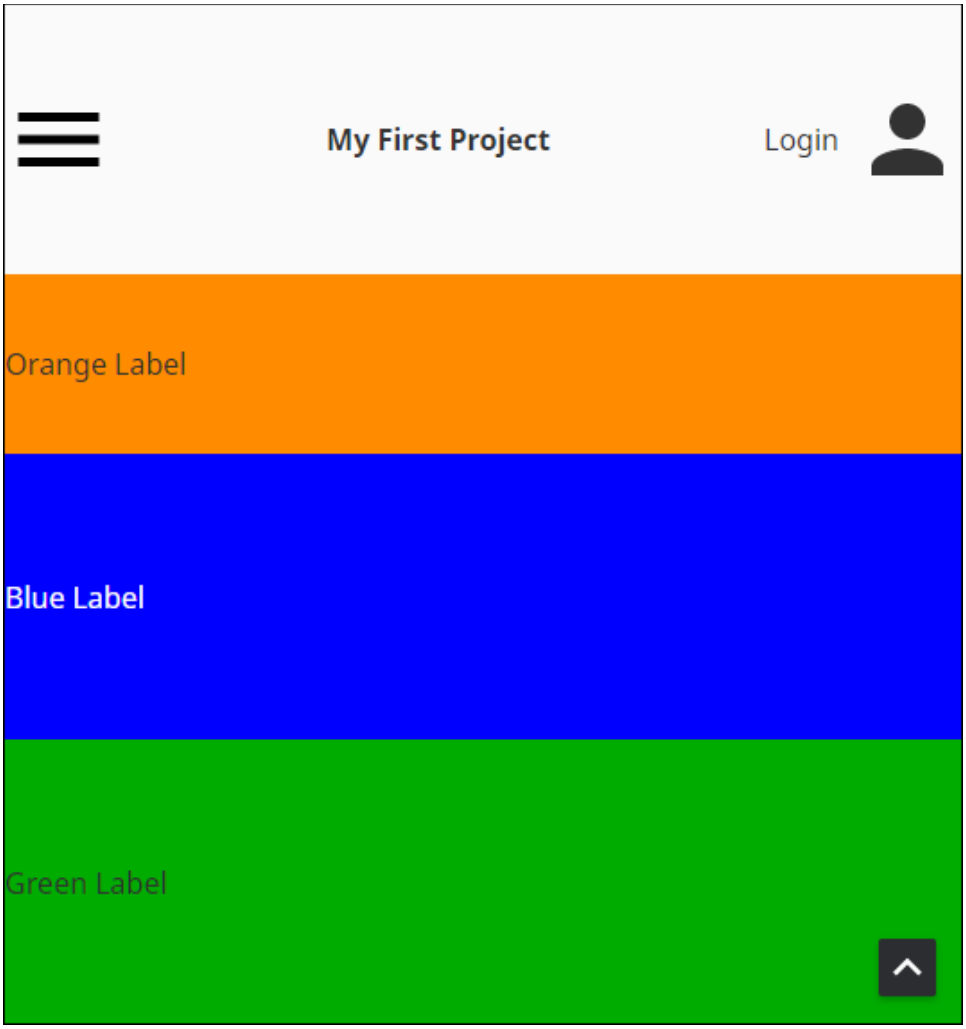
13. In the Layout area, click on the **Add** for the top header section. Select the **Top Header Flex** view and click **OK**. This will make the Top Header Flex View be displayed at the top of the Flex View we created.



14. Right click on the Flex view page and select **Launch Url**.



15. With your Perspective Session open, stretch and shrink your browser window horizontally and you'll see the Header grow and shrink with the icons locked in the corners.



You can modify the different properties to change the behavior for the Label components on your browser.

Related Topics ...

- Perspective - Flex Container
- Pages in Perspective
- Styles

Test Your Responsive Design Using Chrome's Developer Tools

When designing for mobile-responsive applications, designs must work everywhere, on any device from a smartphone to a desktop with full-sized monitors, in a optimized format using one design. Websites not only have to look great on many different screen sizes, but they are also expected to perform well. You can test your responsive designs using Chrome's Developer Tools by integrating an emulation feature called Device Mode. Device Mode can emulate a mobile device environment to test a website's responsiveness in different devices. It can also change the resolution of your page to reflect the size of screens from different manufacturer's devices.

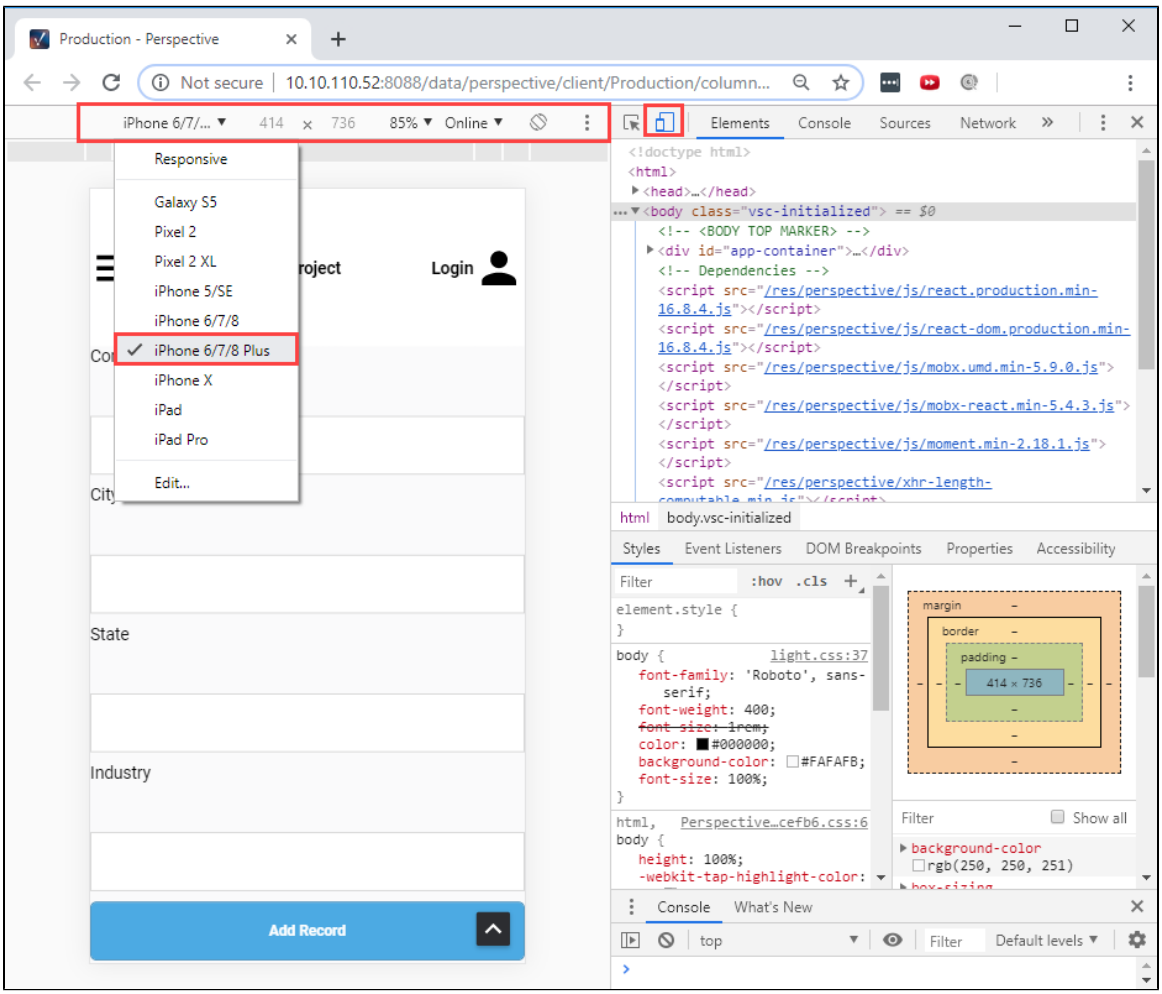
On this page

...

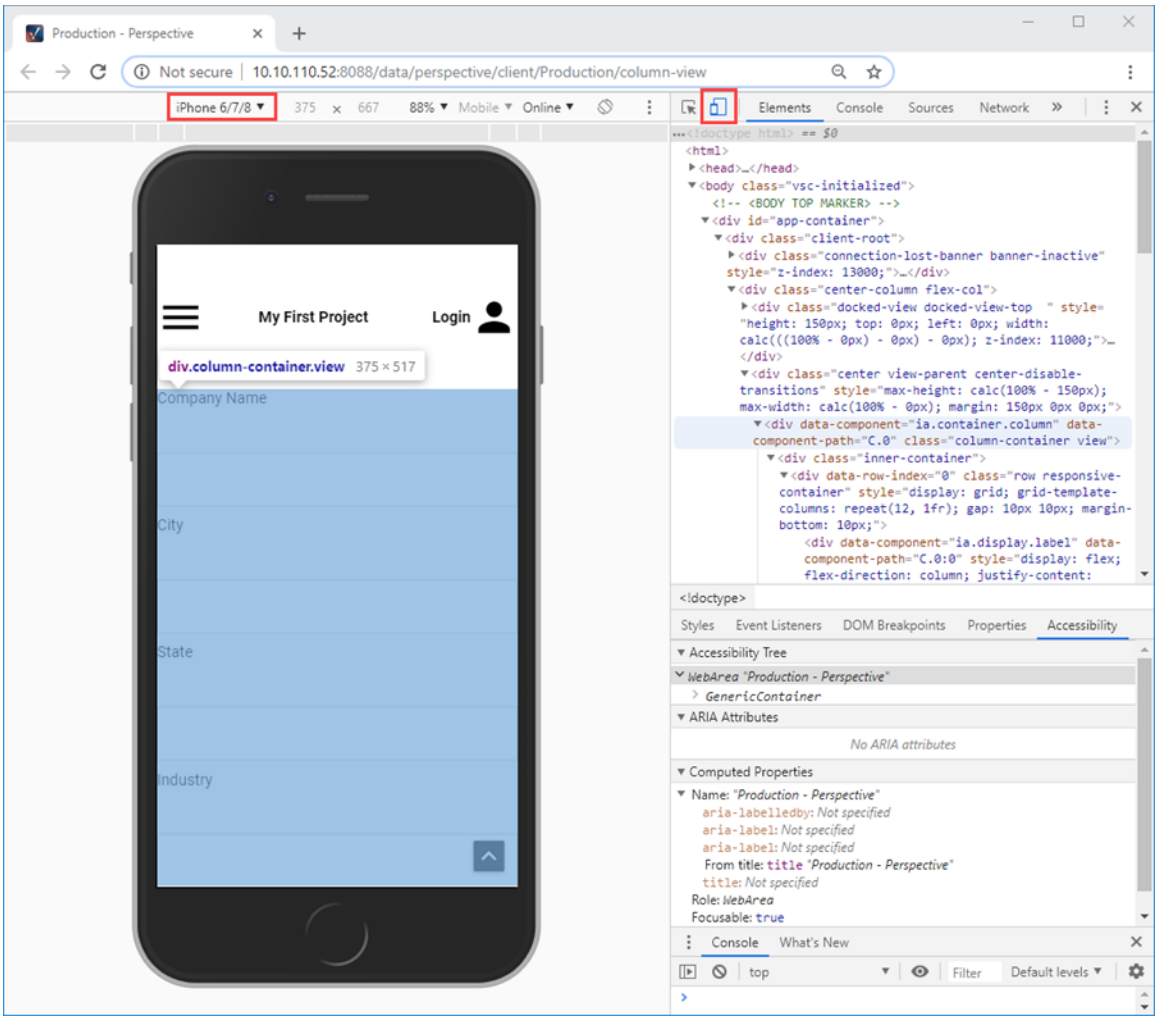
- Developer Tools

Developer Tools

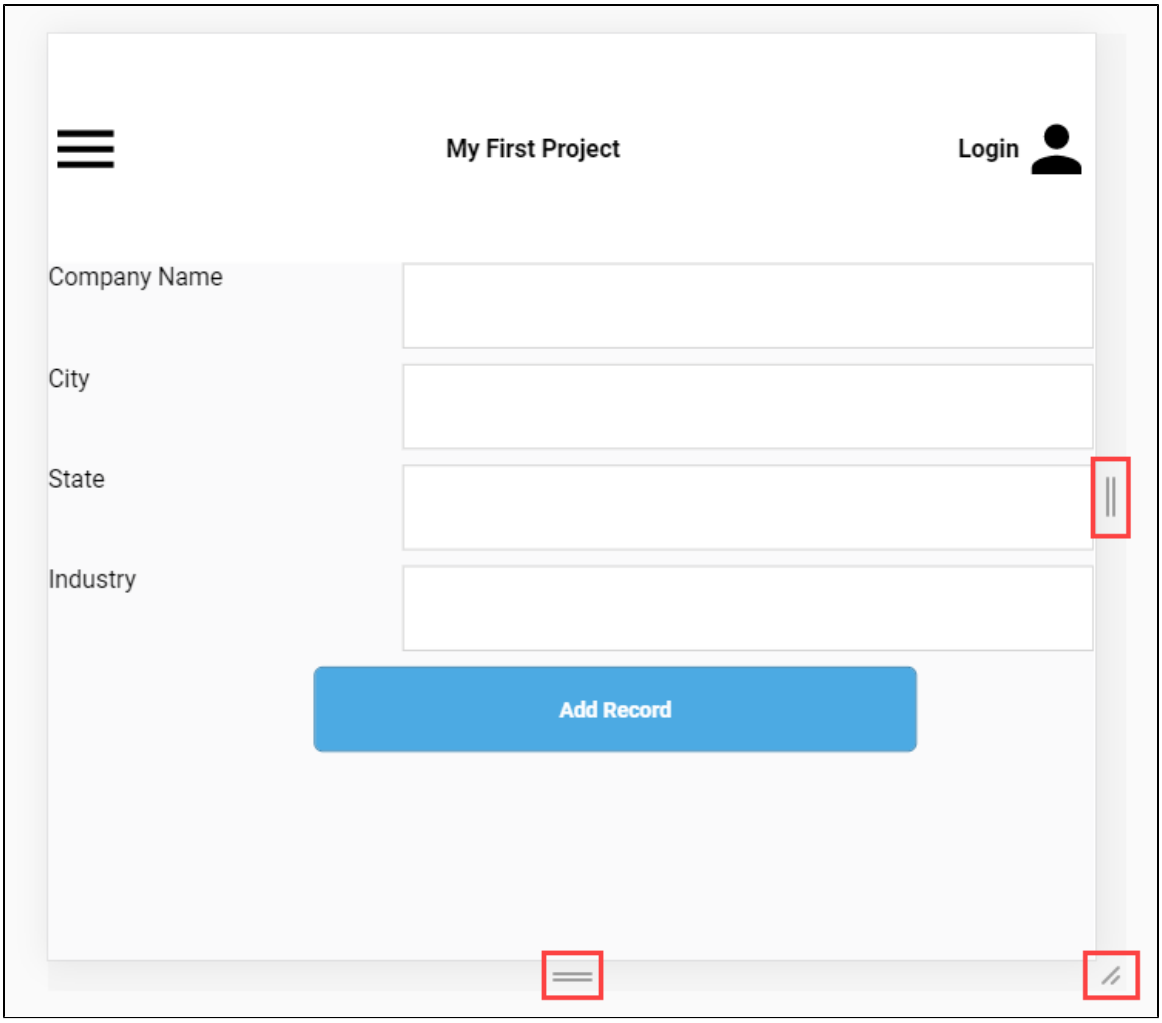
Any modern version of Chrome will work. You can access Chrome's Developer Tools from a Perspective Session by opening **Chrome's menu > More Tools > Developer Tools** or by clicking **F12**. A toolbar will open at the top of the Perspective Session page below the URL you are viewing. A panel is displayed on the right side of the page showing the HTML for the current URL. From the toolbar, you can select a specific mobile device and size as well as show the actual device frame by clicking the **Device Mode** icon. Chrome will emulate different devices with viewport sizes, and other features.



To activate Device Mode, simply click the **Device Mode** icon at the top of the Developer Tools window. You can use the **Device Mode** icon to toggle between devices. You can hover lines of code and Chrome will highlight the specific element on the screen.



In the Perspective Session, you can also use the handles in the Chrome browser to grow and shrink the screen size. There are a host of developer's tools, so it's a good idea to take time to familiarize yourself with Chrome's Developer Tools. It has potential to save you hours of work and help you to make your application site friendly to the increasing variety of devices people will use to access your application.



Related Topics ...

- Pages in Perspective
- Perspective Tab Container

Perspective Designer Interface

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

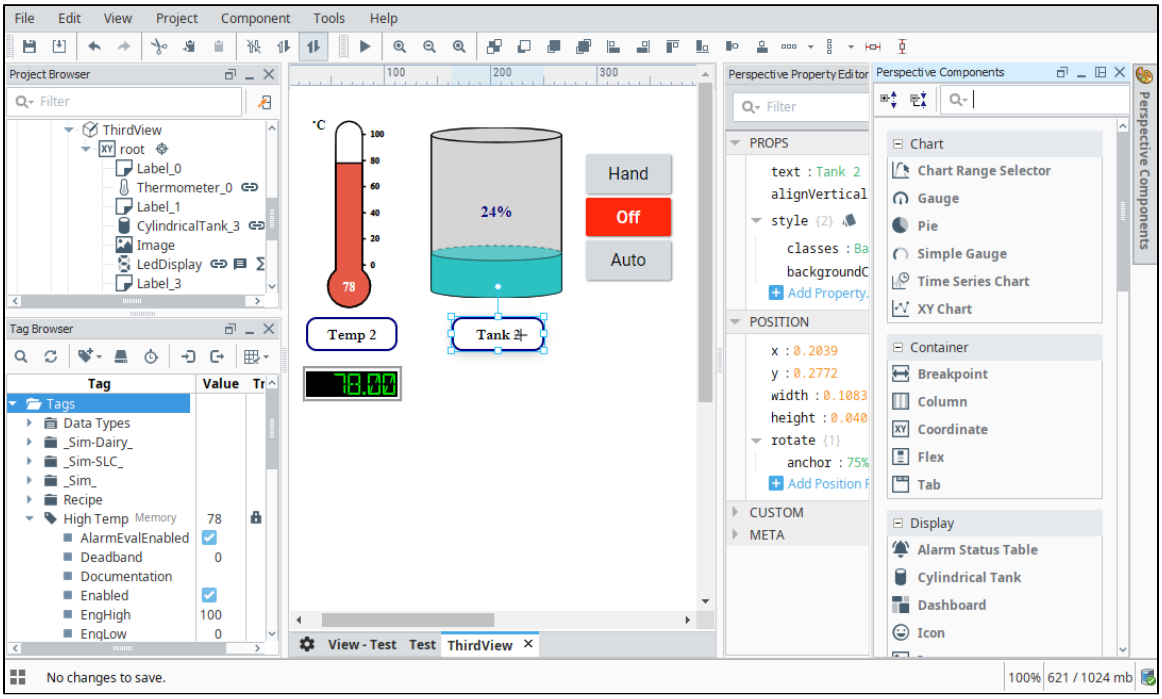
There are many dockable and draggable panels that surround the workspace, as well as the familiar menu bars and toolbars. The default panels include: Component Palette, Property Editor, P roject Browser, and Tag Browser. The dockable panels can be rearranged as you wish and will snap into place as you 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 **Rest Panels**.

On this page

...

- Perspective Menubar
  - File Menu
  - Edit Menu
  - View Menu
  - Project Menu
  - Component Menu
  - Help Menu
- Perspective Toolbar
- Resource Tabs
  - Right-Click Menu
- Project Browser
  - Badge Icons
- Tag Browser
- Perspective Component Palette
- Perspective Property Editor
- Page Configuration
  - Recently Modified Views
- Keyboard Shortcuts
- Vertical and Horizontal Guides



Perspective Menubar

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

File Menu

See General Designer Interface.

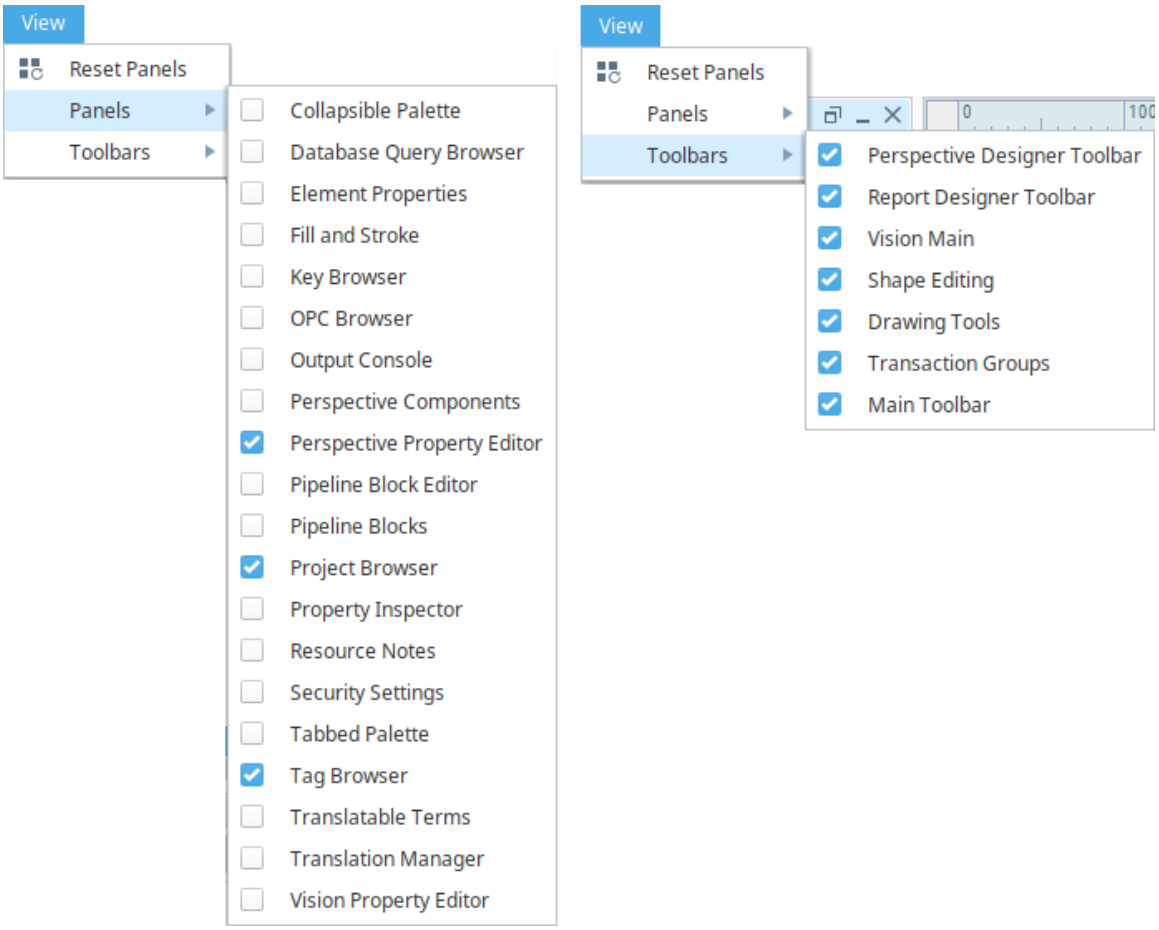
Edit Menu

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

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

View Menu

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



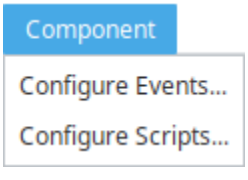
Function	Description
Reset Panels	Resets the Panels in the Designer so that they are in the default configuration.
Panels	A list of all the available Panels for the Designer in Perspective. Select the checkbox next to the Panel name to display that panel.
Toolbars	A list of all the available Toolbars for the Designer in Perspective. Select the checkbox next to the Toolbar name to display that toolbar.

Project Menu

See General Designer Interface.

Component Menu

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



Function	Description
Configure Events...	Displays the Event Configuration screen. For more information, see Perspective Events and Actions.

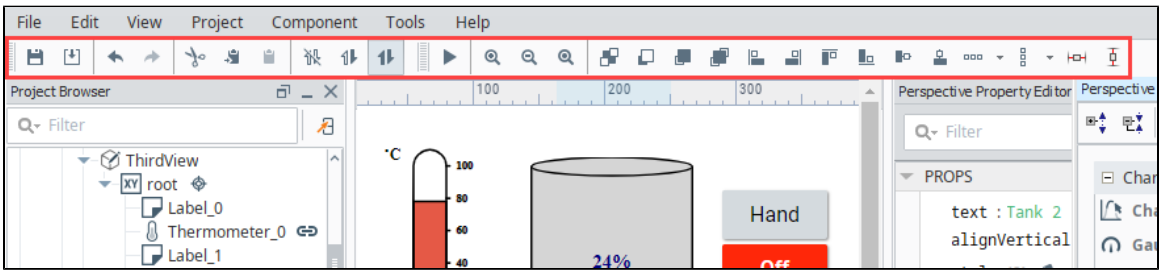
Configure Scripts...	Displays the Script Configuration screen. For more information, see Component Message Handlers.

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.
	The following feature is new in Ignition version **8.0.8** Click here to check out the other new features Zoom into the currently open window.
	The following feature is new in Ignition version **8.0.8** Click here to check out the other new features

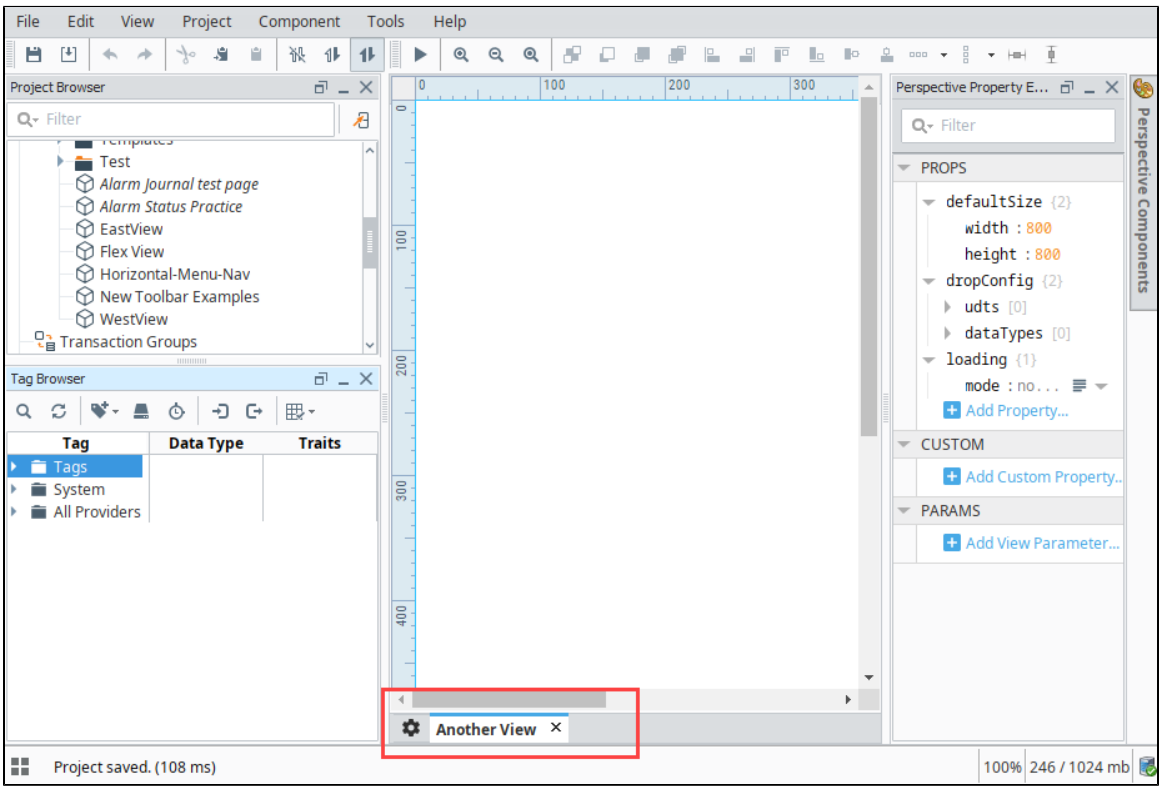
	Zoom out of the currently open window.
	The following feature is new in Ignition version **8.0.8** Click here to check out the other new features Zoom reset to 100%.
	Move the selected components to the back of the z-order.
	Move the selected components backward in the z-order relative to the overlapping 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 following feature is new in Ignition version **8.0.8**
Click here to check out the other new features

The remainder of the icons on the Perspective toolbar new to version 8.0.8 and are for aligning components. See aligning for more information.

Resource Tabs

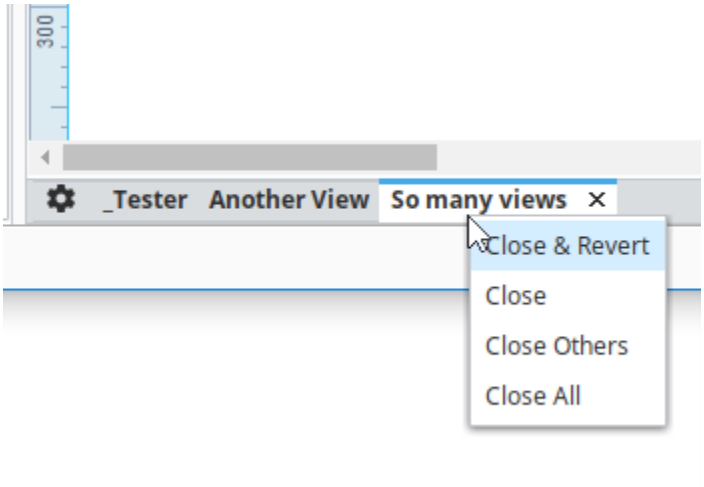
The Resource Tabs allow you to change which resource is being edited in the workspace, as well as navigate to the **Settings**  area of the Perspective Workspace.



Right-Click Menu

The following feature is new in Ignition version **8.0.1**
Click here to check out the other new features

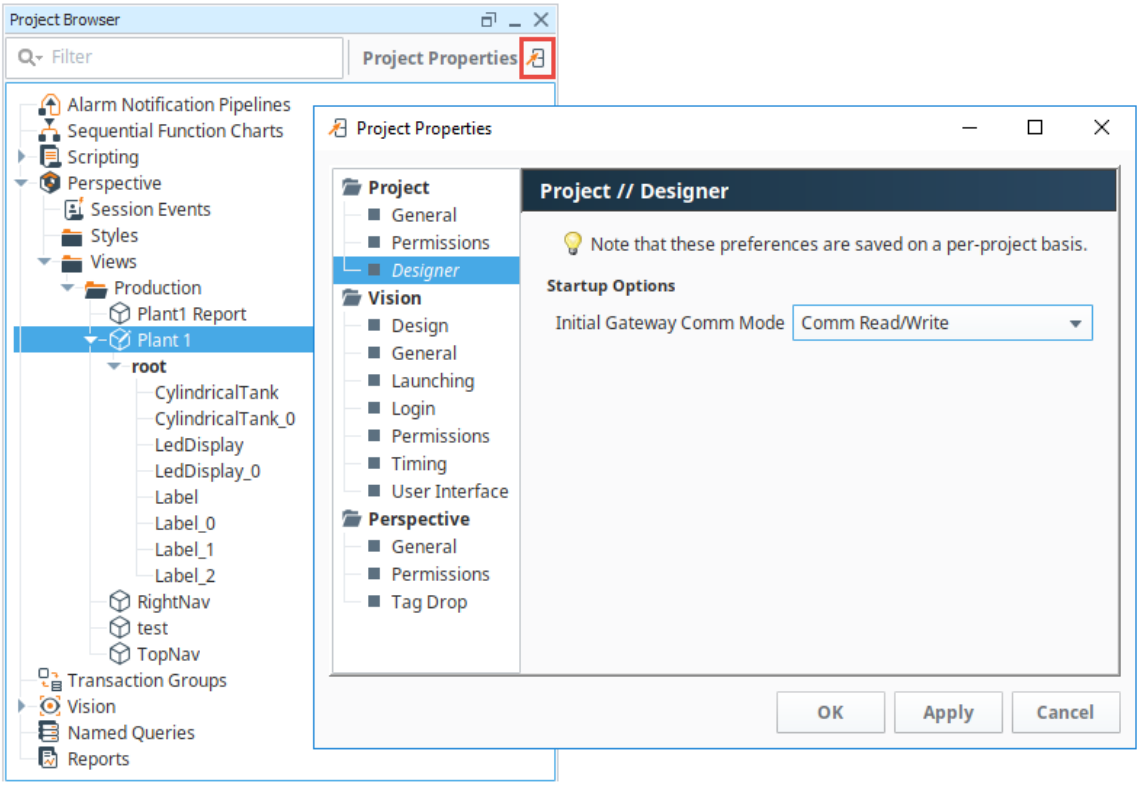
The right-click menu provides multiple ways to close one or more views.



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

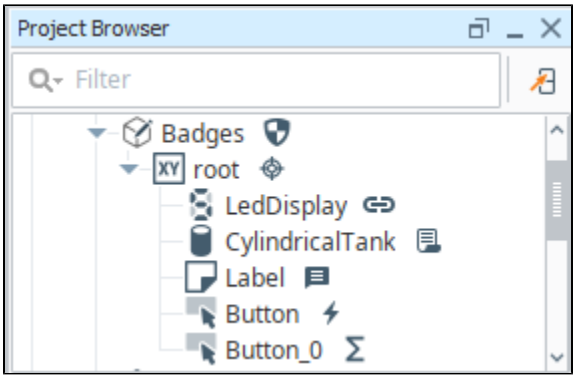
The Project Browser panel allows you to view the different Designer Spaces and their component hierarchies. You can expand folders and navigate down through each folder to see elements of a project such as windows, views, containers, and components. The Project Browser shows the entire tree structure 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.



The following feature is new in Ignition version **8.0.6**
Click here to check out the other new features

Badge Icons

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

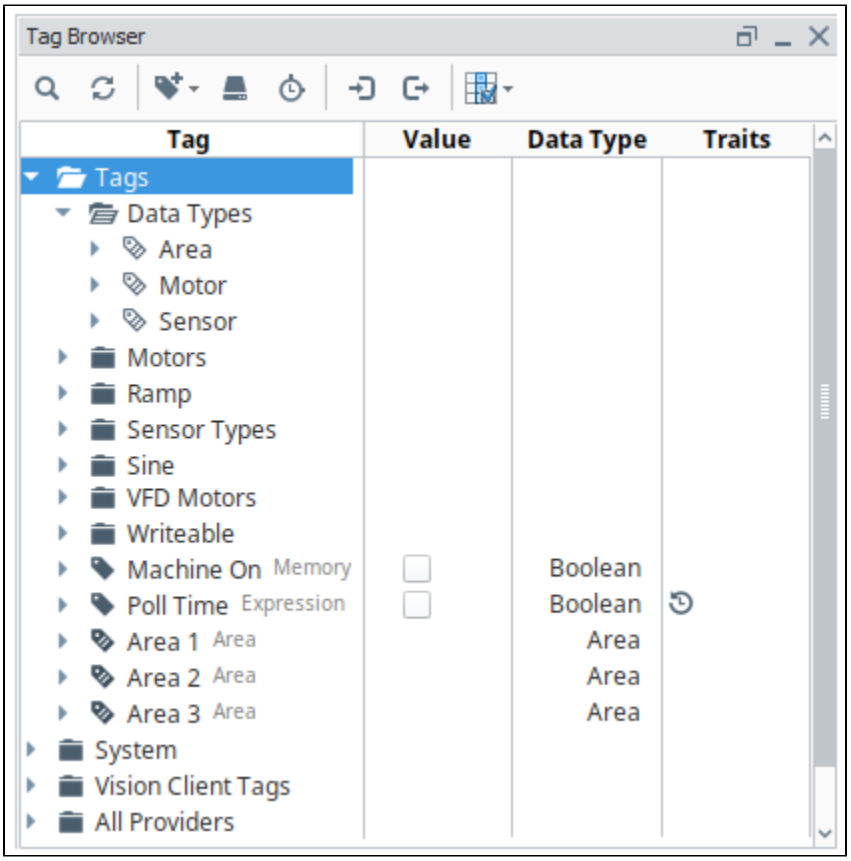


Badge Icon	Name	Description
	Binding	Indicates the component has a binding script. Appears next to the name of the component.
	Custom Method	Indicates the component has a custom method. Appears next to the name of the component.
	Deep Select	Shows a selected component within a selected container. Lets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

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 Understanding Tags.

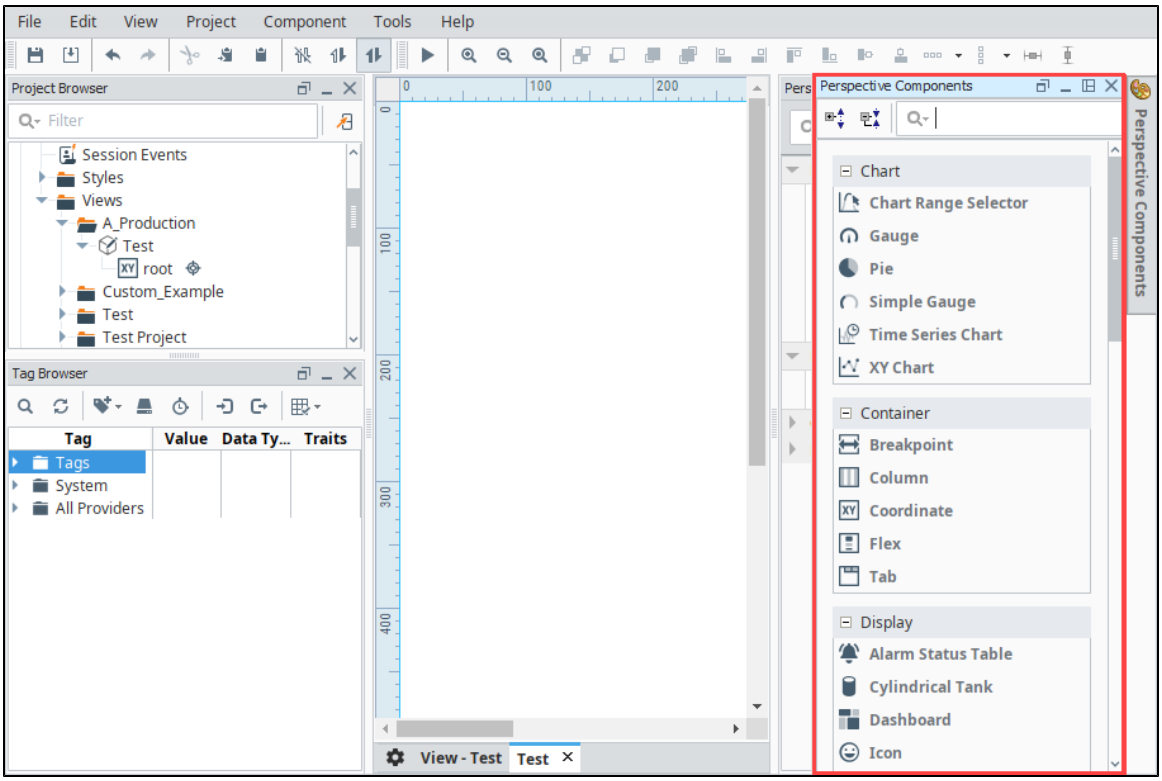


Perspective Component Palette

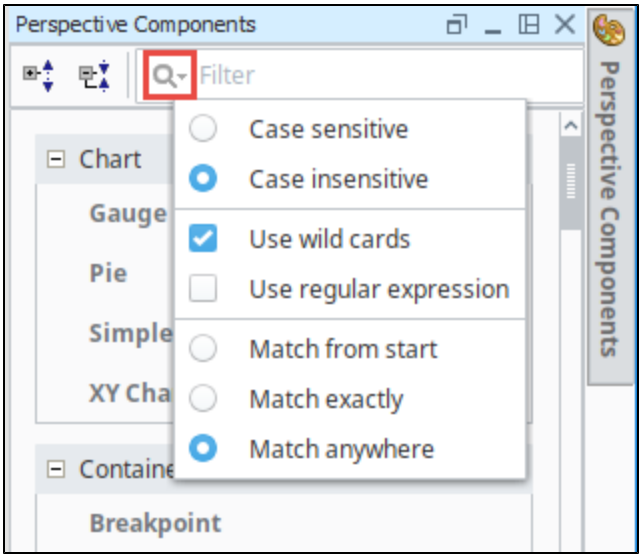
The Perspective module contains numerous components, such as buttons, labels, and charts. Perspective utilizes modern web technologies so many of the built-in components may look reminiscent of components you may have seen on your favorite websites. Components are created by dragging the component from 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.



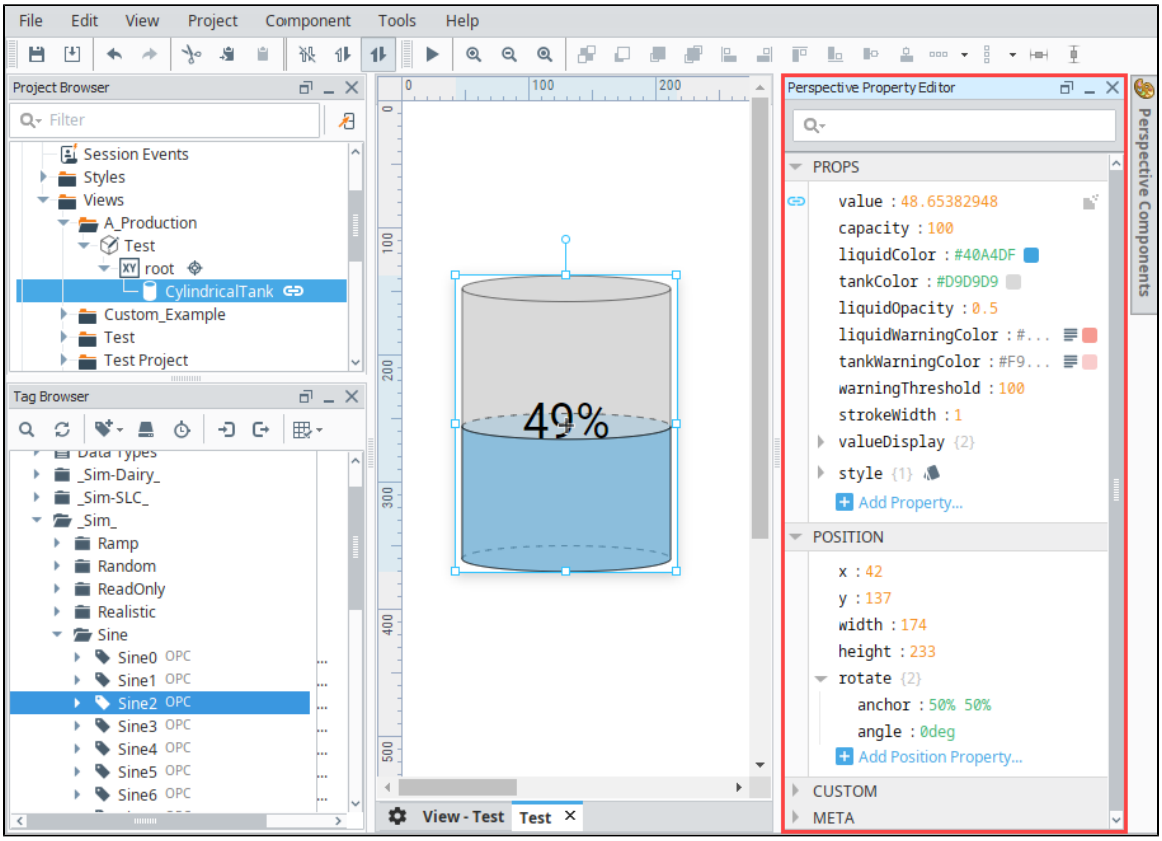
Perspective Property Editor

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.



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

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

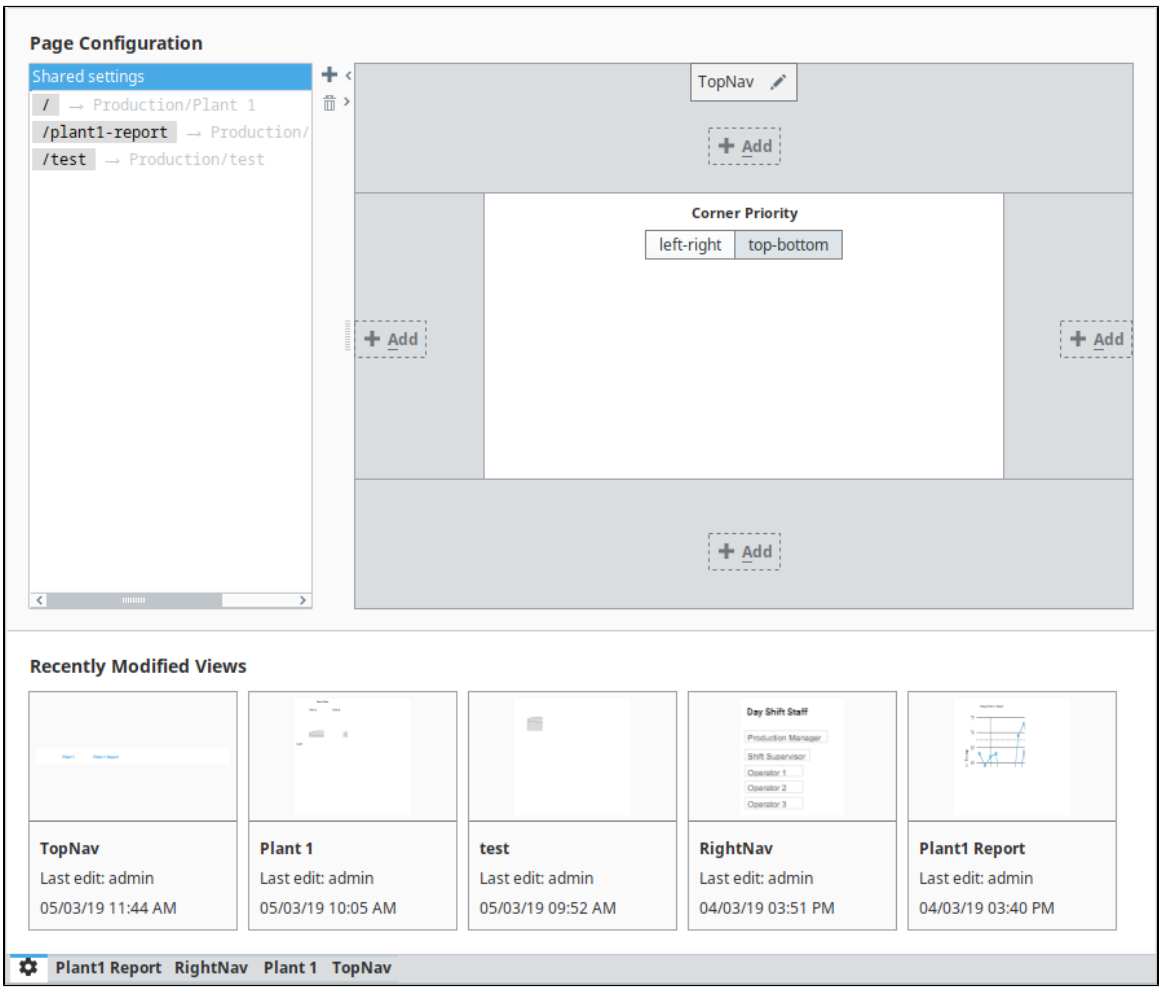
Initially, **Shared Settings** allow you to apply configuration changes to all pages (such as adding an omnipresent docked view), as well as specify the **Corner Priority**.

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.



Keyboard Shortcuts

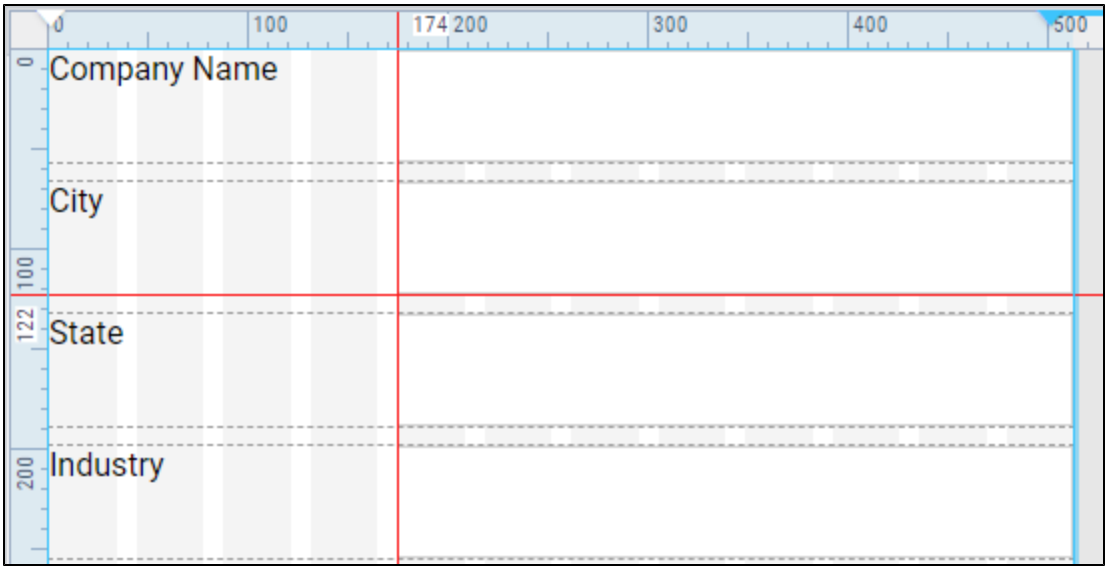
There are a lot of ways to speed up development once you are familiar with how Ignition works. There are many keyboard shortcuts in Perspective Designer that are listed throughout the Designer interface alongside menu options. To learn about keyboard shortcuts, go to the Keyboard Shortcuts page.

Vertical and Horizontal Guides

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

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

You can add multiple vertical and horizontal guides.



The following feature is new in Ignition version **8.0.8**
Click here to check out the other new features

As of release 8.0.8, the Perspective Designer toolbar also has options for aligning components. For more information, see Manipulating Components.

Related Topics ...

- Perspective Components
- Perspective Component Properties
- Views and Containers in Perspective
- General Designer Interface
- Pages in Perspective

Working with Perspective 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.

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.

On this page

...

- Component Categories
  - Component Properties
- Adding Components
- Selecting Components
  - Mouse Selection
  - Tree Selection
  - Deep Selection
- Right-Click Menu
  - Z-order
- Manipulating Components
  - Moving
  - Resizing
  - Aligning
  - Alignment Guides
  - Alignment Guides Example
  - Rotating
  - Rotation Examples
- Image Source
  - Web Address
  - Image Management

Component Categories

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. 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.

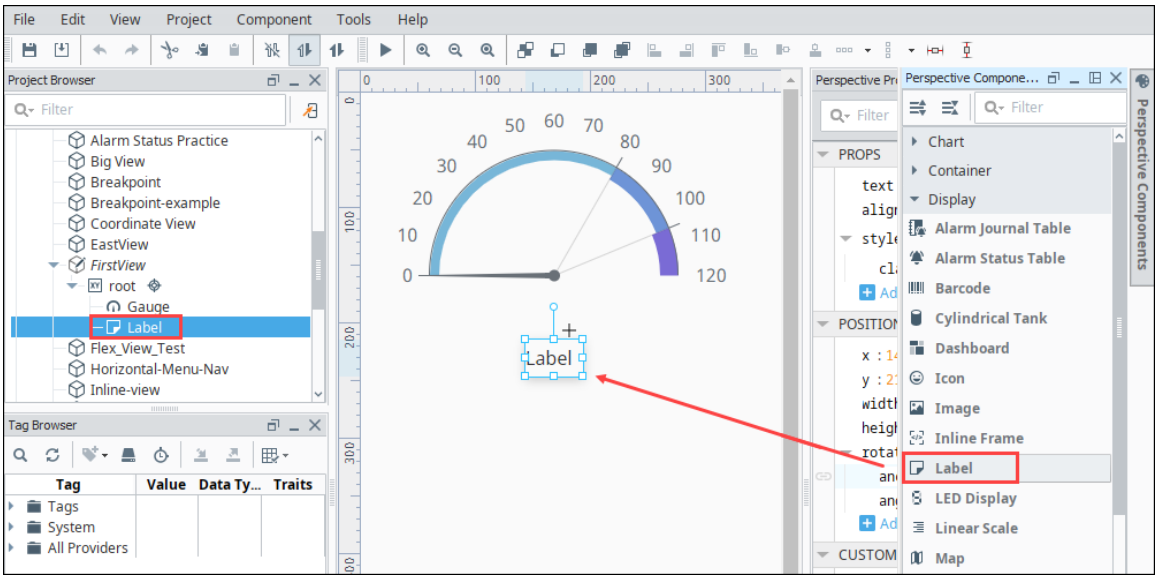
Component Properties

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.

Adding Components

To add a component, open the Perspective Components palette. Click on the component you want and drag it onto your View. In this example, we put a Label component under the Gauge component.

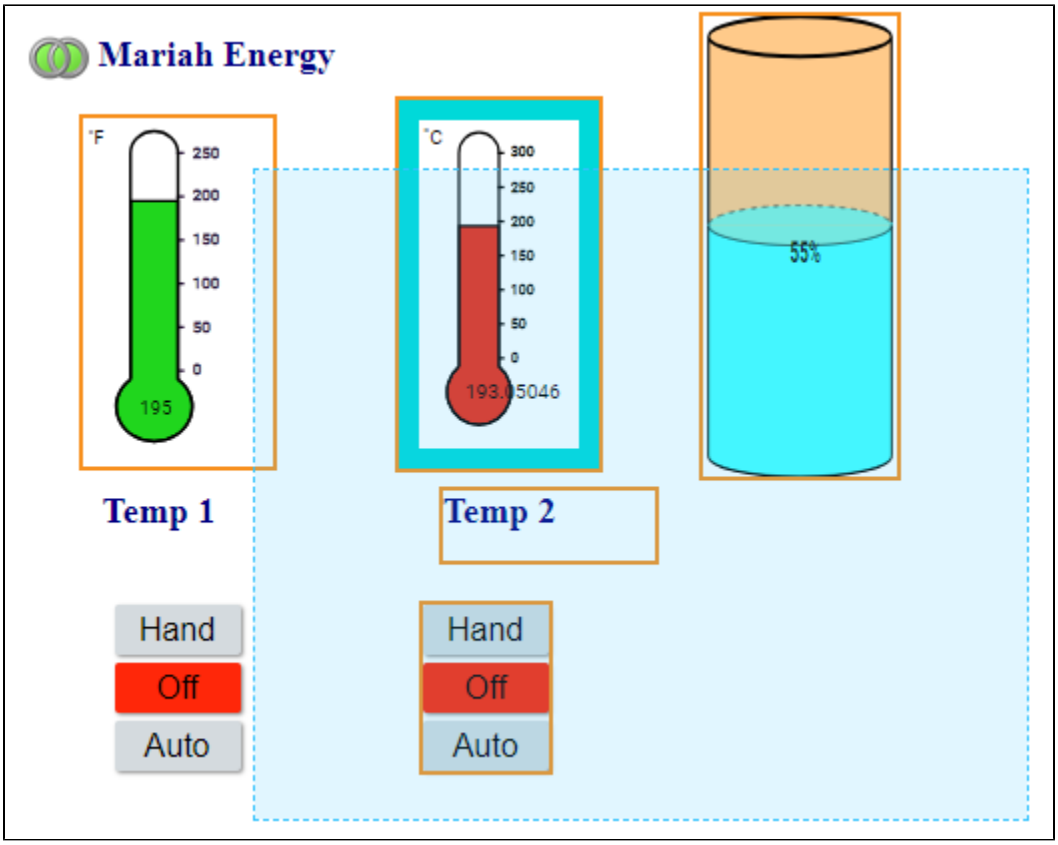


Selecting Components

Mouse Selection

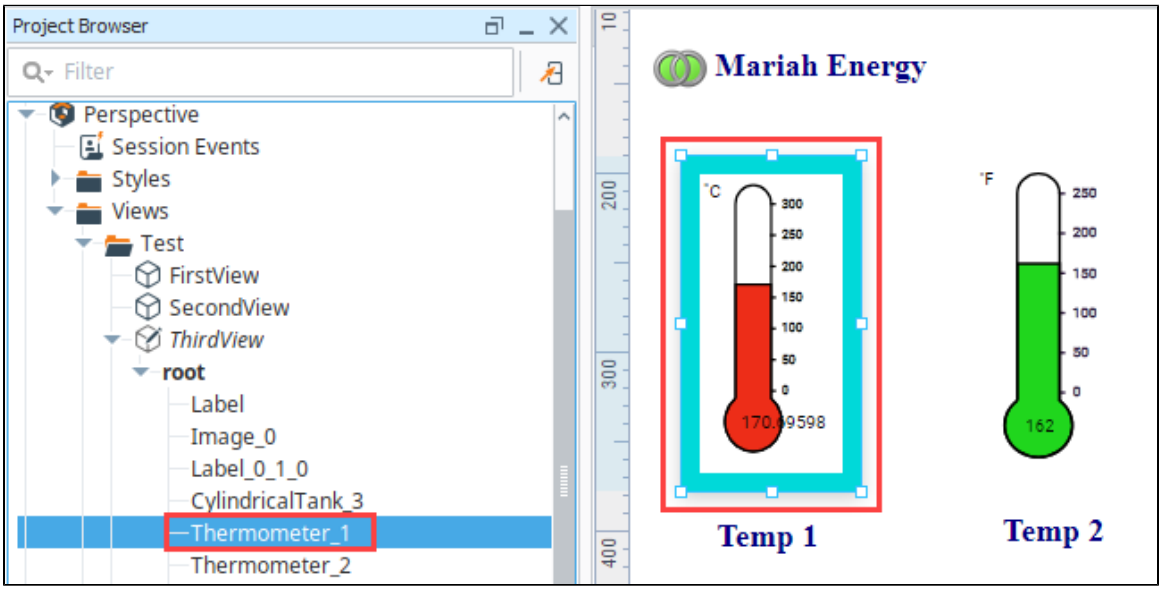
Using the mouse is the most common way to select components. Click on a component to select it, double-click on it to Deep Select it (more on that later).

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

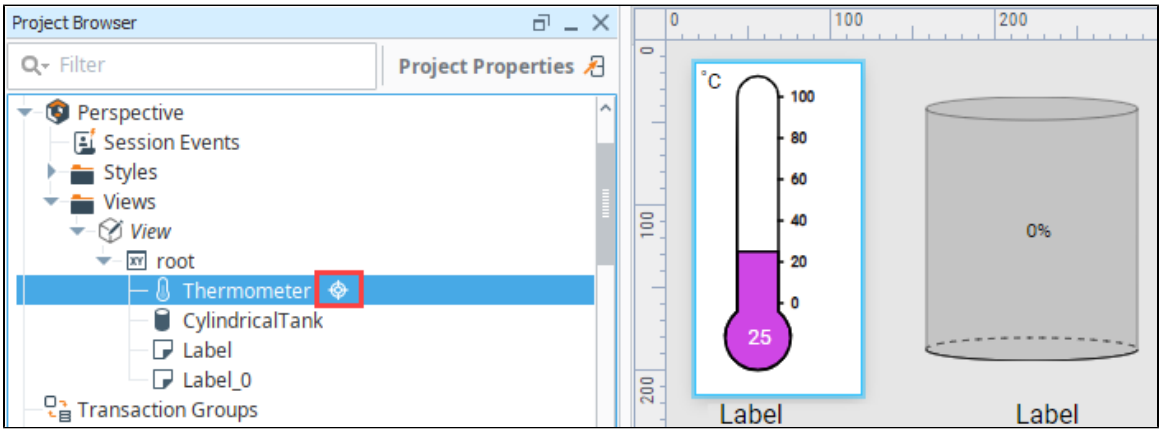
this comes a true tree of components, and selecting components nested inside containers works differently than components on the root. Deep Selection allows you to select into a container and select a component within.

To deep select a component, double click on it. Three things happen to indicate that the component is deep selected:

- 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.

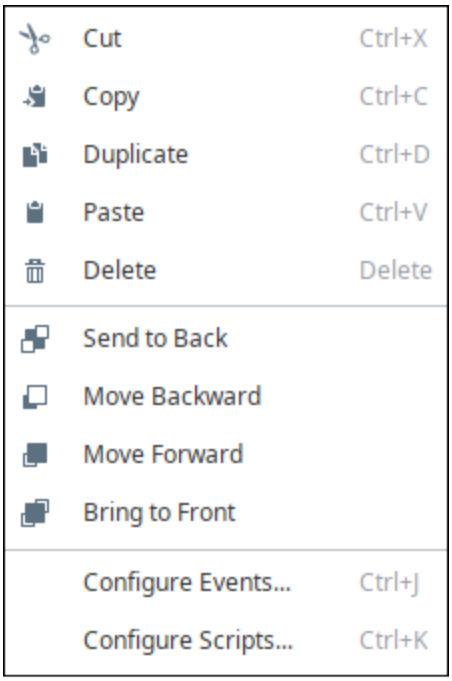
This lets you know that you have deeply selected into the component and not just selected the component itself. Once inside this new deep selection space, you can then select a component much like before, either by clicking or dragging to select components, or by deep selecting into another container inside the original container.

To cancel the deep selection, click anywhere outside of the component. In the following example, the Temperature component is deep selected.



Right-Click Menu

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



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

Function	Icon	Description

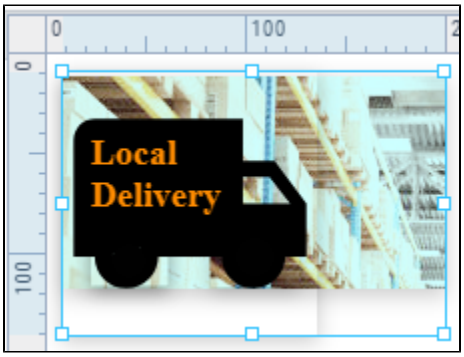
Cut		Removes the selected component but keeps a copy on the clipboard. You can also use the **Ctrl-X** shortcut to quickly cut a component.
Copy		Copies the selected component to the clipboard. You can also use the **Ctrl-C** shortcut to quickly copy a component.
Paste		Pastes the current contents of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

Z-order

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.

Icon	Action
	Move the selected components to the back of the z-order.
	Move the selected components backward in the z-order relative to any overlapping components.
	Move the selected components forward in the z-order relative to any overlapping components.
	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.



Manipulating Components

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

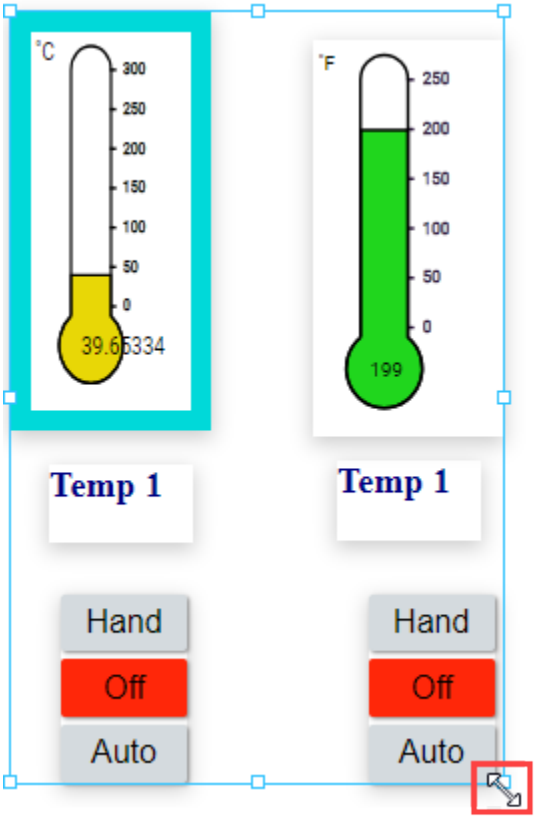
Moving

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.



The following feature is new in Ignition version **8.0.8**
Click here to check out the other new features

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.

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

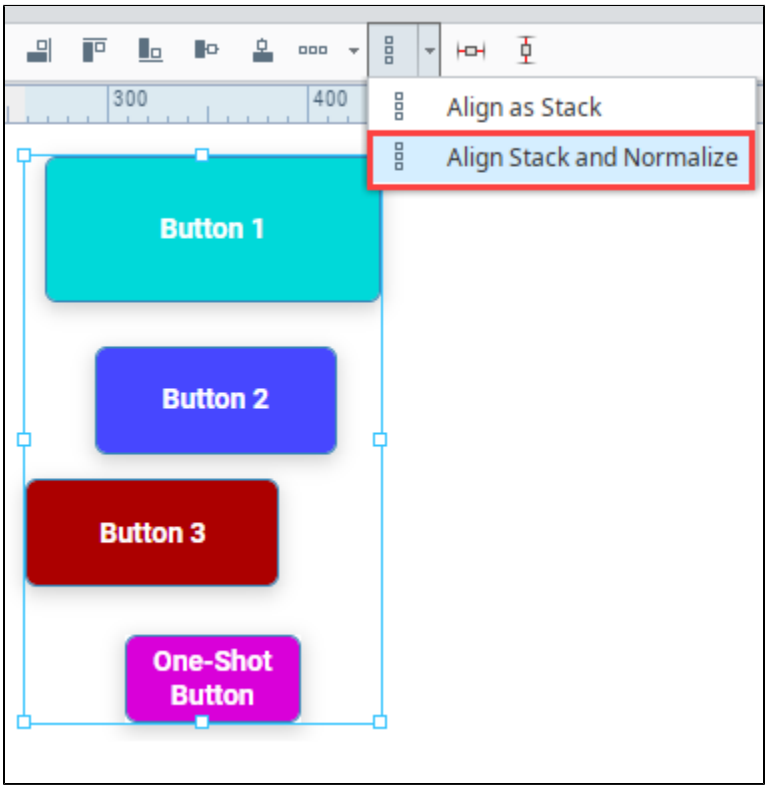
Icon	Function	Description
	Align Left	Align the left edges of a group of components.
	Align Right	Align the right edges of a group of components.
	Align Top	Align the top edges of a group of components.
	Align Bottom	Align the bottom edges of a group of components.
	Align Centers	Aligns all of the selected components horizontally on their centers.

	Horizontal	
	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.

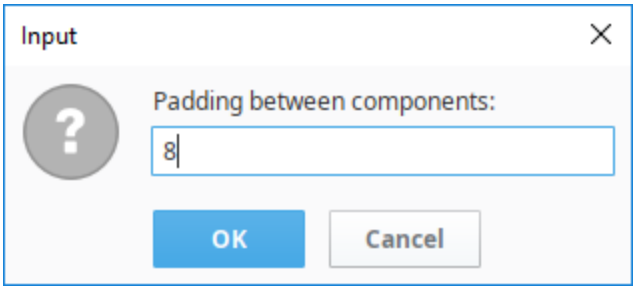
Align and Normalize Example

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.

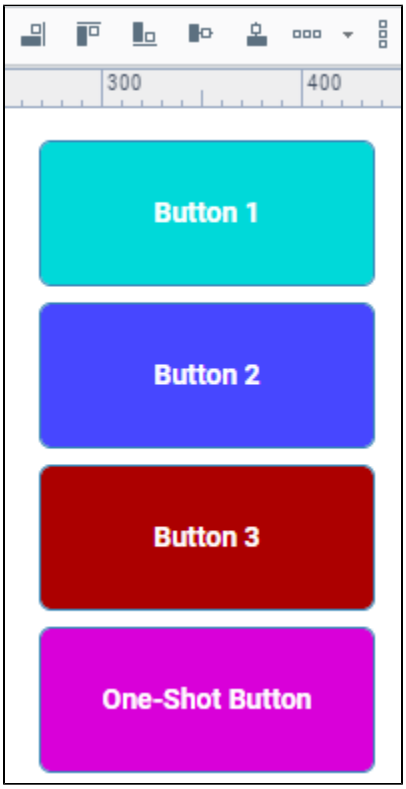
1. Select the buttons, then choose Align as Stack and Normalize.



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



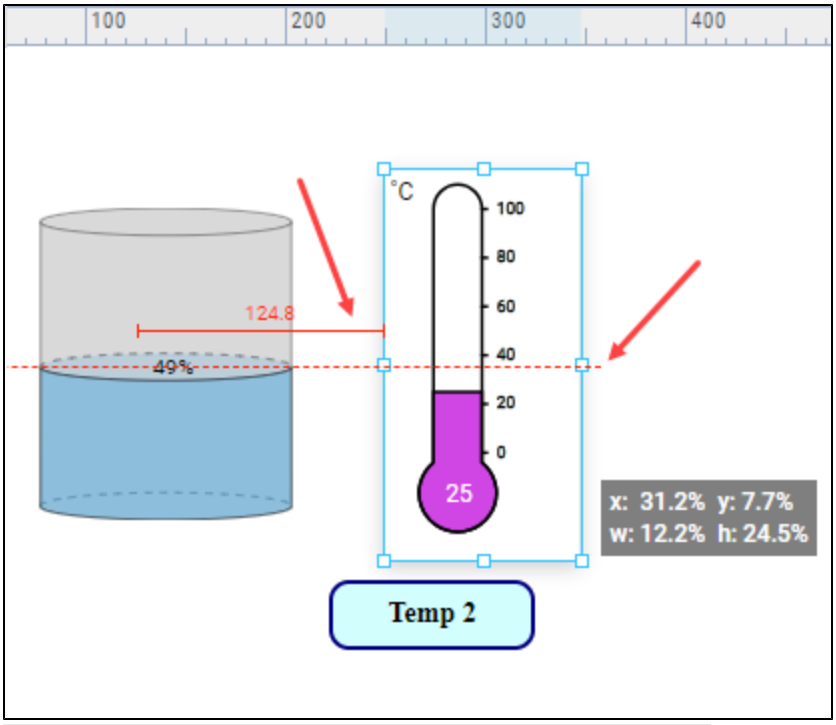
3. Click **OK**. The buttons will be stacked and normalized.

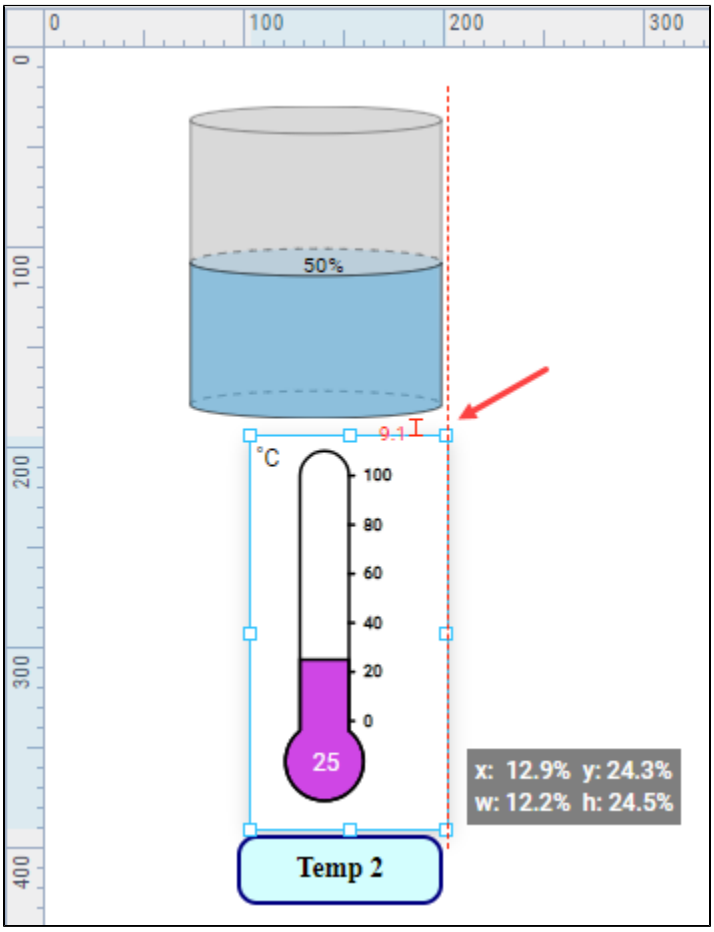


Alignment Guides

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.





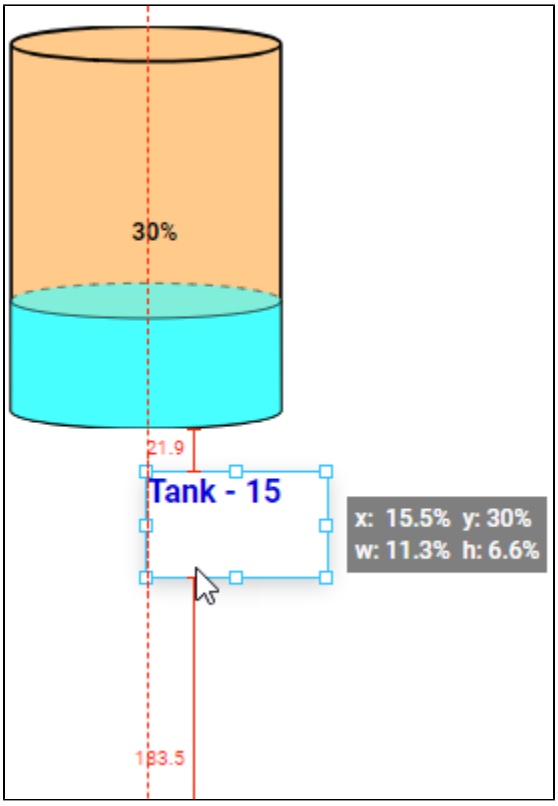
The Designer also has Vertical and Horizontal guide that can be set up to help you align components. For more information, see Vertical and Horizontal Guides.

Alignment Guides Example

Alignment Guides Example

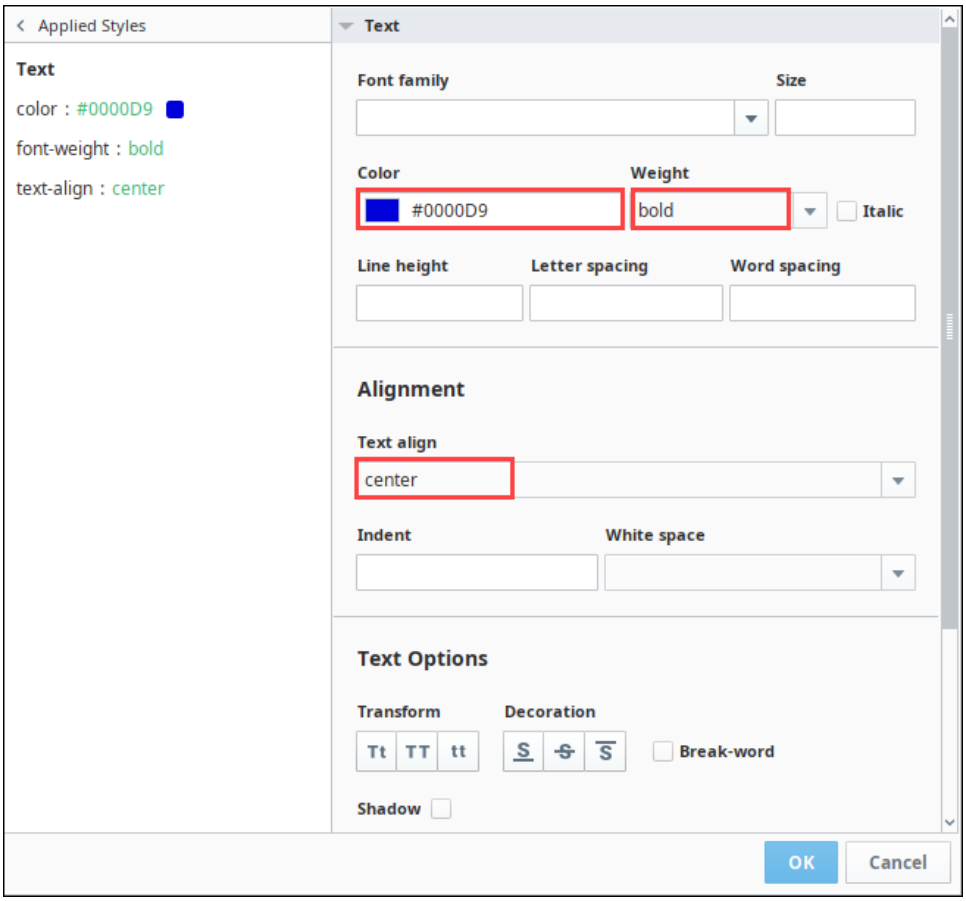
In the following example we've added a Label component beneath a Cylindrical Tank component. I want to align the label so that it is centered with the center line of the tank.

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

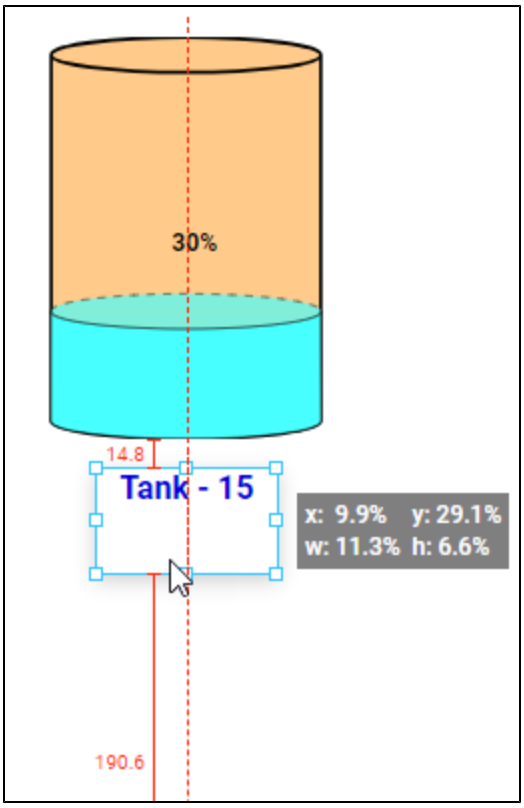


   Now the starting edge of the Label is aligned with the center of the Cylindrical Tank. But we want the text to be centered under the tank.

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.



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



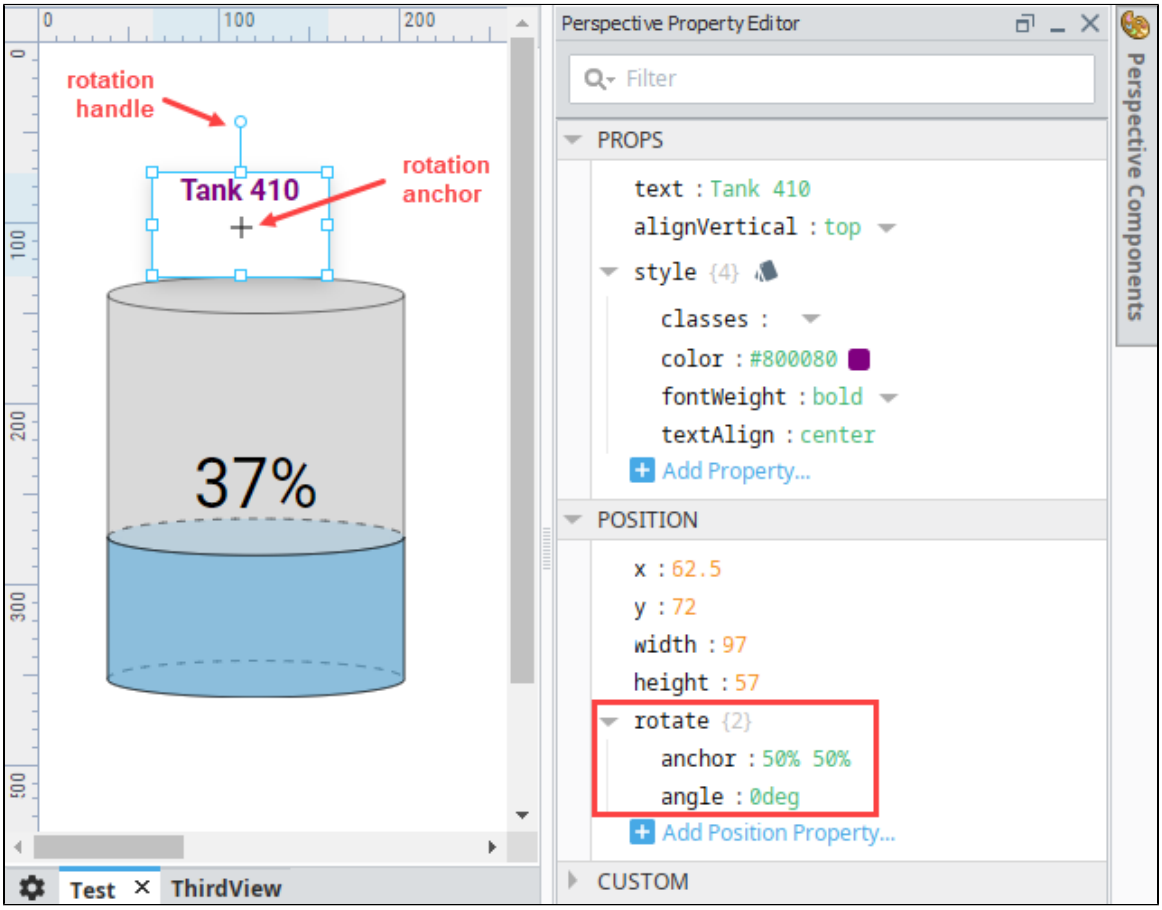
Rotating

Components place on coordinate containers can be rotated. As of 8.0.6 the Rotate property has been moved to the Position Properties section of the Perspective Property Editor. For a definition of all the position properties, see Views and Containers in Perspective and the individual container pages. Rotated components being aligned to other components will correctly align along the top-most, bottom-most, left-most, right-most point of the rotated component.

There are two ways to rotate a component:

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

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

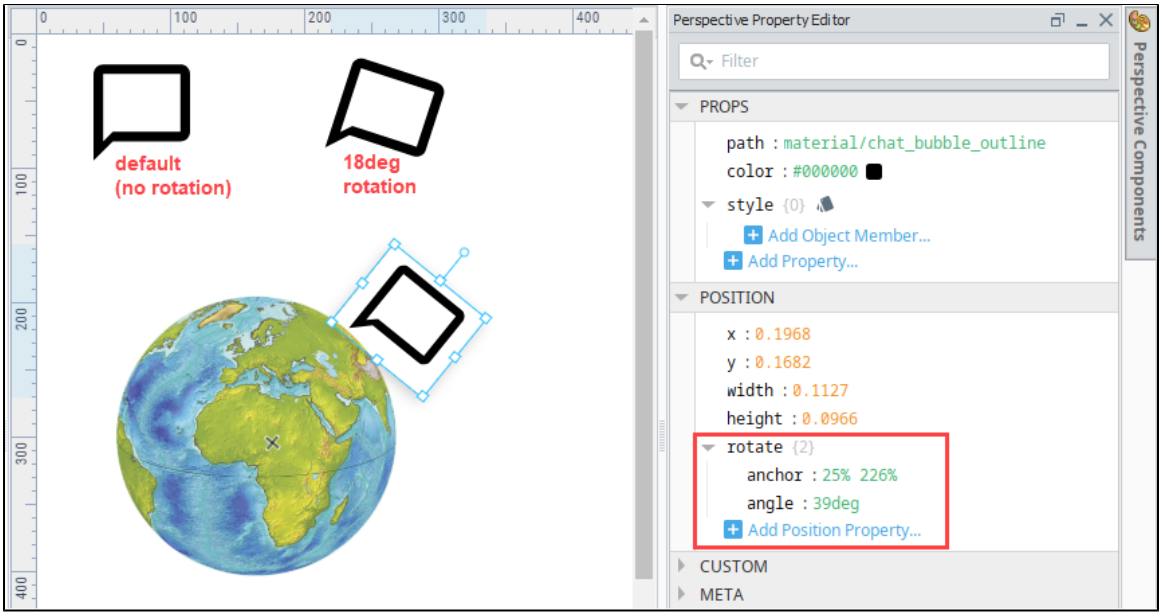


If a component is smaller than 28px by 28px, the rotation handle and anchor + symbol are not displayed. Use the property editor to set rotation and anchor properties in this case.

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.



The following feature is new in Ignition version **8.0.8**
Click here to check out the other new features

Example 2

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

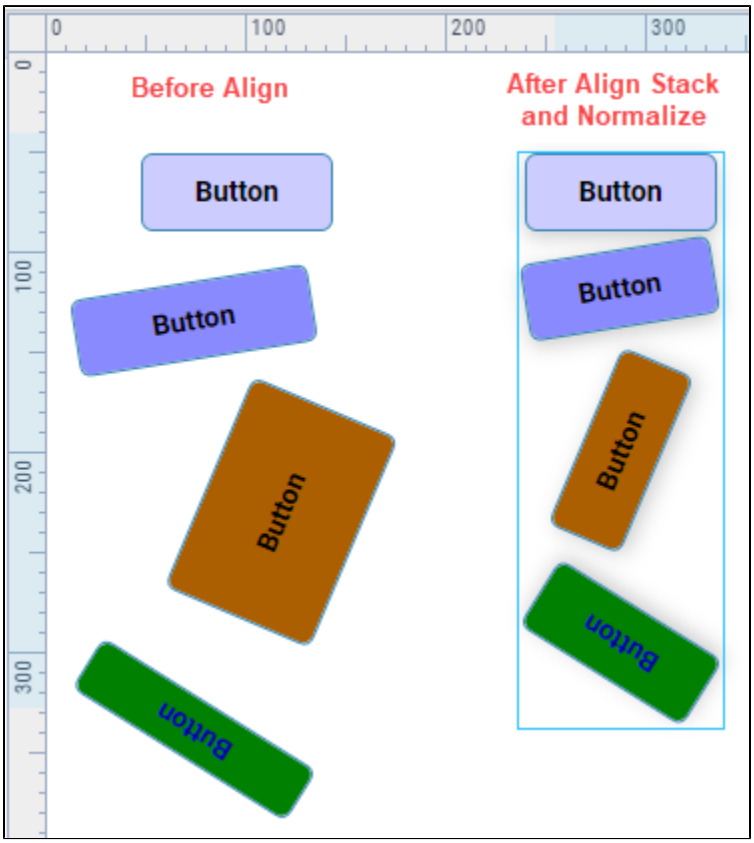


Image Source

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

Web Address

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

Image Management

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

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

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

Related Topics ...

- Perspective Component Properties
- Tag Data Types

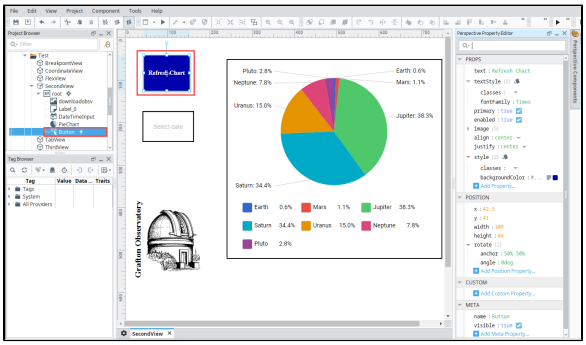
In This Section ...

Perspective Component Properties

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

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

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



On this page

...

- User Created Properties
- Property Data Types
- Property Categories
- Restricting Property Access
  - Restricting Access to Component Properties
  - Writing to Private or Protected Session Prop
- Persistent Properties
  - Persistence and User-Created Properties
  - Bindings and Persistence
- Custom Properties
  - Creating Custom Properties
- Meta Properties
- Params
  - Docked View
  - Embedded View
  - Page
- Search Filter
- Bindings
- Styles
- Action Menu
  - Actions
  - Structure
  - Options

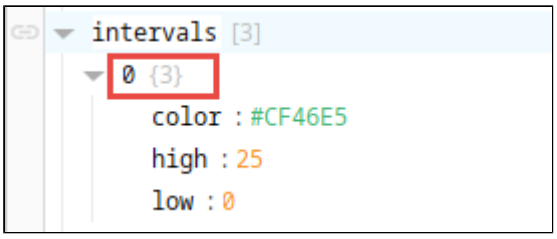
User Created Properties

Perspective has two distinct types of user-created properties:

- Pre-defined properties in a component that has its own schema. We don't recommend creating additional properties to components with their own schema, although it is possible. In the event something should go wrong or inconsistencies occur, it's difficult to figure out if the problem is with a user-created property that's part of the component's schema or with a custom property in the Custom Category.
- Custom Properties fall under the Custom Category in the Property Editor. This type of user created property is unique, and is used to enhance functionality to a component through the use of data bindings and scripts. Custom properties created in the Custom Category will not result in any collisions with any pre-defined properties that are part of the schema of the component. Creating a custom property in the Custom Category is considered best practice.

Property Data Types

Before you create a property, you must first understand the different property data types. There are three property data types available in Perspective's components: value, object, and array. 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. The table below describes each property data type.

Property Types	
Value	A value is an single variable for the property. It has a 'key" and a "value". `key : value` "Key" is the **name** given to the property, and "value" is the actual value of the property. Value types are as follows: • Boolean - A true/false value. • Numeric - An integer up to the maximum value for a long integer. • String - A string of characters can be numeric, alpha, or a combination
Object	An object is a one or more values stored under one variable name. Objects are indicated by curly braces { }. In this example, the Object has three sub- properties. 
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).

Dataset

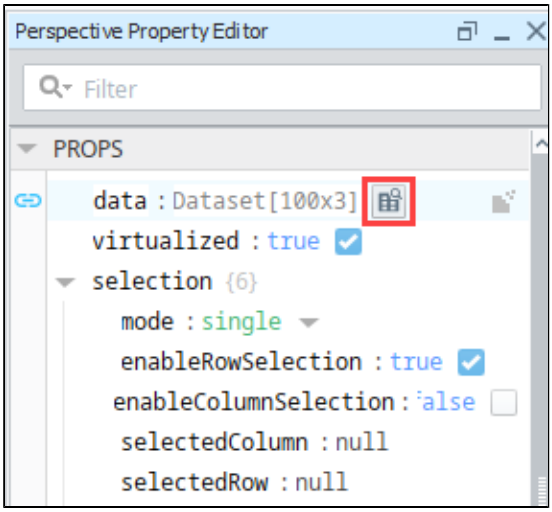
The Dataset property type is not an option when changing a properties data type, it can only be used when a binding returns data in a dataset format.

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.

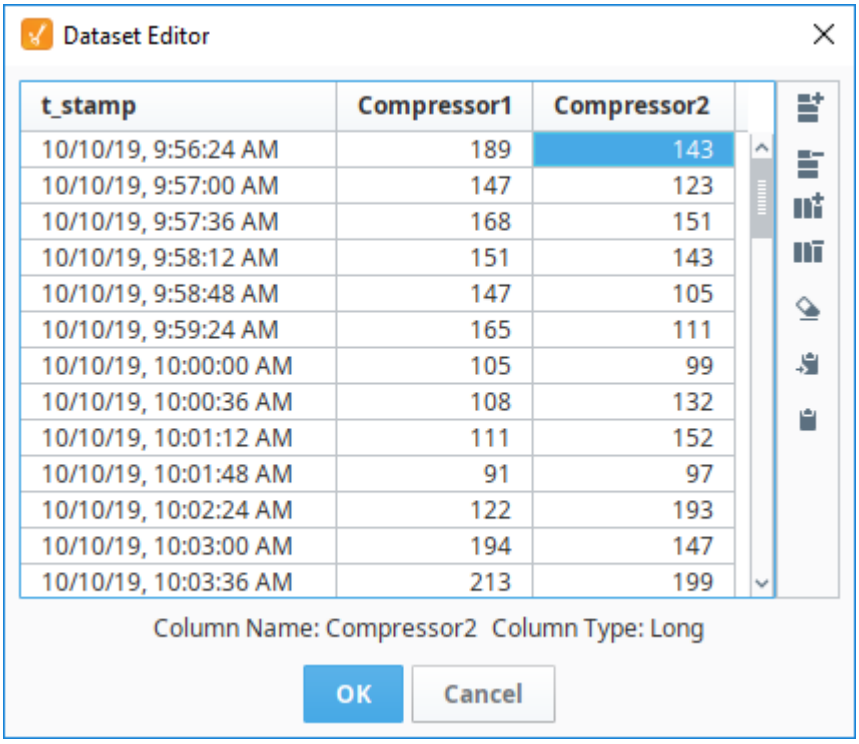
Datasets are generally only returned by SQL queries and Tag History bindings, though both have the ability to select from several different return formats. A good example is a table bound to a Historical Tag query like the image below.

The following feature is new in Ignition version **8.0.5**
Click here to check out the other new features

Dataset properties now 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.



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



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

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

Paste from clipboard

A Note about Numbers in Perspective

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:

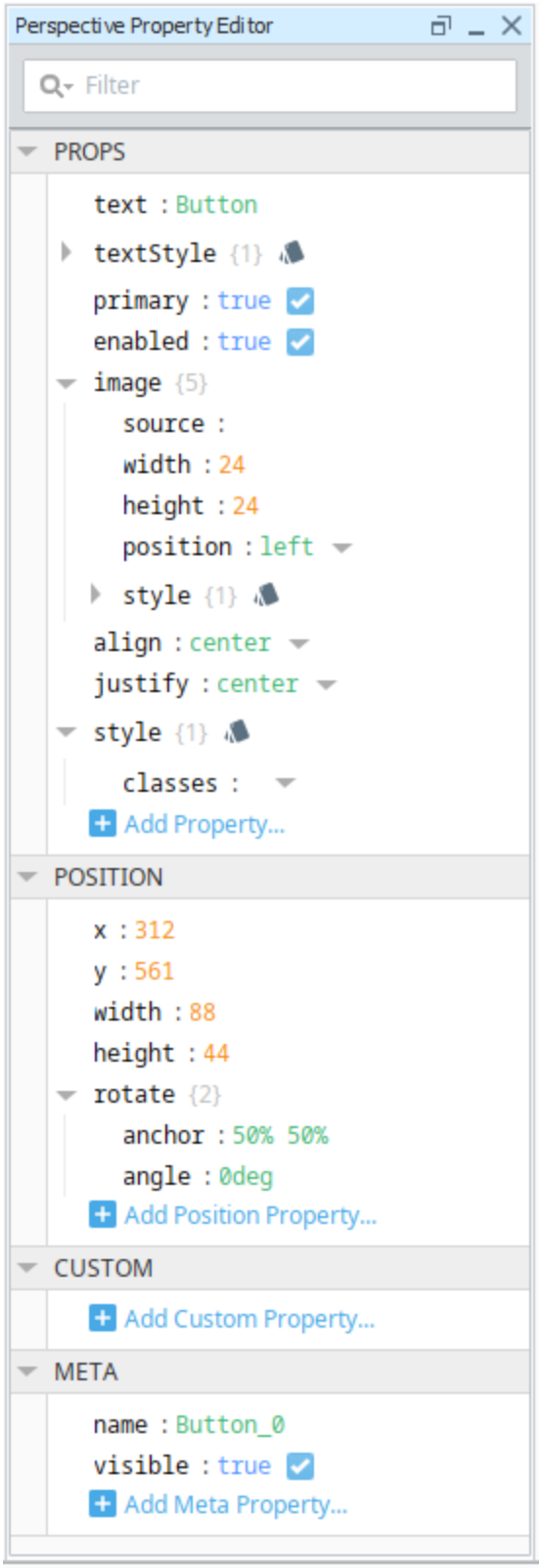
- Modifying a value in the Perspective Dataset Double or Long column could lead to an unexpected value.
- If bound to a Dataset tag, the value of the tag could be changed to an unexpected value.
- A Perspective Component rendering a dataset Double or Long could render an unexpected value.

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.

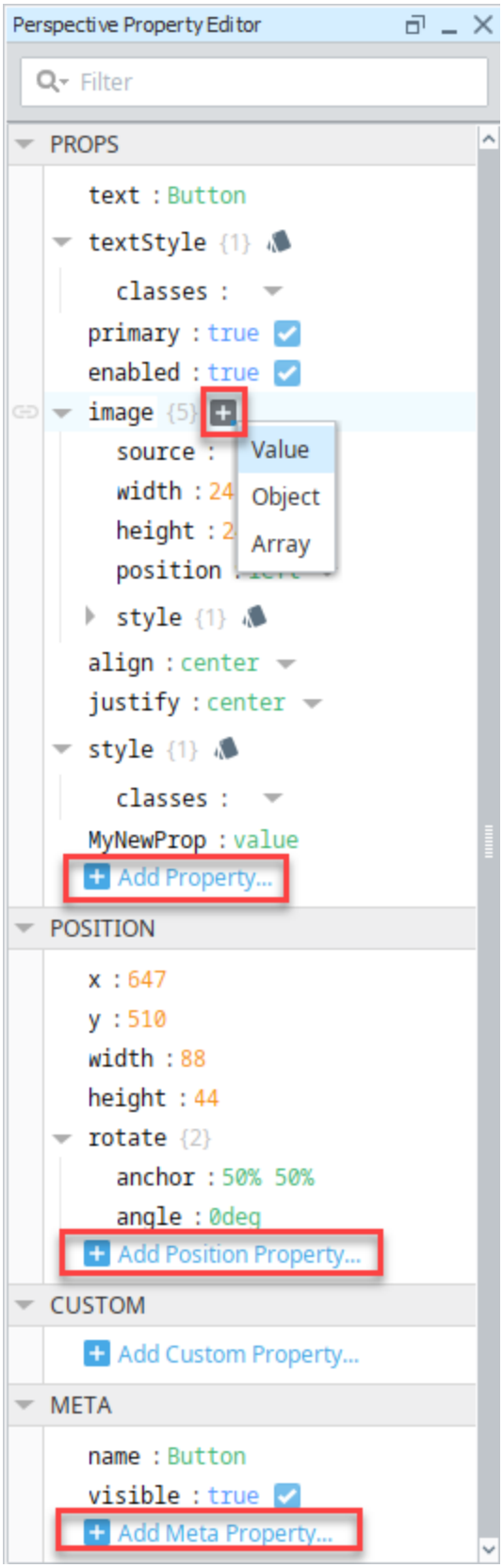
- **Props** - Properties that control the component's configuration and provides the runtime data for how the property appears and behaves in a session. See individual Perspective Components for a list of the properties and their descriptions.
- **Position** - Properties defined by the component's parent 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 Custom Properties. To create a new custom property, all you need to do is select a property type such as Value, Object or Array. For example, if you want to add a single property, select "**Value**" and you'll see it has " **key**" and "**value**" objects. Rename "**key**", to give the property a unique name, and change the "**value**" by entering number or string
- **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. Note that each property is listed under a category: Props, Position, Custom, and Meta.



The following feature is new in Ignition version **8.0.6**
Click here to check out the other new features

Add additional properties by hovering over an existing property and clicking the **Add** icon, then select Value, Object or Array, or by clicking the **Add Property** icon at the bottom of the category.



The following feature is new in Ignition version **8.0.5**
Click here to check out the other new features

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). However the browser-side of the DOM is still interactable, but 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.

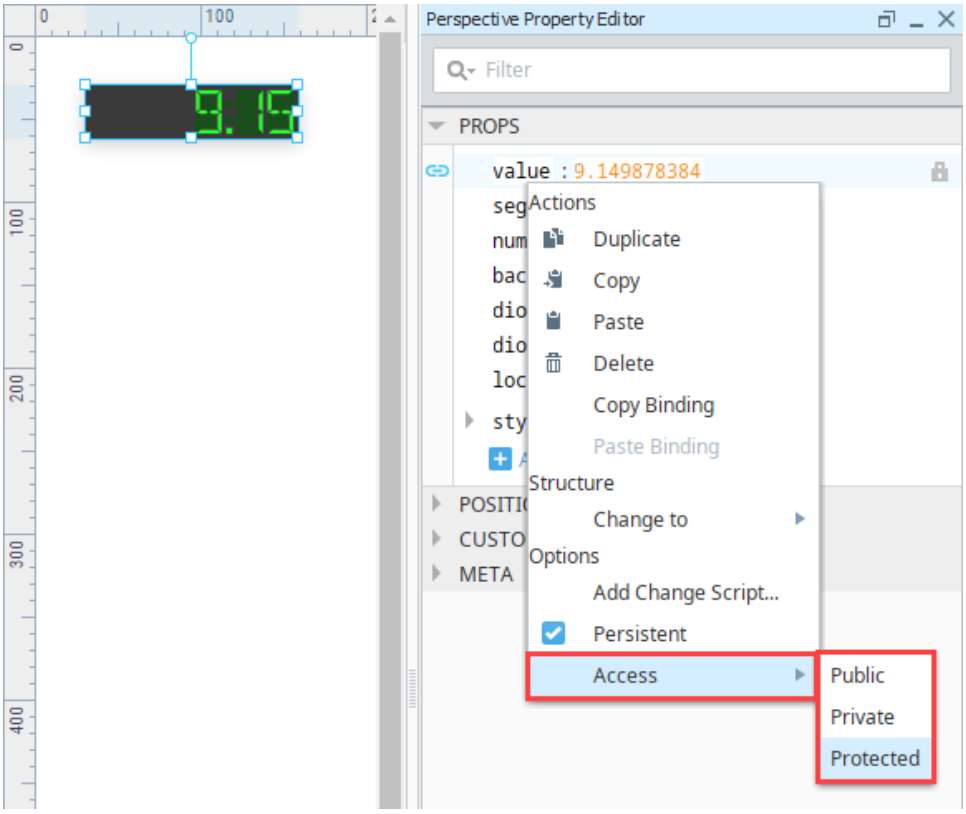
One caveat property access is that the names of style classes are always visible and interactable from the browser, so style class names should not included sensitive information.

Property	Description
Public	Unrestricted Access. This is the default setting for all non-system properties.
Private	Hidden. The property is not readable from JavaScript, and write requests will be ignored (assuming the script correctly guesses the property path)
Protected	Read only. The browser's DOM may be interacted with via JavaScript, but the back-end will ignore any write requests.
System	This property's value is updated automatically. Not user-writable, read-only, and cannot be removed. System properties will not accept writes from the browser, and bindings will not be allowed to write to these properties either.

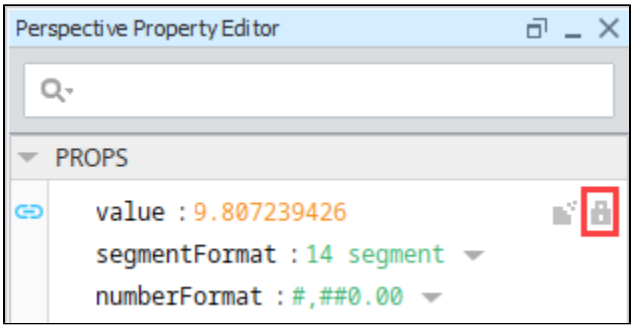
Restricting Access to Component Properties

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

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

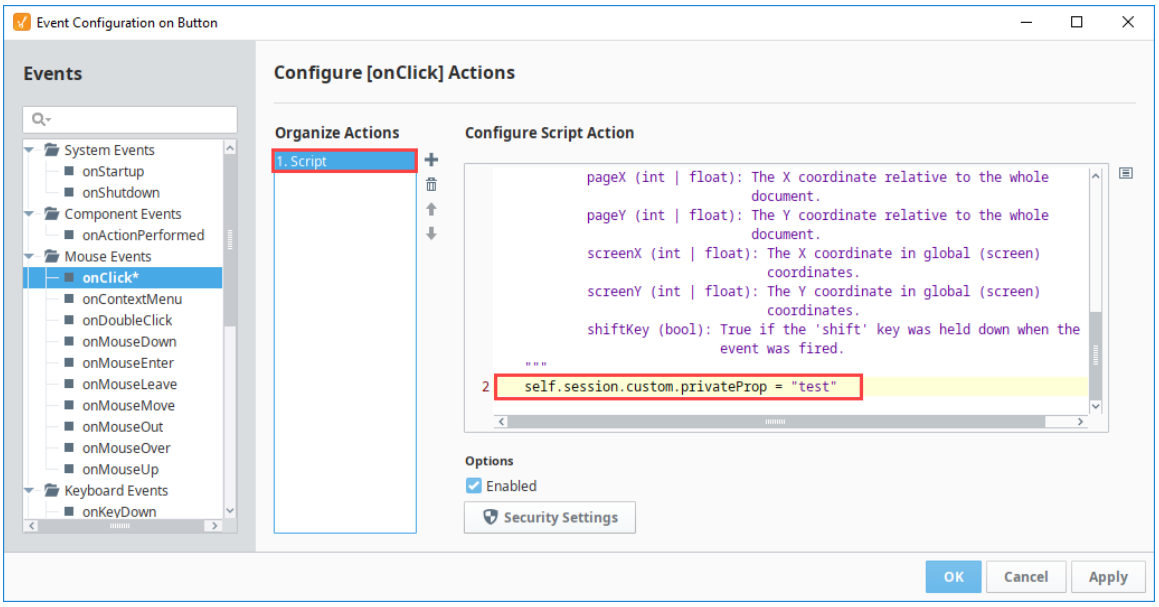


3. Once the restriction access is configured, the **Property Editor** will place a badge on any non-public property: Private , Protected , and System .

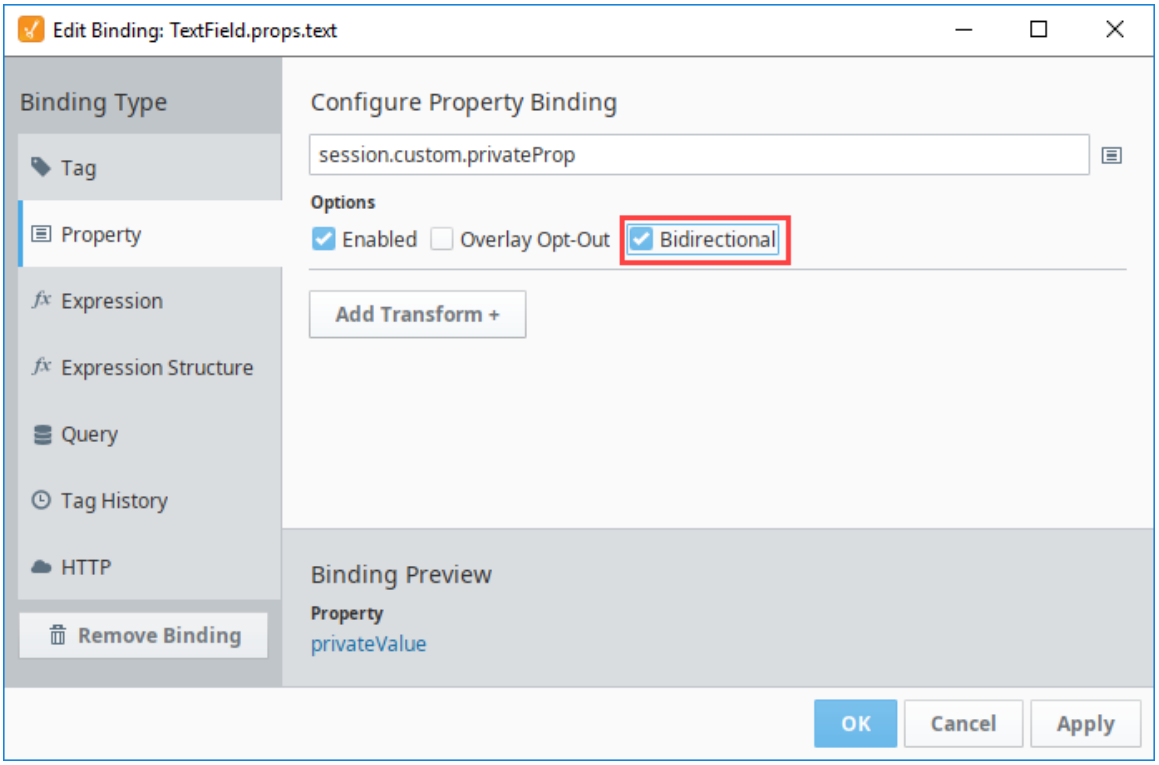


Writing to Private or Protected Session Prop

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



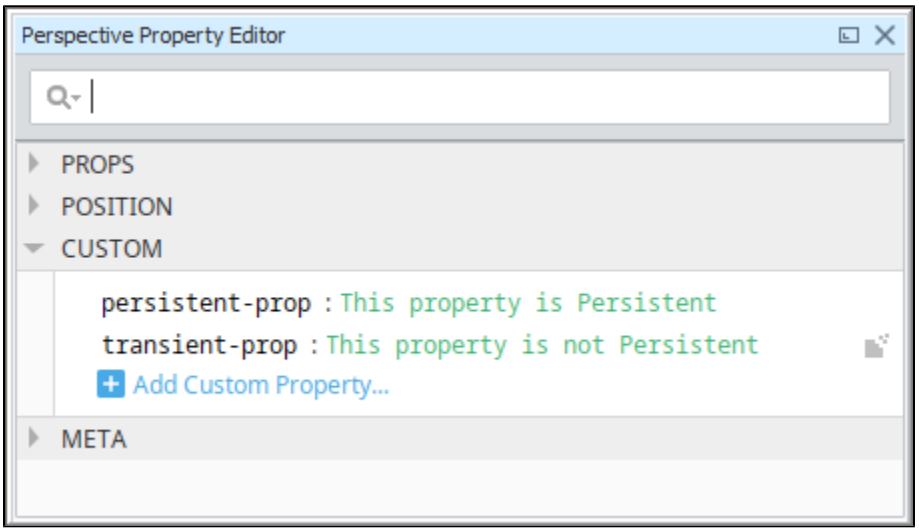
You can also use a property binding to write to a Private or Protected session prop. Remember to enable the **bidirectional** option.



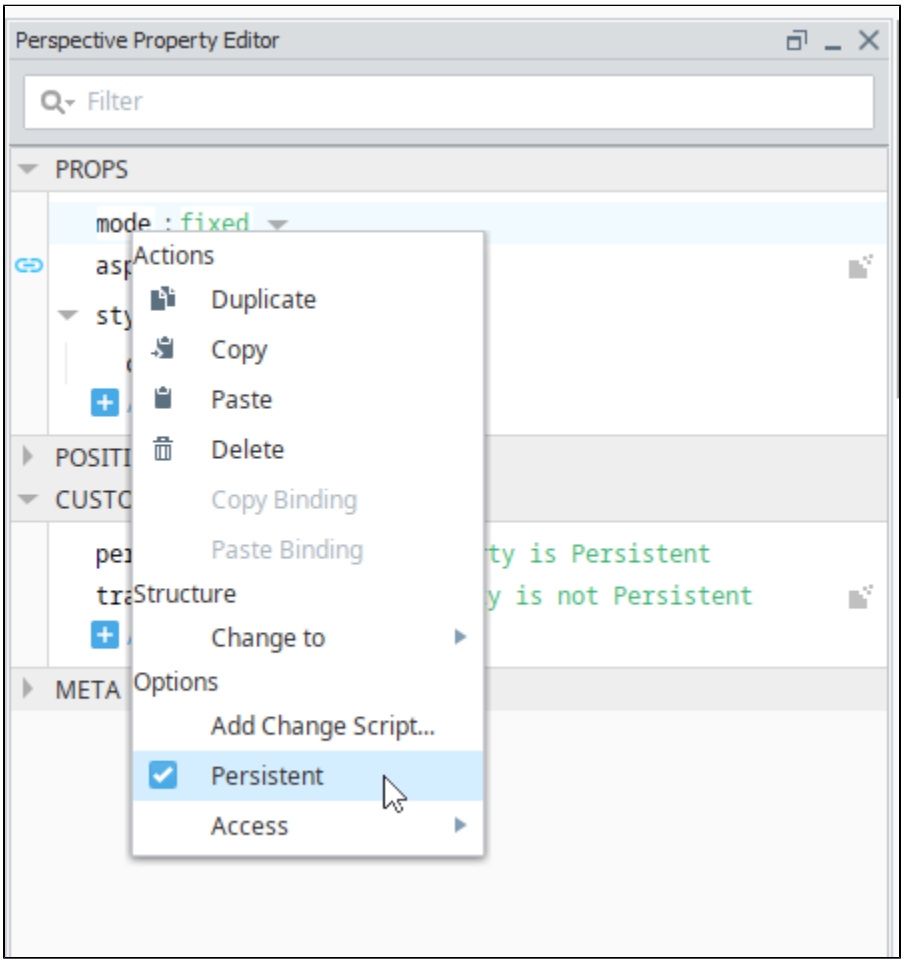
Persistent Properties

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

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



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



Persistence and User-Created Properties

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

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

Bindings and Persistence

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

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

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

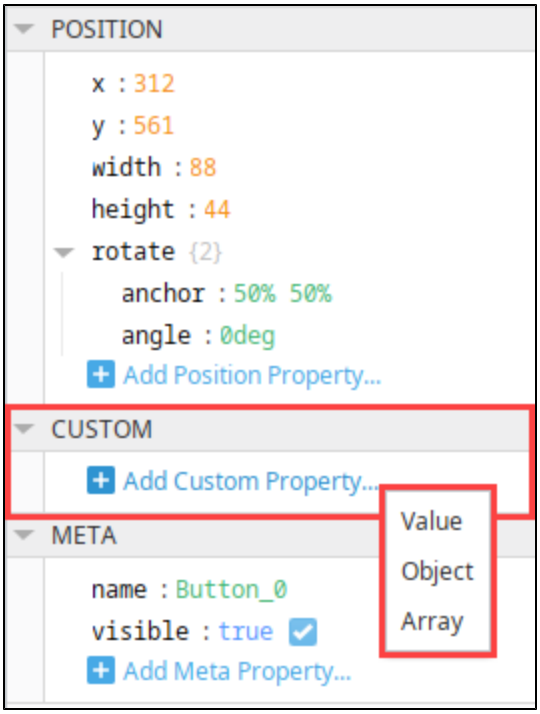
Custom Properties

User-created properties may be added to any property category such as the Props, Position, Custom and Meta Property Categories to enhance functionality. The Custom Category was designed as an ideal location in the Propety 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.

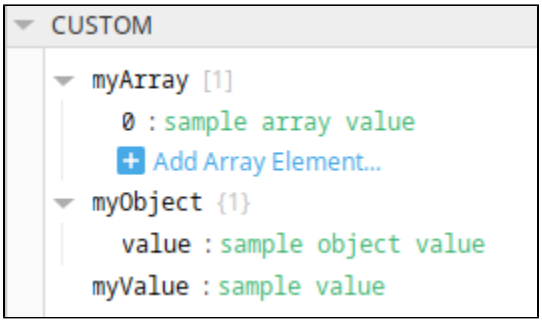
Creating Custom Properties

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

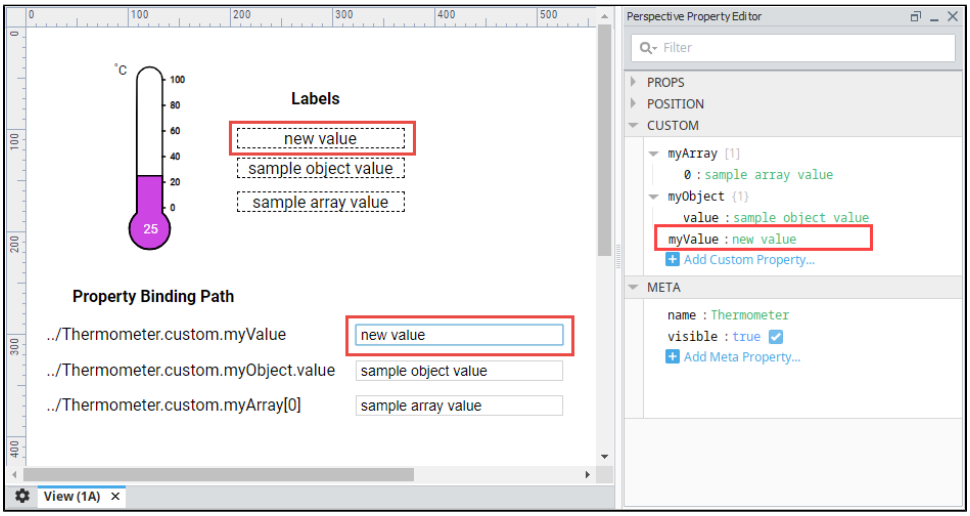
1. Custom Properties are created by clicking on the "Add Custom Property..." link, or by right clicking on a property in the Property Editor to bring up the Action Menu. There are three property types to choose from: Value, Object, and Array.



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.

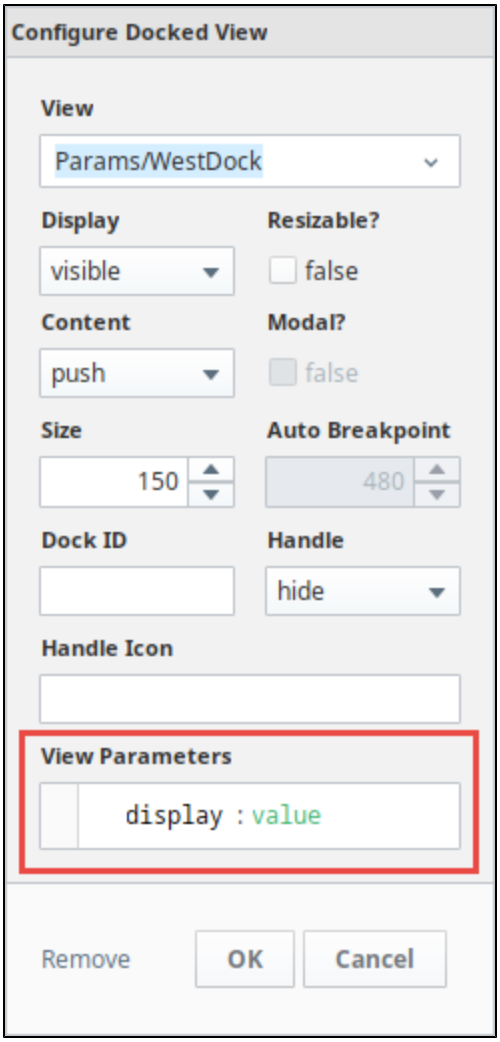
Description	Description	Data Type
`Name`	Name of the component used when navigating tree paths by name.	value: string
`Visible`	Whether or not the component should be visible.	value: boolean
`Rotate`	This feature was removed from Ignition in version 8.0.6 As of 8.0.6 the Rotate property has been moved to the Position Properties section of the Perspective Property Editor.	
`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

Params

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

Docked View

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



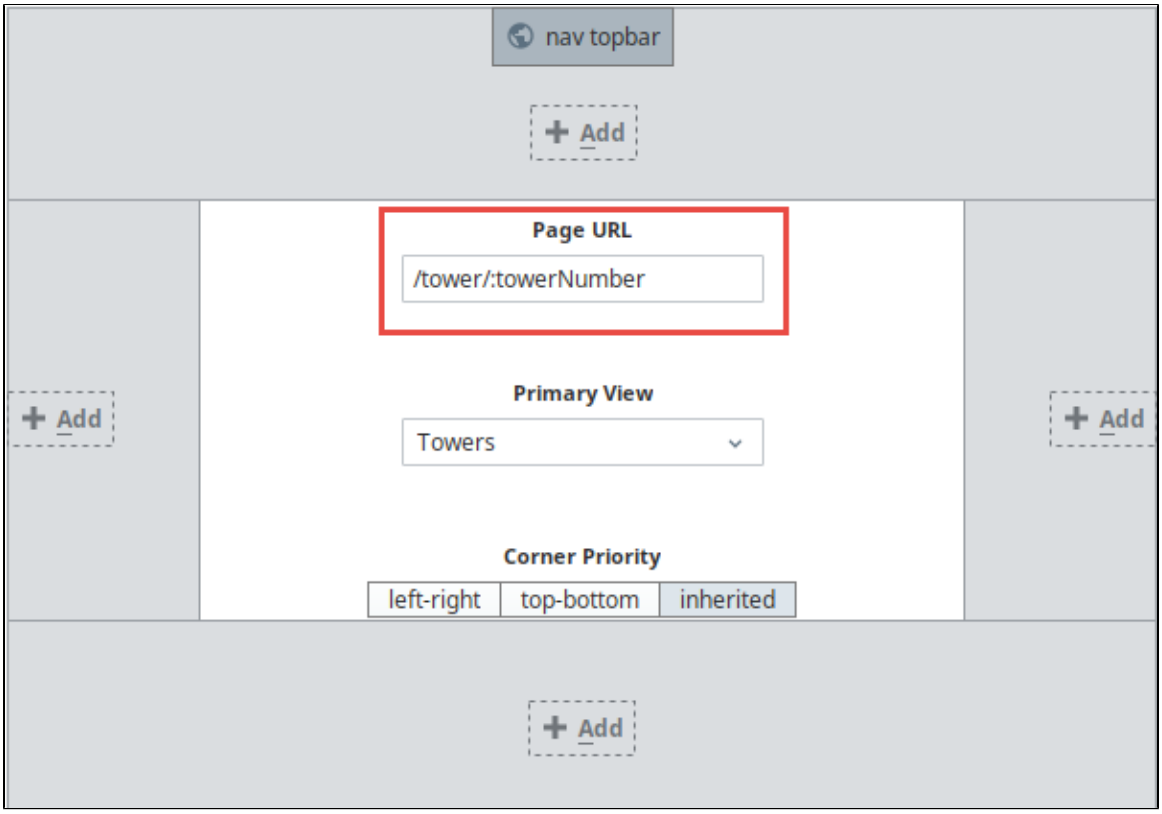
Embedded View

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

Page

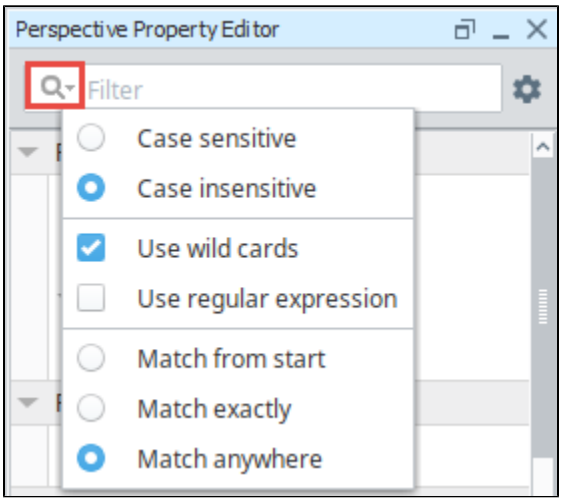
Passing parameters to a Page essentially means passing a URL parameter to a page. Parameters are used to allow a page to be mounted at a dynamic URL, allowing information in the URL to be interpreted as input parameters to the page's primary view. A primary view can see whatever parameters are passed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

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



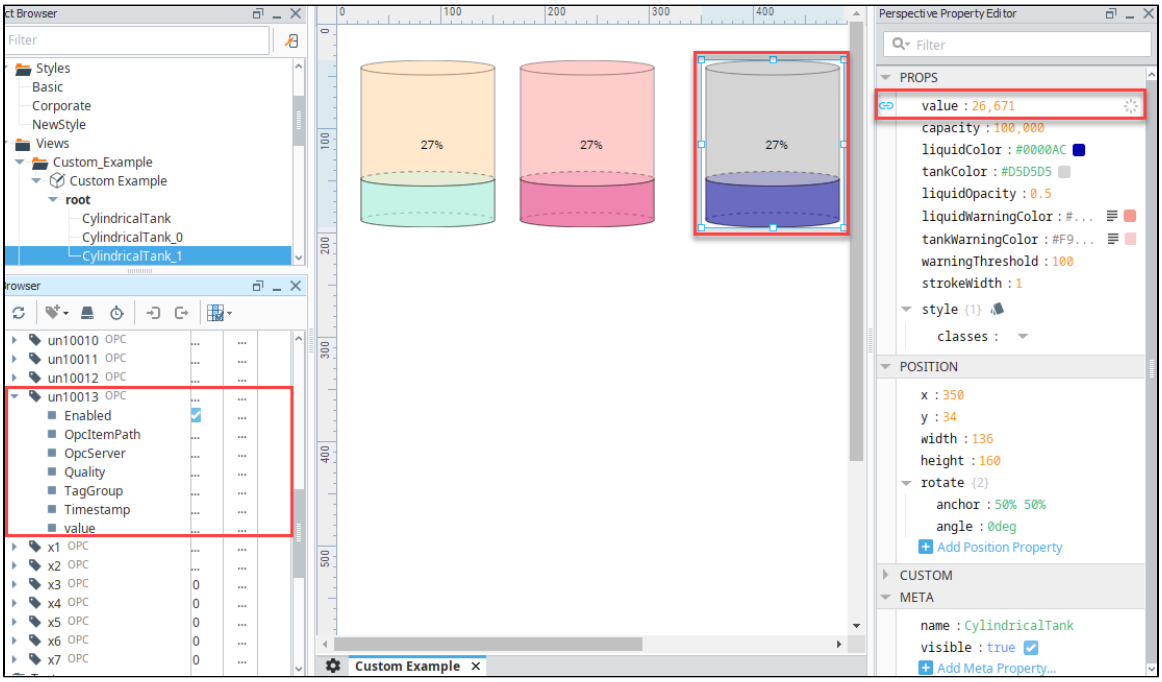
Search Filter

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.



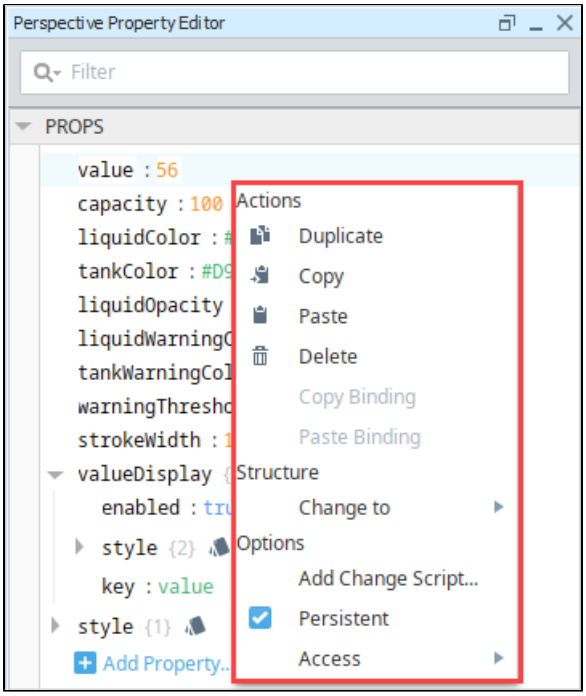
Styles

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.

Action Menu

Actions

Additionally, you can right-click on a property in the Perspective Property Editor to access an **Actions Menu**. This menu 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, as well as some more unique items.

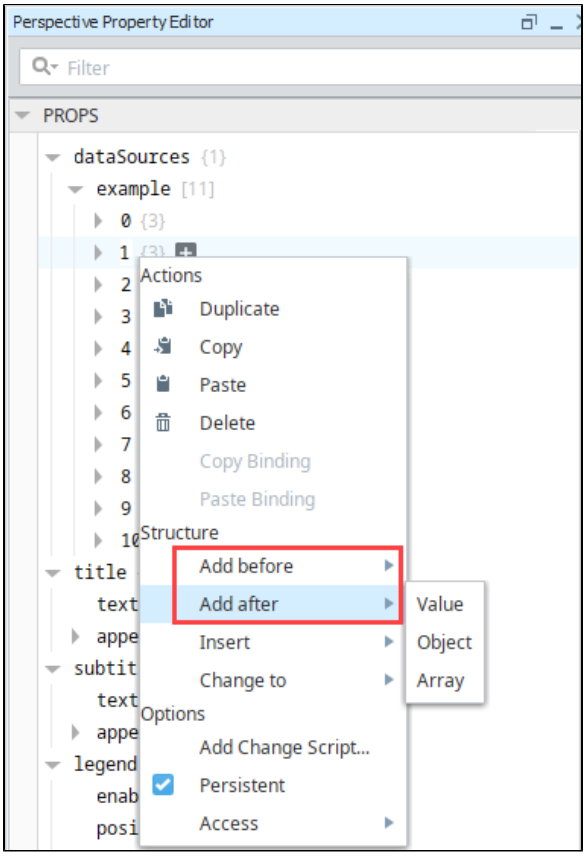


Structure

> The following feature is new in Ignition version **8.0.6**
> Click here to check out the other new features

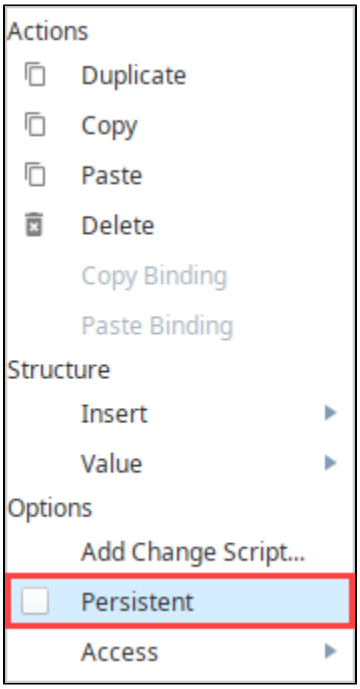
Insert: Allows you to insert a new property into the hierarchy. The "Add Before" and "Add After" Str ucture options in the Perspective Property Editor context menu are two unique items that are only visible when right-clicking on an element in either an Array or Object.

Value: Allows you to change the type of the selected property to a Value, Array, or Object



Options

Persistent: Determines if the property is Persistent or Transient.



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.

Related Topics ...

- Property Bindings in Perspective
- Views and Containers in Perspective
- Pages in Perspective

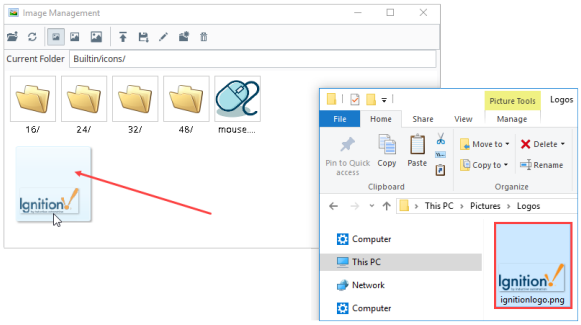
In This Section ...

Ignition Server Migration Best Practices

It is common for Ignition users to find themselves needing to migrate their Ignition Gateway to a new server or virtual machine. A few reasons for server migration include new hardware requirements or old hardware simply failing.

Images, SVGs, and Icons in Perspective

Images such as PNGs, JPGs, GIFs, or SVGs can be uploaded to the Image Management tool and used inside of containers in Perspective. The Image Management tool, available from the **Tools > Image Management** menu, provides an interface to upload, download, or select images.



The following feature is new in Ignition version **8.0.3**
Click here to check out the other new features

The Designer gives you the option to either save the image and link to its location or embed the image directly in the view. However, images larger than 100KB may degrade performance. Save and link is recommended for larger images.

⚠ The Image Management tool does not support bitmap files.

On this page

...

- Using the Image Component
- Drag and Drop an Image onto a View
- Manage Images in the Image Management Tool
  - Upload an Image to the Image Management Tool
  - Download Images from the Image Management Tool
- Import SVGs
- Embed an Image in a View
- Using Icons
  - Use a Custom Icon Repository
  - Example with Custom Icon Repository

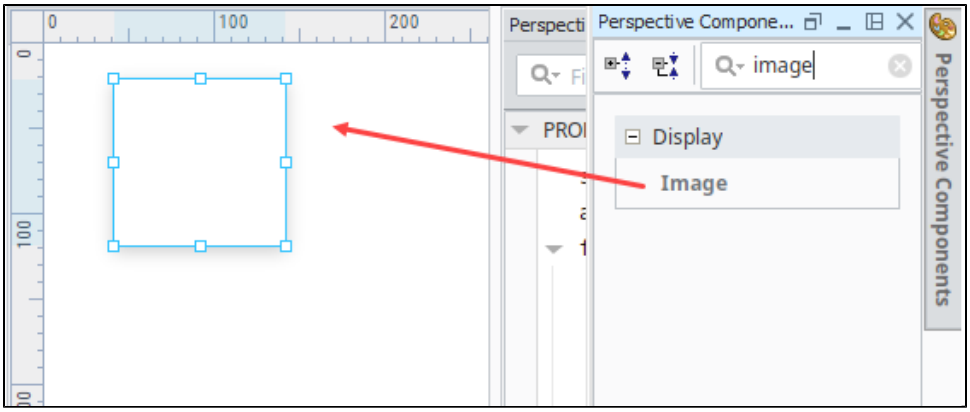
Images in Perspective

Watch the Video

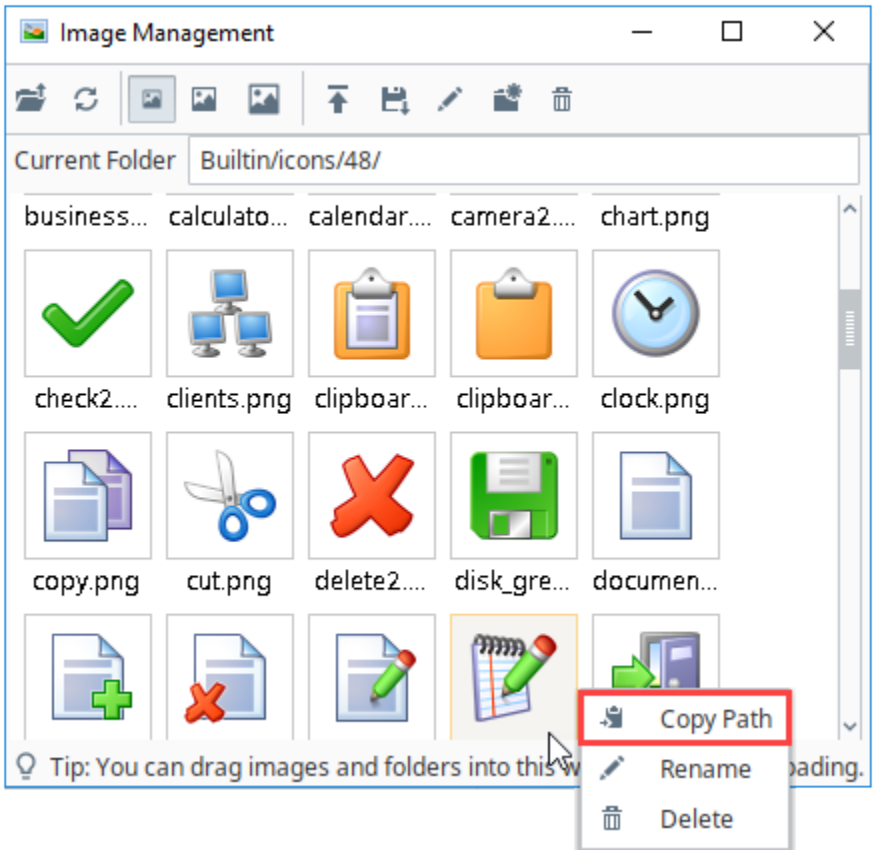
Using the Image Component

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

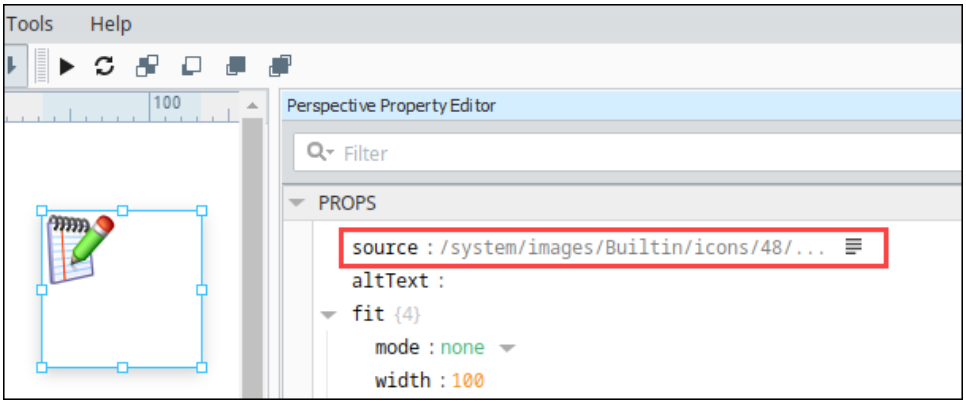
1. Drag an **Image** component from the component palette onto the container.



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



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



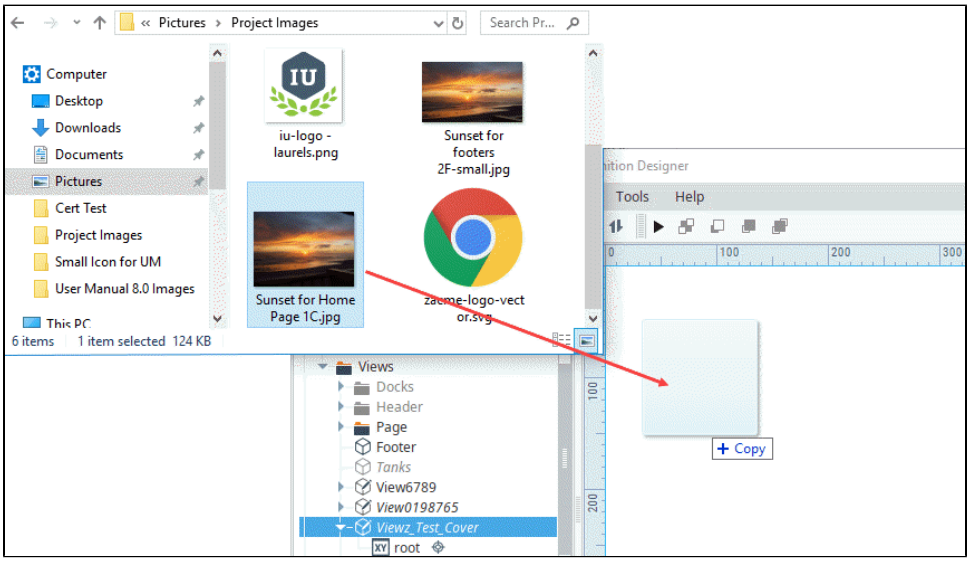
The image now appears inside the Image component on the container.

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

Drag and Drop an Image onto a View

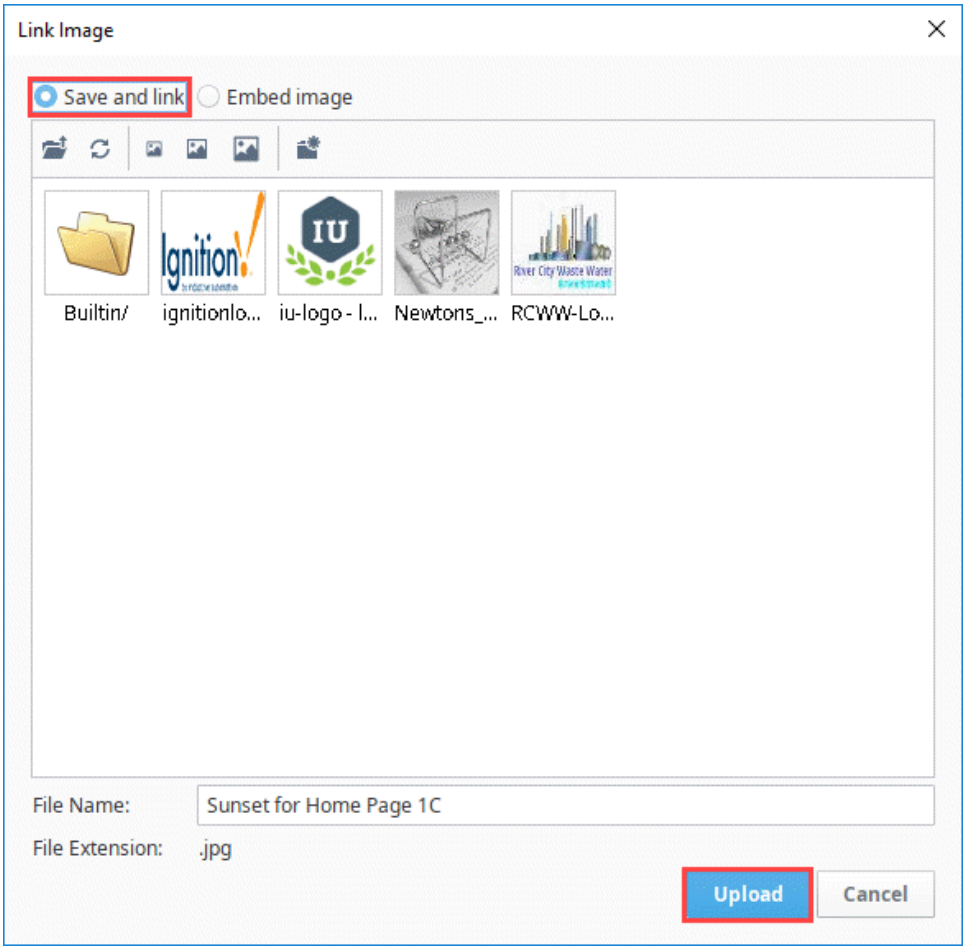
In Perspective, you can drag and drop images from your computer onto a view. In this example, we'll pull in a .jpg photo of a sunset to use as a background on a home page for our application.

1. In the Designer, open the view in which you want to put the image.
2. Locate the image on your computer and drag it onto the view. Note that we already resized this image in a graphics editing program so that it is the size we want.

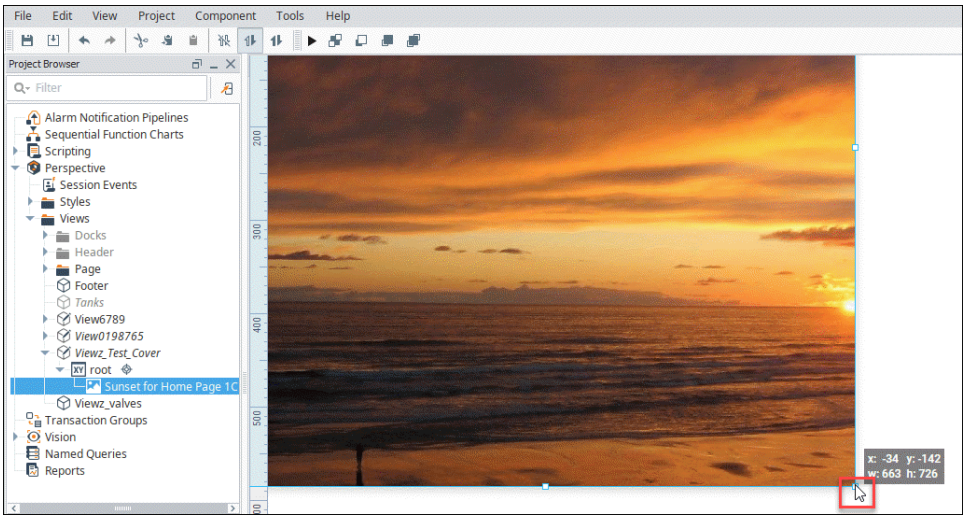


> The following feature is new in Ignition version **8.0.3**
> Click here to check out the other new features

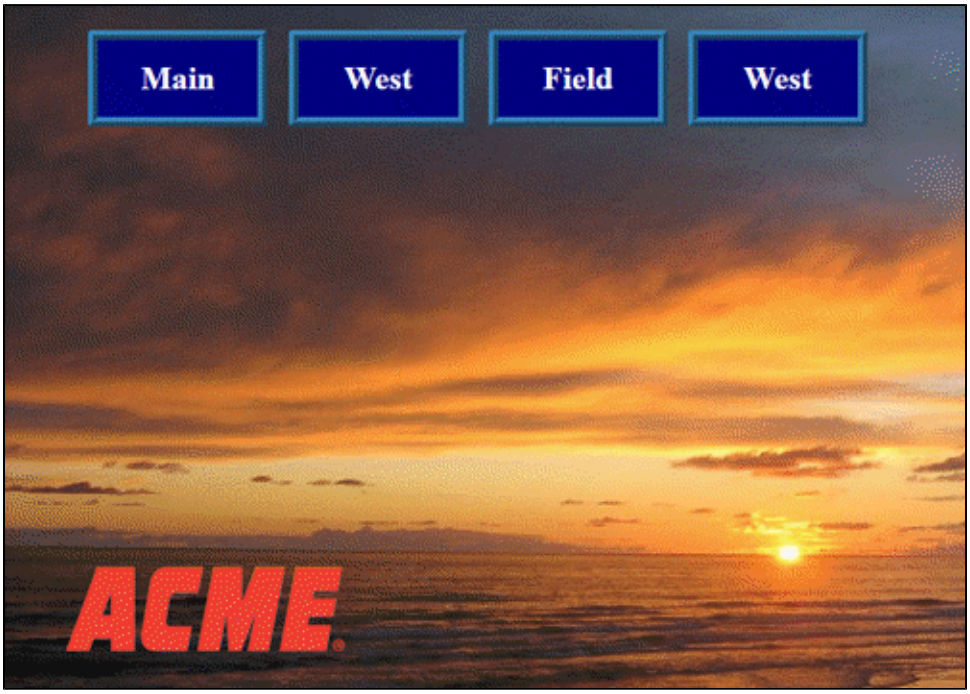
3. Starting in version 8.0.3, you have the option of saving the image as a link or embedding the image in the view. The Link Image screen is displayed. Choose the **Save and Link** radio button and then click **Upload**. Note that this will save the image in the Image Management Tool and it can be used repeatedly.



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



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

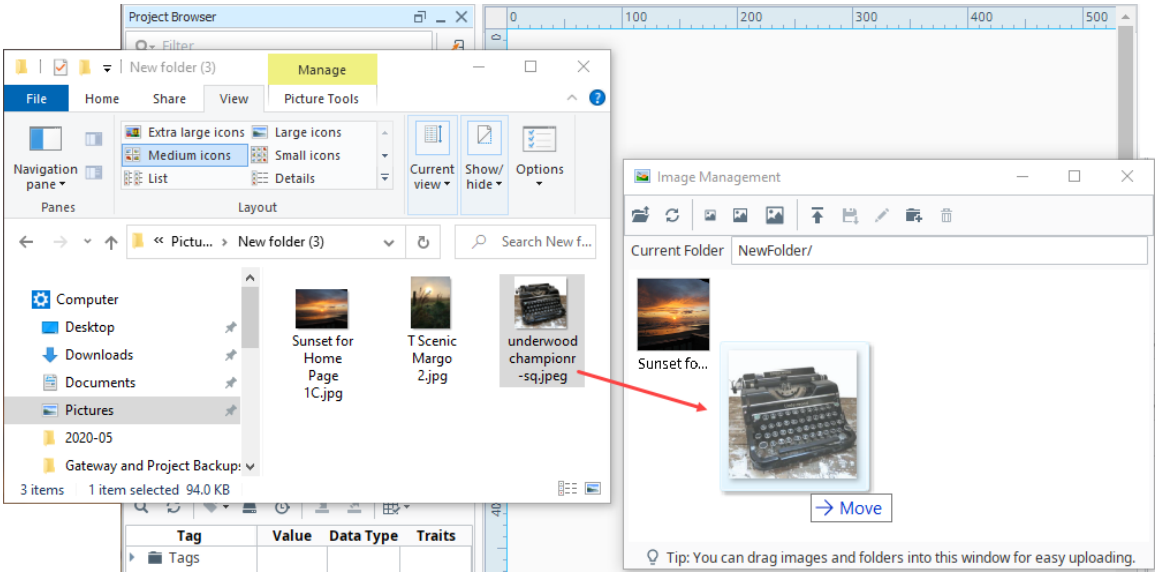


Manage Images in the Image Management Tool

There are additional ways to manage images into the Image Management tool.

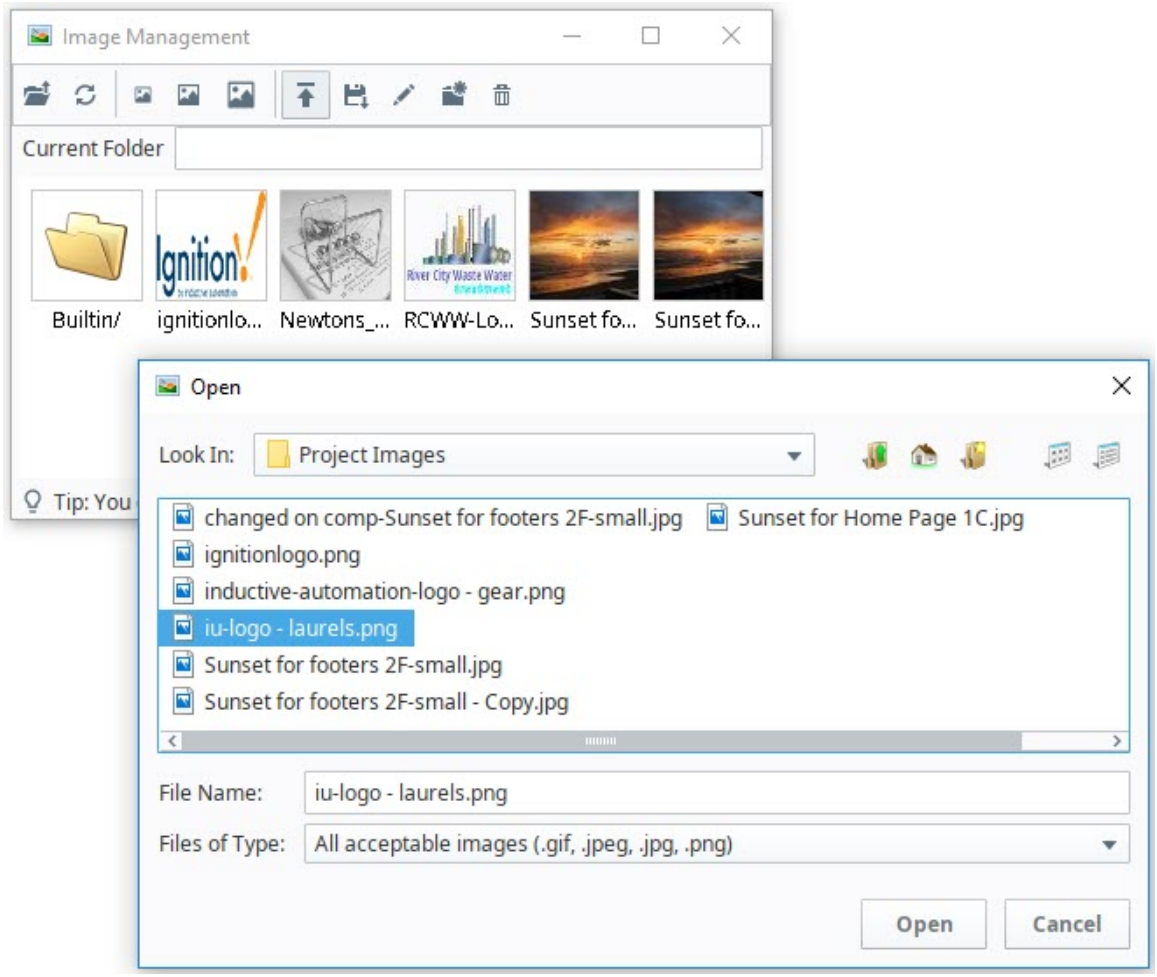
Upload an Image to the Image Management Tool

1. At the top of the Designer in the Menu Bar, select **Tools > Image Management**.
2. Next, drag and drop the image from the local file system directly onto the Image Management window.



Alternatively, the Image Management window has an Upload button to pass images in. Locate the directory you wish to upload the image. You may use the root folder, or create a new folder to keep your images organized.

1. In the Image Management tool, click on the **Upload New Image** icon. An **Open** dialog window appears.
2. Navigate to your image on the local system.
3. Select the image, then click **Open** to upload it.



You can also use a file path to a local image. Enter the file path in the source property for the Image component . This is done by prefixing the file path with `file:///` . An example Image Path would look like this:

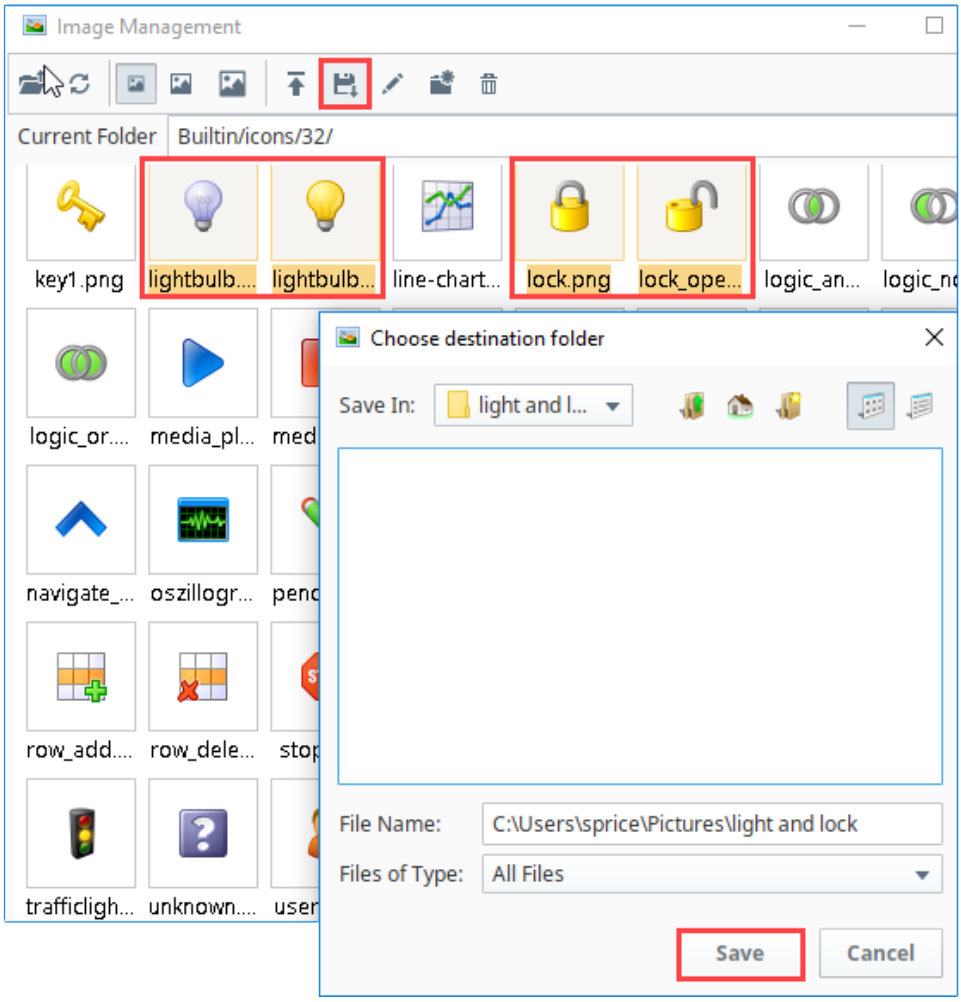
```
file:///C:/Users/Public/Public Pictures/chrysanthemum.jpg
```

It is important to understand that this will only work if the image is accessible from where the client is running. So if you access an image from the designer on the local machine, clients that launch elsewhere may not have the image stored in the same location.

Download Images from the Image Management Tool

Single images, as well as entire directories, may be downloaded from the Image Management tool. This is useful when migrating a project to another Gateway. Image downloads can be taken from either the **Image Management** or **Image Browser** windows.

1. At the top of the Designer in the Menu Bar, select **Tools > Image Management** .
2. Select the image, images, or folder you want and click the **Save As** icon.
3. Choose a destination folder for the downloaded files.



4. Click **Save**. All images and subfolders in the selected folder will be copied to the selected directory.

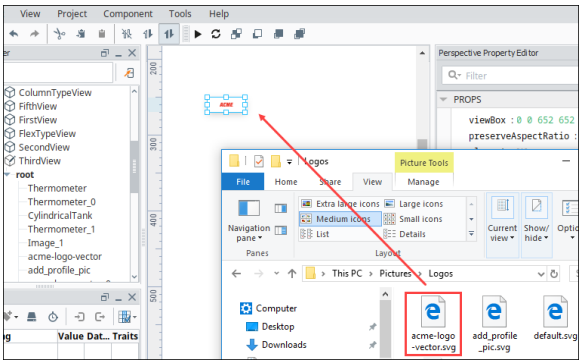
Import SVGs

Ignition can import SVGs (Scalable Vector Graphic) into Perspective containers. Once imported, SVGs can be resized and style elements can be applied to them.

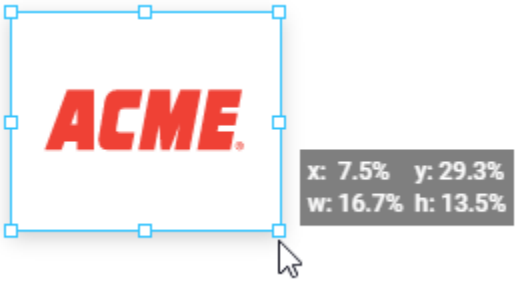
1. To use an SVG in your project, find the file on your local system.

> ⚠ When working with images found online, make sure to follow all applicable copyright laws.

2. Drag the SVG file directly onto the View or Container you want the SVG to appear on.



3. Starting in version 8.0.3, you have the option of saving the image as a link or embedding the image in the view. The Link Image screen is displayed. Choose the **Save and Link** radio button and then click **Upload**. Note that this will save the image in the Image Management Tool and it can be used repeatedly.
4. Sometimes the way the SVG imports may result in the SVG appearing very small. You can resize the SVG by dragging one of the handles or entering width and height properties in the Property Editor.



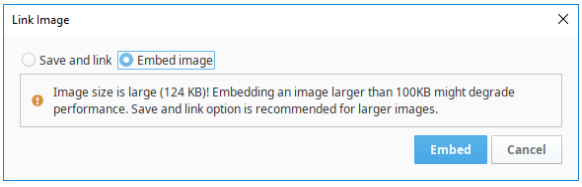
5. You can also apply Style properties such as a border and background color to the image. For more information, see Styles and Style Classes.

The following feature is new in Ignition version **8.0.3**
Click here to check out the other new features

Embed an Image in a View

If you don't want the image to be brought into the Image Management tool you can embed the image instead.

1. Locate the image on your computer and drag it onto the view.
2. After dragging the image onto the view, click the **Embed image** radio button. Note that if the image is larger than 100KB, you see a warning message in the confirmation popup.



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

Using Icons

Icons can also be placed into Perspective containers using the Icon component. Several components, such as the Horizontal Menu, have icon properties that enable you to embed icons into the component. The materials icon library is a great source for icons, see https://fonts.google.com/icons?selected=Material+Icons. For additional information, see Perspective - Icon.

Use a Custom Icon Repository

While materials icon library is a great source for icons, if you have a custom library of icons you can also set it up to be accessible by Ignition. When using a custom icon repository, the format for the file should use the following pattern. Note that each icon is place in an outer enclosing SVG tag that defines values for the viewbox attribute, which is mandatory for icons:

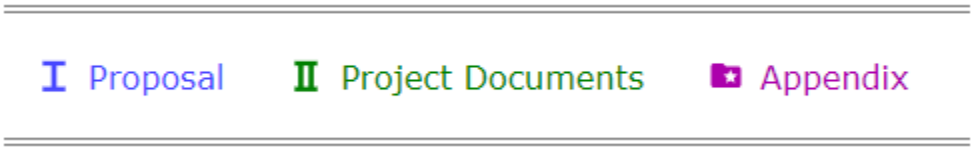
```
<?xml version="1.0" encoding="utf-8"?>

<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>
```

Once you have the file built, you need to place it in the icons directory within your Ignition Installation Directory, for example

`Ignition Installation Directory\data\modules\com.inductiveautomation.perspective\icons`

Example with Custom Icon Repository

In this example we'll make a custom repository file with icons. The name of the file is `example.svg`. We'll then use the icons in this file and other repositories to create a Horizontal Menu component as follows;



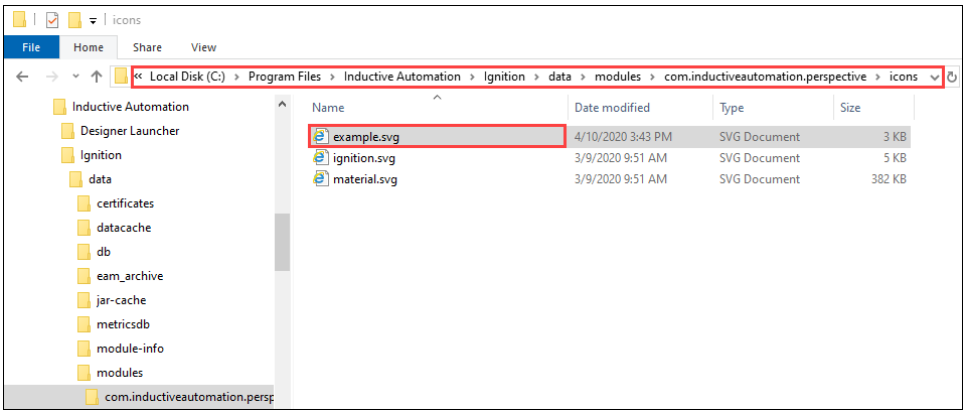
1. First we created the repository file in xml.

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

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

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

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

        <!--roman numeral one icon-->
        <svg viewBox="0 0 24 24">
                <g id="one">
```

```
                        <rect
                           id="rect3723" width="15.09025" height="3.0526078" x="4.4548745" y="
2.4385488" />
                        <rect
                           id="rect3723-3" width="15.09025" height="3.0526078" x="4.4548745" y="
19.510113"/>
                        <rect
                           id="rect3755" width="3.4557824" height="16.990929" x="10.272108" y="
3.5045362" />
                </g>
        </svg>
        <!--roman numeral two icon-->
        <svg viewBox="0 0 24 24">
                <g id="two">
                        <rect
                           id="rect3723" width="15.09025" height="3.0526078" x="4.4548745" y="
2.4385488" />
                        <rect
                           id="rect3723-3" width="15.09025" height="3.0526078" x="4.4548745" y="
19.510113" />
                        <rect
                           id="rect3755" width="3.4557824" height="16.990929"  x="6.8435426" y="
3.5045362" />
                        <rect
                           id="rect3755-9" width="3.4557824" height="16.990929" x="13.700686" y="
3.5045002"/>
                </g>
        </svg>
</svg>
```

2. Once created, we saved the file in the following directory:

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



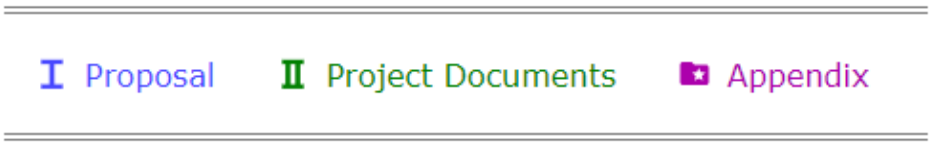
3. Now we can use the icons in this icon repository in our components. For this example, we put a Horizontal Menu component in a Coordinate view.
4. We use two icons from the new example repository and one from the materials repository. We set the properties as follows for the Horizontal Menu component:

Top Level Properties

Property	Value
props.items.0.enabled	true
props.items.0.icon.path	example/one
props.items.0.icon.color	#4747FF
props.items.0.label	Proposal
props.items.0.style.color	#4747FF
props.items.0.style.fontFamily	Verdana

props.items.1.enabled	true
props.items.1.icon	example/two
props.items.1.color	#008000
props.items.1.label	Project Documents
props.items.1.style.color	#008000
props.items.1.style.fontFamily	Verdana
props.items.2.enabled	true
props.items.2.icon.path	material/folder_special
props.items.2.icon.color	#AC00AC
props.items.2.label	Appendix
props.items.2.style.color	#AC00AC
props.items.2.style.fontFamily	Verdana
props.style.borderBottomStyle	double
props.style.borderLeftStyle	none
props.style.borderRightStyle	none
props.style.borderTopStyle	double
props.style.borderTopWidth	4
props.style.borderBottomWidth	4

5. When we put the Designer into Preview ▶ mode. The component appears as follows:



Related Topics ...

- Image Management Tool
- Symbol Factory

Localization in Perspective

Localization in the Perspective module utilizes terms 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.

On this page

...

- Selecting a Language
- Switching Languages within a Session
- Related Topics ...

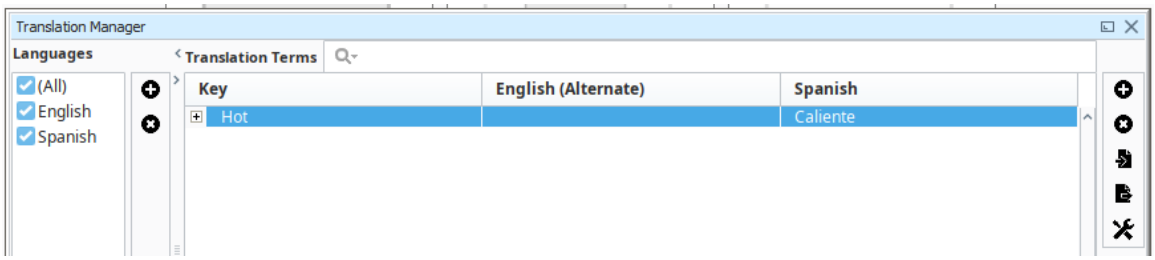
Selecting a Language

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

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

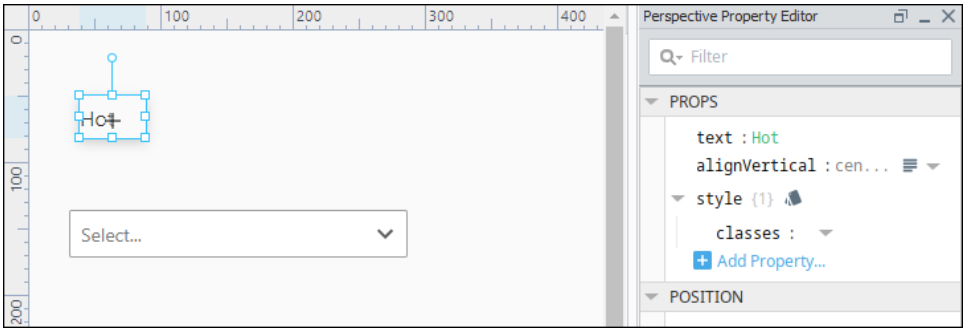
Switching Languages within a Session

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".

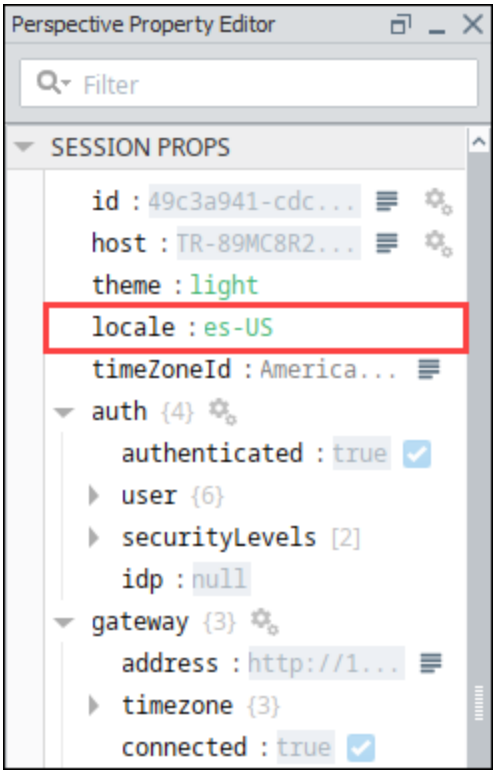


Now we'll setup a view where the user can choose between English and Spanish.

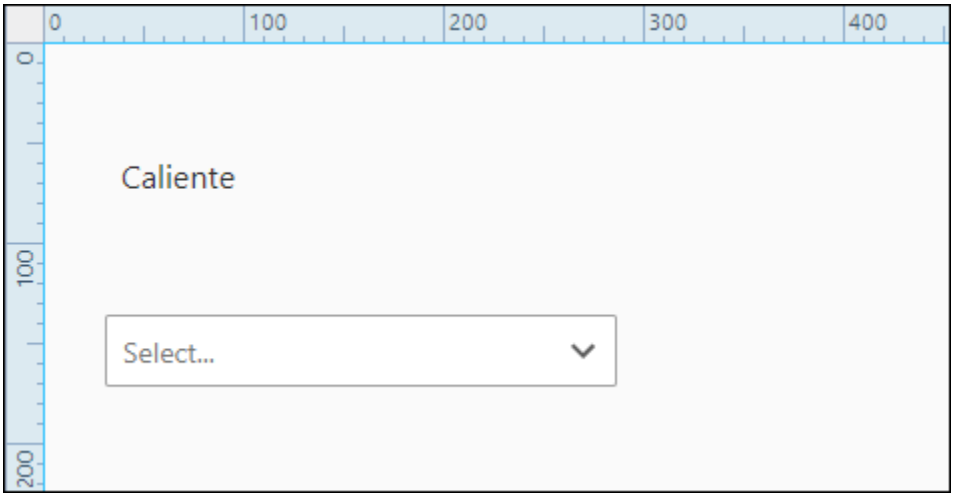
1. Create a new view and put a Label component and a Dropdown component on the view.
2. Set the text property of the label to "Hot."



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).

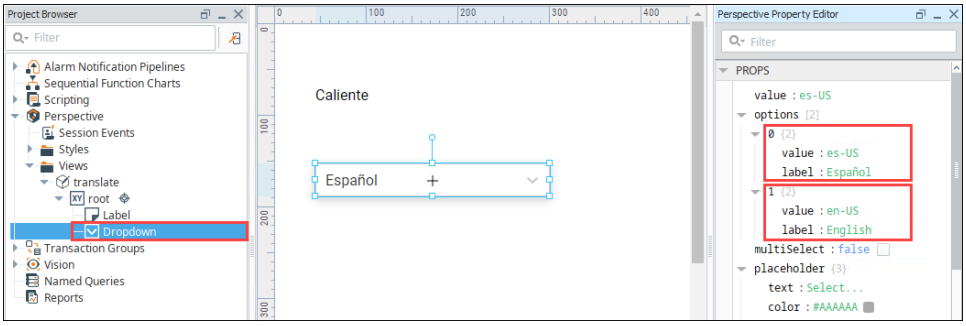


This results in anything in your session that contains the word "Hot" to be translated to its Spanish translation.

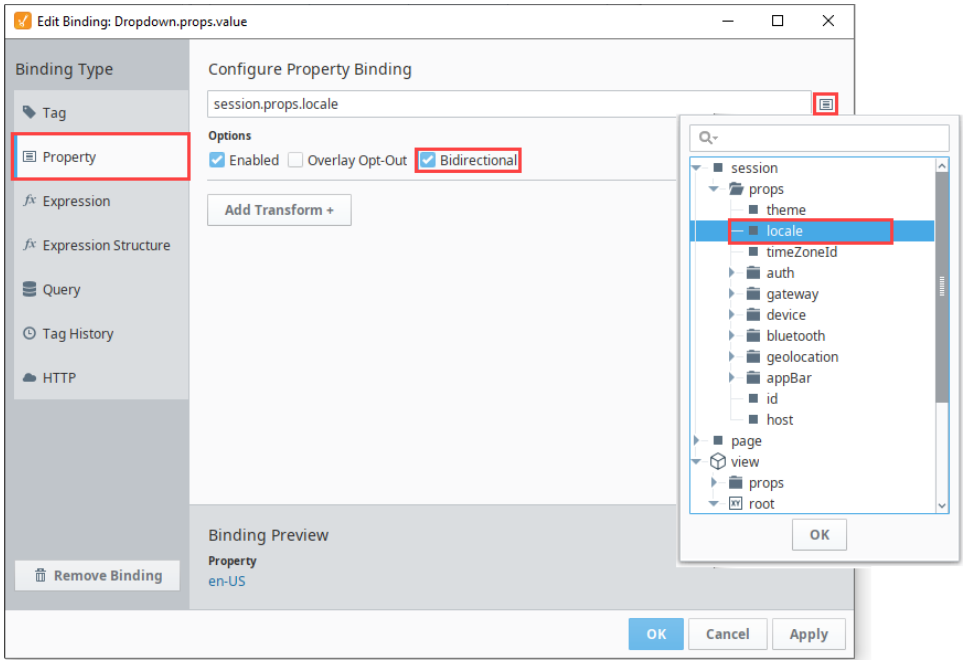


Similarly, if you set the locale session property back to "en-US", Ignition will translate your session text back to English.

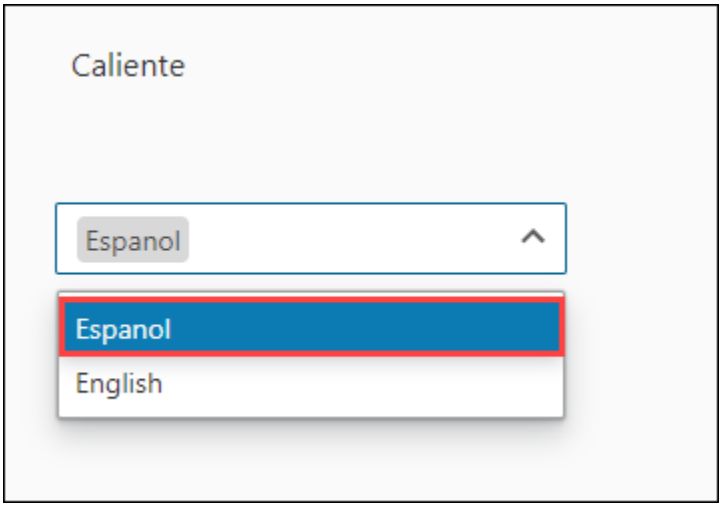
4. Next we'll set up the Dropdown component with two options configured as shown below where the Spanish option is tied to the "es-US" language tag and the English option is tied to the "en-US" language tag.
   options.0.value: **es-US**
   options.0.label: **Espanol**
   options.1.value: **en-US**
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



Related Topics ...

- Localization and Languages

Component Events and Actions

In Perspective, events and actions are some of the fundamental building blocks of project functionality. Actions give you the ability to respond to specific user inputs (such as mouse, keyboard, and touch inputs), as well as broad session events (like the beginning and end of the session) in many different ways. Thus, *actions* are a response to *events*.

Example uses for events and actions include:

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

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

On this page

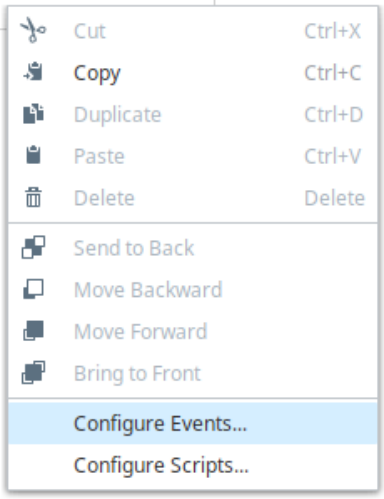
...

- Configure Events and Actions
- Perspective Events
- Action Types
  - Shared Action Options
  - Accelerometer Action
  - Alter Logging Action
  - Dock Action
  - Fullscreen Action
  - Login and Logout Actions
  - Navigation Action
  - Popup Action
  - Refresh Action
  - Scan Barcode Action
  - Scan Ndef NFC Action
  - Script Action
  - Theme Action

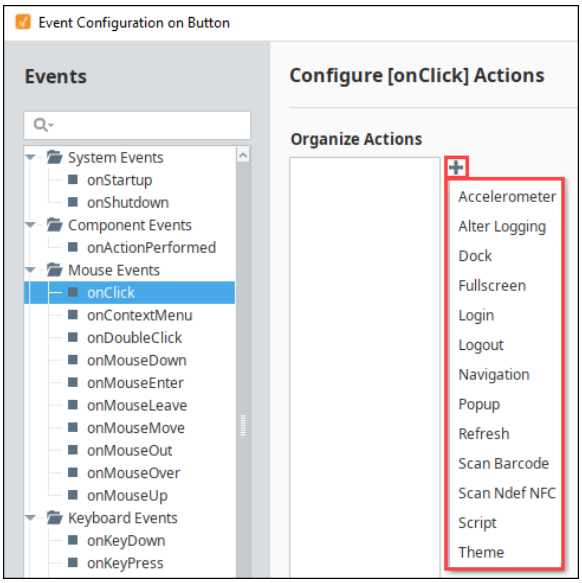
Configure Events and Actions

To configure an event and action on a component:

1. Select the component.
2. **Right click** on the component, and select **Configure Events**.



3. Choose an event from the left hand side. Note that 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 then click **OK**.

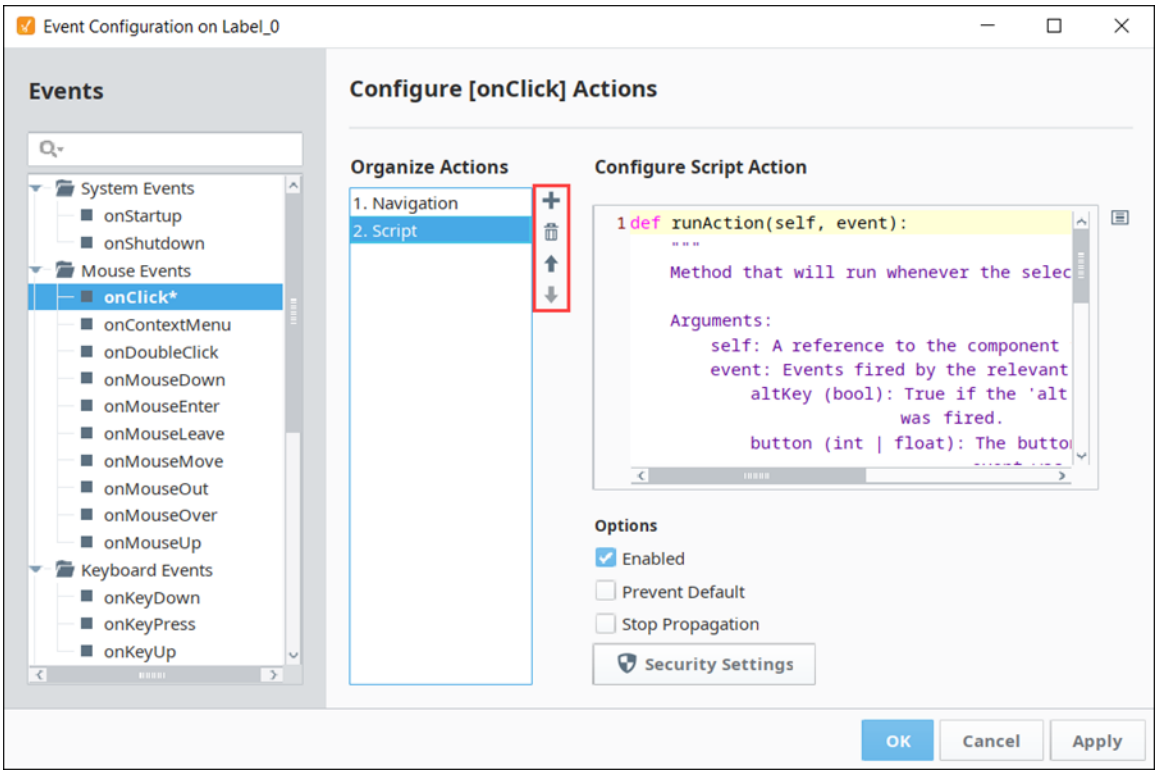
Perspective Events

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.

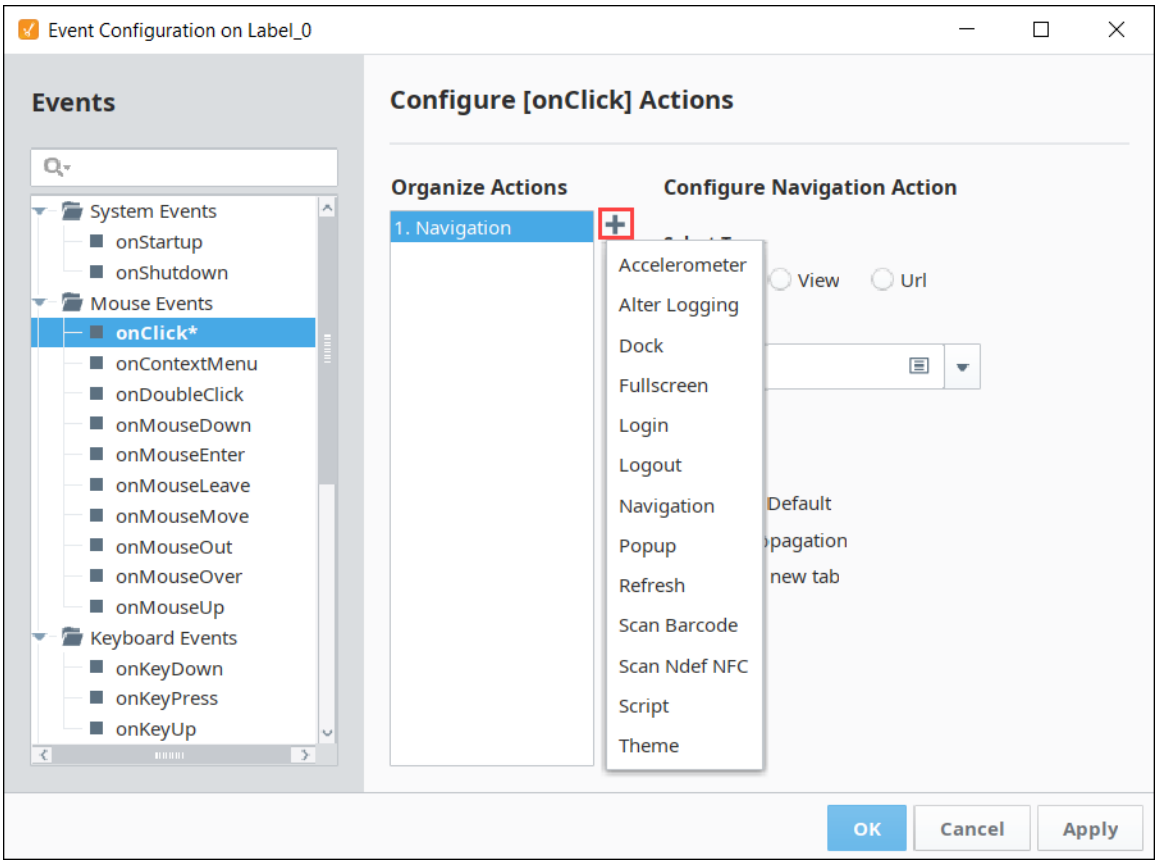
Action Types

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 table below are shared across all action types.

Enabled	Specifies whether the action should be used or ignored.
Prevent Default	The following feature is new in Ignition version **8.0.15** Click here to check out the other new features 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	The following feature is new in Ignition version **8.0.15** Click here to check out the other new features Prevents events from higher up in the component hierarchy from triggering when the selected event triggers.

Accelerometer Action

Retrieves accelerometer data from the device's accelerometer (a common feature on smartphones). This is a Native App Action, designed to help in gathering data from a mobile device. Each action needs to be configured in two parts:

1. The action is run on a mobile device, indicating that a type of data should *begin* to be gathered.
2. As the data is gathered, or once it's finished being gathered, it is sent to the Gateway to be handled by the corresponding *Session Event*.

⚠ 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. Note that 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 that a **session** object is already provided to that event script.
Off	Turns off accelerometer data recording.

Alter Logging Action

Perspective sessions log their session activity and errors to the *logs of the browser they run in*. Alter Logging allows us to change how verbose this logging is. This is useful for session debugging.

Action Setting	Description
Remote Logging Enabled	Indicates whether the browser logs should also be sent to the Gateway to log in the Gateway logs. Note that for all messages to be visible, the **Perpective.Client** logger must be set to the same logging level as the level indicated in this action.
Set Logging Level	Dictates how verbose the session's logging should be. A logging level of **all** (or **trace**) shows all possible records, while a level of **off** shows none. Options are: • all • trace • debug • info • warn • error • fatal • off

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. The dock ID can be whatever you'd like. We'll need it here in order to run the action.

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

The following feature is new in Ignition version **8.0.10**
Click here to check out the other new features

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. Note that 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.

Navigation Action

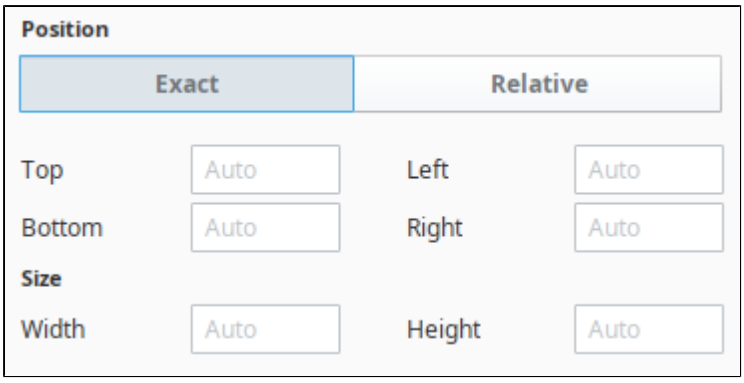
The Navigation action allows you to navigate to different views, pages, or URLs from an event. The Navigation action has several modes. Each mode allows for a different type of navigation and different options. The following table lists the types of navigation:

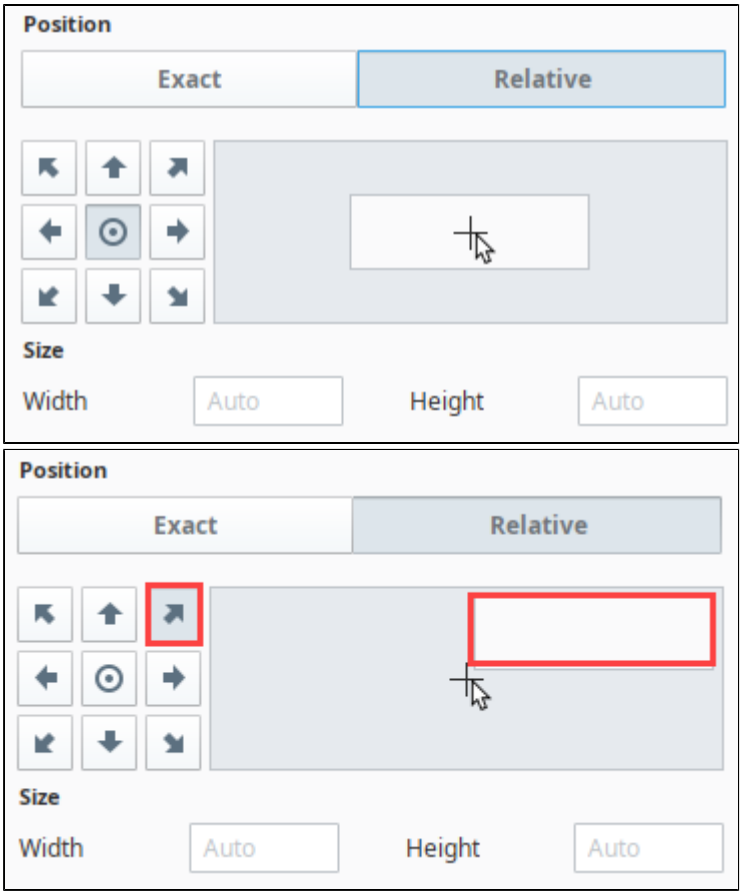
<table>
<tr><th>Type</th><th>Description</th></tr>
<tr><td>Page</td><td>Navigates to a separate page.
<table>
<tr><th>Type</th><th>Description</th></tr>
<tr><td>Set Page</td><td>A string denoting the page URL of the target page.</td></tr>
<tr><td>Open in new tab</td><td>Specifies whether the newly opened page should replace the current page, or open in a new browser tab.</td></tr>
</table></td></tr>
<tr><td>View</td><td>Replaces the current main View with a new main View.
<table>
<tr><th>Type</th><th>Description</th></tr>
<tr><td>Select View</td><td>The path to the view that should be used.</td></tr>
<tr><td>Parameters</td><td>A set 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</td></tr>
</table></td></tr>
<tr><td>Url</td><td>Navigates to an external web address.
<table>
<tr><th>Type</th><th>Description</th></tr>
<tr><td>Enter Url</td><td>The URL that the action should navigate to. Example: <code>https://inductiveautomation.com/</code></td></tr>
</table></td></tr>
</table>

Open in new tab:	Specifies whether the newly opened page should replace the current page, or open in a new browser tab.

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	The following feature is new in Ignition version **8.0.6** Click here to check out the other new features Exact Positioning controls where in the session the popup should appear, and how large it should be. If no specifications are given, the popup will open centered at its default size. • **Top**, **Left**, **Bottom**, and **Right** control how far the popup should be offset from each margin of the session. • **Width** and **Height** specify how large the popup should appear. 
Position Relative	The following feature is new in Ignition version **8.0.6** Click here to check out the other new features 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.



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 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.

This is a Native App Action, designed to help in gathering data from a mobile device. Each action needs to be configured in two parts:

1. The action is run on a mobile device, indicating that a type of data should *begin* to be gathered.
2. As the data is gathered, or once it's finished being gathered, it is sent to the Gateway to be handled by the corresponding *Session Event*.

Action Setting	Description
Barcode Type	Indicates the format of the barcode to be scanned. **Any** can be used to catch all barcodes.
Context	Provides the opportunity to pass a custom object through to the Barcode Scanned Session Event. Note that a **session** object is already provided to that event script.

Scan Ndef NFC Action

Allows the Perspective app to catch any scans by the phone using the NFC data exchange format (Ndef), which is then sent to the Gateway and handled by the NFC Ndef Scanned Session Event. This suppresses any default behavior of the phone in catching the scan.

> ⚠ This action is designed specifically for the Ignition Perspective App for Android and iOS devices. If it is run in a browser session, it will be ignored.

Action Property	Description
Single Mode	Listens for a single NDEF scan to send.
Continuous Mode	Listens indefinitely for NDEF scans, which are sent to the gateway as they are received.
Off Mode	Turns off listening for NDEF scans.

Script Action

Write a script that happens on the event specified. See Perspective Component Methods and system.perspective Functions for more details on how to configure script actions.

> The following feature is new in Ignition version **8.0.5**
> Click here to check out the other new features

Script actions contain a built-in "event" object, that further contains values pertaining to the underlying event. As of 8.0.5 these values and descriptions are displayed in the docstring.

Theme Action

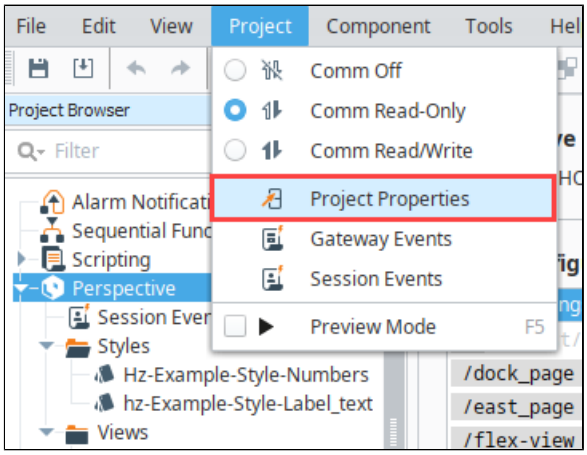
Changes the theme for the session.

Action Setting	Description
Select Theme	Theme that will be used, for example dark, light, light-cool etc. Dropdown list of all available themes.

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 and configuring how the sessions receive updates.

To access the Project Properties, in the Designer, click on **Project** tab on the menu bar. Then select **Project Properties**.

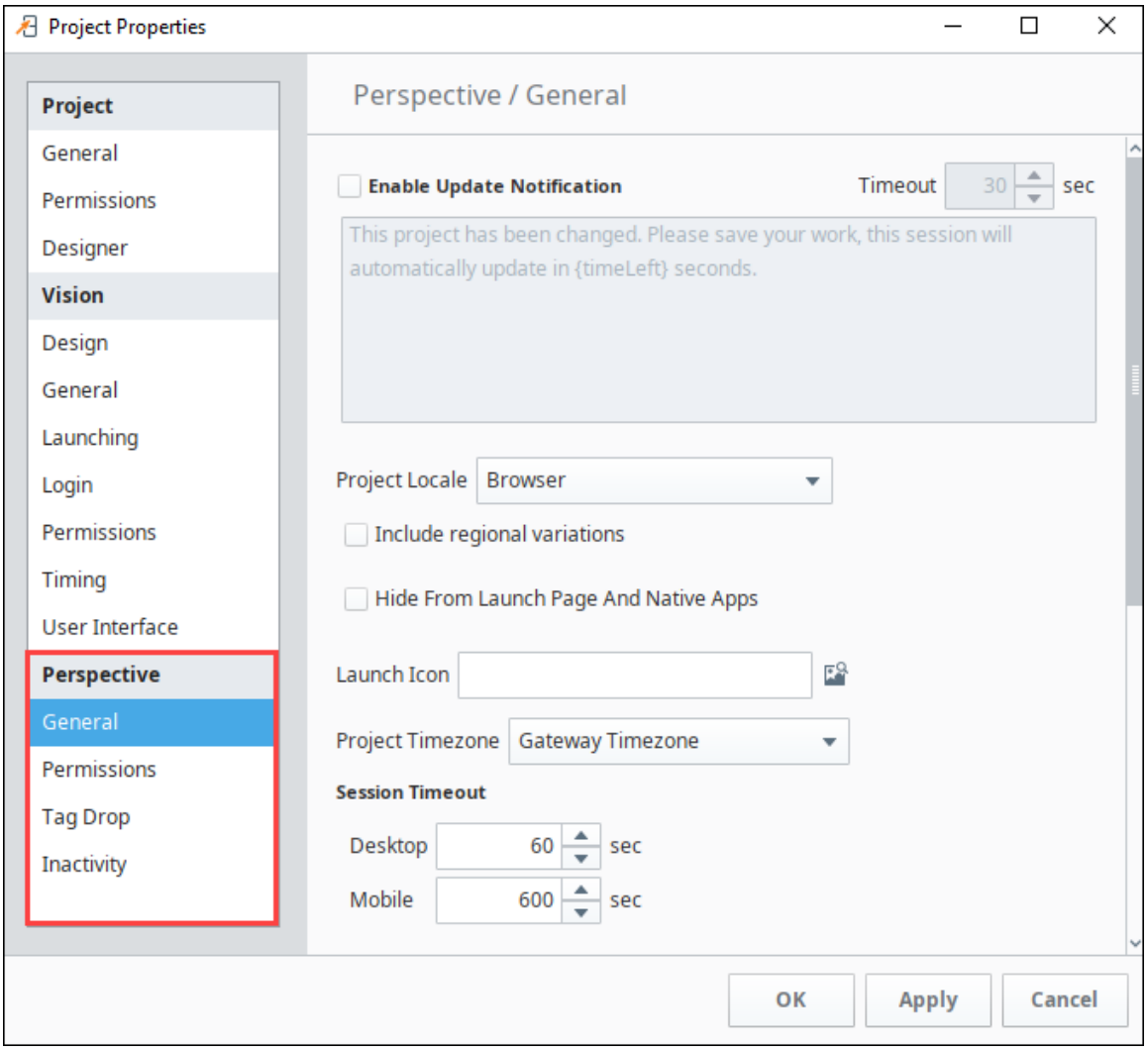


Project properties span several functional areas each containing settings applicable to that area. Scroll down to the Perspective section.

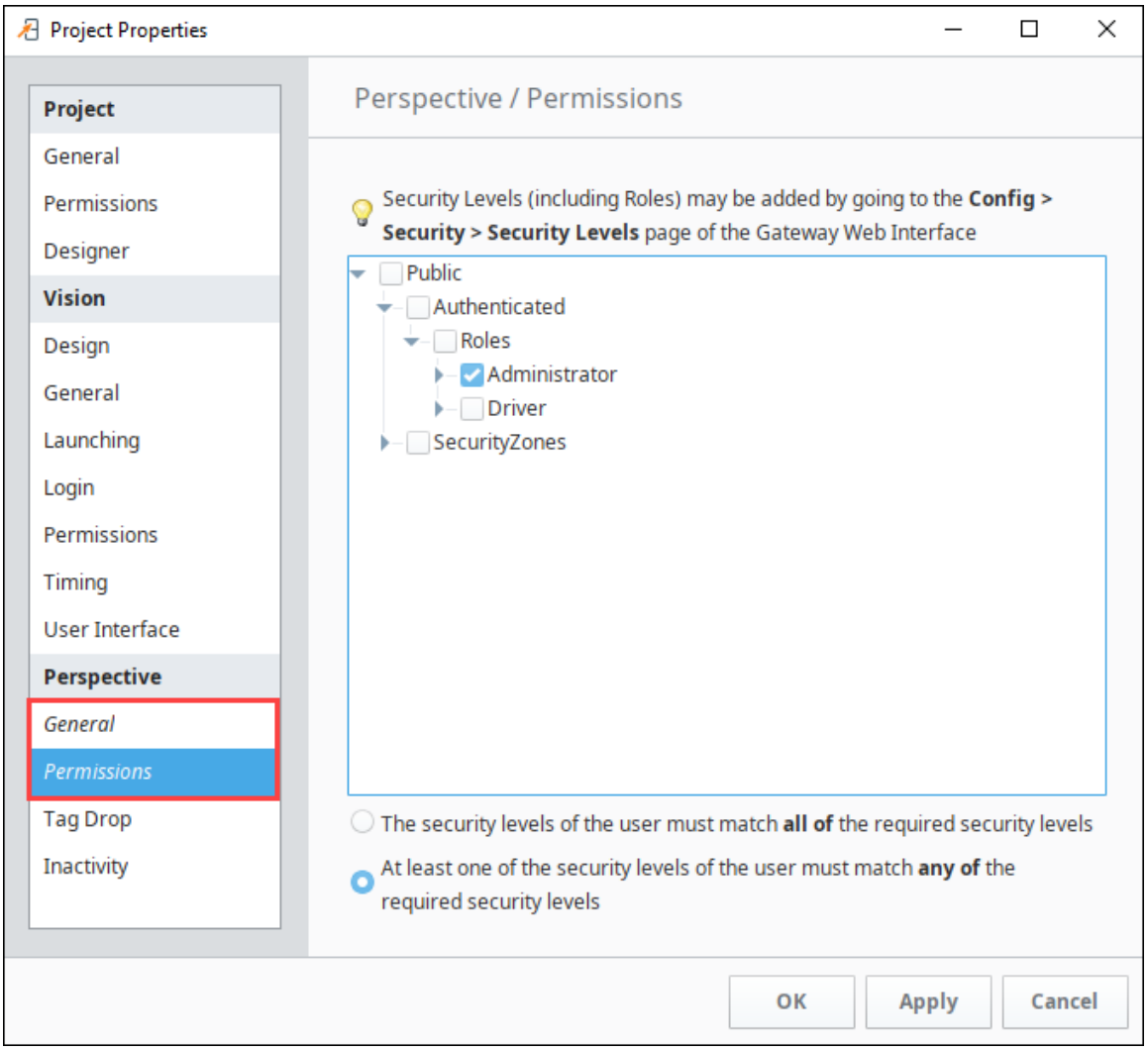
On this page

...

- Perspective General Properties
- Perspective Permissions Properties
- Perspective Tag Drop Properties
- Perspective Inactivity Properties



When properties in a section have been updated but not saved yet, the section heading will change to italicized text. In the following example, changes have been made in the the Perspective General and Permissions properties but they have not been saved or applied yet.



Perspective General Properties

These general properties apply to the Perspective Sessions.

General	
Property	**Description**
Enable Update Notification	Enables notifications for sessions when the project is changed. When notification is enabled, this is the message that will be displayed. The token {timeLeft} will be replaced with the seconds remaining until update. If false, update will be immediate.
Project Locale	Select a project locale.
Include regional variations	Enables regional variations for the locale.
Hide from Launch Page and Native Apps	The following feature is new in Ignition version **8.0.6** Click here to check out the other new features 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 desktop shortcut. This needs to be a path to an image that has been uploaded to the Gateway. Use the browse button to choose or upload a new image.
Project Timezone	Timezone for this project. Dropdown list includes the Gateway timezone, Session timezone, or specific timezones around the globe.

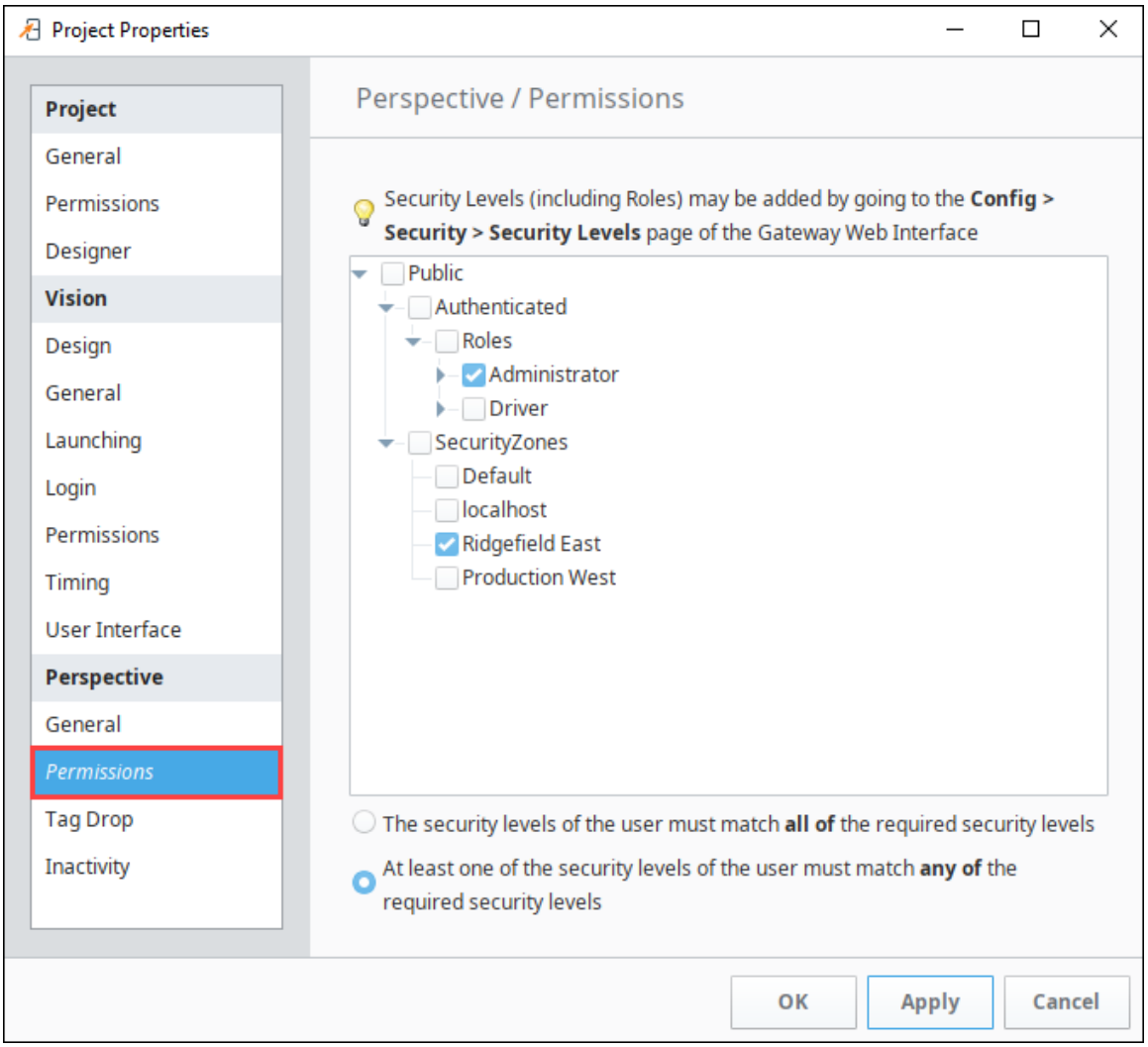
Identity Provider	Identity Provider (IdP) for this project. Dropdown list of available IdPs. **The following feature is new in Ignition version 8.0.6** **Click here to check out the other new features** After release 8.0.6, the Identity Provider setting will only remain in this location if your Perspective Project already has an IdP set for it. If you change this property to none, it will no longer remain in this location. It moves to the Project Properties > Project General. For new installations, the Identity Provider property is now located in Project Properties > Project General. For more information, see Project Properties.
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	The following feature is new in Ignition version **8.0.3** Click here to check out the other new features Message to be displayed if system.perspective.closeSession is invoked.
Page Closed Message	The following feature is new in Ignition version **8.0.4** Click here to check out the other new features Message to be displayed if system.perspective.closePage is invoked.

Perspective Permissions Properties

The Perspective Permissions properties restrict project access to specific security levels and security zones.

Permissions	
Property	Description
(Security Levels Tree)	Interactive tree that shows the current authenticated roles and security zones.
(First radio button)	If selected, then the security levels of the user must match **all of** the required security levels
(Second radio button)	If selected, then at least one of the security levels of the user must match **any of** the required security levels.

In the following example, a user must have the Administrator security level and Ridgefield East security zone to be able to access this project.



Perspective Tag Drop Properties

Tag Drop properties provide a way to link certain data types of Tags to commonly used components when dragging-and-dropping a Tag onto a view. This is done in **Project Properties** under **Tag Drop** in three steps: select the data type, select the component to create, and select the bindings that are created on that component.

- **Data Type Configuration:** This setting determines which components show up in the popup list when dragging a Tag into a view.
  You can switch between the different Tag data types to see what sorts of components are allowed to be dropped for each of those types. Add or remove new entries in the list using the **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.

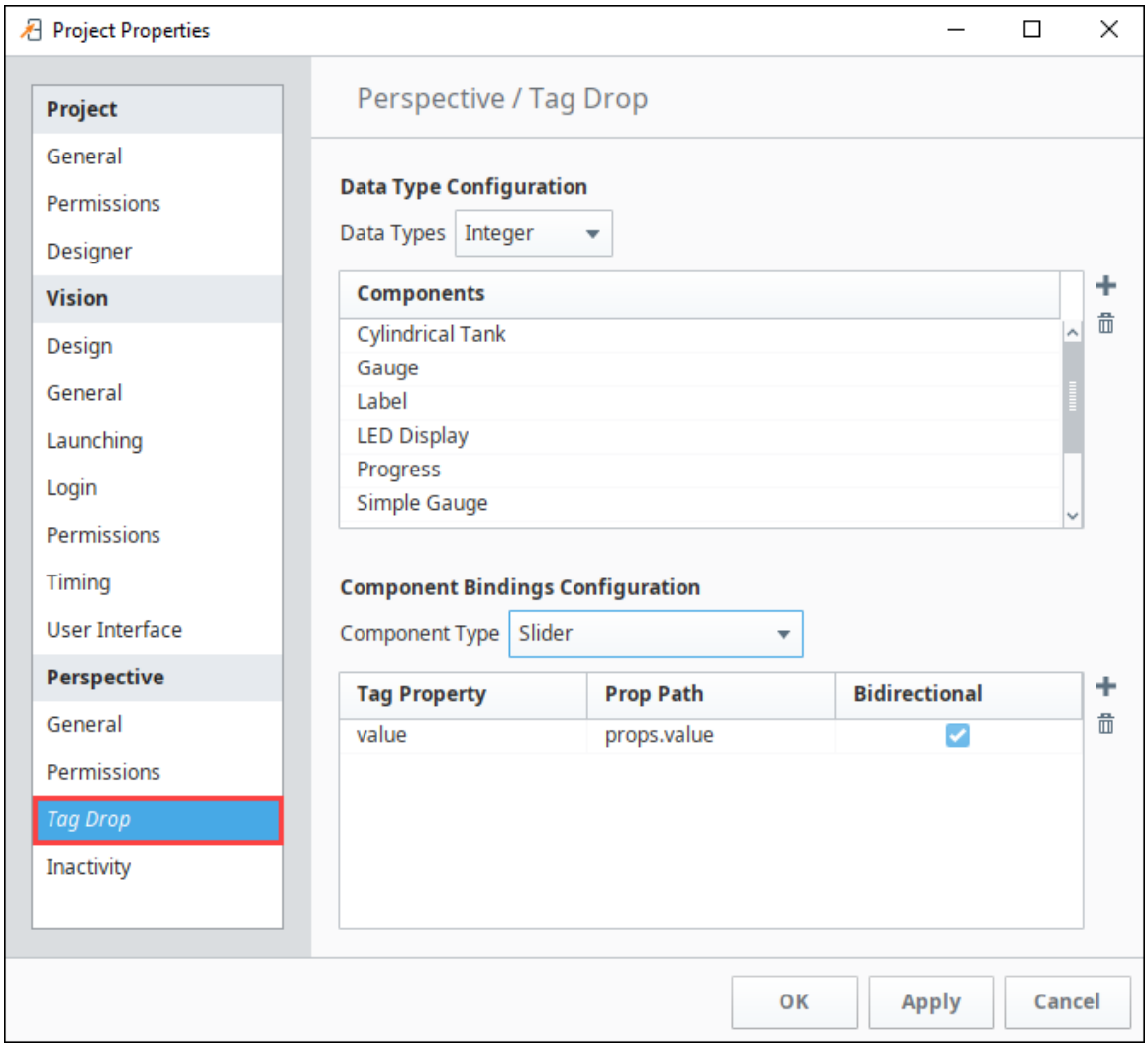
⚠ The Component Binding Configuration table values are tied to the component currently selected in the Component Type dropdown, they are not related to what is selected in the Data Type Configuration table above.

The following feature is new in Ignition version **8.0.6**
Click here to check out the other new features

Dropping tags onto existing components can now create bindings on those components if the data type of the tag matches a Tag Drop configuration in the project. Tag Drop configuration is what determines if a binding is configured or not.

The following feature is new in Ignition version **8.0.5**
Click here to check out the other new features

The Tag Drop interface enables the **Bidirectional** setting by default when adding Input Components.



Tag Drop	
Property	**Description**
DataType Configuration	This section is to set up a list of components to show for each data type that can be dragged onto the view.
Data Types	Dropdown list of the available datatypes in Ignition. Choose a datatype to see a list of components associated with it.
Components	List of components associated with selected data type.
Component Bindings Configuration	This section is to set up bindings when a Tag is dropped.

Component Type	Dropdown list of the available components in Perspective.
Tag Property	The name of a Tag's property to be used in a binding. "`value`" is most commonly used.
Prop Path	The property path on the component where the binding will be created. props.value or props.text are most commonly used.
Bidirectional	Check box to indicate if the binding should be bidirectional. The following feature is new in Ignition version **8.0.5** Click here to check out the other new features The **Bidirectional** property is set by default when adding Input Components.

The following feature is new in Ignition version **8.0.16**
Click here to check out the other new features

Perspective Inactivity Properties

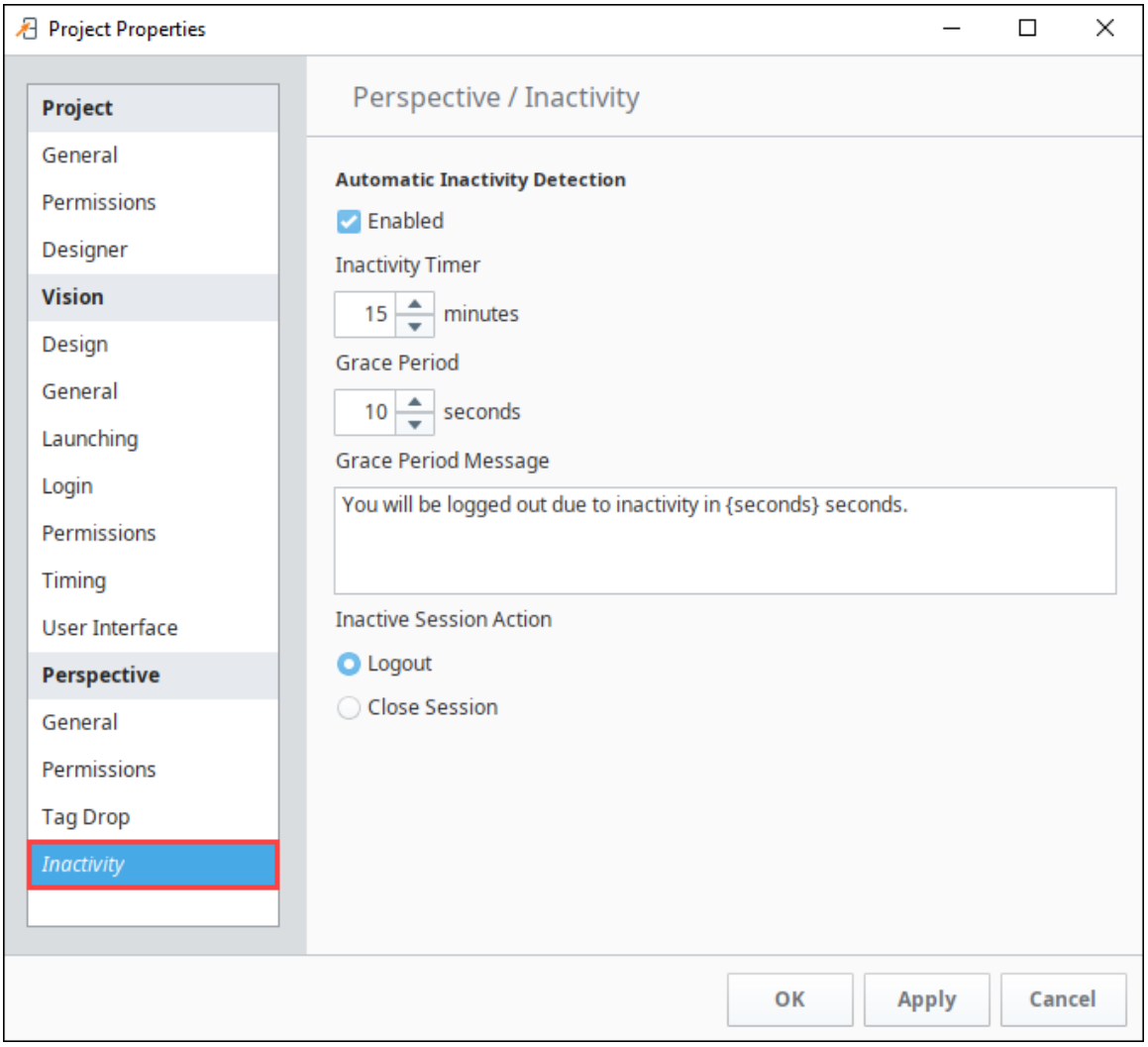
As of 8.0.16, Perspective has Inactivity Timeout settings in Project Properties. You can set an Inactivity timer that either closes the Perspective session or logs the user out if no activity is detected after a specified number of minutes. Activity is considered opening a new tab, clicking, typing. 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.

When enabling this inactivity feature, it is highly recommended that you also enable the "Always ask the IdP to re-authenticate users by default" setting under Project Properties, as this will require that any user attempting to log into a perspective session will have to provide their credentials first.

The following are the properties for the Inactivity Timeout:

Inactivity	
Property	**Description**
Enabled	Whether the automatic inactivity detection option is enabled.
Inactivity Timer	Time (in minutes) in which a session can be inactive before the user is logged out or the session is closed.
Grace Period	Grace period (in seconds) after the Inactivity Timer time has passed but before the user will be logged out or the session closed.
Grace Period Message	Message to be displayed before a session becomes inactive. Use the placeholder {seconds} to indicate remaining time. (Optional.)
Inactive Session Action	Action that occurs when the session becomes inactive. Options are: • Logout - Log the user out of the session. Note that the user will still be logged in with the • Close Session - Close the session.


Project Properties
Project
General
Permissions
Designer
Vision
Design
General
Launching
Login
Permissions
Timing
User Interface
Perspective
General
Permissions
Tag Drop
Inactivity
Perspective / Inactivity
Automatic Inactivity Detection
Enabled
Inactivity Timer
15
minutes
Grace Period
10
seconds
Grace Period Message
You will be logged out due to inactivity in {seconds} seconds.
Inactive Session Action
Logout
Close Session
OK
Apply
Cancel

Styles

Perspective components have Style properties that enable you to customize the look and feel of components on the screen through various properties. These styles are based on Cascading Style Sheets (CSS), a style sheet language used for describing the presentation of a document or webpage. CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. The Perspective module uses the popular standard HTML5/CSS3 technology for its user interface (UI) layer.

Using CSS, the Perspective Style options for components are both detailed and flexible. Styles are used in multiple places, and components can accept style information from multiple sources. There are many CSS settings available; they are all described in the Style Reference.

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

> The following feature is new in Ignition version **8.0.8**
> Click here to check out the other new features

As of version 8.0.8, the Style Editor has a more robust user interface. This includes a new Applied Styles panel that displays the style names and settings as you add them to a component.

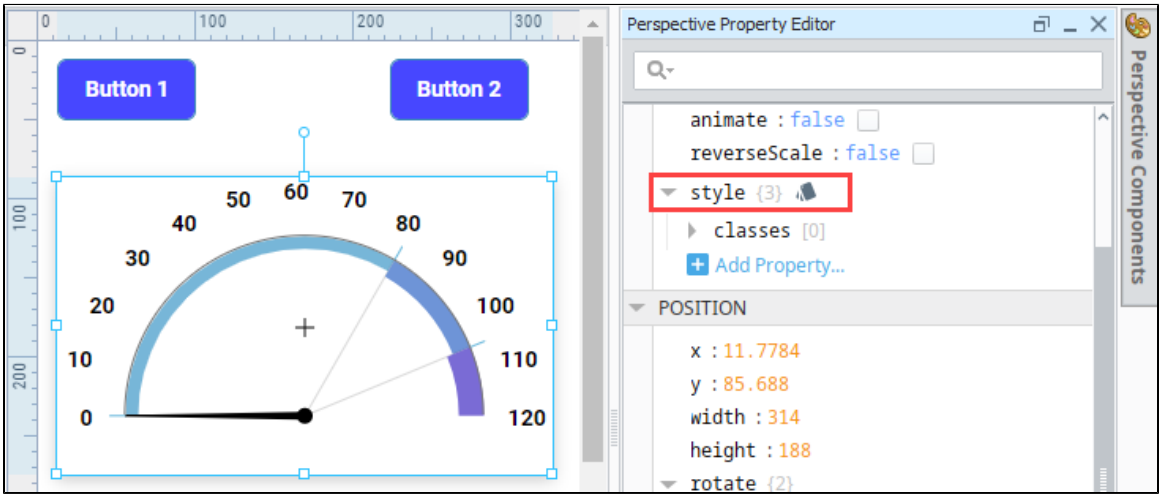
On this page

...

- Style Editor
- Style Example

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.

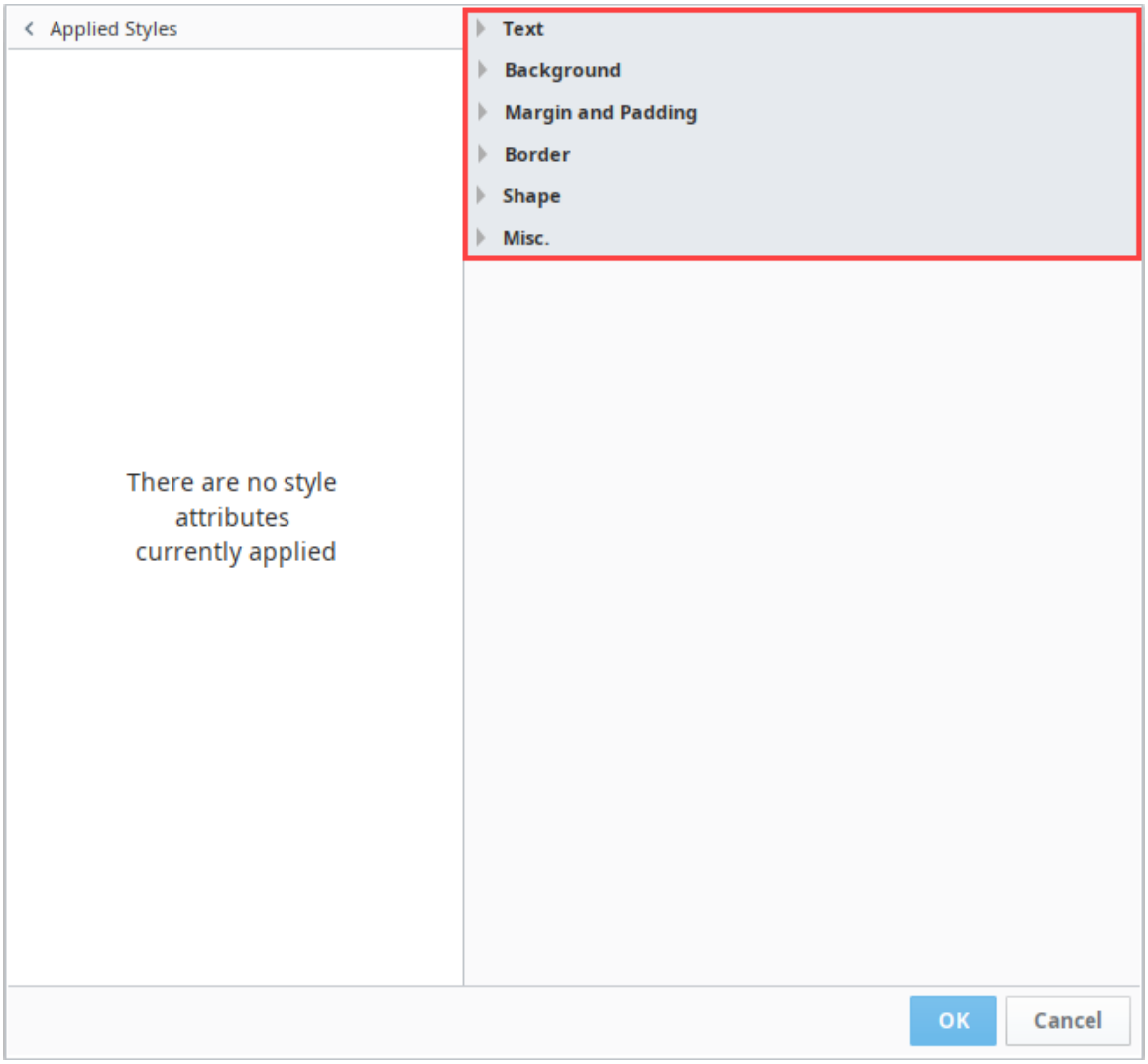


The following menus are available in the style editor:

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

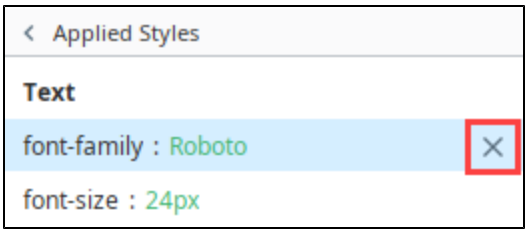
> The following feature is new in Ignition version **8.0.10**
> Click here to check out the other new features

In release 8.0.10, the Padding menu was updated to include Margin and Padding settings. For complete information, see Style Reference.

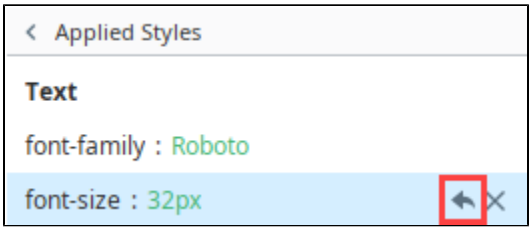


You can quickly make adjustments to the styles by clicking on them in the Applied Styles panel.

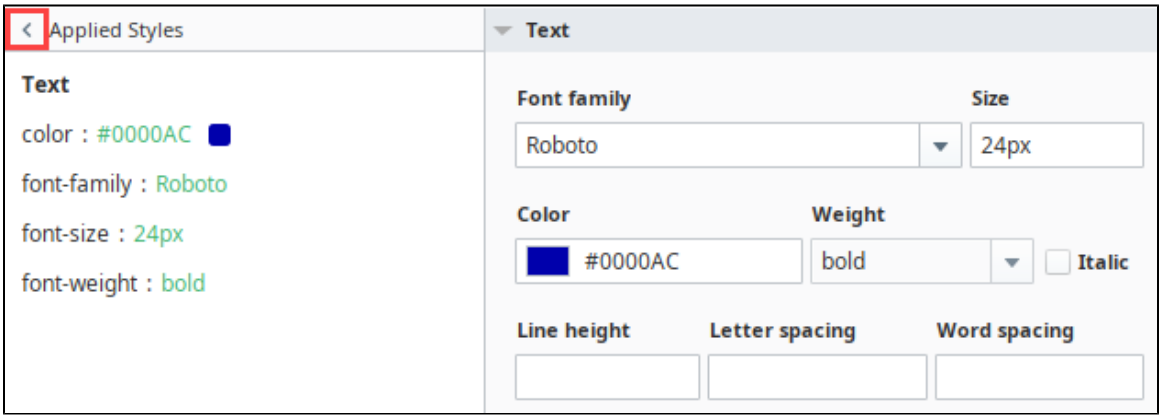
When you hover over a style listed in the panel, a **Delete** icon is displayed. Click on it to delete the style.



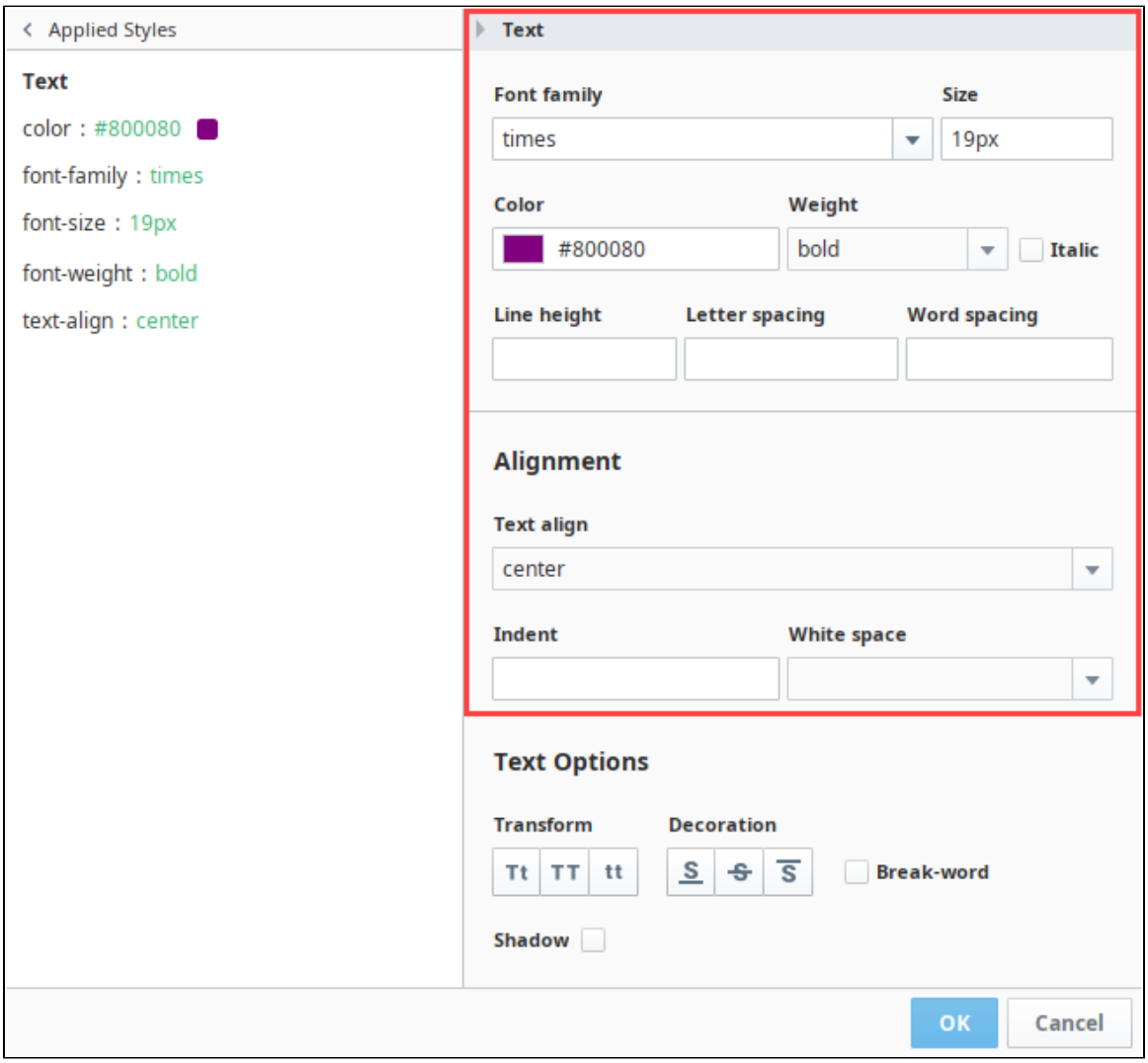
If you made new edits to an existing style attribute, you can hover over the style and an **Undo** icon is displayed. Click on it to revert to the previously saved setting.



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:

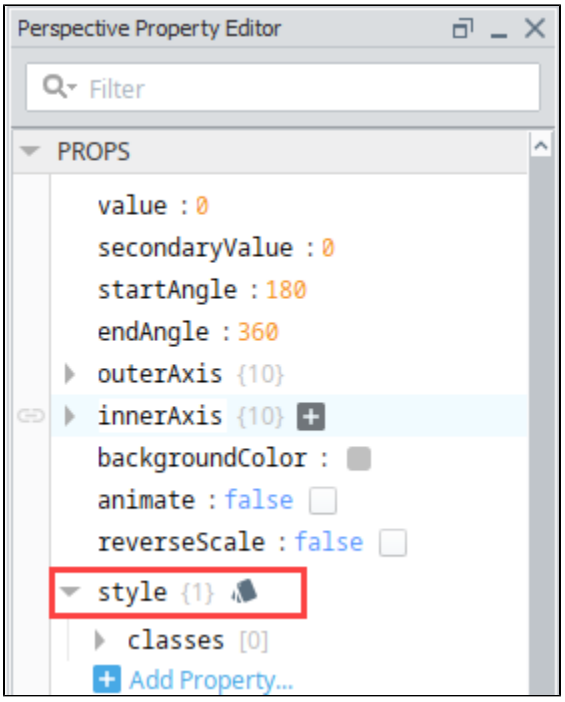


⚠ Applying individual style elements to a component will overwrite the settings for the same style elements being applied from a Style Class.

⚠ 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.

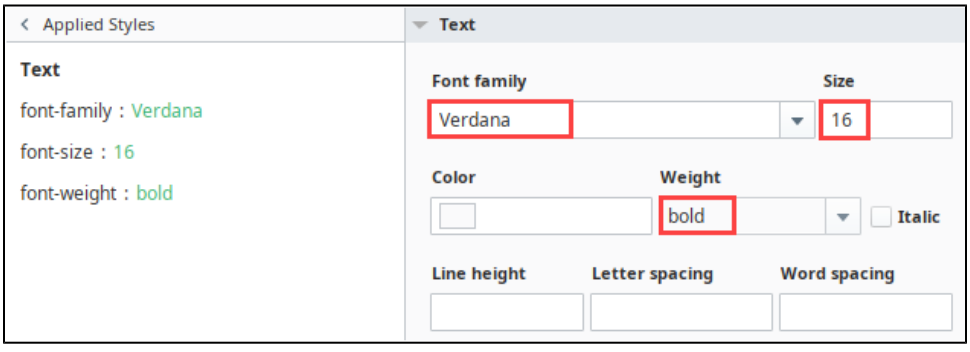
Style Example

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.

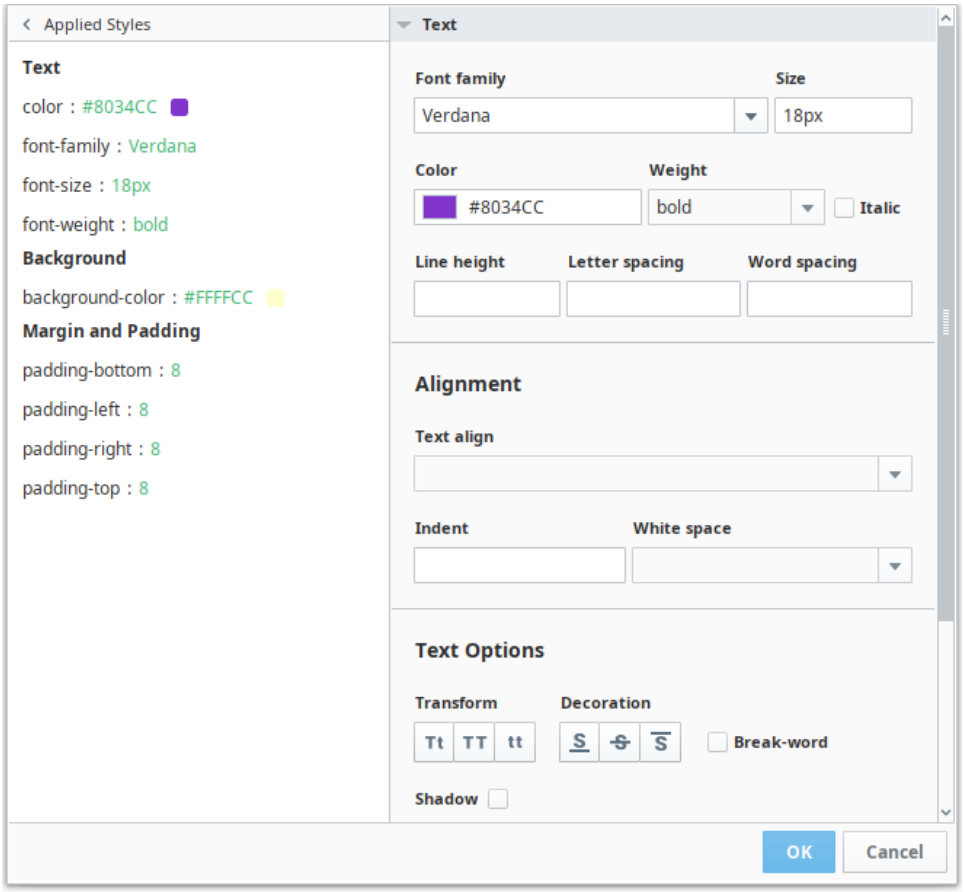


   The style editor is displayed. Each of the pull down menus has options. These options are described in the Style Reference.
3. Click on the **Text** menu then set the following style options:
   Font Family: **Verdana**
   Size: **16**
   Weight:**bold**

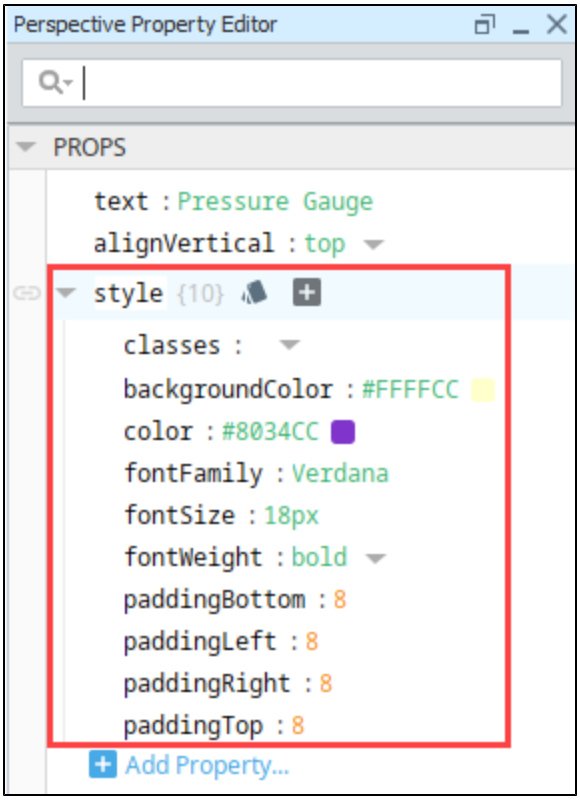
   As you select Style elements, they appear in Applied Styles column on the left.



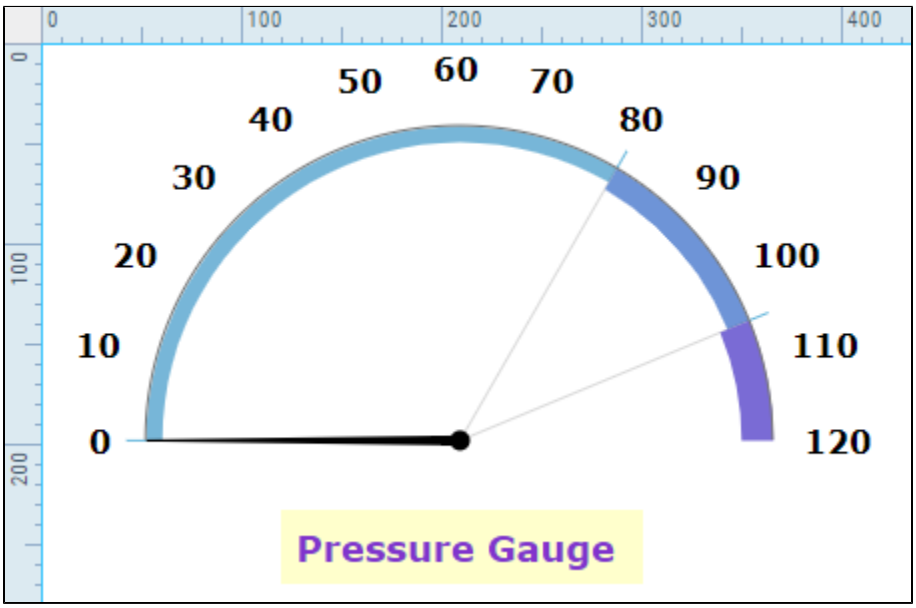
4. Click **OK**.
5. Next select the Label component and click the **Styles** icon.
6. Expand the **Text** menu and set the following style options:
   Font family: **Verdana**
   Size: **18px**
   Weight: **bold**
   Color: **#8034CC**



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.



Our finished example looks like this:



Refer to Style Reference for a complete reference of all the available settings for Styles.

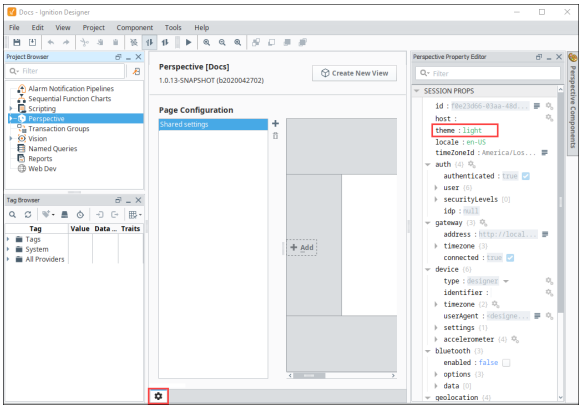
In This Section ...

Copy of Perspective Themes

Themes

Perspective comes with several themes, providing initial styling to all components. A theme effectively provides the default style to a component, allowing styles to override and modify any styling rules that are defined in the theme.

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.



On this page

...

- Themes
  - Initial Theme
  - Theme Examples
- Setting a Theme
  - Setting a Theme Using a Component
  - Theme Component Action
- Modifying a Theme - Using a Different Font
- Theme Colors
  - Creating a Color Variable.
  - Creating a Custom Theme Using Custom Variables. (This really should just say, creating your own custom theme, should be broken up ideally - YN)
  - Using Theme Colors
  - Built-in Theme Colors

Initial Theme

Ignition installations come with the following themes:

- light - white background
- dark - dark background

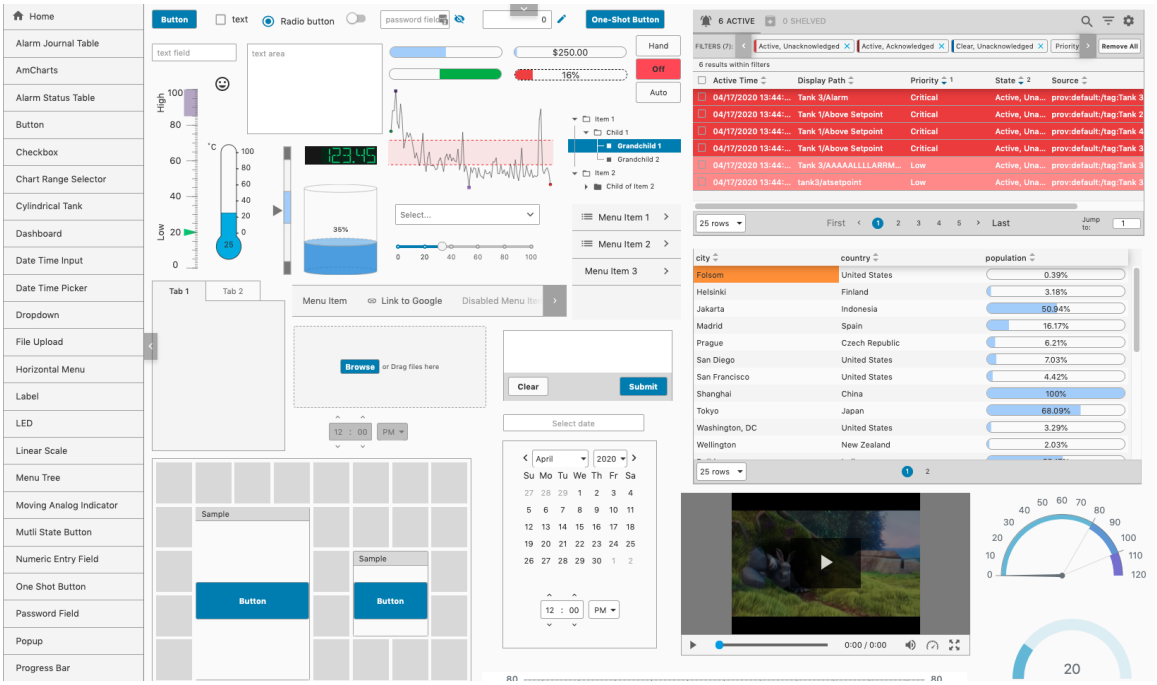
> The following feature is new in Ignition version **8.0.13**
> Click here to check out the other new features

Ignition version 8.0.13 introduced the following new themes:

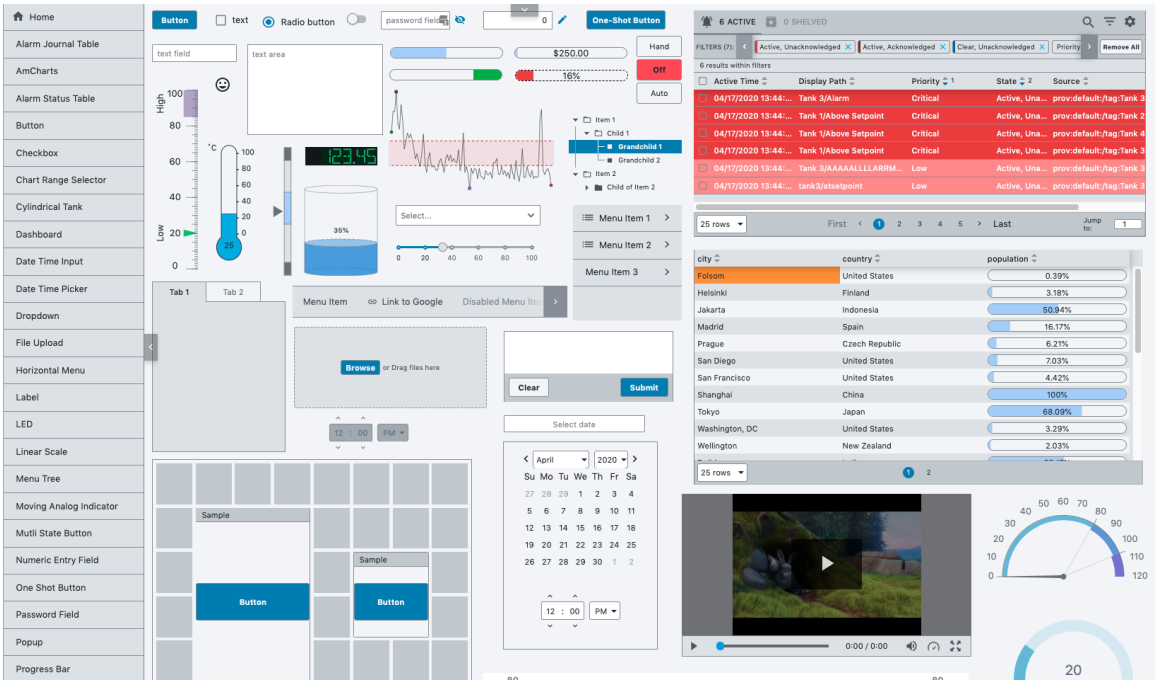
- light-warm
- light-cool
- dark-warm
- dark-cool

Theme Examples

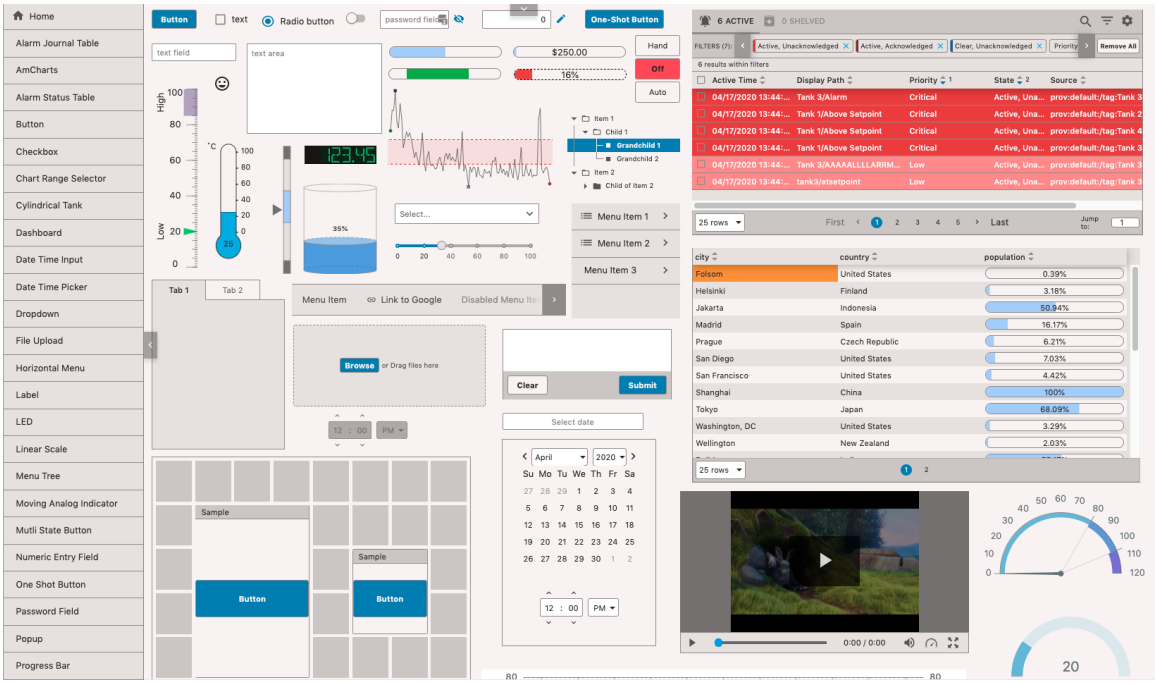
Light:



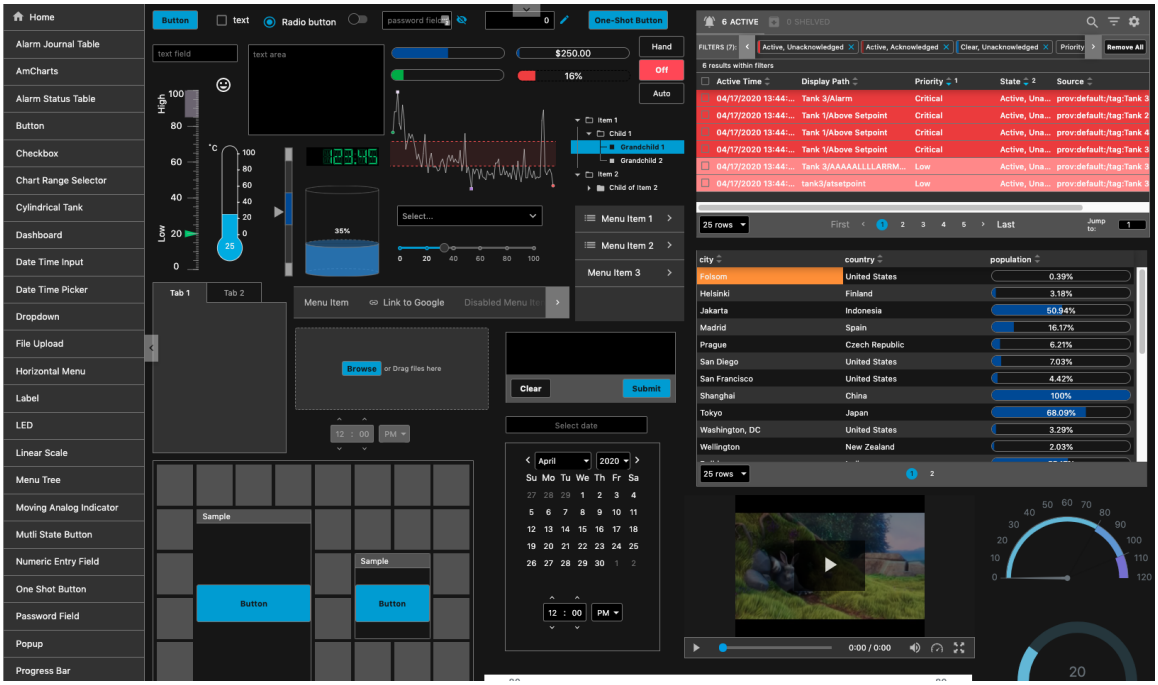
Light-Cool:



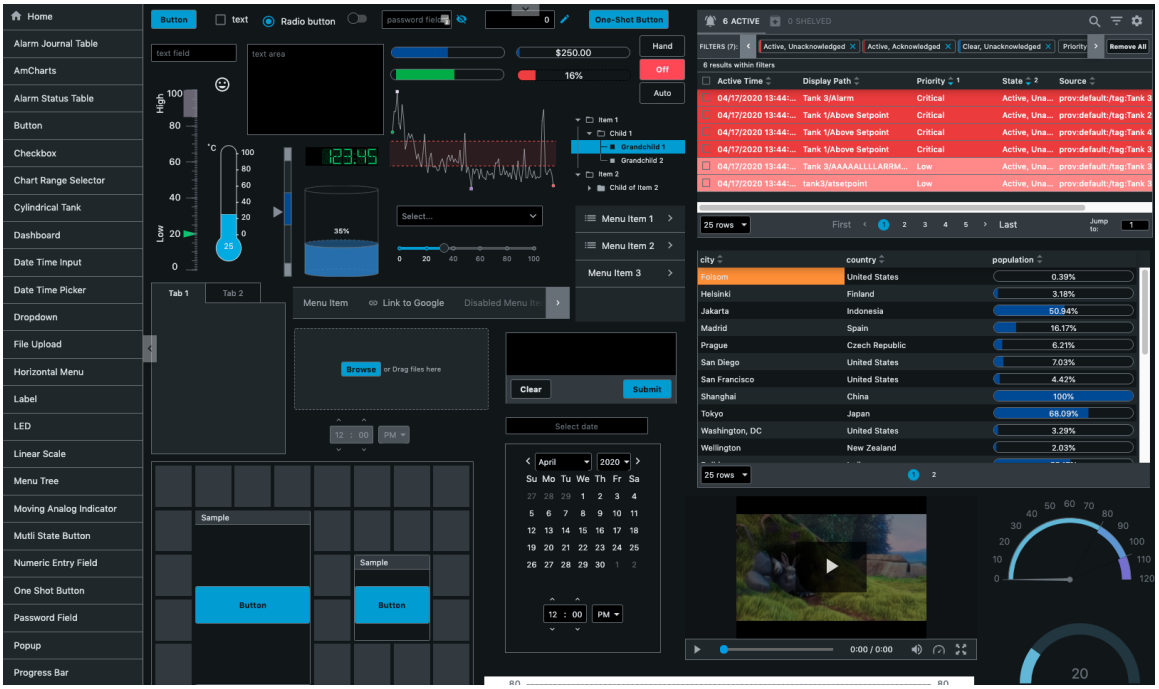
Light-Warm:



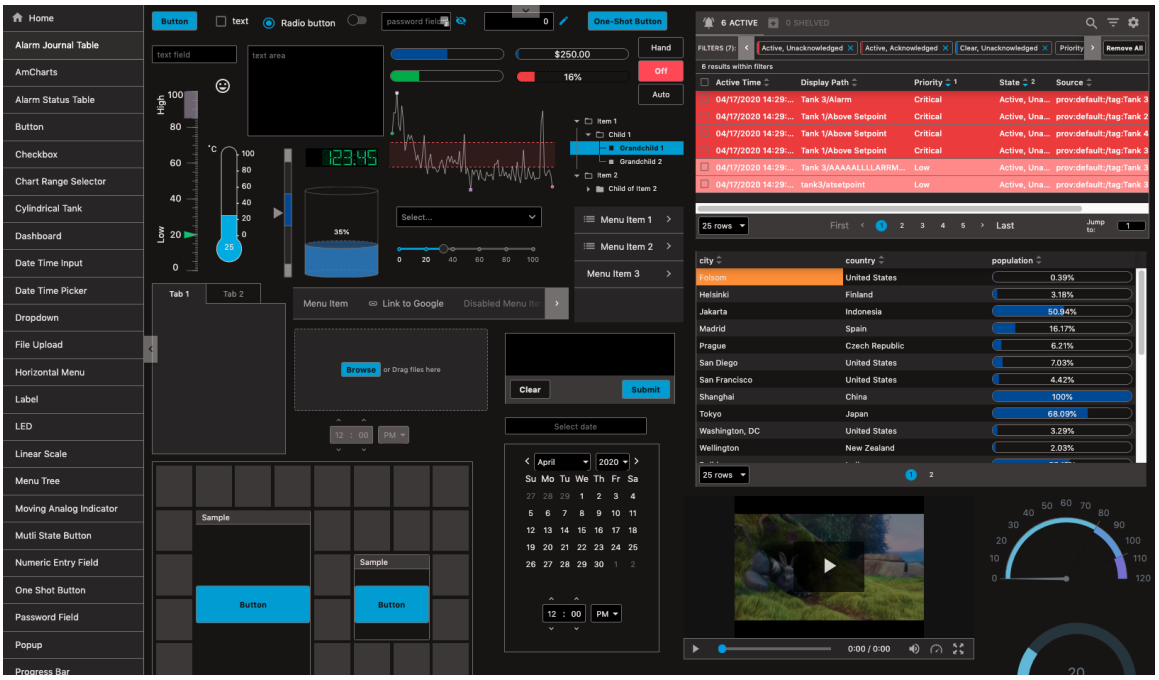
Dark:



Dark-Cool:



Dark-Warm:



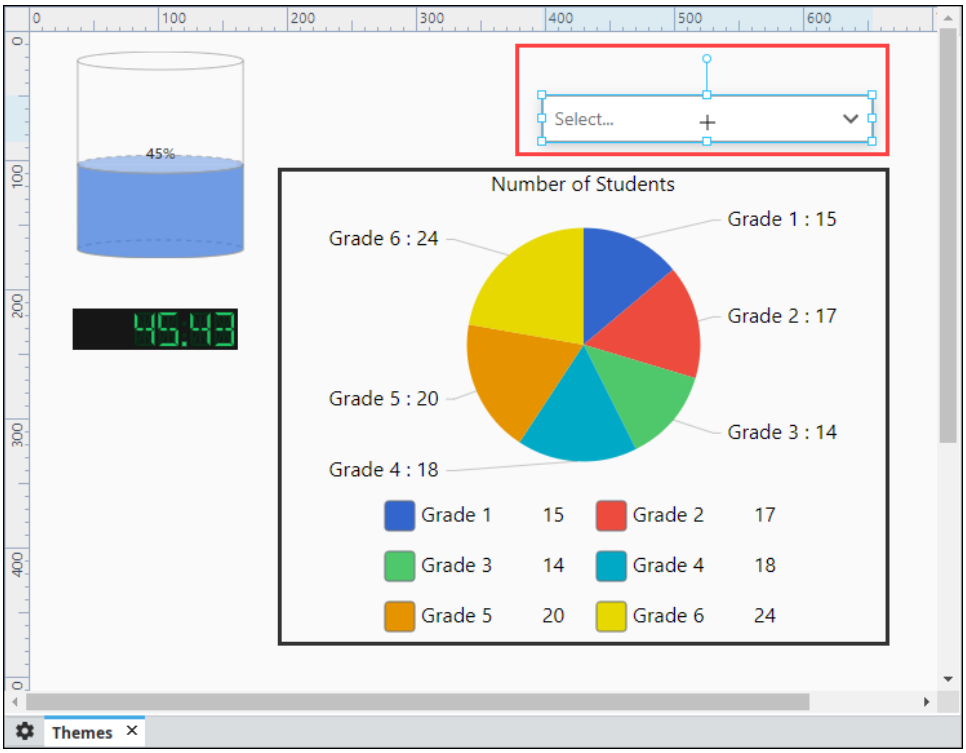
Setting a Theme

A theme can be set a number of ways. This section demonstrates a couple of approaches.

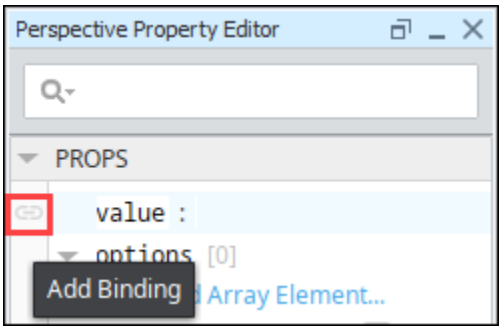
Setting a Theme Using a Component

Simply writing to the "theme" session property will change the theme used in the session. In this example, we used a Dropdown component to change the theme in a Perspective Session.

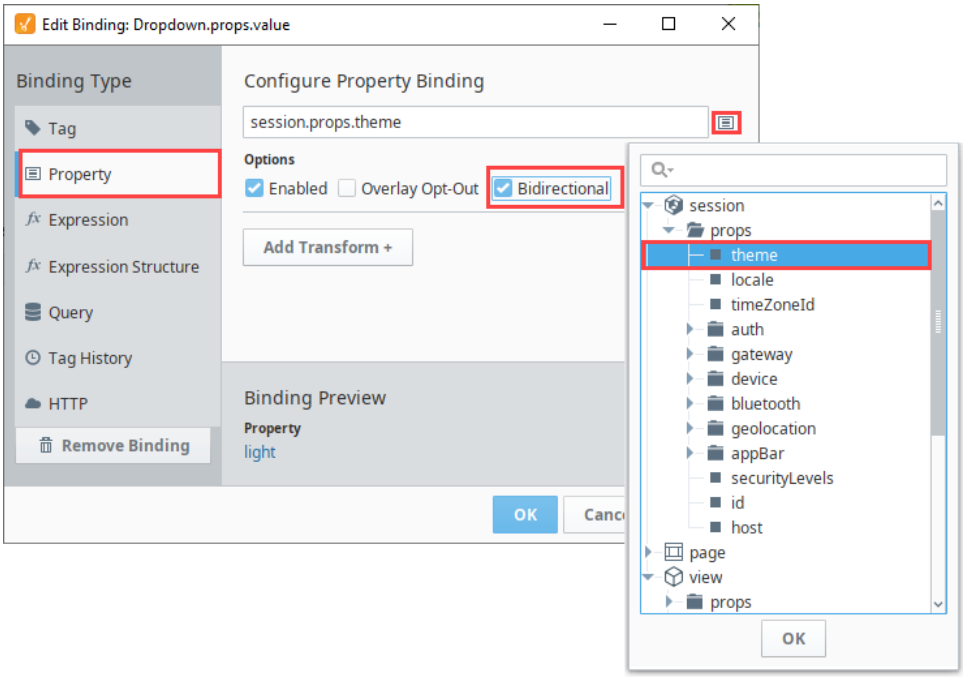
1. Open a view in the Designer that contains a few components and drag in a Dropdown component.



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



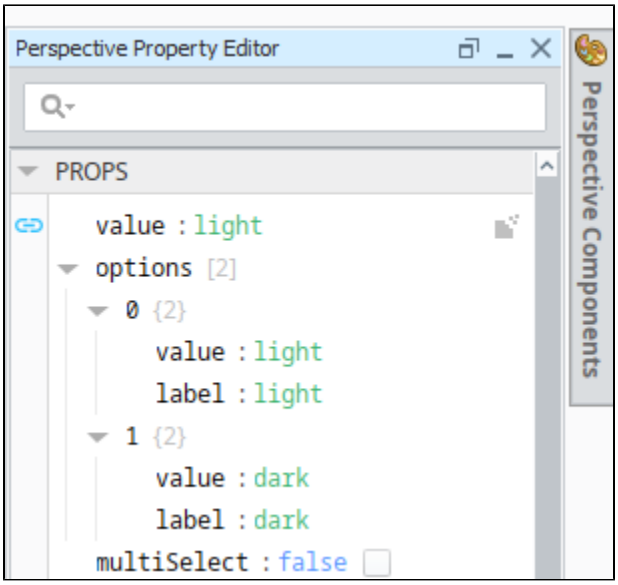
3. The Edit Binding window will open. The theme property is a session property. Click on the **Property** icon and expand **session > props**, and select **theme**. Click **OK**.
4. Check **Bidirectional** and click **OK** to save your property binding.



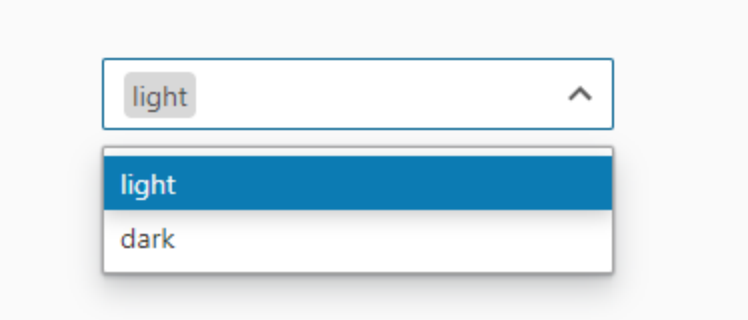
5. In the Property Editor, enter the theme values under options.

   Paste the following onto the "options" property of a dropdown component.

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



6. **Save** your project and open your view in a Session or in Preview Mode.
7. From the Dropdown component, select a desired theme (i.e., light).

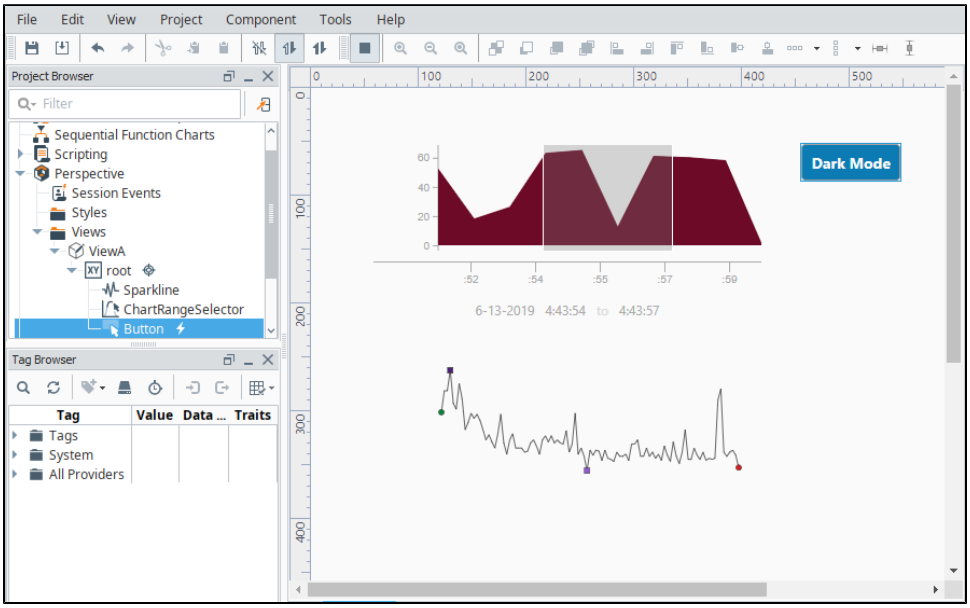


The selected theme will now be applied for the entire Perspective Session.

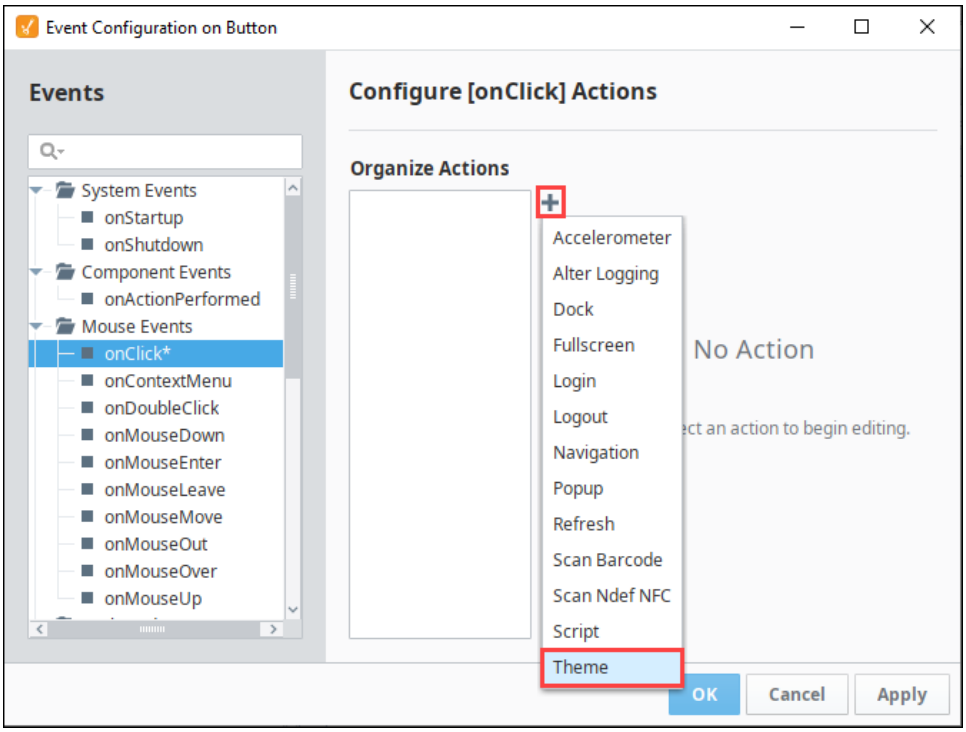
Theme Component Action

You can also set a theme by configuring a Component Action.

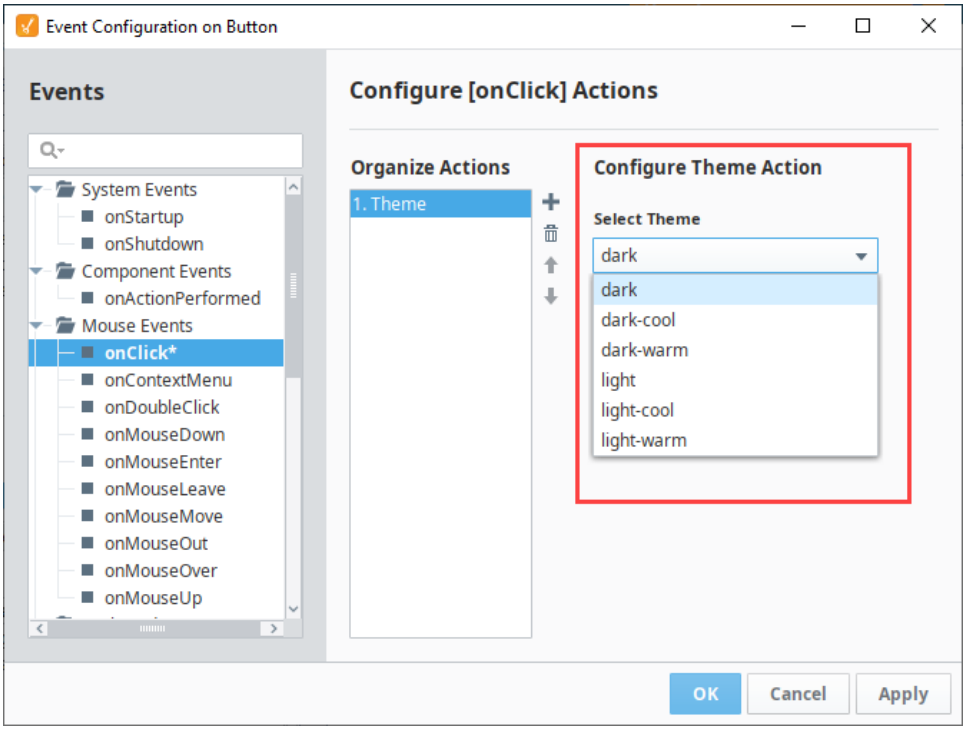
1. Open a view in the Designer with a couple of components, including a button component. The goal here is to change the active theme to "dark" when a user presses the button.
2. Select the Button component, and click on the **Components > Configure Events** at the top of the Designer menubar.



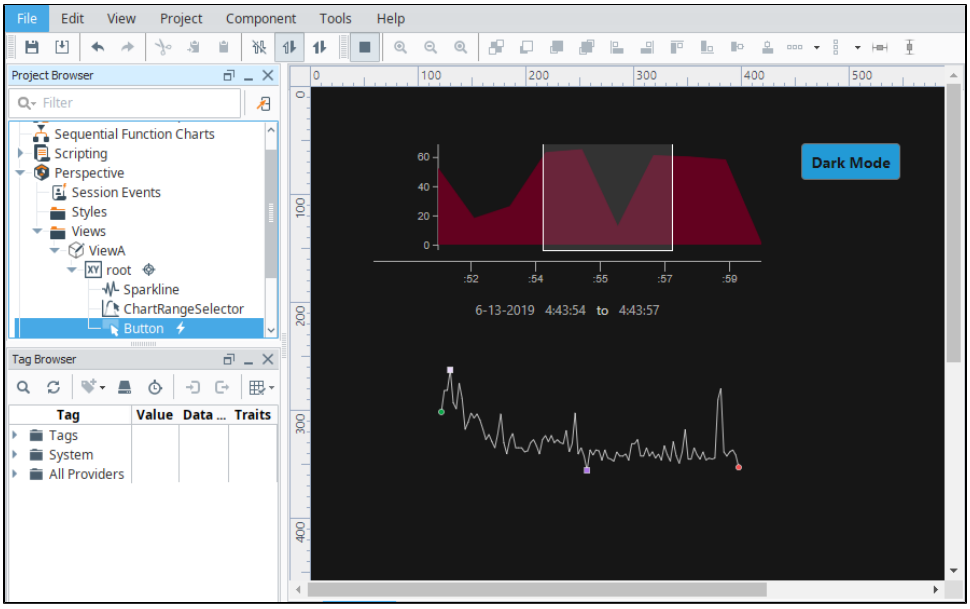
3. The Event Configuration window will appear. Select the event you want to put the Theme on. In this example, we used **onClick.** Click the **Add** icon and choose **Theme.**



4. Select the desired Theme name (i.e., dark) and click **OK**.



5. Click **Save** and open either a Perspective Session, or click Preview Mode if you are in the Designer.
6. Click the **Button** and the theme will change to Dark Mode.



Modifying a Theme - Using a Different Font

Creating a custom Perspective theme is considered an advanced feature. As such, we operate under the assumption that anyone altering these files has some knowledge of CSS. By making use of CSS variables, imports, and descriptive class selectors, you will find that we provided features and structure to help alleviate some of the frustrations inherent to managing large amounts of CSS.

So, before we even get to this next section about using a different font, I think it would be very useful to discuss the general structure of theming, and how it makes use of a series of import statements, etc. A general idea of how it works. Pulling a quick overview from that readme. It just feels like we are missing some sort of intro. It makes it hard to follow, just like the color of this font 😉. - YN

More information on modifying a theme, or making a new theme, can be found at "*%ignitionInstallationDirectory% \data\modules\com.inductiveautomation.perspective\themes.README.md*" file (added in 8.0.13).

Note that modifying or creating new themes is unsupported, so our support department will not be able to assist with any questions you may have. However, feel free to ask any questions on the forums.

Need to elaborate on "using a different font." At least, mention somewhere that this walk through is all about modifying the themes global font. Also, there should be a discussion about when it makes sense to create your own theme versus when it makes sense to just override. I believe we discussed this last week, but let me know if you want it in writing. - YN

For this example, modify the existing Light theme in *"%ignitionInstallationDirectory%\data\modules\com.inductiveautomation.perspective\themes\"* to have it use the Ubuntu font.

1. Download the Ubuntu family font files here: https://fonts.google.com/specimen/Ubuntu. Put all of the font files from your download in a folder named "Ubuntu" and place this folder in `%ignitionInstallationDirectory%\data\modules\com.inductiveautomation.perspective\fonts\`.
2. Go to `%ignitionInstallationDirectory%\data\modules\com.inductiveautomation.perspective\themes\` and create a folder named `overrides`.
3. Inside your `overrides` folder, create two CSS files, one called `fonts` and the other called `globals`. **`You planning on mocking out a folder structure, here? This is how it should look kind of thing. -YN`**

   Your `fonts.css` file should look like this: **Provide a link to what font-face actually does https://developer.mozilla.org/en-US/docs/Web/CSS/@font-face -YN**

fonts file example

```
@font-face {
    font-family: var(--font-Ubuntu);
    font-style: normal;
    font-weight: 400;
    src: local('Ubuntu'), local('Ubuntu'),
         url('/data/modules/perspective/fonts/Ubuntu/Ubuntu-Regular.ttf') format('ttf');
```

```
}

@font-face {
    font-family: var(--font-Ubuntu);
    font-style: normal;
    font-weight: 500;
    src: local('Ubuntu-Medium'), local('Ubuntu-Medium'),
         url('/data/modules/perspective/fonts/Ubuntu/Ubuntu-Medium.ttf') format('ttf');
}

@font-face {
    font-family: var(--font-Ubuntu);
    font-style: normal;
    font-weight: 700;
    src: local('Ubuntu-Bold'), local('Ubuntu-Bold'),
         url('/data/modules/perspective/fonts/Ubuntu/Ubuntu-Bold.ttf') format('ttf');
}
```

Your `globals.css` file should look like this: **Let them know that this is setting the font on the body, input, textarea, etc... to Ubuntu. Also, you haven't declared your variable yet. Might want to stick to `Ubuntu` and not `--font-Ubuntu`. Also, rename global.css to my_globals_overrides.css or something. -YN**

```
body, input, textarea, keygen, select, button {
    font-family: var(--font-Ubuntu);
}
```

Modify your `light.css` file in "*%ignitionInstallationDirectory%\data\modules\com.inductiveautomation.perspective\themes*" as follows:

light.css file example

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

Order of the import statements matter. Variables, fonts, and globals, should always be at the top so that when the CSS is parsed, these rules are already available to be used since they were parsed first. Also, it would help if my override import statements were more easily distinguishable in the example. - YN

Note how you're basically overriding the `/light/fonts.css` file with your own `/overrides/fonts.css` file. The same thing is happening with the `/light/globals.css` file. The order of these imports is important, as an import at the bottom of the list could override imports at the top of the list. **Not basically, definitely overriding. Would help to elaborate on this. What actually happened here. - YN**

By following these steps, your Light theme is ready to use in your Designer using the Ubuntu font. **Right, not only have you declared a new font-face and imported the font-face files onto the page in the browser so that they can be used by other components, but more importantly you have overridden the globally used font. Side note: fonts and font colors cascade, the C in CSS. Meaning, if the html body is using the Ubuntu font, unless anything below it declares it's own font family, this font will be inherited automatically as a result of the cascade. -YN**

Theme Colors

Creating a Color Variable.

The built-in themes make heavy use of CSS variables for colors. For any of the default themes, colors are defined in the `variables.css` file. Defining your own color variable can be done by following these steps: **CSS variables are limited to only colors by the way. Read up on the docs here https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_custom_properties - YN**

Be aware that changes made to the built-in theme files will be replaced on Gateway start up (including restarts caused by a Gateway Restoration) and moved to a backup folder on upgrade. As a result, users who want to modify a theme must have their own custom css files in a directory separate from the built-in theme files. Custom css files can then be imported into the entry point css files in \themes\ which do not get overwritten or modified on Gateway start up.

For more information, see the markdown README file located in the Gateway's installation directory: `%installDirectory%\data\modules\com.inductiveautomation.perspective\themes\README.md`

This should be described earlier on. Seem out of place, doesn't it? - YN

1. Create a folder named "variables" in `%installDirectory%\data\modules\com.inductiveautomation.perspective\themes.` The purpose of this folder is to contain any and all variable definitions. Since this folder is not part of any of the built-in theme files, it will not get touched on system start up or system restore.
2. Inside this newly created variables folder, place a .css file named "variables" with the following variable definition:

```
:root {
    /* Variables: */
    --myvariable: #000080; /*This is the variable I created*/
}
```

3. Now that the variable has been created in this variables.css file, we must import it in the specific .css file corresponding to the theme where you wish for this variable to be used. For this example, the variable will be imported into the light theme so the light.css file in /themes/ will be modified as below:

```
@import "./light/index.css";
@import "./variables/variables.css"; /* This is where the variable is imported */
```

After following these steps, the variable will be ready to be used. **You don't actually need to create your own folder if you don't want to. You can declare the variables directly in the light.css. Assuming you're not concerned with having it override anything. - YN**

Creating a Custom Theme Using Custom Variables. (This really should just say, creating your own custom theme, should be broken up ideally - YN)

To create a custom theme, it is important to take advantage of the default themes. **(it is highly advised and recommend that you extend one of the default themes, light.css in particular. Many if not all of the components rely on styles provided by the themes. If those styles don not exist on the page, your project will reflect that. - YN)** The 'light-cool', 'dark', and 'dark-warm' themes are all derived from the 'light' theme. We can similarly take advantage of the 'light' theme to create a new, custom theme. The new theme will override an IA owned css file with a custom css file which sets container background color to blue using the color variable created in the example above. Follow these steps to create a new, custom theme:

1. Create a css file named 'navy' in `%installDirectory%\data\modules\com.inductiveautomation.perspective\themes\` with the following:

```
@import "./light/index.css";
@import "./variables/variables.css"; /* Import custom variables. */
```

Since our new theme is derived from the 'light' theme, this css file is a direct copy of the light.css file. We want to use custom variables in this new theme so we import the custom variables created in the example above. Once this navy.css file is created, a theme named 'navy' will become available for use in the Designer.

2. To make the theme's background blue, the container.css file in `%installDirectory%\data\modules\com.inductiveautomation.perspective\themes\light\common\` must be modified. Since we cannot modify any of the built-in theme files, we will create a new folder named "overrides" in `%installDirectory%\data\modules\com.inductiveautomation.perspective\themes.` This folder will contain any and all css overrides. In this folder, place a copy of the container.css and modify it as below: **I'm confused, are we creating a custom or overriding the existing themes? -YN**

```
.ia_container--root {
    background-color: var(--myvariable); /* We will override the container background-color to be
--myvariable*/
}
.ia_container--secondary {
    background-color: var(--container);
}
```

This is a pretty heavy example, since that css selector (ia_container-–root) is used on a bunch of fundamental container types. You could really mess things up. Unless this is something highly requested, I'd avoid doing it this way, or better

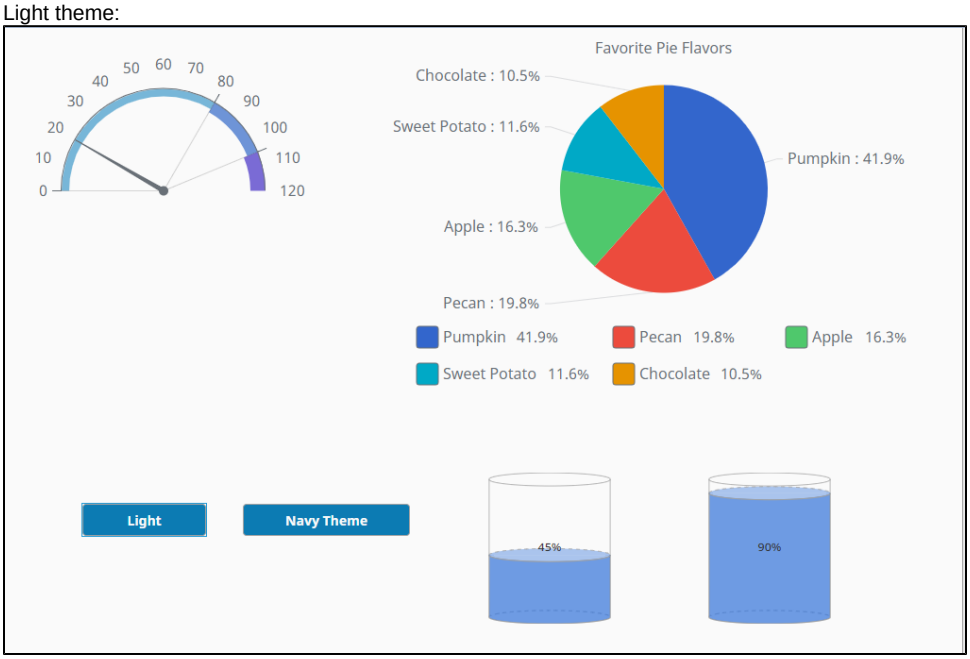
yet, this example entirely. A simpler example would be better. In general, I'd really prefer that they override the value of an existing css variable, and not modify a css ruleset or declaration block to use their own. This is pretty advanced and you can really mess things up unless you really know what you're doing. -YN

3. Now that overrides for our custom theme were created, we can import them into our 'navy' theme by modifying the navy.css file as below:

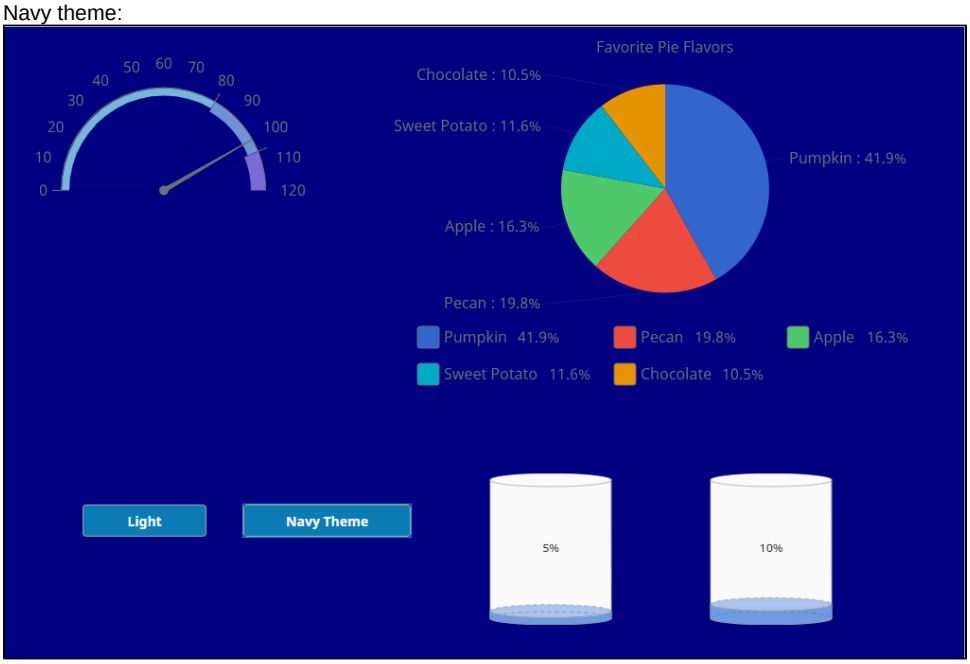
```
@import "./light/index.css";
@import "./variables/variables.css"; /* Import custom variables. */
@import "./overrides/container.css"; /* Import overrides. */
```

4. Our 'navy' theme is now configured to include all the attributes from the IA owned 'light' theme with the exception of an overwritten container background color. To verify this is working, we create two buttons with Theme Component Actions. One button will set the Theme to 'light' while the other will set the Theme to 'navy'.

Light theme:

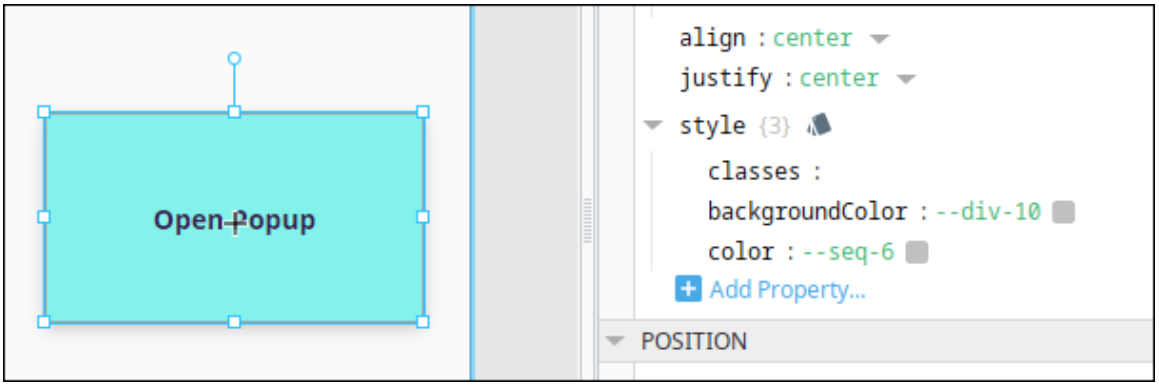


Navy theme:

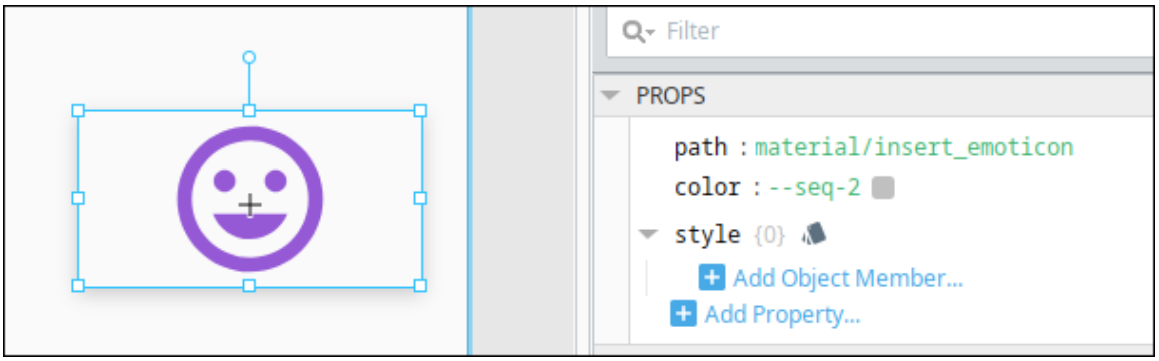


Using Theme Colors

Theme colors can be used on components by simply providing the variable name. For example, we can change the **backgroundColor** and **color** of a button component by just stating the variable name for the appropriate styling properties on the component's **style** object.

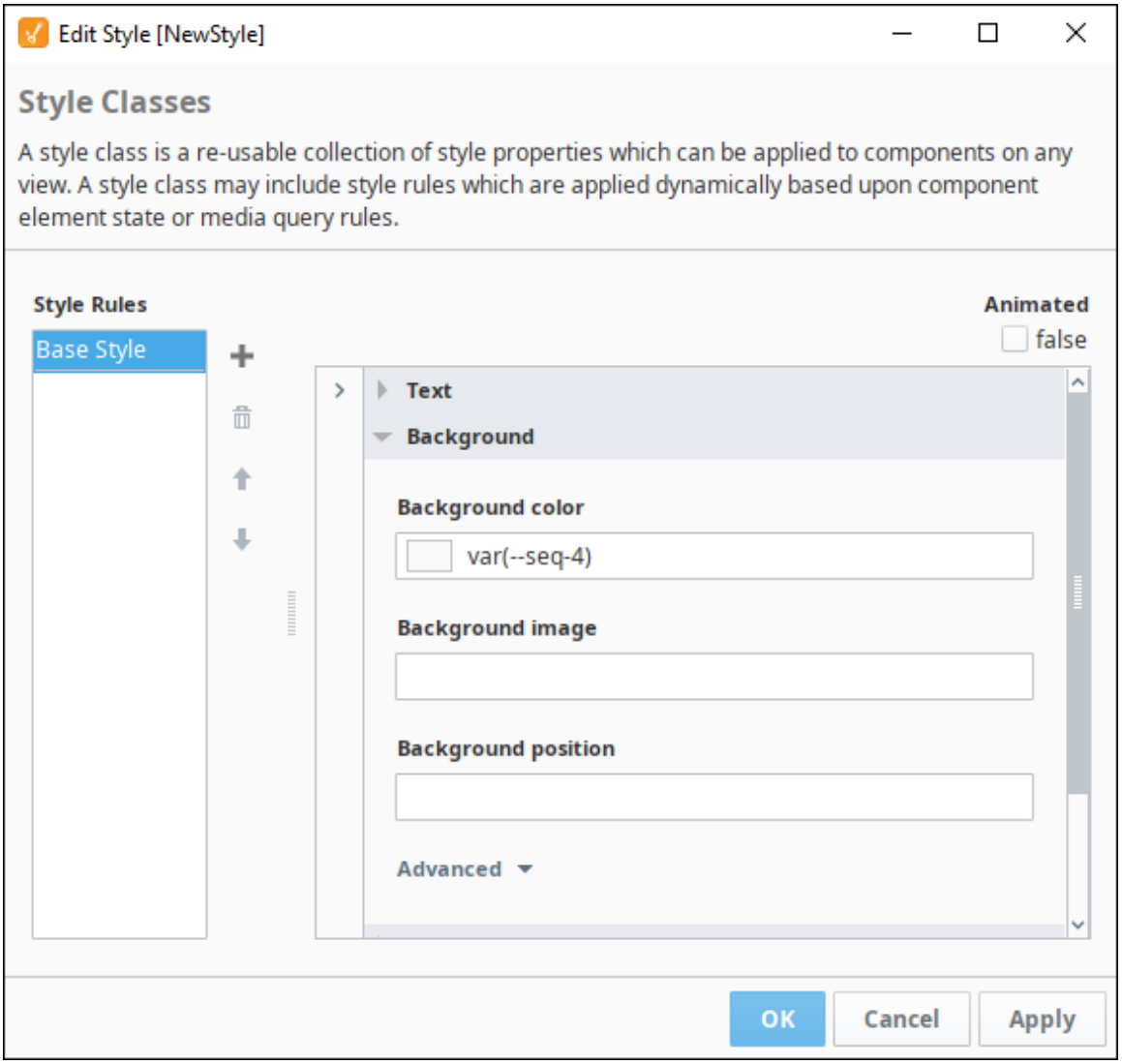


If a component has a **color** property outside of a style object, such as the Icon component, the same rules apply; simply set the value of the color property to the name of the variable.



Style Classes

When using a Theme Color in a Style Class, the variable must be wrapped in the `var()` method, as shown below.



Built-in Theme Colors

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

	light	light-cool	light-warm	dark	dark-cool	dark-warm
--neutral-10						
--neutral-20						
--neutral-30						
--neutral-40						
--neutral-50						
--neutral-60						
--neutral-70						
--neutral-80						
--neutral-90						
--neutral-100						
--seq-1						

--seq-2						
--seq-3						
--seq-4						
--seq-5						
--seq-6						
--div-1						
--div-2						
--div-3						
--div-4						
--div-5						
--div-6						
--div-7						
--div-8						
--div-9						
--div-10						
--div-11						
--div-12						
--div-13						
--div-14						
--div-15						
--div-16						
--qual-1						
--qual-2						
--qual-3						
--qual-4						
--qual-5						
--qual-6						
--qual-7						
--qual-8						
--qual-9						
--qual-10						
--callToAction						
--callToActionHighlight						
--callToAction--hover						
--callToAction--active						
--callToAction--disabled						
--error						
--info						
--infoSecondary						
--warning						

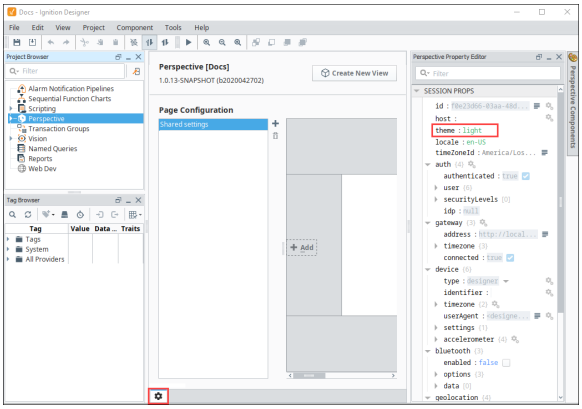
--warningSecondary						
--success						
--indicator						
--indicatorOff						

Perspective Themes

Themes

Perspective comes with several themes, providing initial styling to all components. A theme effectively provides the default style to a component, allowing styles to override and modify any styling rules that are defined in the theme.

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.



On this page

...

- Themes
  - Initial Theme
  - Theme Examples
- Setting a Theme
  - Setting a Theme Using a Component
  - Theme Component Action
- Modifying a Theme - Using a Different Font
- Theme Colors
  - Using Theme Colors
  - Built-in Theme Colors

Initial Theme

Ignition installations come with the following themes:

- light - white background
- dark - dark background

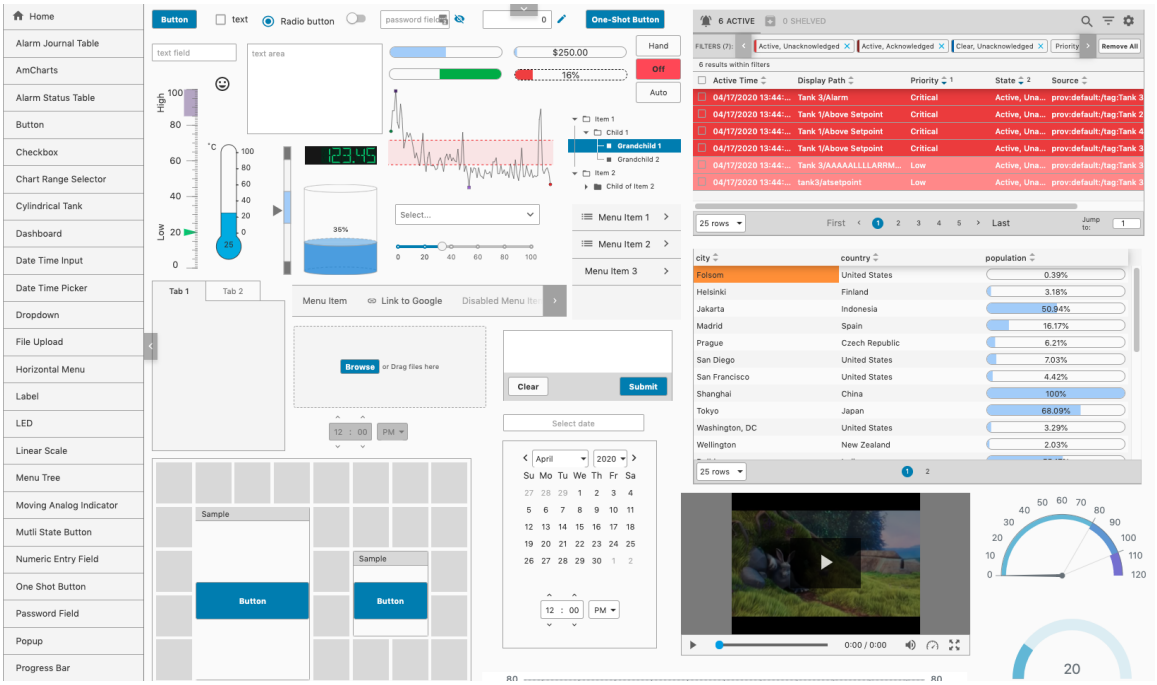
> The following feature is new in Ignition version **8.0.13**
> Click here to check out the other new features

Ignition version 8.0.13 introduced the following new themes:

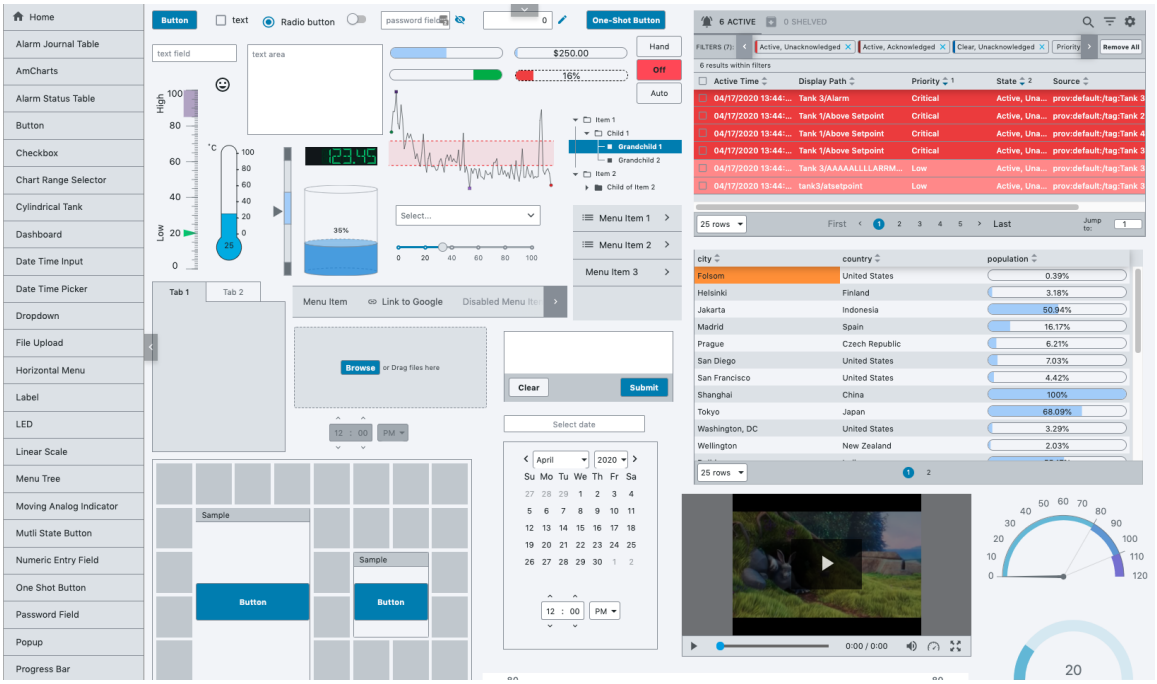
- light-warm
- light-cool
- dark-warm
- dark-cool

Theme Examples

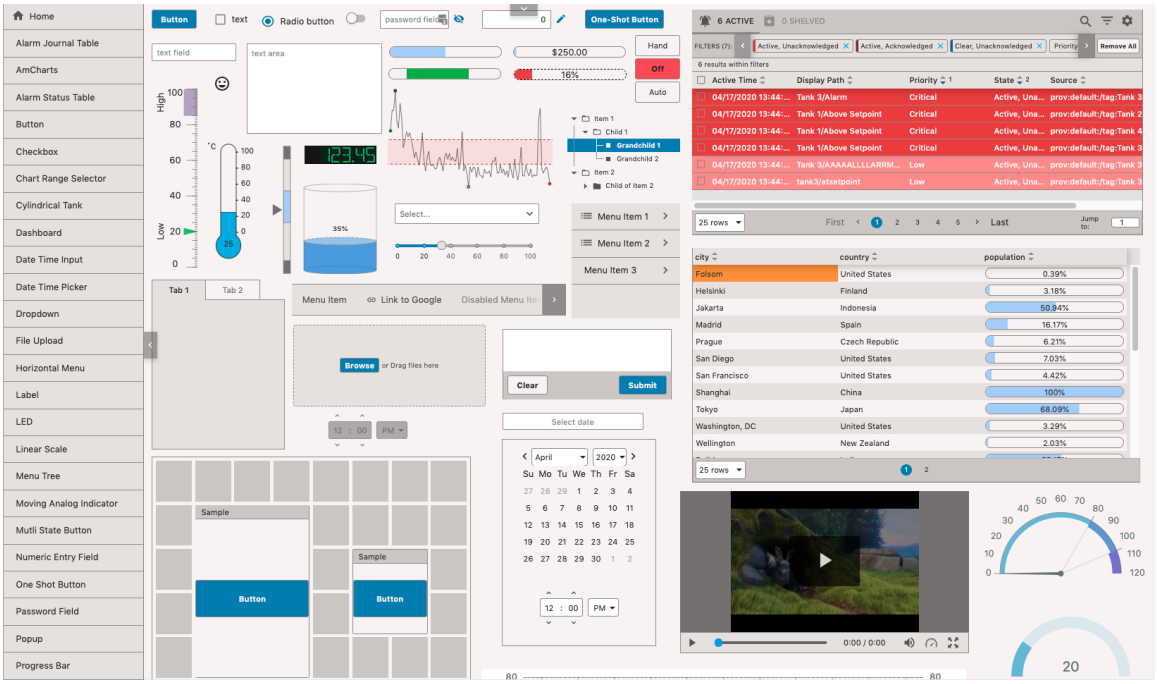
Light:



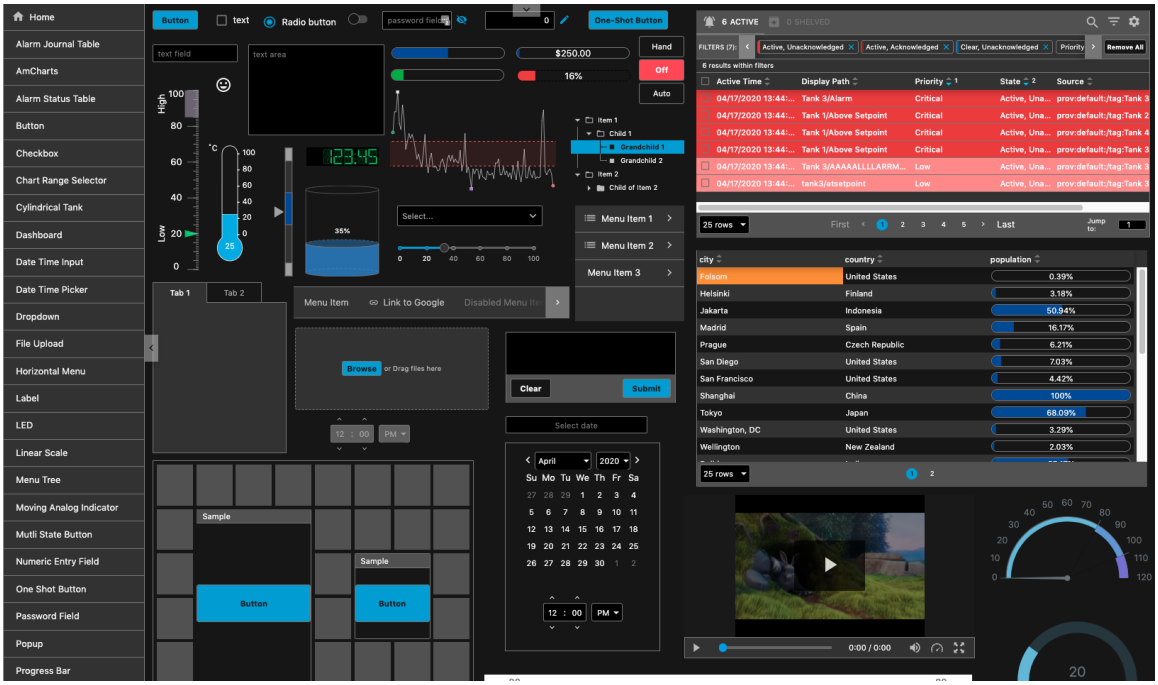
Light-Cool:



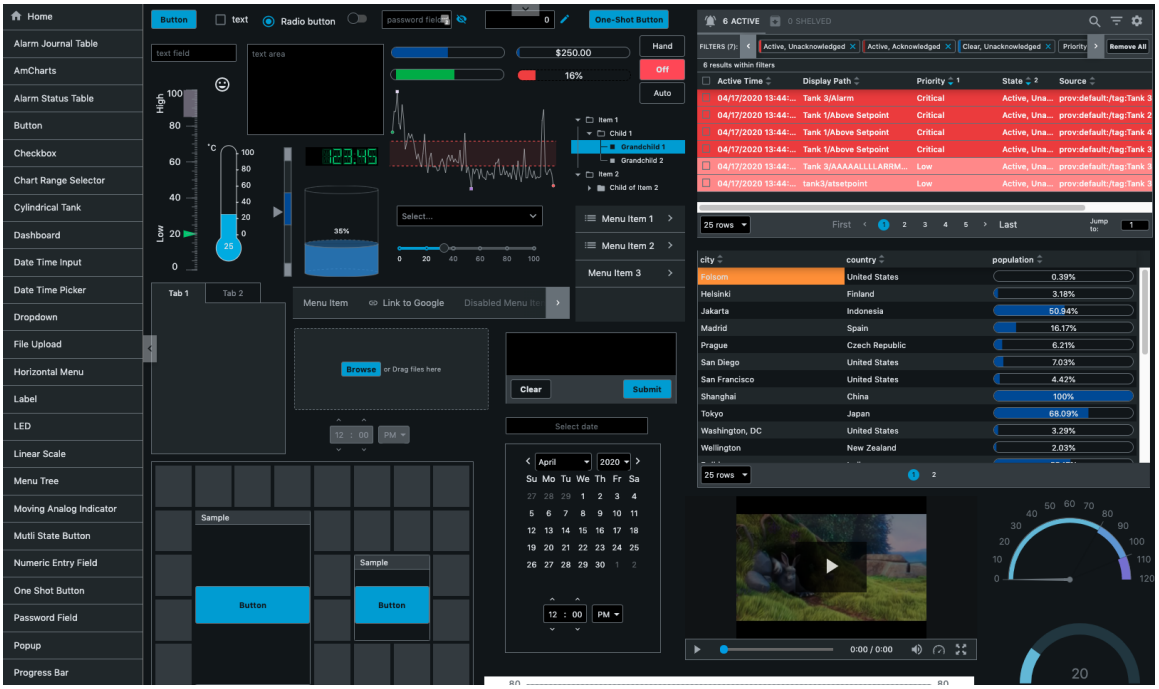
Light-Warm:



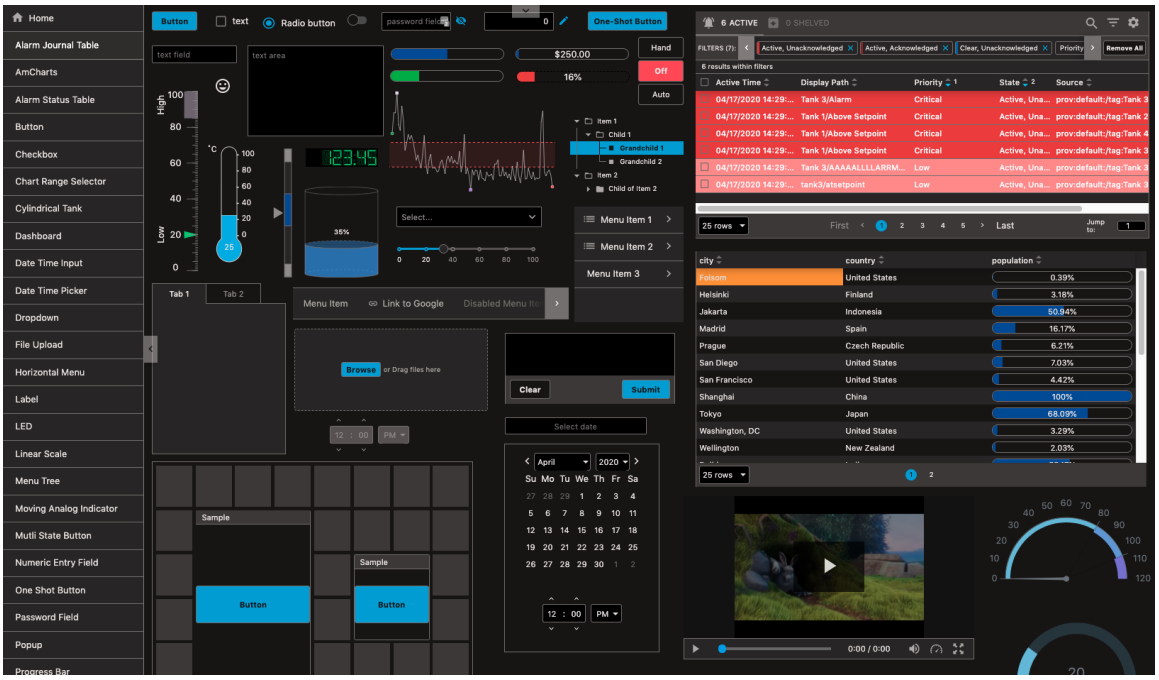
Dark:



Dark-Cool:



Dark-Warm:



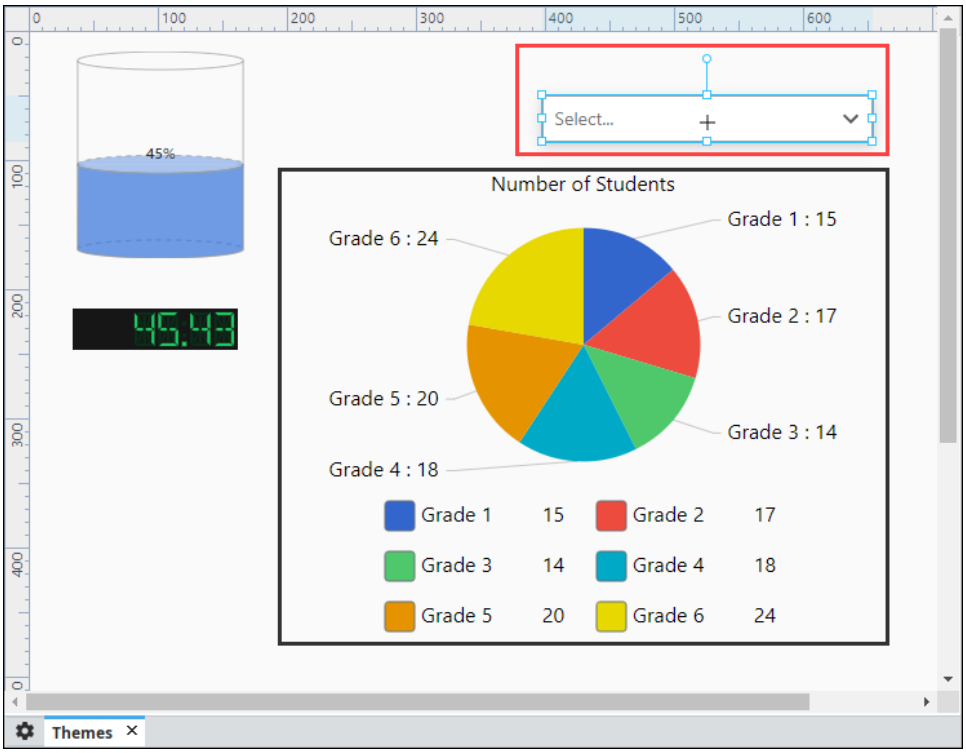
Setting a Theme

A theme can be set a number of ways. This section demonstrates a couple of approaches.

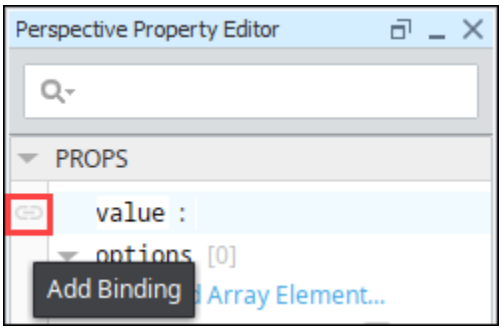
Setting a Theme Using a Component

Simply writing to the "theme" session property will change the theme used in the session. In this example, we used a Dropdown component to change the theme in a Perspective Session.

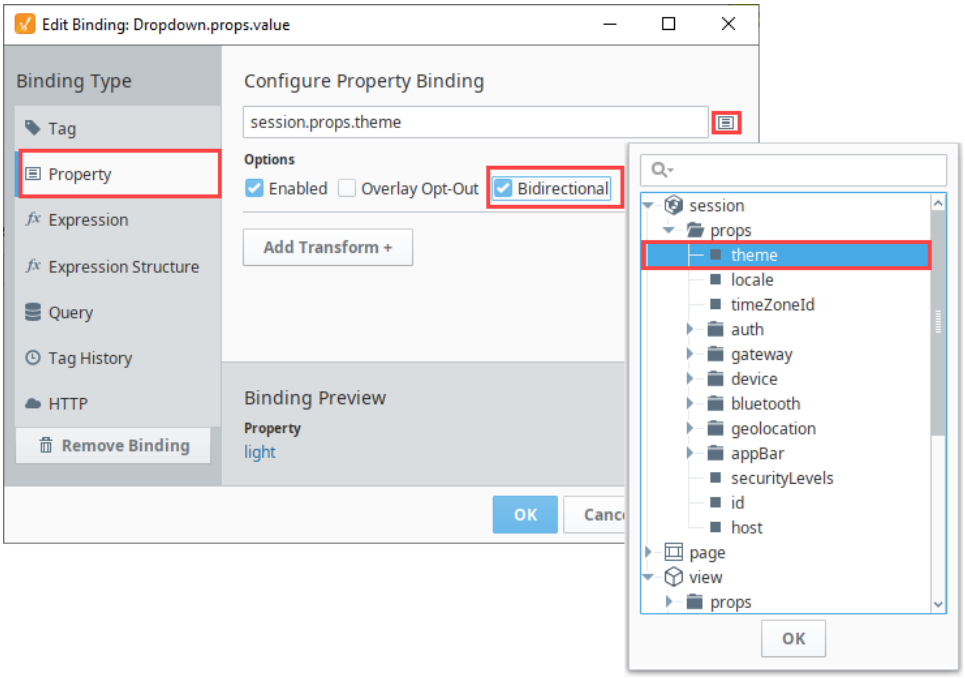
1. Open a view in the Designer that contains a few components and drag in a Dropdown component.



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



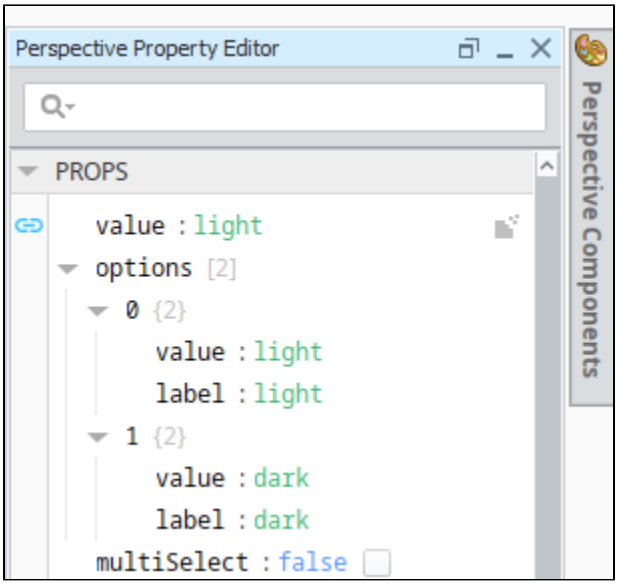
3. The Edit Binding window will open. The theme property is a session property. Click on the **Property** icon and expand **session > props**, and select **theme**. Click **OK**.
4. Check **Bidirectional** and click **OK** to save your property binding.



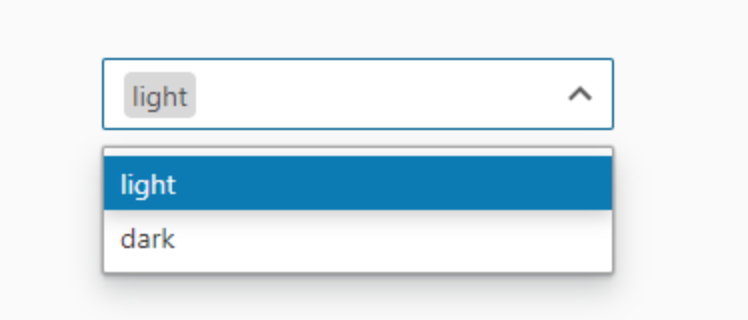
5. In the Property Editor, enter the theme values under options.

   Paste the following onto the "options" property of a dropdown component.

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



6. **Save** your project and open your view in a Session or in Preview Mode.
7. From the Dropdown component, select a desired theme (i.e., light).

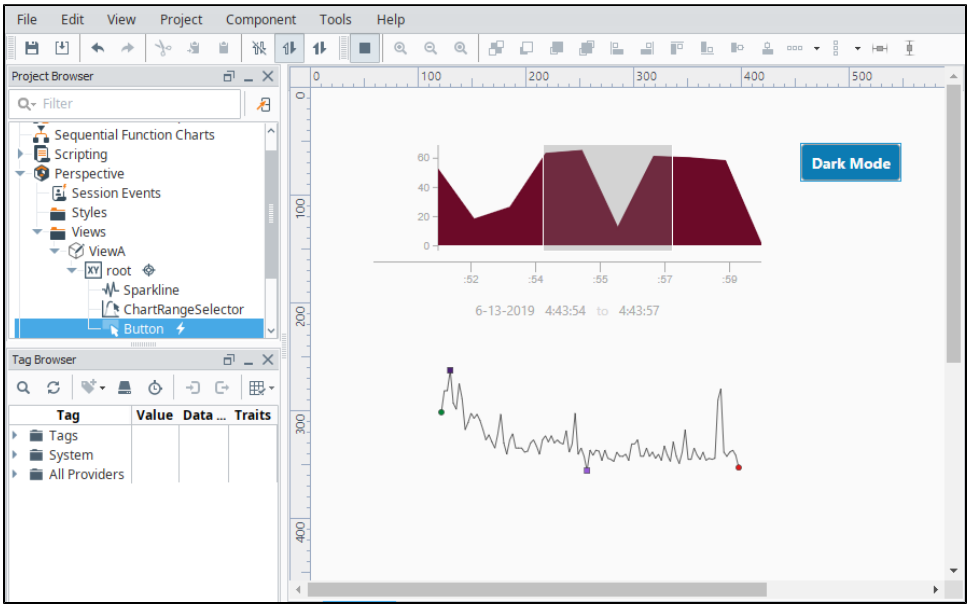


The selected theme will now be applied for the entire Perspective Session.

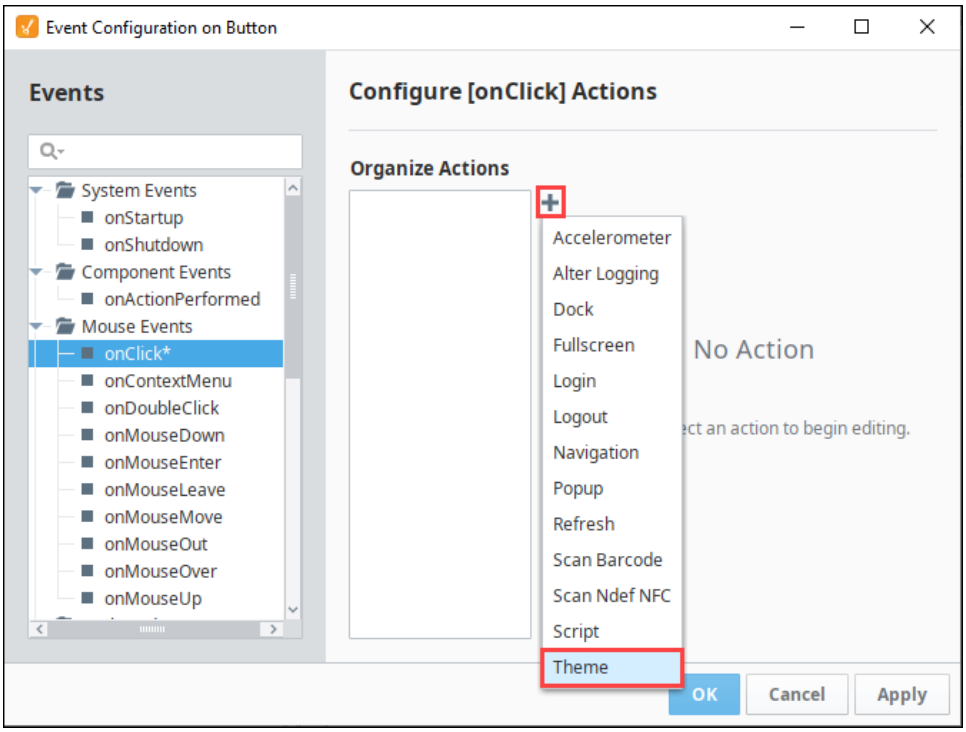
Theme Component Action

You can also set a theme by configuring a Component Action.

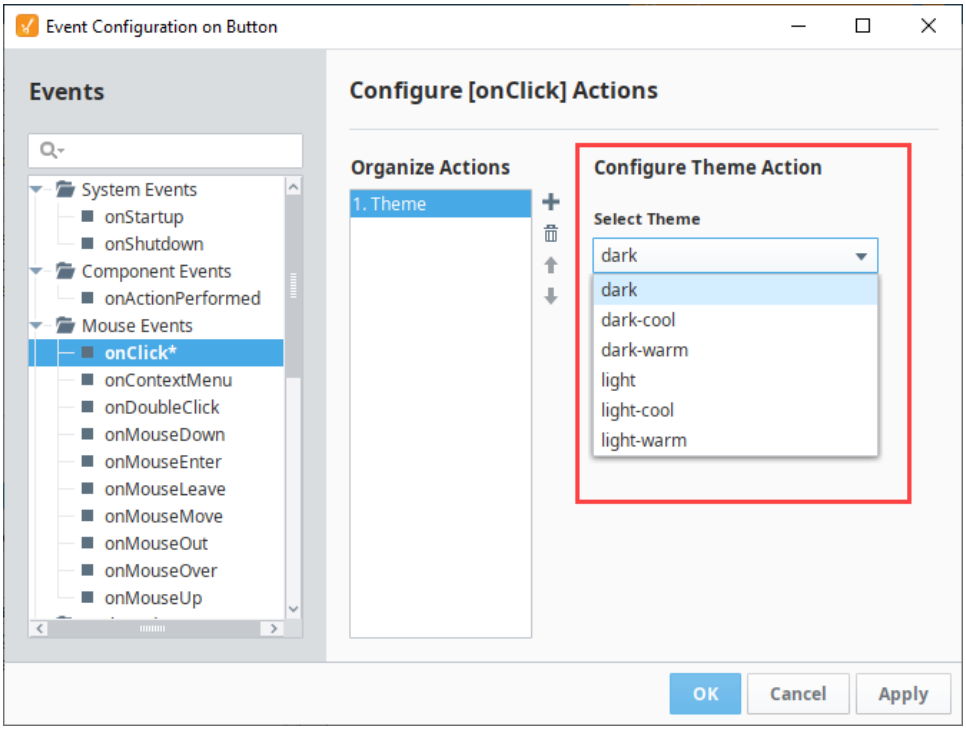
1. Open a view in the Designer with a couple of components, including a button component. The goal here is to change the active theme to "dark" when a user presses the button.
2. Select the Button component, and click on the **Components > Configure Events** at the top of the Designer menubar.



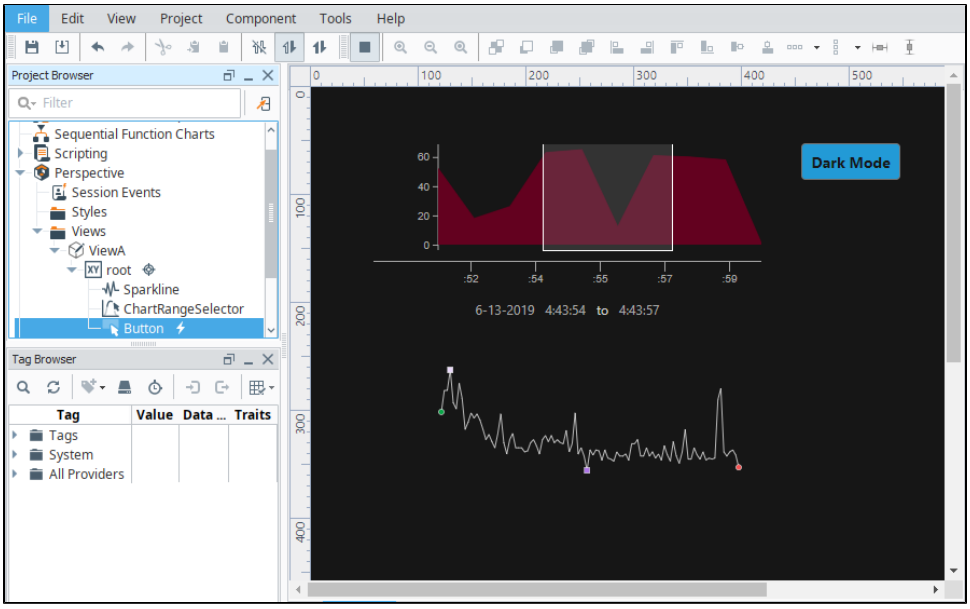
3. The Event Configuration window will appear. Select the event you want to put the Theme on. In this example, we used **onClick.** Click the **Add** icon and choose **Theme.**



4. Select the desired Theme name (i.e., dark) and click **OK**.



5. Click **Save** and open either a Perspective Session, or click Preview Mode if you are in the Designer.
6. Click the **Button** and the theme will change to Dark Mode.



Editor notes are only visible to logged in users

Modifying a Theme - Using a Different Font

Creating a custom Perspective theme is considered an advanced feature. As such, we operate under the assumption that anyone altering these files has some knowledge of CSS. By making use of CSS variables, imports, and descriptive class selectors, you will find that we provided features and structure to help alleviate some of the frustrations inherent to managing large amounts of CSS.

For this example, modify the existing Light theme in *"%ignitionInstallationDirectory%\data\modules\com.inductiveautomation.perspective\themes\"* to have it use the Ubuntu font.

More information on modifying a theme, or making a new theme, can be found at *"%ignitionInstallationDirectory%\data\modules\com.inductiveautomation.perspective\themes\README.md"* file (added in 8.0.13).

Note that modifying or creating new themes is unsupported, so our support department will not be able to assist with any questions you may have. However, feel free to ask any questions on the forums.

1. Download the Ubuntu family font files here: https://fonts.google.com/specimen/Ubuntu. Put all of the font files from your download in a folder named "Ubuntu" and place this folder in `%ignitionInstallationDirectory%\data\modules\com.inductiveautomation.perspective\fonts\`.
2. Go to `%ignitionInstallationDirectory%\data\modules\com.inductiveautomation.perspective\themes\` and create a folder named `overrides`.
3. Inside your `overrides` folder, create two CSS files, one called `fonts`, and the other called `globals`.

   Your `fonts.css` file should look like this:

fonts file example

```
@font-face {
    font-family: var(--font-Ubuntu);
    font-style: normal;
    font-weight: 400;
    src: local('Ubuntu'), local('Ubuntu'),
         url('/data/modules/perspective/fonts/Ubuntu/Ubuntu-Regular.ttf') format('ttf');
}

@font-face {
    font-family: var(--font-Ubuntu);
    font-style: normal;
    font-weight: 500;
    src: local('Ubuntu-Medium'), local('Ubuntu-Medium'),
         url('/data/modules/perspective/fonts/Ubuntu/Ubuntu-Medium.ttf') format('ttf');
}

@font-face {
    font-family: var(--font-Ubuntu);
    font-style: normal;
    font-weight: 700;
    src: local('Ubuntu-Bold'), local('Ubuntu-Bold'),
```

```
        url('/data/modules/perspective/fonts/Ubuntu/Ubuntu-Bold.ttf') format('ttf');
}
```

Your `globals.css` file should look like this:

```
body, input, textarea, keygen, select, button {
    font-family: var(--font-Ubuntu);
}
```

Modify your `light.css` file in *"%ignitionInstallationDirectory%\data\modules\com.inductiveautomation.perspective\themes\"* as follows:

light.css file example

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

Note how you're basically overriding the `/light/fonts.css` file with your own `/overrides/fonts.css` file. The same thing is happening with the `/light/globals.css` file. The order of these imports is important, as an import at the bottom of the list could override imports at the top of the list.

By following these steps, your Light theme is ready to use in your Designer using the Ubuntu font.

Theme Colors

The built-in themes make heavy use of CSS variables for colors. For any of the default themes, colors are defined in the `variables.css` file. Defining your own variable is simple. Add a line with the following to the variables file:

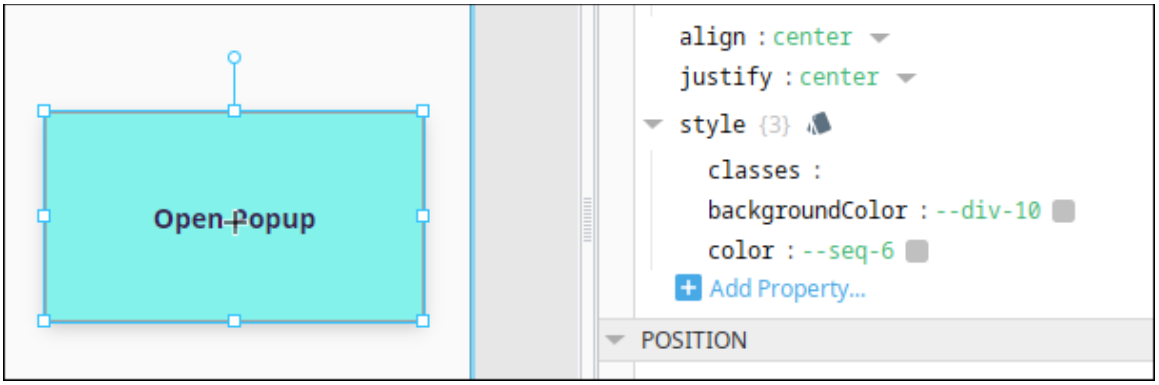
```
--variable-name: #FF0000;
```


Be aware that changes made to the built-in theme files will be replaced on Gateway start up (including restarts caused by a Gateway Restoration) and moved to a backup folder on upgrade. As a result, it is highly recommended that you create a custom CSS file that can then be imported into the entry point CSS files.

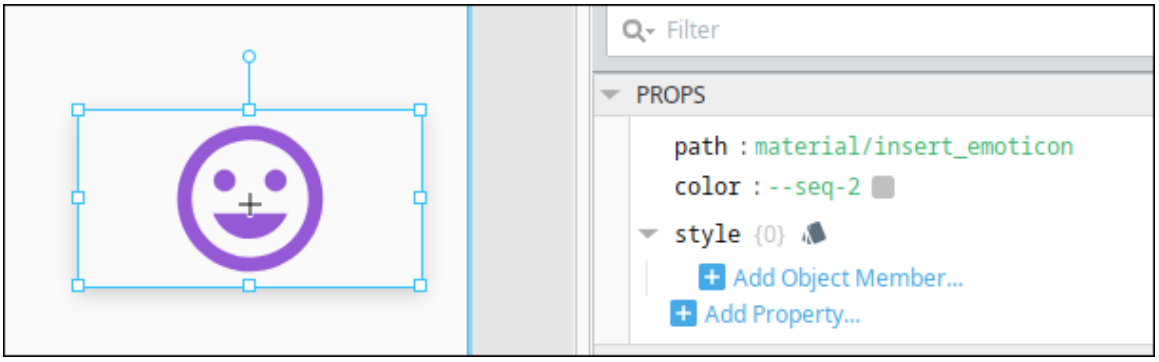
For more information, see the markdown README file located in the Gateway's installation directory: `%installDirectory%\data\modules\com.inductiveautomation.perspective\themes\README.md`

Using Theme Colors

Theme colors can be used on components by simply providing the variable name. For example, we can change the **backgroundColor** and **color** of a button component by just stating the variable name for the appropriate styling properties on the component's **style** object.

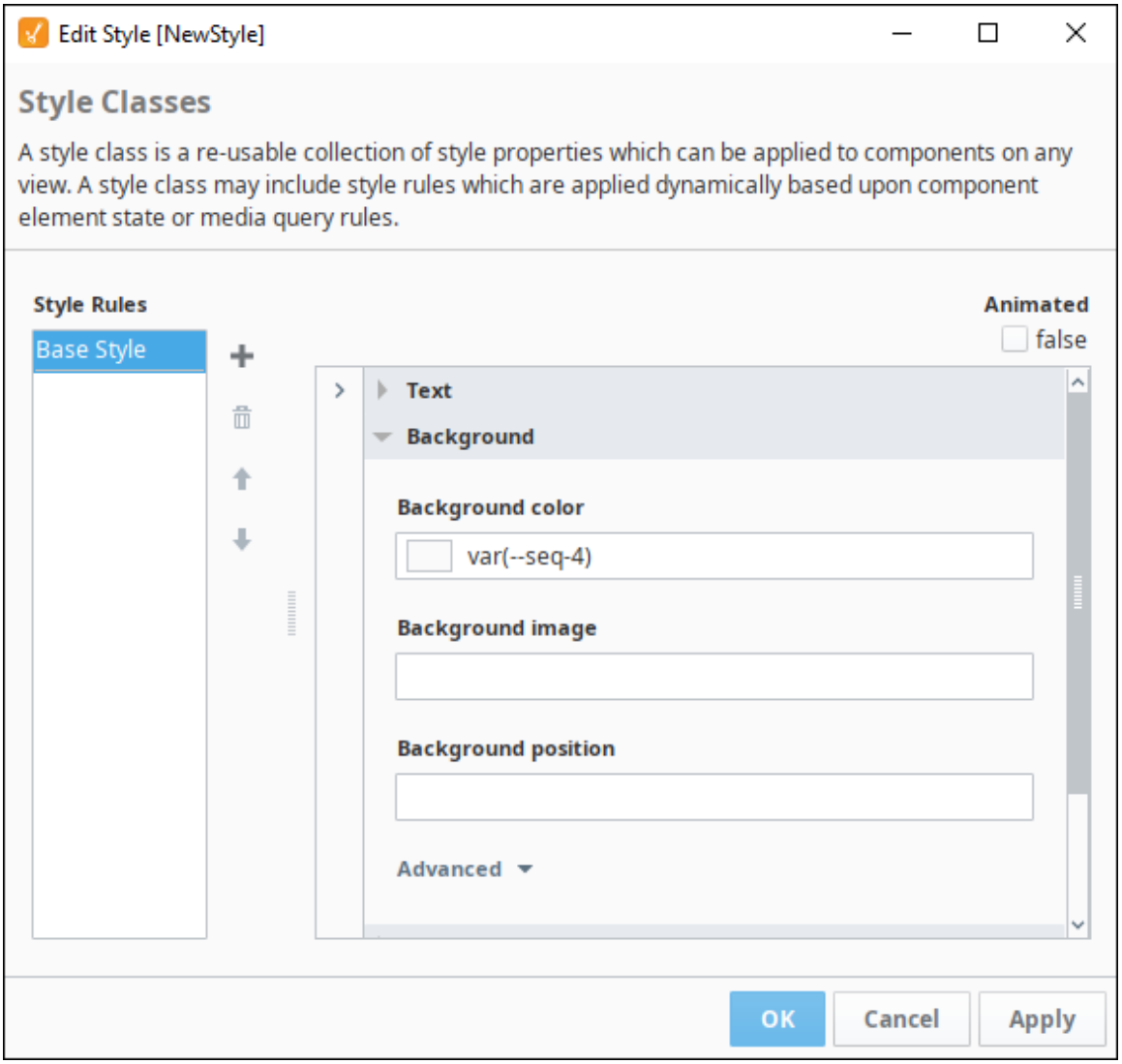


If a component has a **color** property outside of a style object, such as the Icon component, the same rules apply; simply set the value of the color property to the name of the variable.



Style Classes

When using a Theme Color in a Style Class, the variable must be wrapped in the `var()` method, as shown below.



Built-in Theme Colors

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

	light	light-cool	light-warm	dark	dark-cool	dark-warm
--neutral-10						
--neutral-20						
--neutral-30						
--neutral-40						
--neutral-50						
--neutral-60						
--neutral-70						
--neutral-80						
--neutral-90						
--neutral-100						
--seq-1						

--seq-2						
--seq-3						
--seq-4						
--seq-5						
--seq-6						
--div-1						
--div-2						
--div-3						
--div-4						
--div-5						
--div-6						
--div-7						
--div-8						
--div-9						
--div-10						
--div-11						
--div-12						
--div-13						
--div-14						
--div-15						
--div-16						
--qual-1						
--qual-2						
--qual-3						
--qual-4						
--qual-5						
--qual-6						
--qual-7						
--qual-8						
--qual-9						
--qual-10						
--callToAction						
--callToActionHighlight						
--callToAction--hover						
--callToAction--active						
--callToAction--disabled						
--error						
--info						
--infoSecondary						
--warning						

--warningSecondary						
--success						
--indicator						
--indicatorOff						

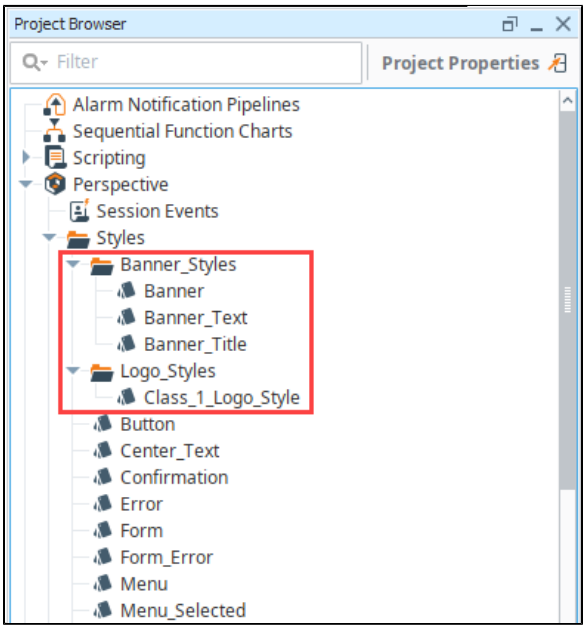
Style Classes

Style Class Basics

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 allows 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.

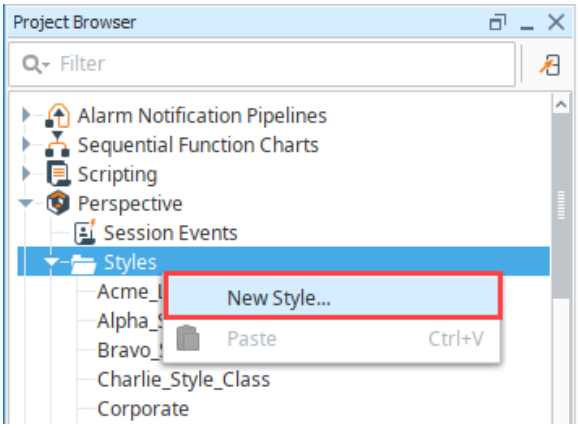
> The following feature is new in Ignition version **8.0.3**
> Click here to check out the other new features

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.



Creating a Style Class

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

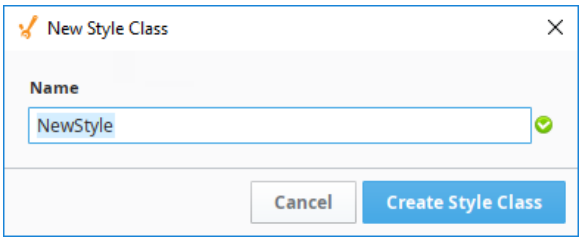


On this page

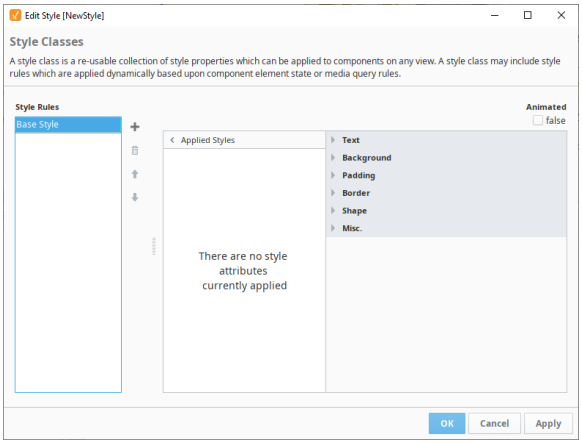
...

- Style Class Basics
  - Creating a Style Class
  - Delete a Style Class
  - Rename a Style Class
  - Protect a Style Class
- Multiple Style Classes
- Animated Style Classes
  - Animated Settings
  - Animated Style Class Example
- Bindings on Style Classes
  - Dynamic Style Class Example
- Element States on Style Classes
  - Element States Example
- Media Query on Style Classes
  - Media Query Settings
  - Media Query Example

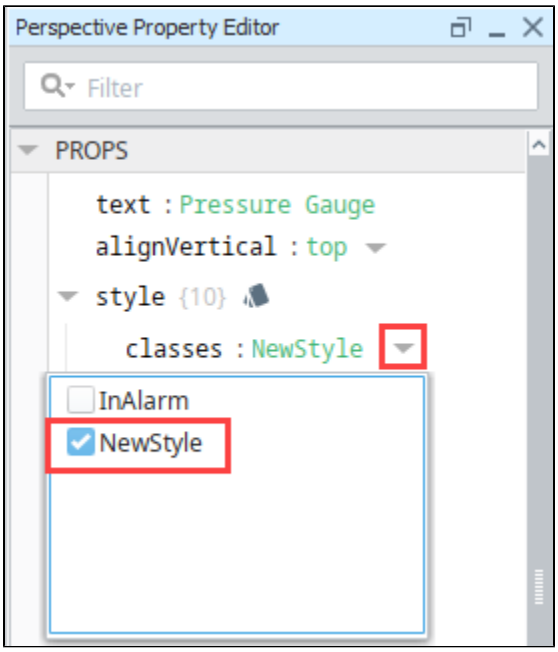
2. Enter a name for the new Style Class. Style Class names must be unique to the project. You will see a green check icon if the name you enter is acceptable.



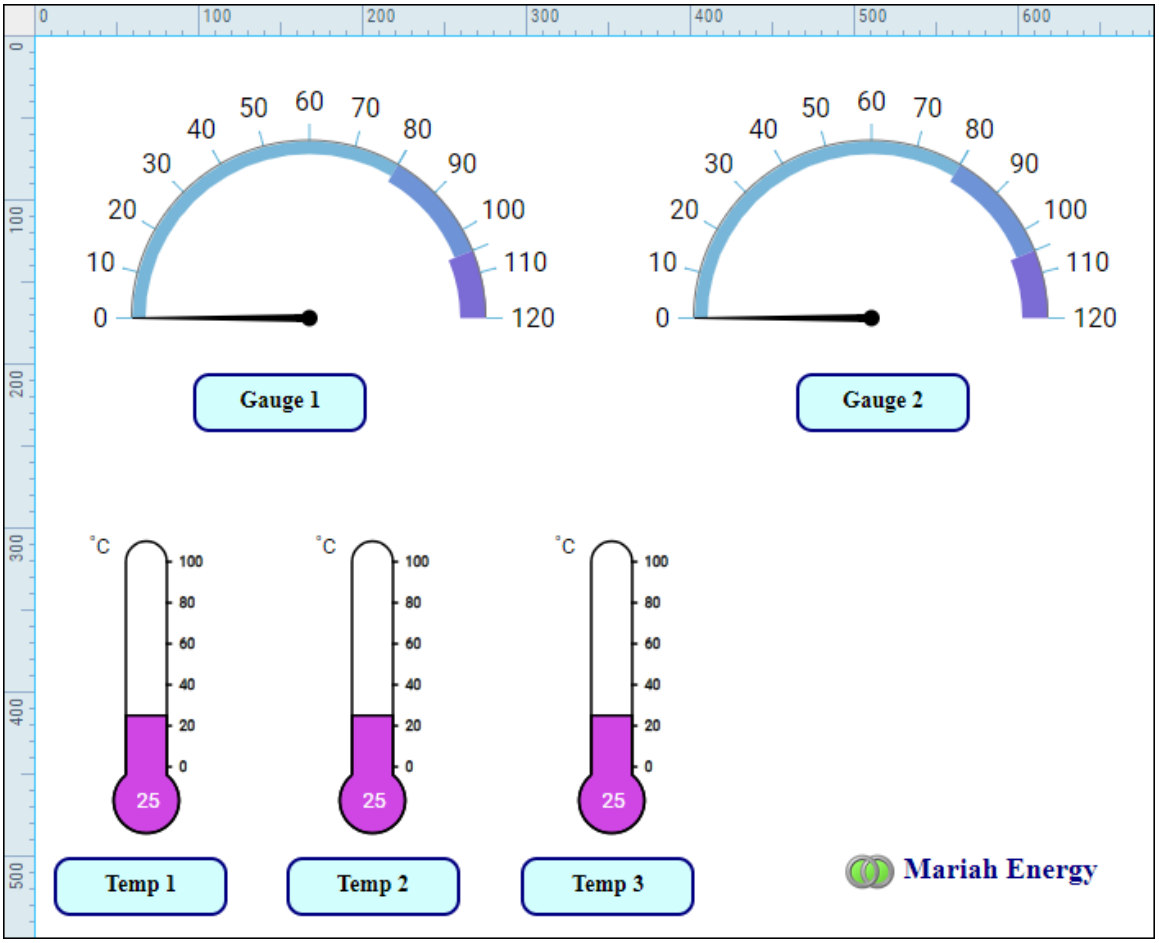
3. Click **Create Style Class**.
4. On the Edit Style screen, use the style editor to set the text, background, border, and so forth, for that style.



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.

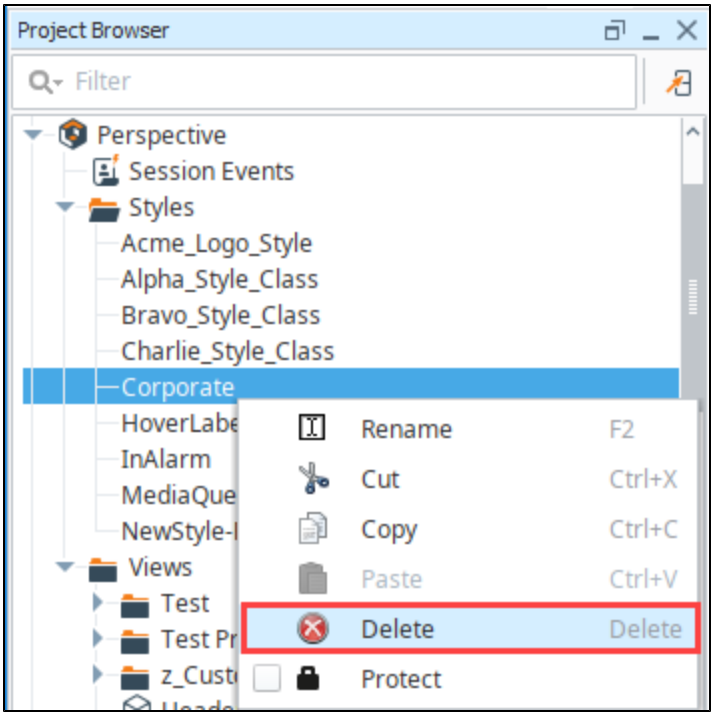


The following is an example screen where the label components were created then all given the same Style Class, which has settings for text weight, font family, background color, and border.



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.

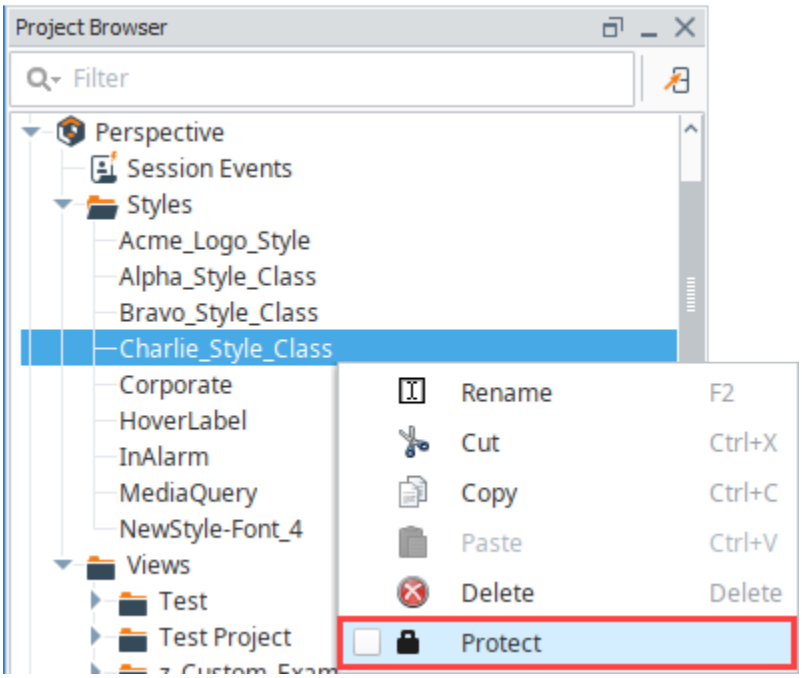


Rename a Style Class

To rename a Style Class, right-click on the Style Class name in the Project Browser, then select **Rename**. Once a Style Class is renamed, it will no longer be applied to any components it was previously applied to.

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 Designer.

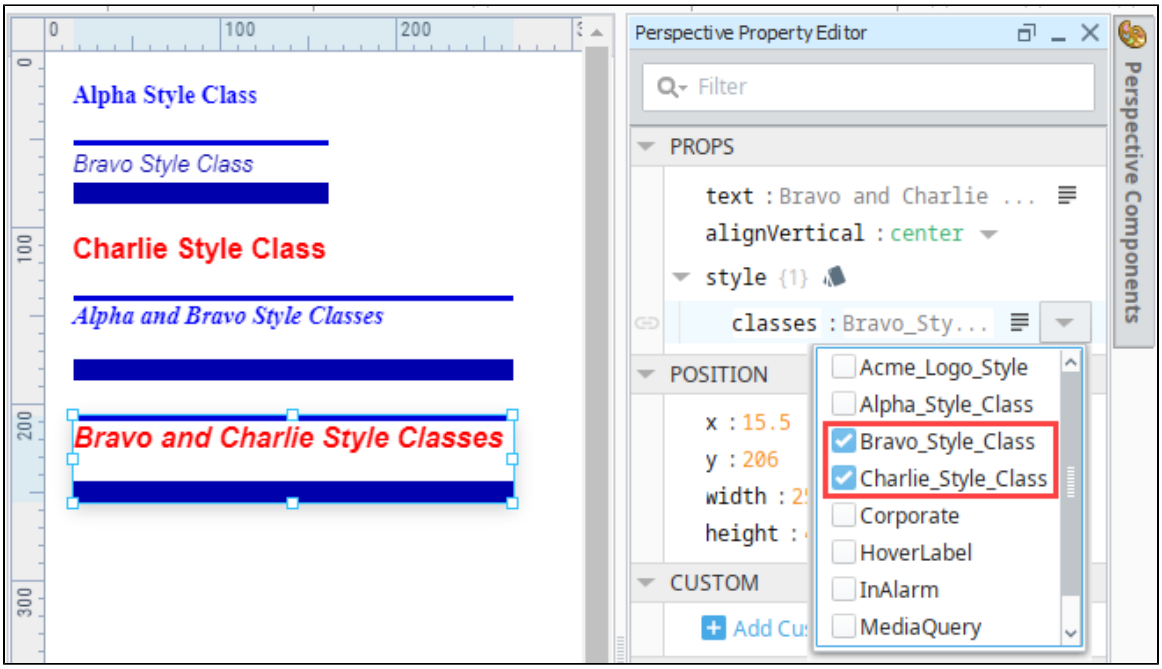


Multiple Style Classes

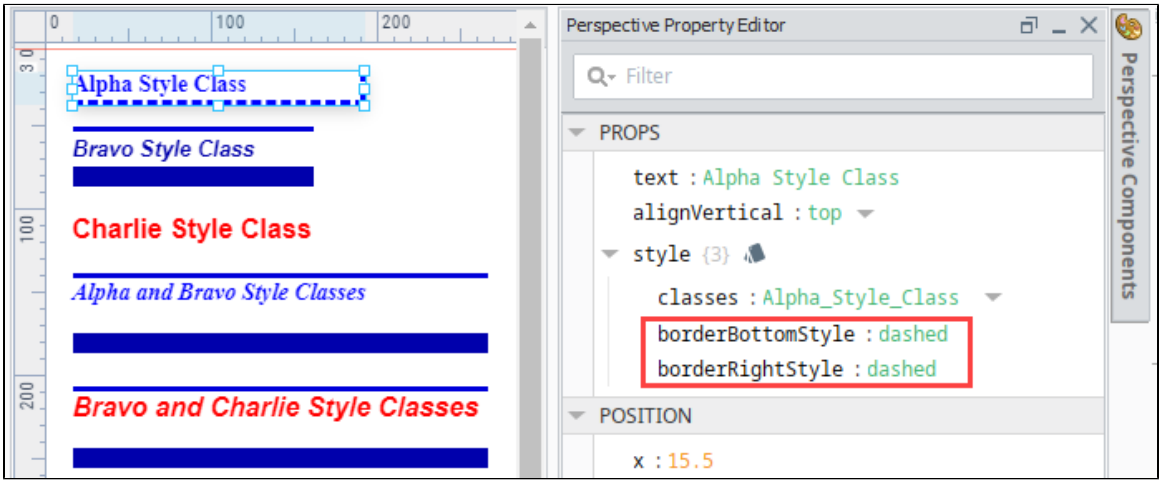
Perspective allows you to select more than one Style Class for a component. Multiple Style Classes are applied in alphabetical order. Style classes further along in the alphabet will override earlier style classes. The overrides occur only for properties that are set in multiple classes.

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.



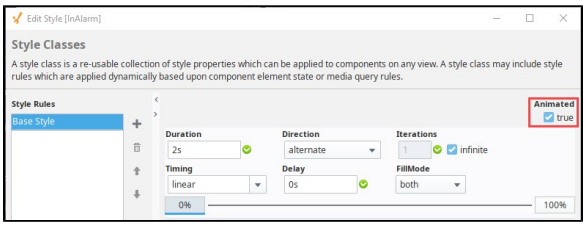
You can also apply inline styles to components that have a style class. The inline style properties override any properties in the style class. For example, if the style class has a properties of 22pt bold text with a color of blue, but there is an inline style property of 18 pt bold text with a color 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

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.



INDUCTIVE UNIVERSITY

Animated Style Classes

Watch the Video

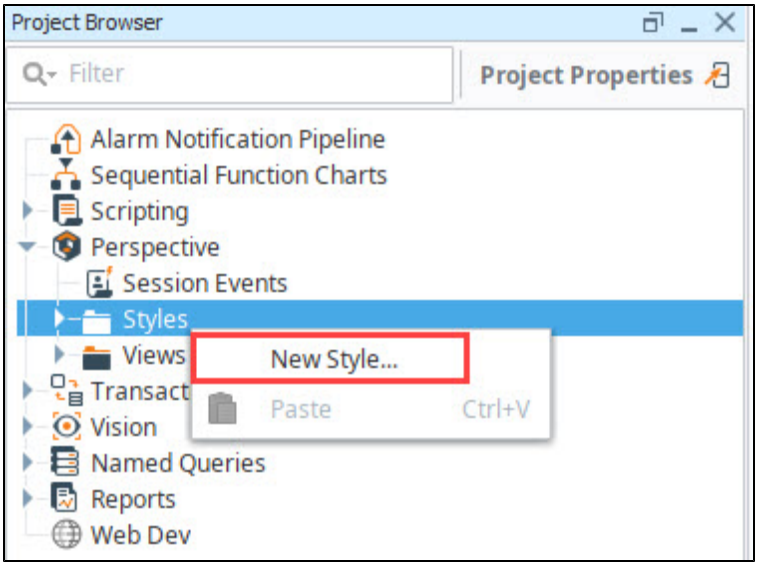
Animated Settings

The following properties appear when the Animated checkbox is set to True.

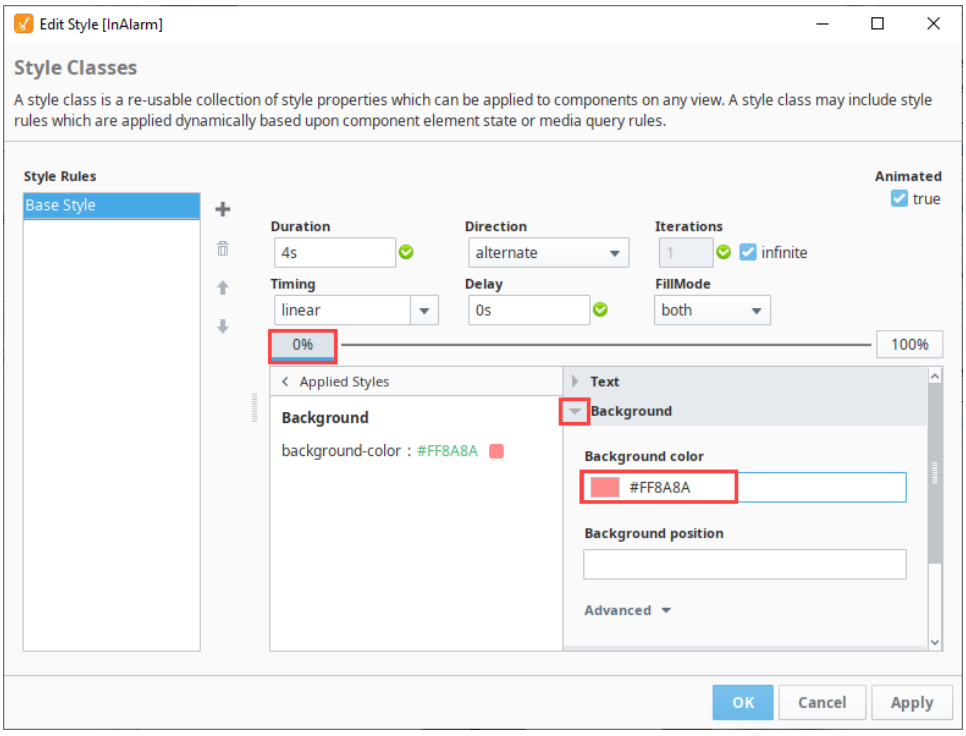
Name	Description
Duration	Number of seconds for each animation stop.
Direction	Options are normal, reverse, alternate, and alternate reverse.
Iterations	Number of times you want the animation to run. Enter an integer, or check the infinite box to have the animation run continually.
Timing	Options are 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	The following feature is new in Ignition version **8.0.8** Click here to check out the other new features Displays the style names and settings as you add them to a component.

Animated Style Class Example

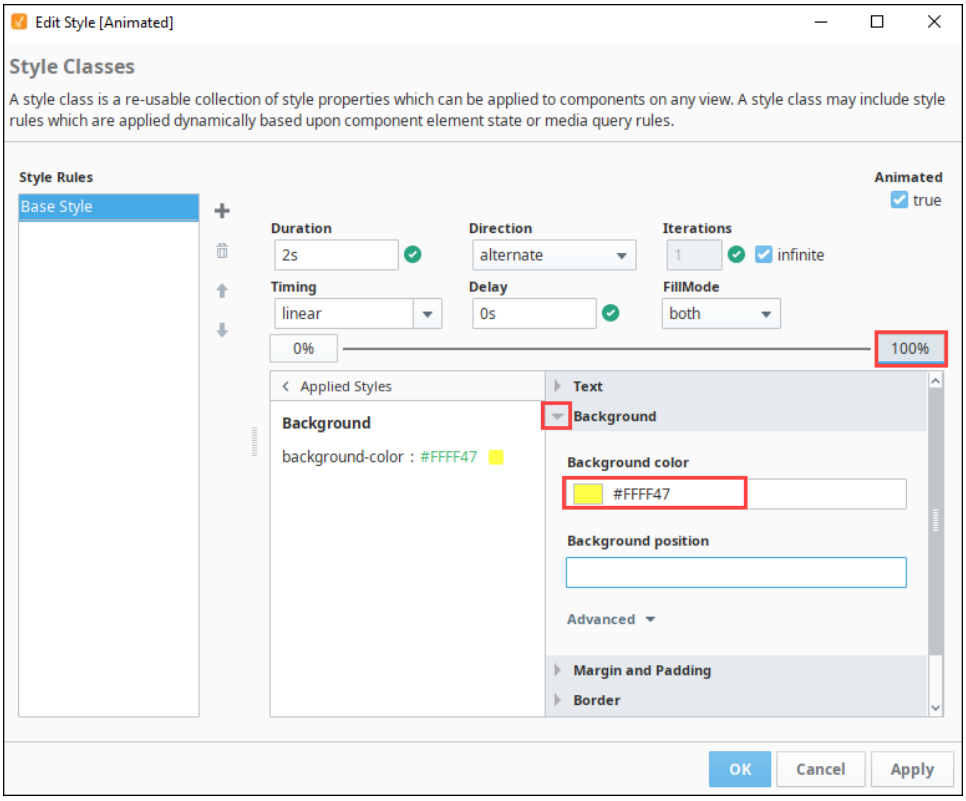
1. In the Project Browser, right-click Styles and select **New Style**.



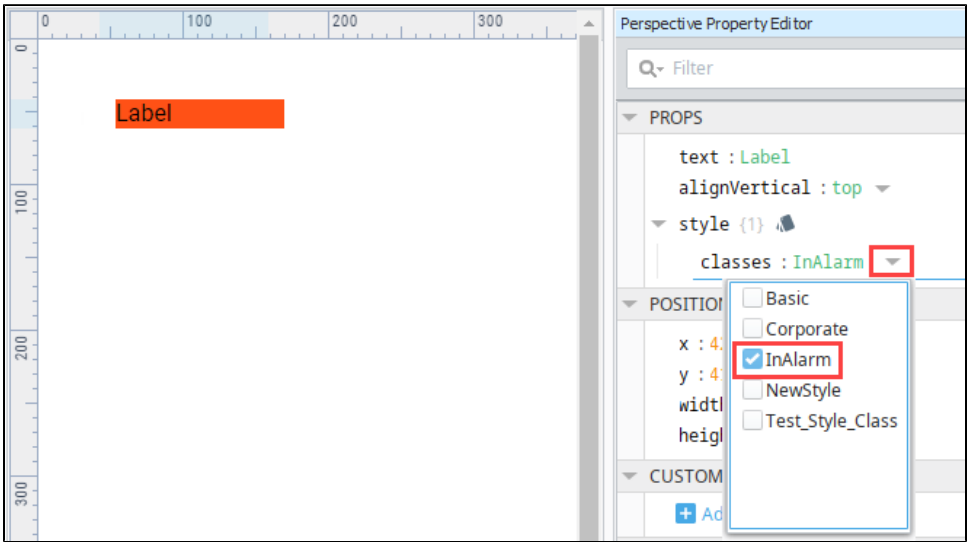
2. Enter a Style Class name, for this example we used "InAlarm", and click **Create Style Class**.
3. On the Edit Style screen, check the **Animated** option in the upper right corner. For this example, we'll leave the default settings in the fields.
4. Next click on **0%** to set the style for the beginning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

Bindings on Style Classes

In Perspective, you can dynamically change the style class on a component. Styles and Style Classes can have binding options. For example, a component could use a style class when a Tag has one value, and not use the class when the Tag has a different value.

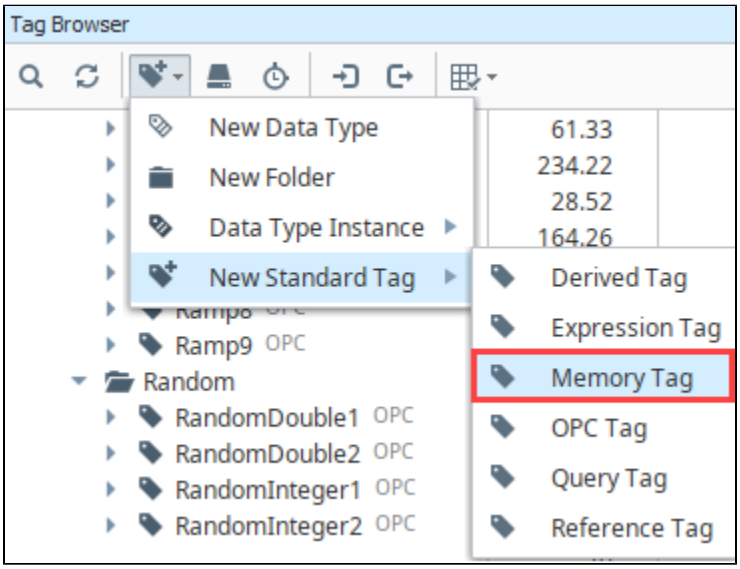
Dynamic Style Classes

Watch the Video

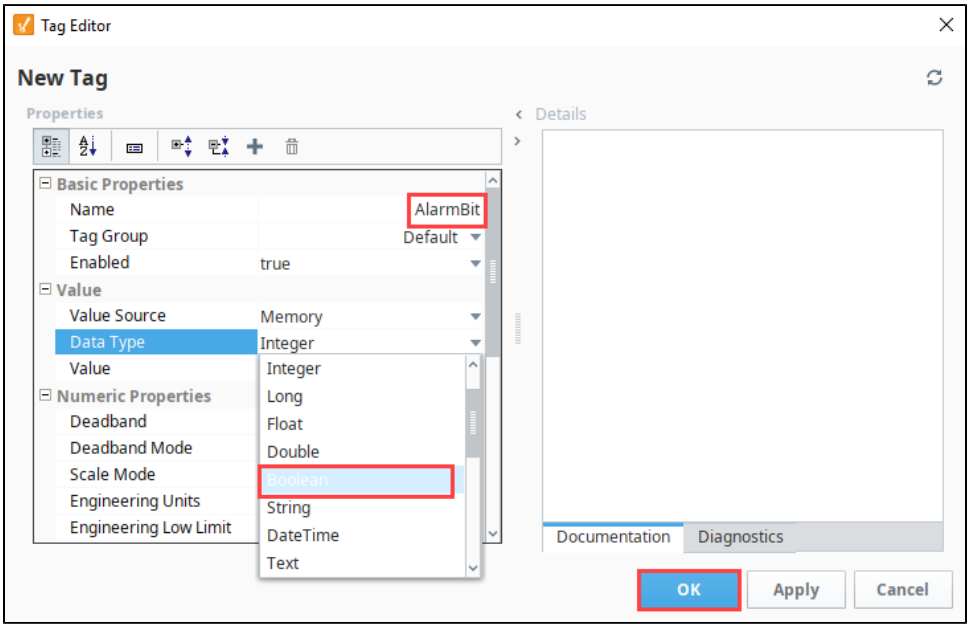
Dynamic Style Class Example

In this example, we use the InAlarm style class and Label component we set up in the previous example.

1. Click on the New Tag icon and select **New Standard Tag** and **Memory Tag**.

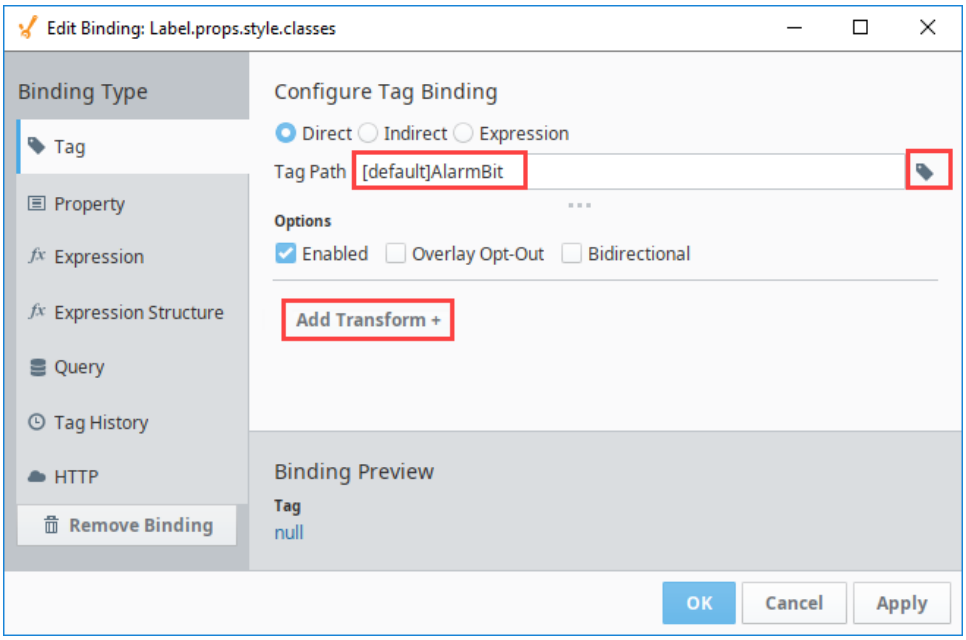


2. Give the Tag a name.
3. Set the data type to **boolean**, and click **OK** to save the Tag.

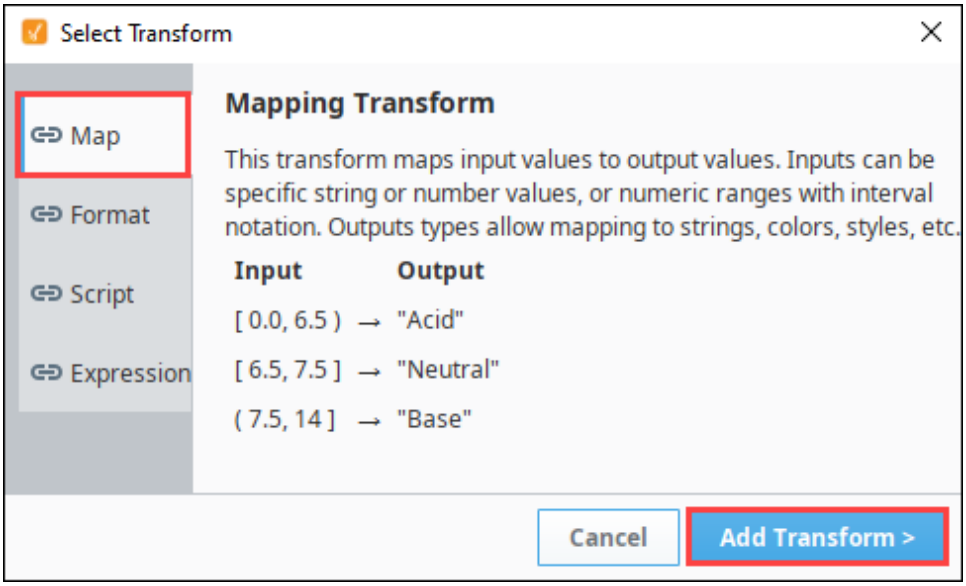


4. On the view, select the Label component.
5. In the Property Editor, click on the **Binding** icon next to classes.
6. Select the Tag Binding type.
7. On the Edit Binding screen, click the **Browse Tags** icon and select the AlarmBit tag.

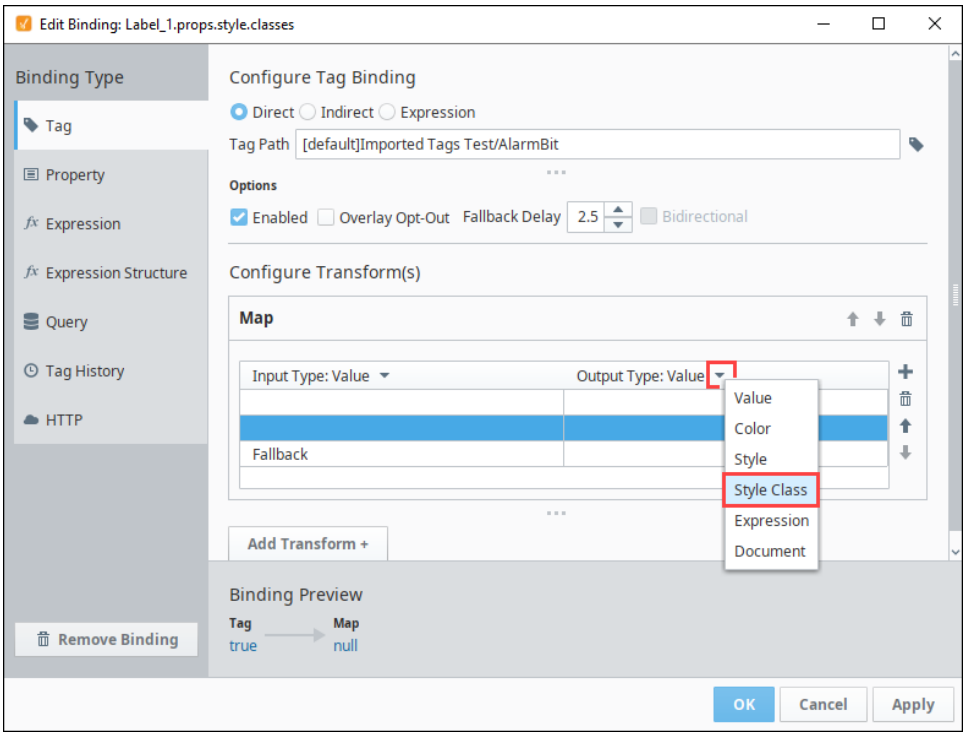
8. Next, click on **Add Transform**.



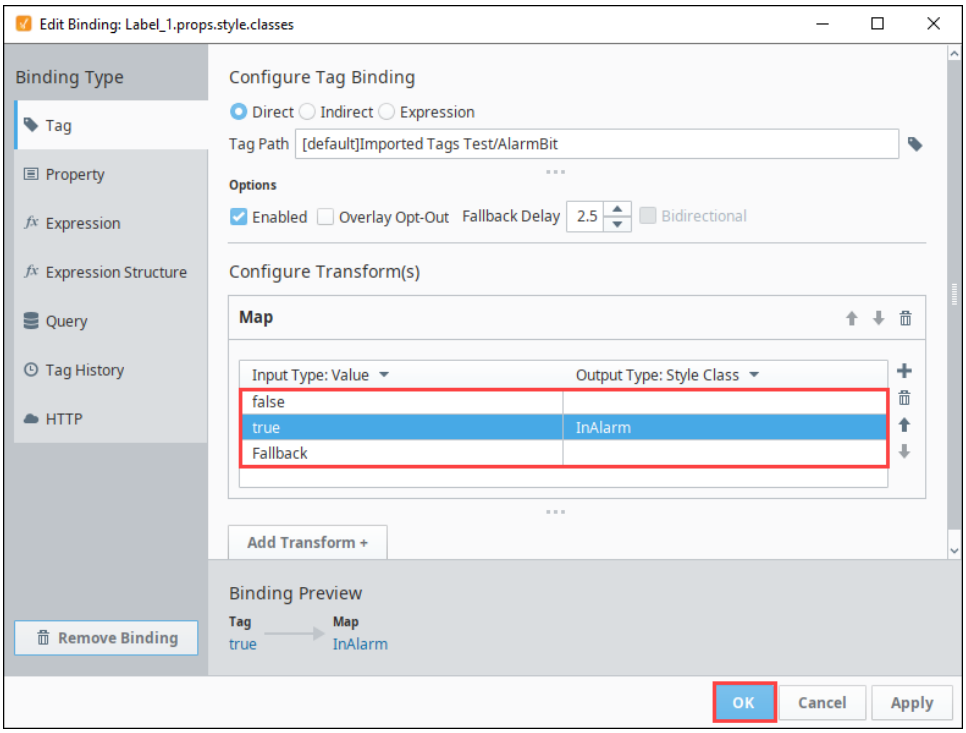
9. On the Select Transform screen, choose Map and click **Add Transform**.



10. Click the **Add** ✚ icon twice to add two mappings.
11. Click on the **Output Type** dropdown and choose Style Class.



12. For the first mapping enter false as the input type and leave the output type blank. This will cause the Label component to have no style class when the value on the Tag is false.
13. For the second mapping enter true as the input type and use the dropdown to select "InAlarm" as the output type. This will cause the Label component to the InAlarm style class when the value on the Tag is true.
14. For the Fallback, leave the output type blank.



15. Click **OK** to save the binding.
16. To test the binding, write to the Tag. The component will change depending the Tag's value.

Element States on Style Classes

Element States are used on Style Classes to change the style configuration on a component based on the state of the component. It is an additional styling configuration you can make on a style class. For example, you could have the border on a component change when it is disabled. Or you could have the background change when the user hovers the mouse over the component. See the table below for element states that are available. These states are based in CSS pseudo classes.

Element States

Watch the Video

Name	Description
Element State	State of the component. Options are as follows:

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.
disabled	Component is disabled that is 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	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 "tab" key.
hover	User hovers mouse over the component.
in-range	Element whose current value is within the range limits specified by the `min` and `max` attributes.
read-only	Component is read-only.
read-write	Component is available for read and write.
visited	Component links that the user has already visited.

Editor notes are only visible to logged in users
Per dev, some of these element states will not work yet in Perspective. There's a ticket. Some may end up being dropped /deleted and some may be implemented in the future.

default-choice, invalid, last child, link, only-child, out-of-range, required, valid

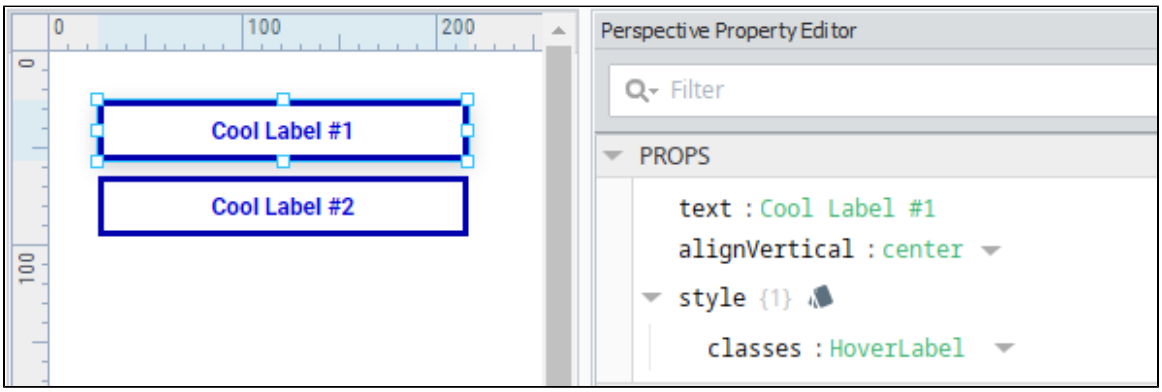
Animated	If checked, the Element State can be animated. The animate options are as follows:

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,

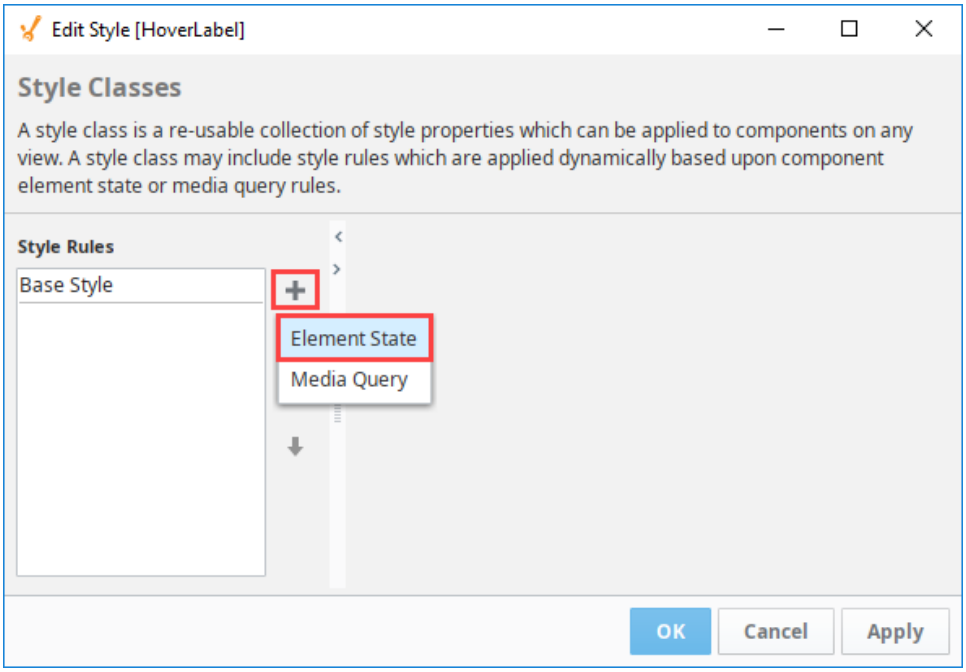
Delay	Delay, in number of seconds.
FillMode	Options are none, forwards, backwards, or both.
Style Settings	Sets a style animation stop. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous.

Element States Example

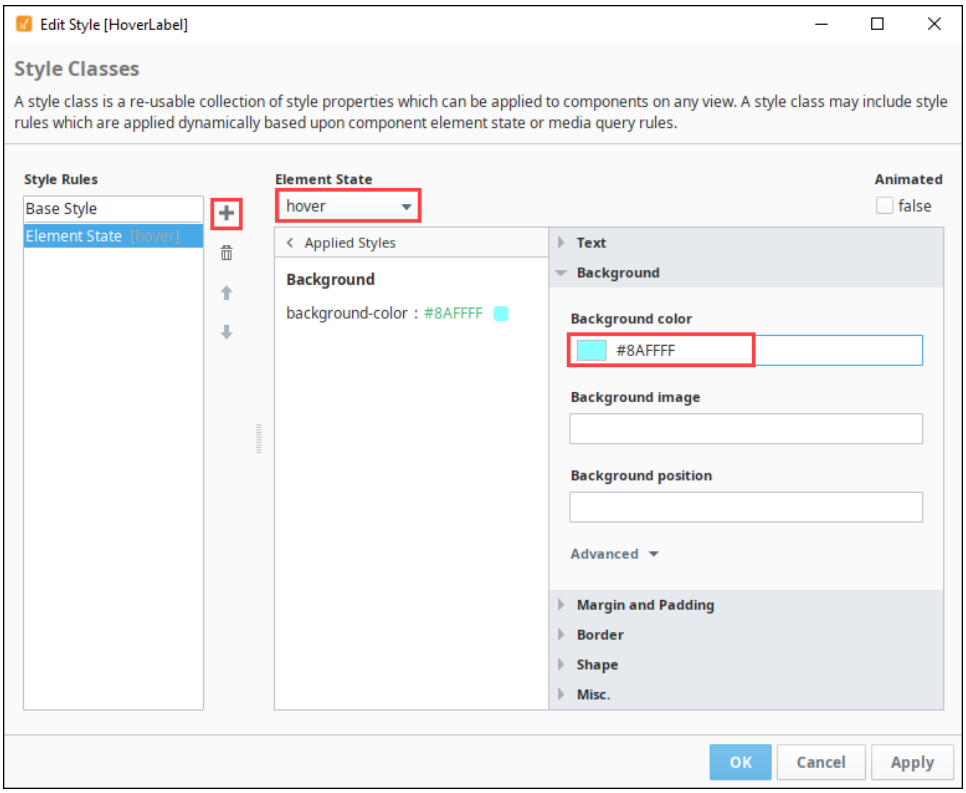
In this example, we have a Style Class called **HoverLabel**. The base style for this class has some text settings and border settings. We applie d the class to two Label components on view. Let's set up an element state that will change the background color of the component if a user hovers over the component with a mouse.



1. To modify the Style Class, double click on it in the Project Browser.
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 for the first animation stop. We chose **#8AFFFF**, a light blue.



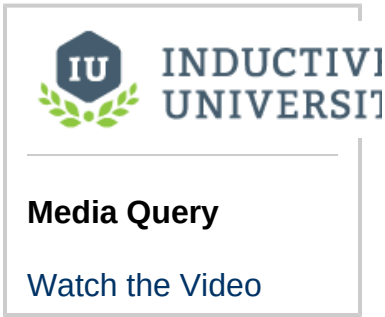
6. Click **OK**. Now the background color will change on the Label component that the user hovers over.

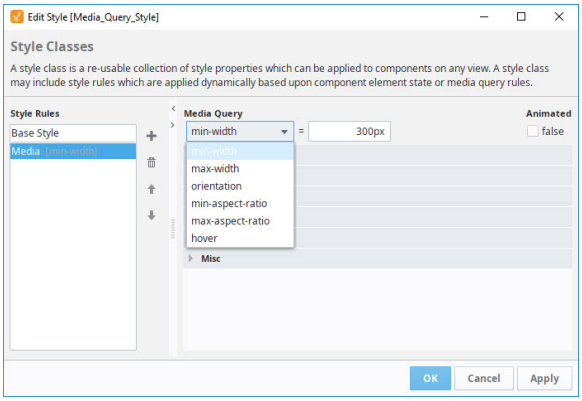
Media Query on Style Classes

A Media Query can be applied to a Style Class in order to change the style of your Perspective components based on the device your session is running on. That is, you can change the style of your Perspective components based on the device your session is running on.

Media queries don't change anything about your layout, rather they react to those changes in the device and then choose from the various style rules you defined.

For example, you could make changes to your styles based on the width of the session. Media queries in style classes are a direct import of CSS rules. Depending on the particular media query, the selector will apply at less than or equal to, or vice versa.





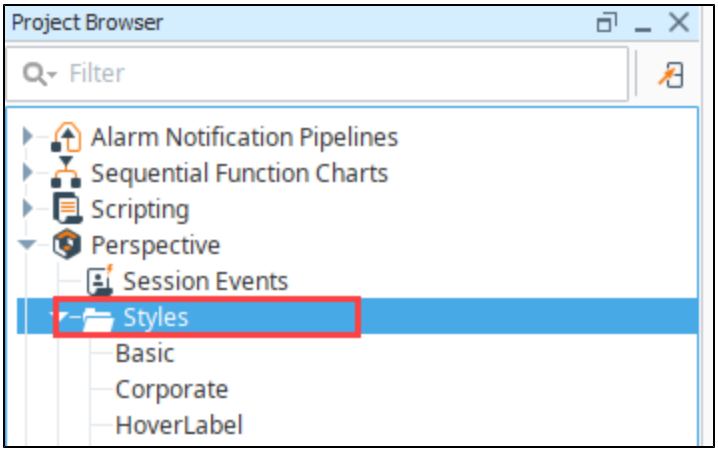
Media Query Settings

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.

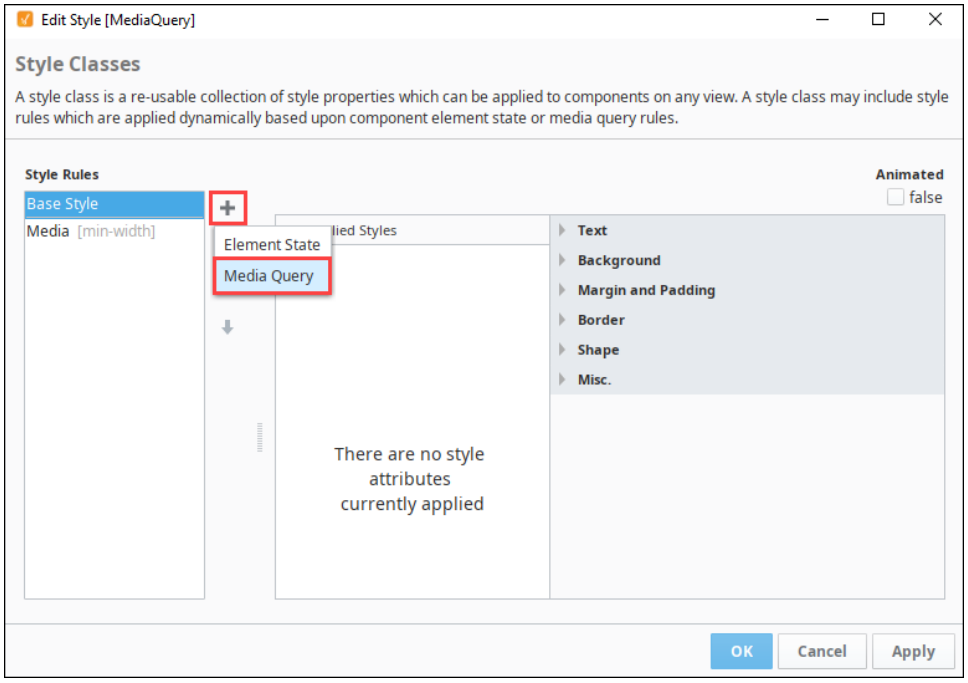
Media Query Example

For the following example, we placed a Label component with no style class on a view.

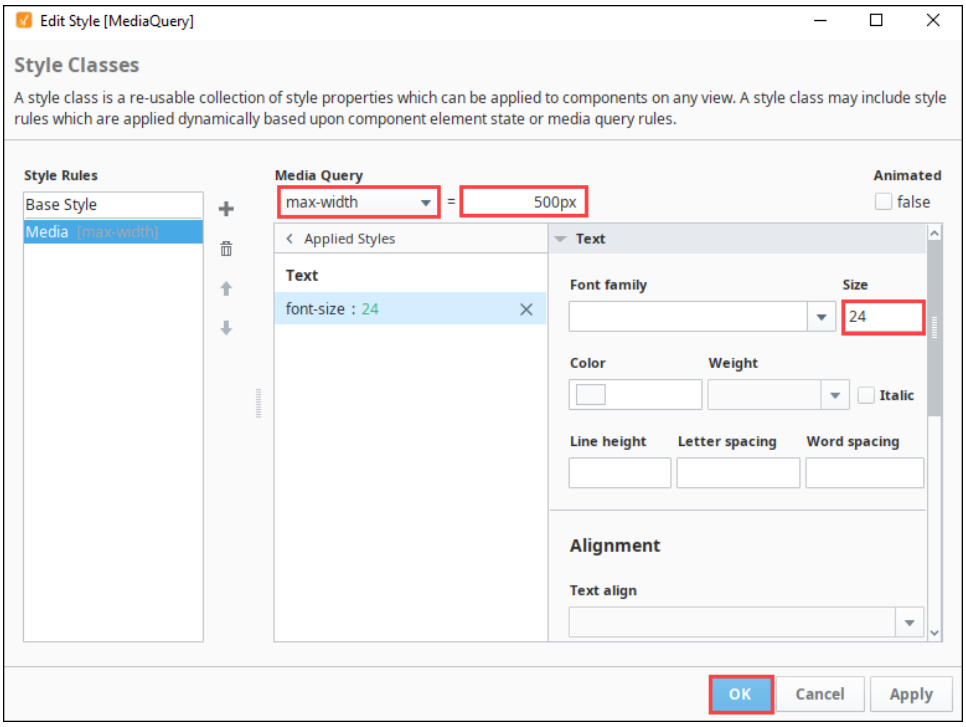
1. To create a Style Class, right-click on the **Styles** folder and select **New Style**.



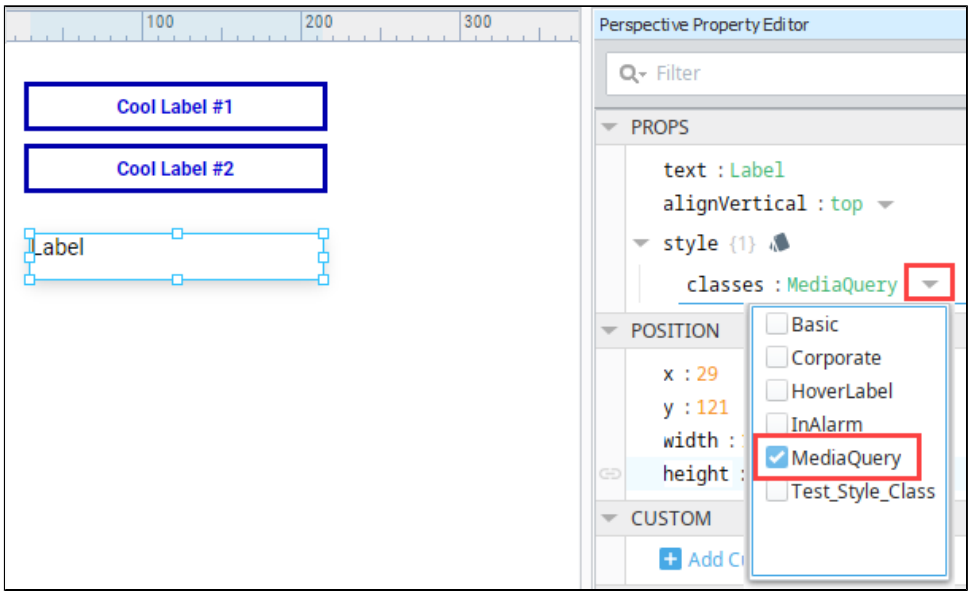
2. Give the new Style Class a name. We chose MediaQuery.
3. Click **Create Style Class**.
4. Under Style Rules, click on the **Add** ✚ icon and select Media Query.



5. Next we updated a few of the settings. Change Media Query to max-width and the pixel value to 500px. We also set the Text Font size to 24px.
6. Click **OK** to save the Style Class.



7. Back in the Designer, select the Label component.
8. In the Property Editor, click on the **Expand** ▶ icon next to classes.
9. Select the MediaQuery Style Class.



10. **Save** the project.
11. Launch a Perspective Session. Now you should be able to see the font size on the label change as you adjust the size of your session window.

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. For Vision users you will find bindings in Perspective operate very similar to the way they work in Vision.

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.

On this page

...

- Bidirectional Bindings
- Binding Interface
- Property Paths
- Tag Binding
  - Direct Example
  - Indirect Example
  - Expression Example
- Property Binding
- Expression Binding
  - Expression Binding Examples
- Expression Structure
- Query Binding
- Tag History
  - Direct Tag Example
  - Expression Tag Example 1
  - Expression Tag Example 2
- HTTP Binding
  - Weather Data Example

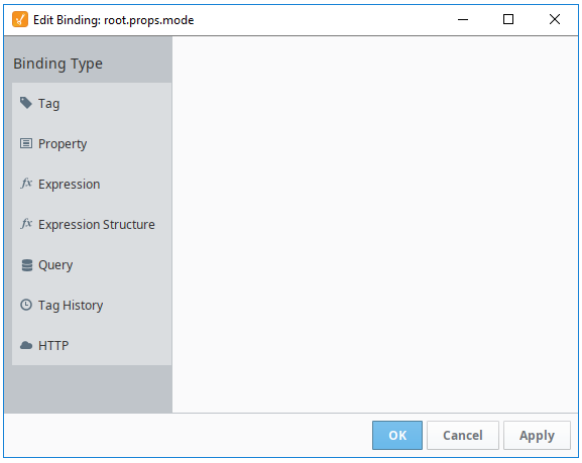
Bidirectional Bindings

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

Binding Interface

A property can have many different types of bindings, for example it can have a Tag or an Expression binding. Instead of setting a label statically, the text might change based on a PLC value or on-screen selection. There many ways to bind your components to show values from PLCs, databases, other components, or user input. You can even bind some or all of the properties on each component. You can bind component values using:

- **Tag** - Binds a property directly to a Tag which sets up a Tag subscription for that Tag, and every time the chosen property changes, the binding is evaluated, and pushes the new value into the inbound property.
- **Property** - Simply binds one property to another. When that property changes, the new value is pushed into the property that the binding is setup on.
- **Expression** - The most powerful type of property binding. It uses simple expression language to calculate a value which can involve lots of dynamic data.
- **Expression Structure** - A powerful type of property binding. It uses the property structure to pass data.
- **Query** - A polling binding type that runs a structured Query against any of the database connections configured in the your Gateway.
- **Tag History** - Used for dataset type properties. It runs a query against the Tag Historian.
- **HTTP** - Used for passing data directly to and from a URL link.



Property Paths

Many of the bindings can utilize a string property path, such as a Property Binding, or Indirect Tag Binding. 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. You may not have a binding refer to a property in another view instance, even view instances that may be embedded in your view. (Views may expose property values to their parent via output parameters.)

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
    - LabelA
    - LabelB
    - Sub_Container1
        - ButtonA
        - ButtonB
    - Sub_Container2
        - ButtonA
        - ButtonB
    - root

Operator /Keyword	Description	Example

`/`	Slash Operator - When a path starts with this operator, then it defines an absolute path. That is, a path that starts at the top of the view hierarchy and is not relative to where the binding is being configured.
	
	When not at the start of a path, the / operator moves further into a container, drilling further down into the hierarchy.
		`// Absolute path. Sequential slashes allow for movement into a container`
		`/root/LabelA.position.x`
		`/root/Sub_Container1/ButtonA.position.y`

`.`	Dot Operator - You may access properties deep within a component's property document structure using the Dot Operator. Assuming the component LabelA had a META property named "foo", then we could use the example on the right to retrieve the value of foo. The Dot Operator can also be used to move further into a complex component. Assuming LabelA has object under META named "rotate", we can move into rotate with further use of the Dot Operator.	`/root/LabelA.meta.foo` `/root/LabelA.meta.rotate.angle`
`[]`	Brackets - When referencing an array property, brackets allow you to specify an individual index within the array.	`/root/LabelA.props.complex.bar[5]`

`../`	Parent Container Operator - This operator acts as a shorthand reference to the parent container. Because the operator always returns the immediate parent container, the operator is relative to the component trying to utilize the operator.
	
	When moving up in the hierarchy, multiple uses of this operator may be used in sequence to climb up multiple containers.
	
	Alternatively, you may simply add additional dots to move up levels. Each additional dot moves up another level.
		`// From ButtonA, we can use this operator quickly move to a sibling component`
		`../ButtonB.position.x`
		
		Move Up Multiple Parent Containers
		`// Moving up multiple parent containers`
		`../../LabelA.position.x`

		`// Also moves up: each additional dot is another parent container .../LabelA.position.x`
`./`	Container Self Operator - When configuring a binding from a container, this operator acts as a shorthand reference to the container. This is similar in concept to the this keyword, but still allows for the user of the other operators. Note that this operator only works when the path is on a binding configured on a container.	`./LabelA.position.x`
`this`	The this keyword allows you to easily reference the same component the binding has been placed on. This works on any object, including containers, views, and even the session.	`this.meta.name`
parent	Parent shortcut - References your immediate parent. This keyword is only valid when being evaluated from the scope of a component. For example, LabelA could reference the root container variables. Note that all of these shortcuts cannot be used with any other path separators, so a path like this/MyChild.position.x is invalid, for that, you'd use ./	`parent.props.complex.foo or parent.position.x`
view	View keyword - Refers to the view that a component is contained in. This is only valid when being evaluated from the scope of a component. Lastly, the view shortcut references the view itself. Views may have input and output parameters, and to reference these parameters simply specify the category and name of the parameter, as shown in the example.	`view.params.paramName`
page	Page keyword - refers to the page that the object is contained in. This is only valid when being evaluated from the scope of a view.	
session	Session keyword - Refers to the session object. This keyword is valid from any object type.	

Tag Binding

A Tag Binding allows a tag value (or property) to be bound to a property of a component. A typical example would be a temperature Tag linked to the text property of a Label component.

The following modes are available:

- **Direct**: Bind the property to a Tag path.
- **Indirect**: Allows properties to be placed in the Tag path, providing a way to make the binding dynamic.
- **Expression**: Utilizes the Expression language to build a Tag path. The Tag path in the Expression is expected to be a string. Unlike the Expression binding, this mode allows the bound property (Tag) to be bidirectional.

These options are available:

- **Enabled:** Allows the component to be active/in use /interactive on the screen.
- **Overlay Opt-Out:** Choosing the Overlay Opt-out option will ignore the quality of the chosen Tag (or expression), making it have no effect on the component's quality overlay.
- **Bidirectional:** Allows user input or parameter changes on the component to be passed back to the Tag or property that the binding refers to.

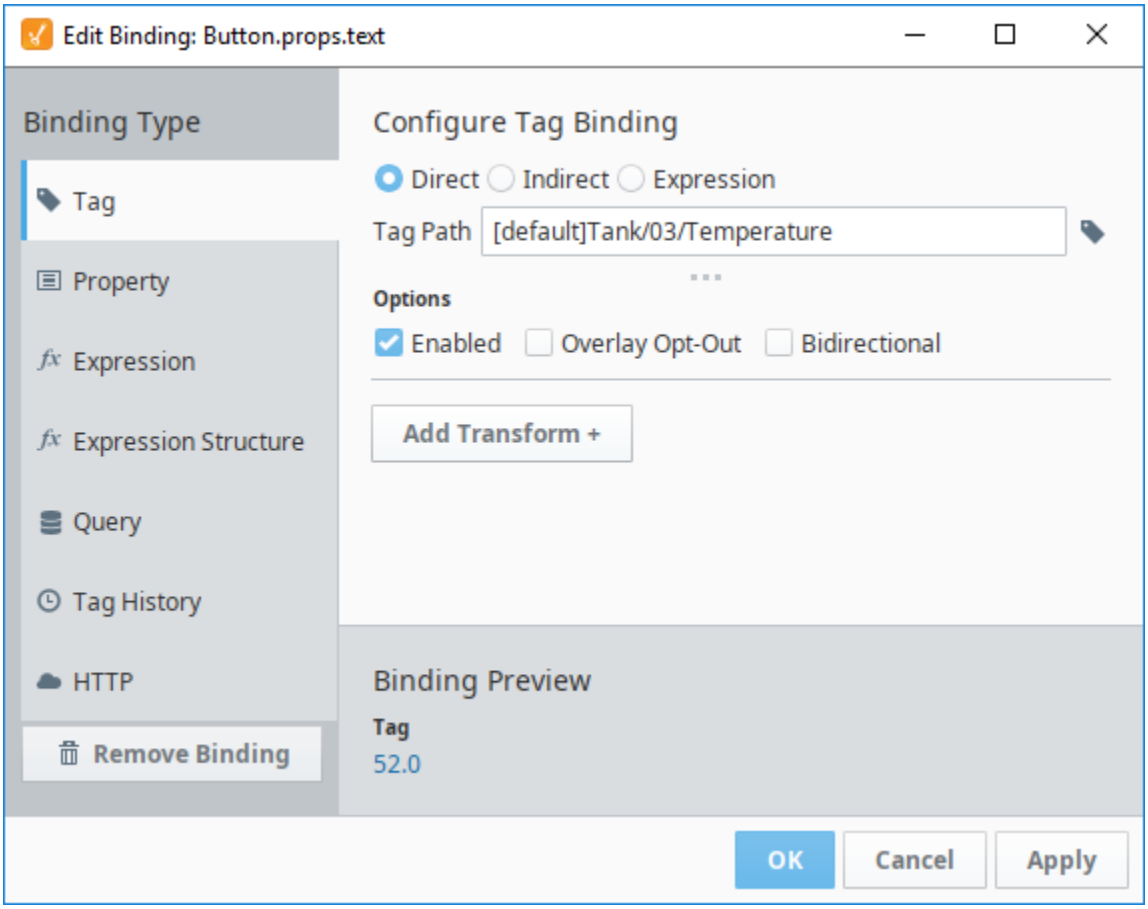
Transforms can be added:

Transforms offer a chance to alter the value returned from a binding. For example, you can bind a p roperty to an integer value and use a transform to map the numerical value to a particular color, all from the same interface. For more information, see Transforms in Perspective.

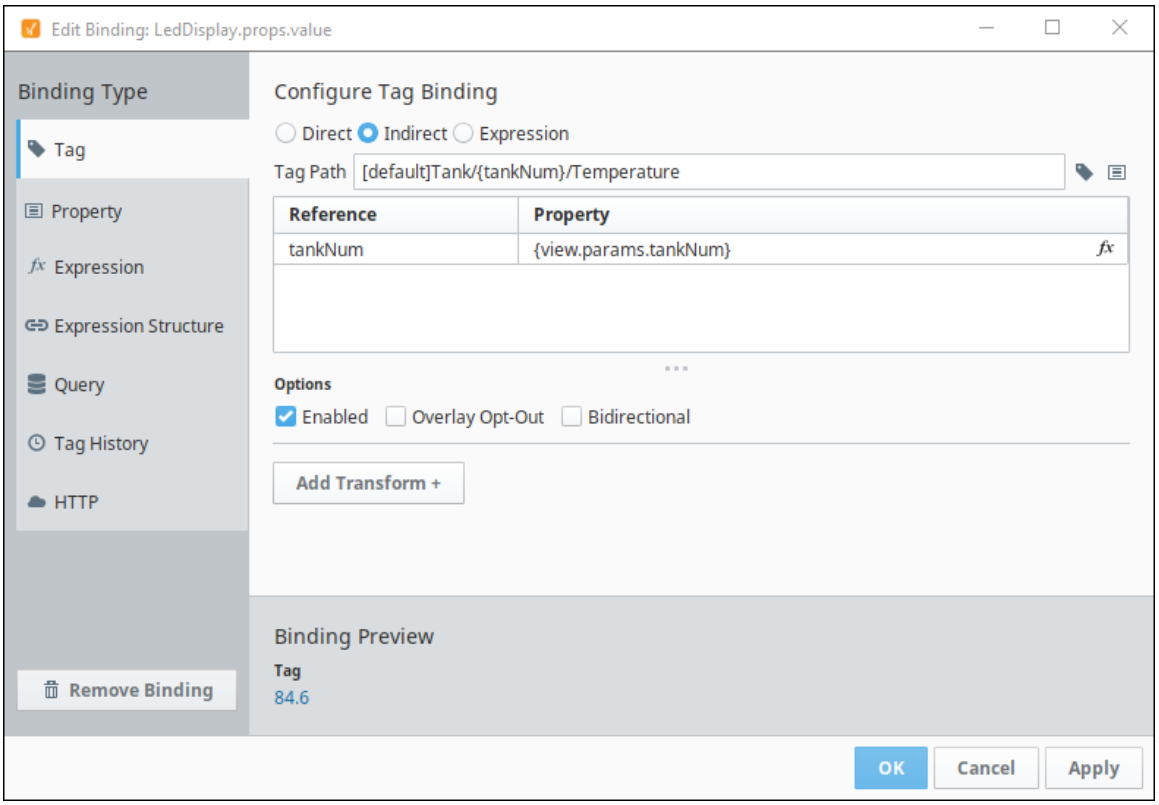
Tag Binding

Watch the Video

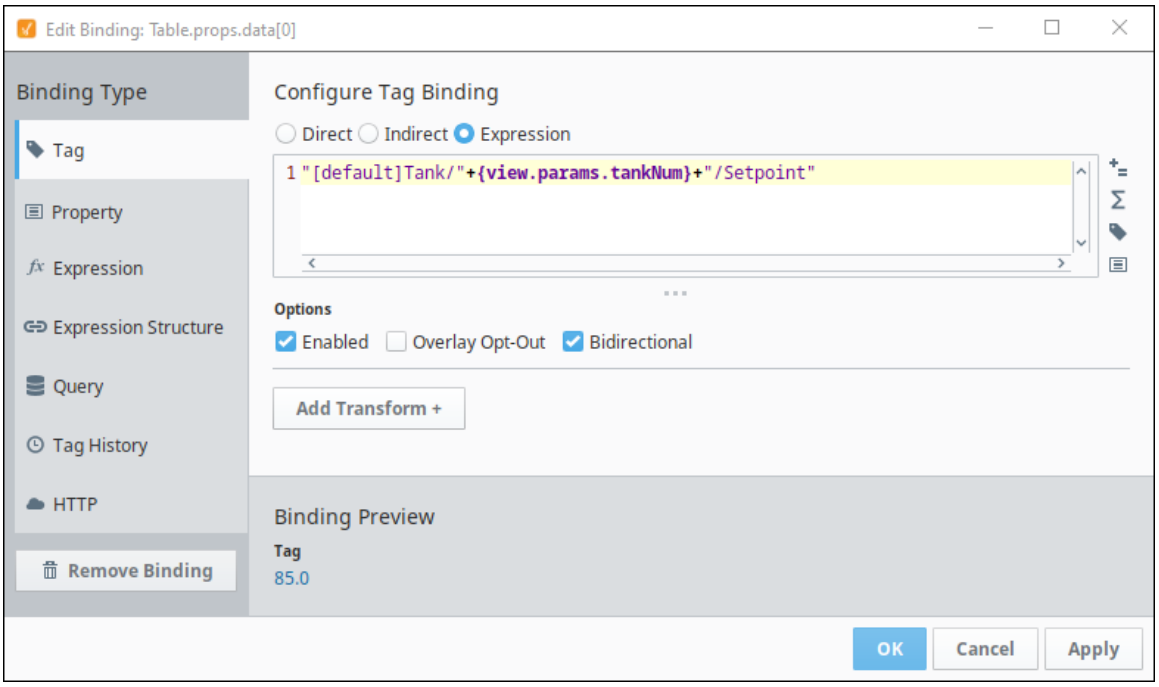
Direct Example



Indirect Example



Expression Example



Property Binding

A Property Binding binds the value of one property, to another. This binding is initially unidirectional.

These options are available:

- **Enabled:** Allows the component to be active/in use /interactive on the screen.

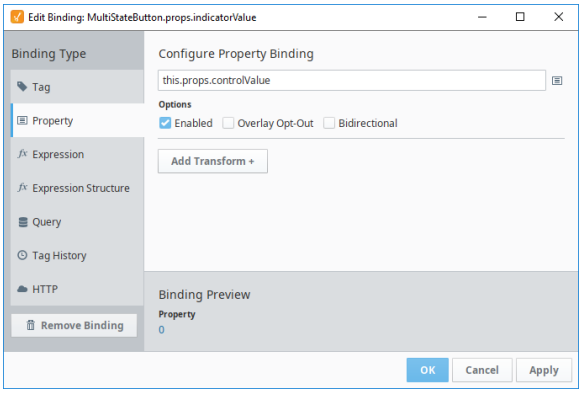
- **Overlay Opt-Out:** Choosing the Overlay Opt-out option will ignore the quality of the chosen Tag (or expression), making it have no effect on the component's quality overlay.
- **Bidirectional:** Allows user input or parameter changes on the component to be passed back to the Tag or property that the binding is reading.

Transforms can be added:

Transforms offer a chance to alter the value returned from a binding. For example, you can bind a p roperty to an integer value and use a transform to map the numerical value to a particular color, all from the same interface. For more information, see Transforms in Perspective.

Property Binding

Watch the Video



Expression Binding

An Expression binding uses the expression language to generate a value. This value is passed onto the Property that is linked to the binding. The Expression Binding is unidirectional only.

These options are available:

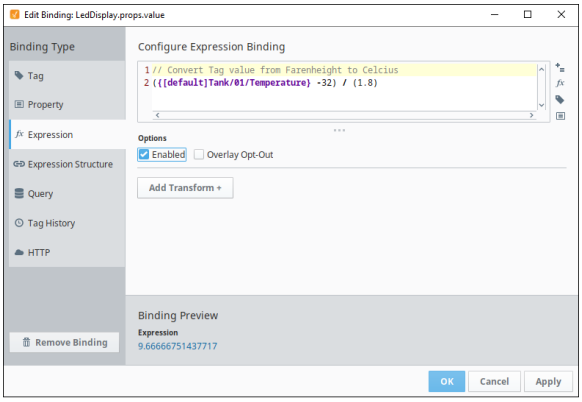
- **Enabled:** Allows the component to be active/in use /interactive on the screen.
- **OverLay Opt-Out:** Choosing the Overlay Opt-out option will ignore the quality of the chosen Tag (or expression), making it have no effect on the component's quality overlay.

Transforms can be added:

Transforms offer a chance to alter the value returned from a binding. For example, you can bind a p roperty to an integer value and use a transform to map the numerical value to a particular color, all from the same interface. For more information, see Transforms in Perspective.

Expression Binding

Watch the Video



Expression Binding Examples

Expression on a button that references its custom property
`if({this.custom.sourceString}!="abc","Return String 1","Return String 2")`

Expression on a label that references a button's custom property

```
if({../Button.custom.selected},"Return String 1","Return String 2")
```

Expression on a container that references a button inside it.

```
if({./Button.custom.intValue}>15,1,0)
```

Expression Structure

An Expression Structure binding allows you to build a json document and bind it to a property. Each value within the Expression Structure can be bound to an Expression builder. This allows for a dynamically changing values based on bindings.

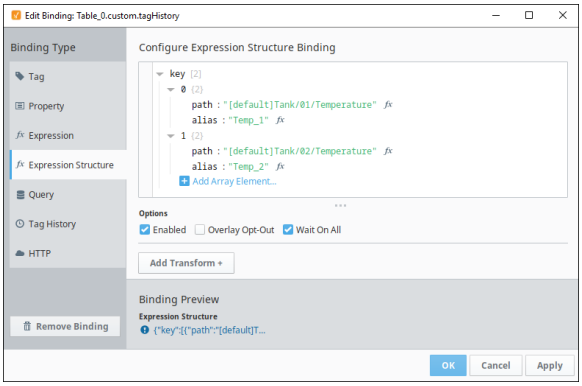
These options are available:

- **Enabled:** Allows the component to be active on the screen.
- **Overlay Opt-Out:** Choosing the Overlay Opt-out option will ignore the quality of the chosen Tag (or expression), making it have no effect on the component's quality overlay.
- **Wait On All:** Waits to evaluate the Expression Structure binding until all Bindings within the Expression Structure have been evaluated.

For this example we also linked a Tag History Expression binding to this expression structure. By using Binding Expressions to modify the "path" field This method can be used to create indirect Tag history bindings. (See Tag History (Expressions) for the other part of this example).

Transforms can be added:

Transforms offer a chance to alter the value returned from a binding. For example, you can bind a p roperty to an integer value and use a transform to map the numerical value to a particular color, all from the same interface. For more information, see Transforms in Perspective.



Query Binding

A Query Binding is a polling binding type that runs a structured Query against any of the database connections configured in the Gateway.

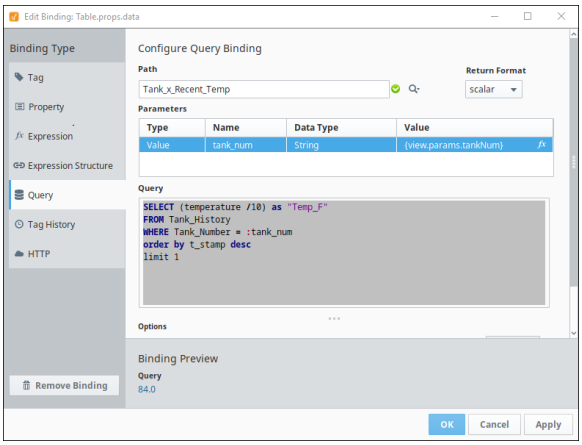
Return Format: the Return format specifies how the query results are returned.

- **auto**: Query results are returned in the format native to the database (typically dataset).
- **json**: Query results are returned in json format. This format is recommended for XY Charts.
- **dataset**: Query results are returned in dataset format. This format is recommended for tables.
- **scalar**: Returns the first element from the query result. This format is best when a single value is expected.

Parameters: If the structured query you select requires parameters you can add them here. The value you enter for the parameters can be modified using an expression builder (*fx*).

Options:

- **Enabled:** Allows the component to be active/in use /interactive on the screen.
- **Overlay Opt-Out:** Choosing the Overlay Opt-out option will ignore the quality of the chosen Tag (or expression), making it have no effect on the component's quality overlay.
- **Bypass Cache:** This will cause the query to bypass/Ignore any cached values from the Named Query and run every time it is called.
- **Designer Limit:** This setting will force the results of the query to be limited to a few rows when run in the Designer.
- **Polling:** This setting will cause the query to run/poll the database using based on a poll time (set) in the Designer.



Tag History

A Tag History binding runs a query against the Tag Historian. There are several ways data can be polled and returned.

Return Format:

- **Wide**: Returns data value and time stamp value from the Tag Historian.
- **Tall**: Returns additional data such as quality code of data from the Tag Historian.
- **Calculations**: Performs calculations on the data (average, sum, count, etc).

Query Mode:

- **PointCount**: Data returned will be the amount of datapoints specified and spread evenly across the date range specified.
- **Periodic**: Data returned will be sampled from the Tag Historian at regular intervals. The regular interval adjustable in the Period filed of the binding configuration.
- **AsStored**: Data returned is exactly as is from the Tag Historian. No data interpolation of the time or points are done.

Time Range:

- **Realtime**: The data is sampled from the most recent time and back from a specified time range. The time range can be adjusted using an Expression builder (*fx*).
- **Historical**: The Start date and End Date can be defined using an expression builder (*fx*).

Select Tags:

- **Direct**: Tags are referenced using a builder that looks directly at the Tag Browser. You can rename the display name of the tag by entering a name in the Alias column.
    - **Alias:** A different name to display for the tag being shown.
    - **Aggregate:** Determines how the tag value will be interpreted. Several functions are available: (Average, MinMax, LastValue, SimpleAverage, Sum, Minimum, Maximum, DurationOn, DurationOff, CountOn, CountOff, Count, Range, Varience, StdDev, PctGood, PctBad).
- **Expression** : The Tag History binding expects a JSON document of objects to read in the desired Tag paths. The best way to do this is to create a custom property and make an array of objects. Each object should contain a value called "path". Optional values to include are "alias" and "aggregate". You can create a custom property structured as mentioned or you can create an "Expression Structure" (refer to Expression Structure Bindings in Perspective for further details). This is the preferred way to make a dynamic Tag path for fetching Tag History values.

Default Aggregation Mode: Any tags left with the Aggregation Mode set to "(default)" will use this setting.

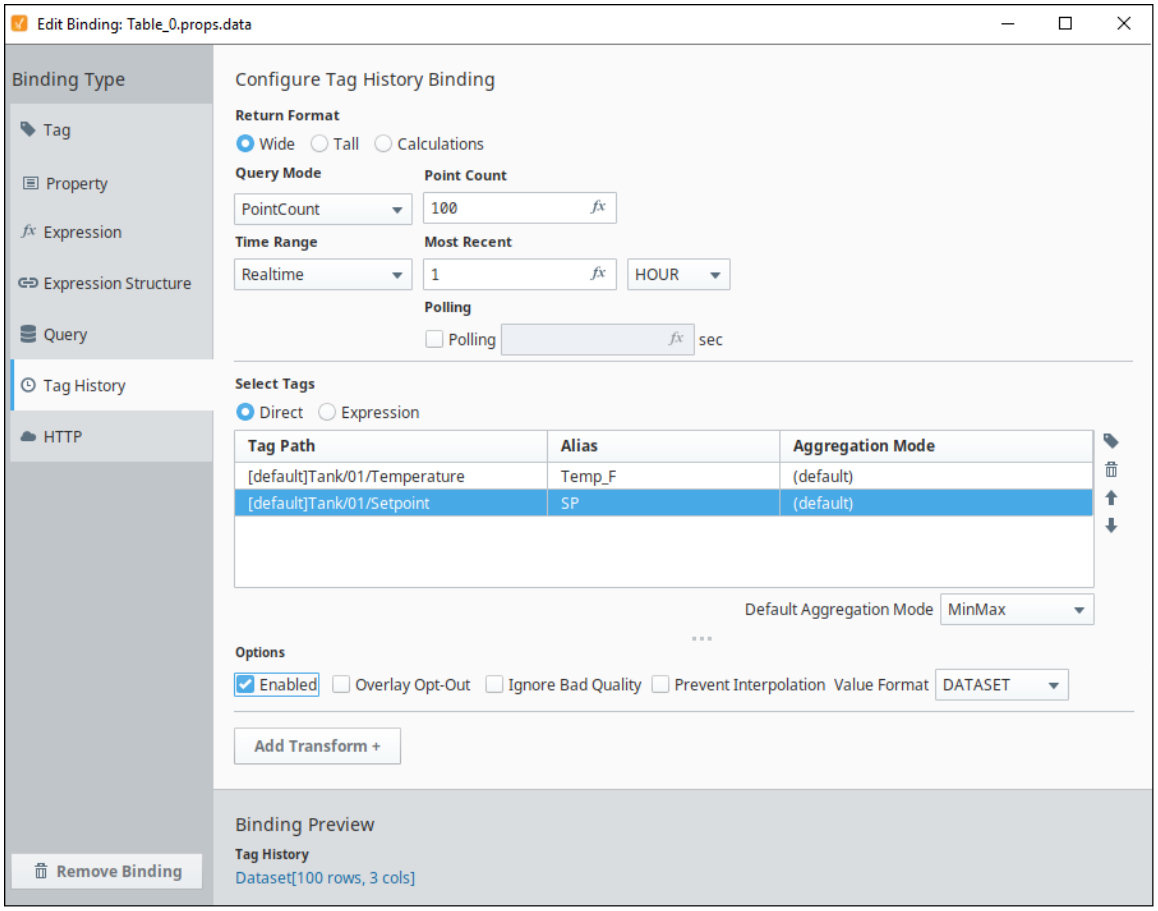
Options:

- **Enabled**: Allows the component to be active/in use /interactive on the screen.
- **Overlay Opt-Out**: Choosing the Overlay Opt-out option will ignore the quality of the chosen Tag (or expression), making it have no effect on the component's quality overlay.
- **Ignore Bad Quality**: This will force the system to ignore any data results that have a bad quality associated with it.
- **Prevent Interpolation**: This forces the data displayed to not use any interpolation. The data will not be truncated or averaged to to fit the display in a nice manor.

Value Format:

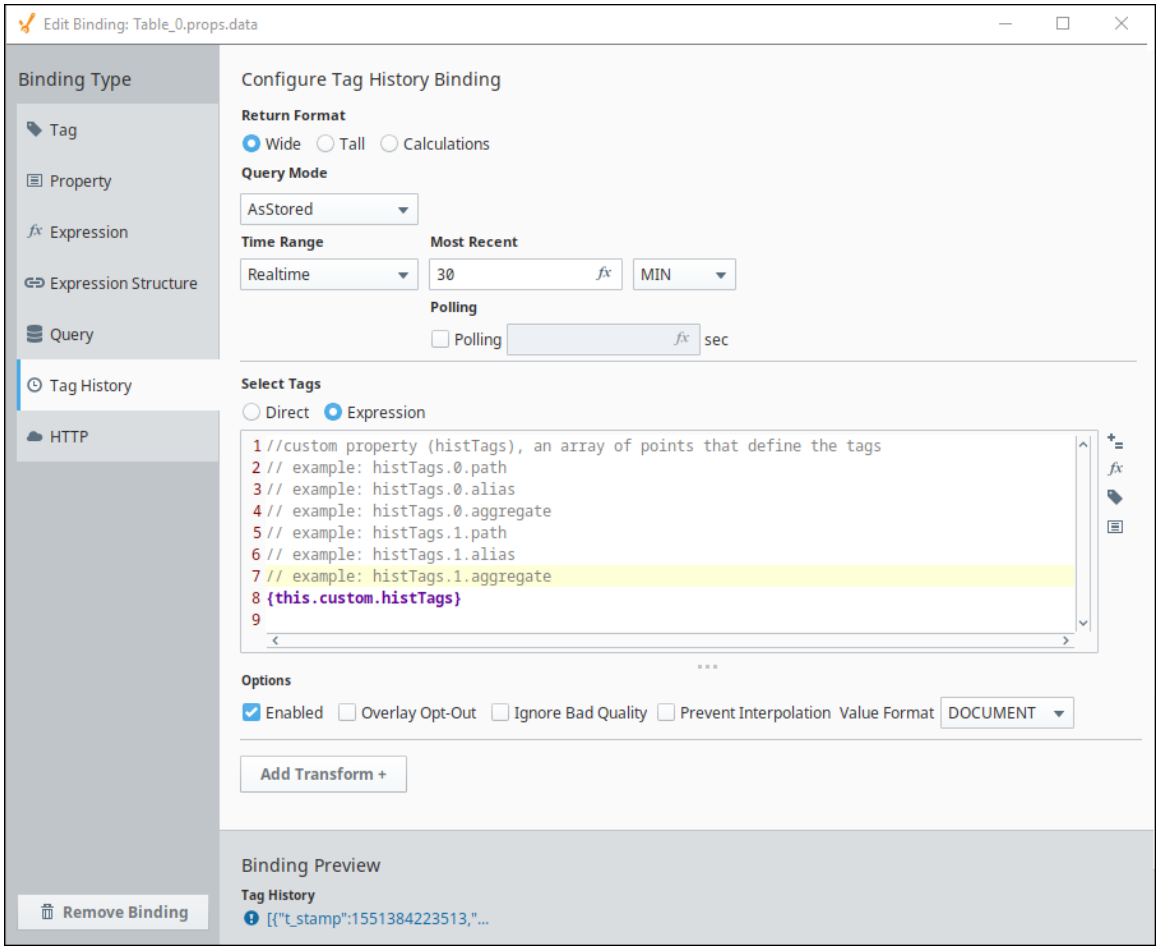
- **Dataset**: Returns the data in a dataset. This format is best suited for use in a table component.
- **Document**: Returns the data in a JSON document. This format is best suited for use in an XY Chart.

Direct Tag Example



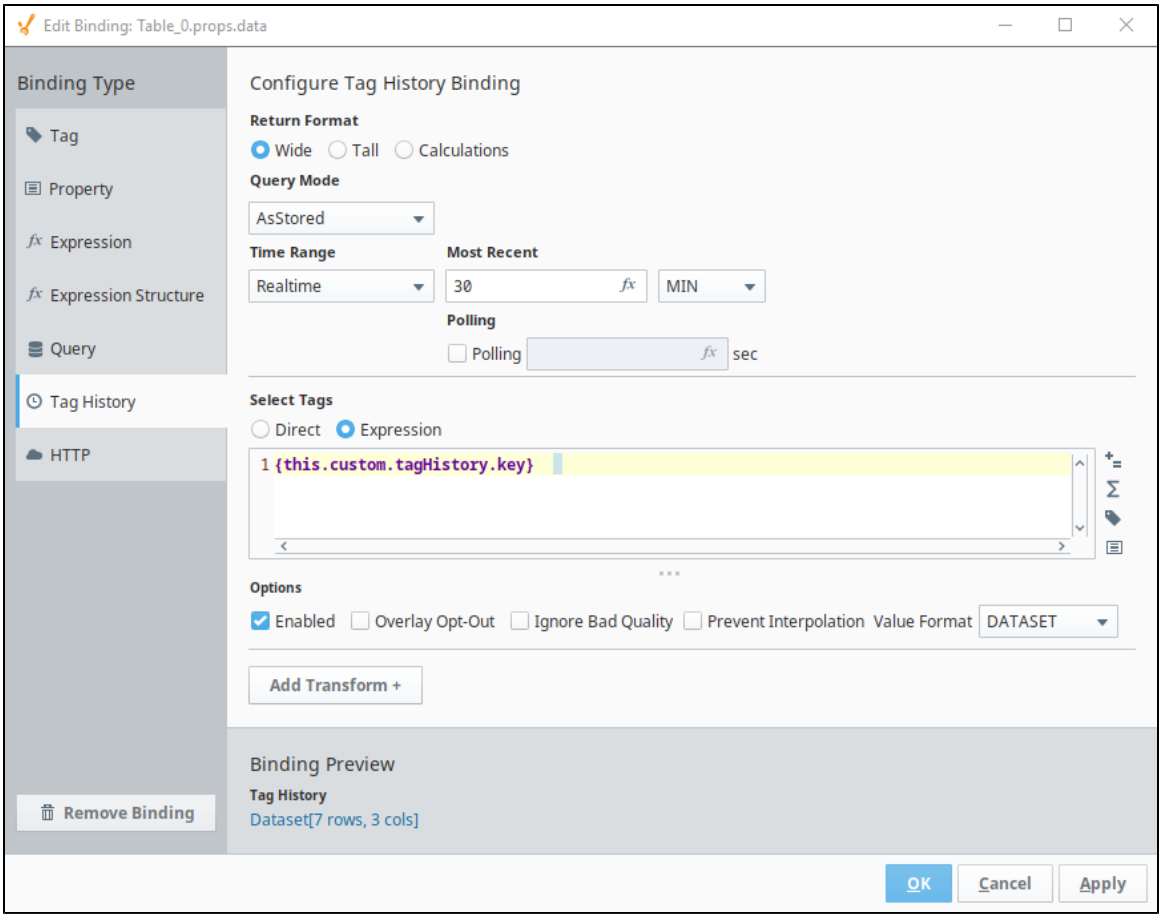
Expression Tag Example 1

For this example, we created a static custom property. The comments in the example show the structure used.



Expression Tag Example 2

For this example we created an Expression Tag structure.



HTTP Binding

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

URL: The web address you are communicating with.

> ⚠ The URL should be inside quotations as it needs to be a string for this binding to work.

Method: The method to communicate with the URL. Available methods are listed here [GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH].

Headers: If any headers are needed they are filled in here:

- **Key**
- **Column**

Body: The body of the HTTP message. This is most commonly used in POST method where it represents the information that is being posted.

Authentication Type: Authentication methods required by the URL. Options are:

- **None**
- **Basic**
- **Bearer**
- **Digest**

Authentication Value: Authentication credentials can be edited via an Expression binding.

Connect Timeout: Amount of time to wait before failing a connection attempt.

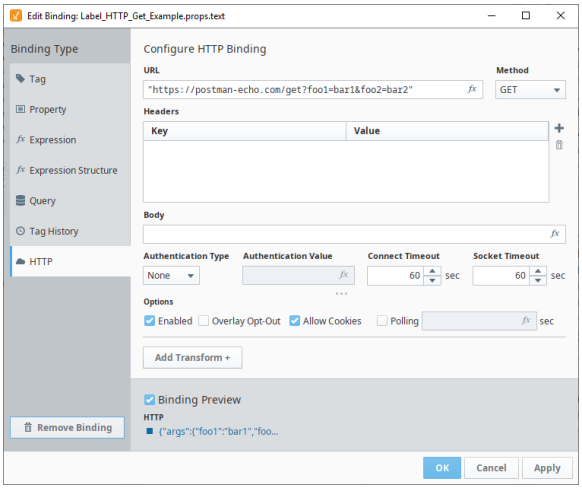
Socket Timeout: Amount of time to wait before failing a connection attempt.

Options:

- **Enabled**: Allows the component to be active/in use /interactive on the screen.
- **Overlay Opt-Out**: Choosing the Overlay Opt-out option will ignore the quality of the chosen Tag (or expression), making it have no effect on the component's quality overlay.
- **Allow Cookies**: Allow cookies from the URL.
- **Polling**: How often to poll the URL (an Expression binding is available to define the poll time (in seconds)).

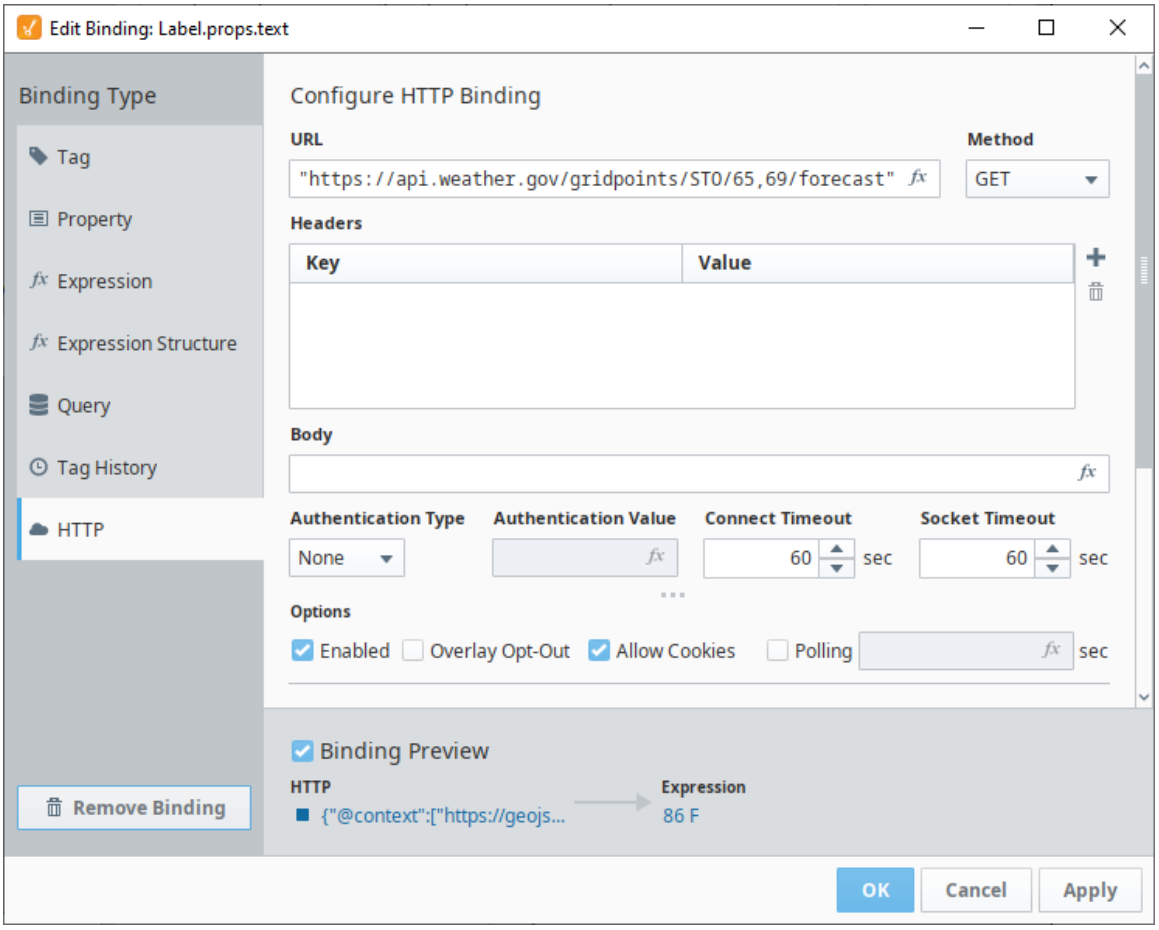
> The following feature is new in Ignition version **8.0.15**
> Click here to check out the other new features

Binding Preview: The Binding Preview checkbox can cancels/stops the binding preview display. Uncheck the Binding Preview option to disable the binding preview. Disabling the binding preview is ideal in cases where you don't want the binding to trigger frequently while configuring it, such as when using a HTTP binding.

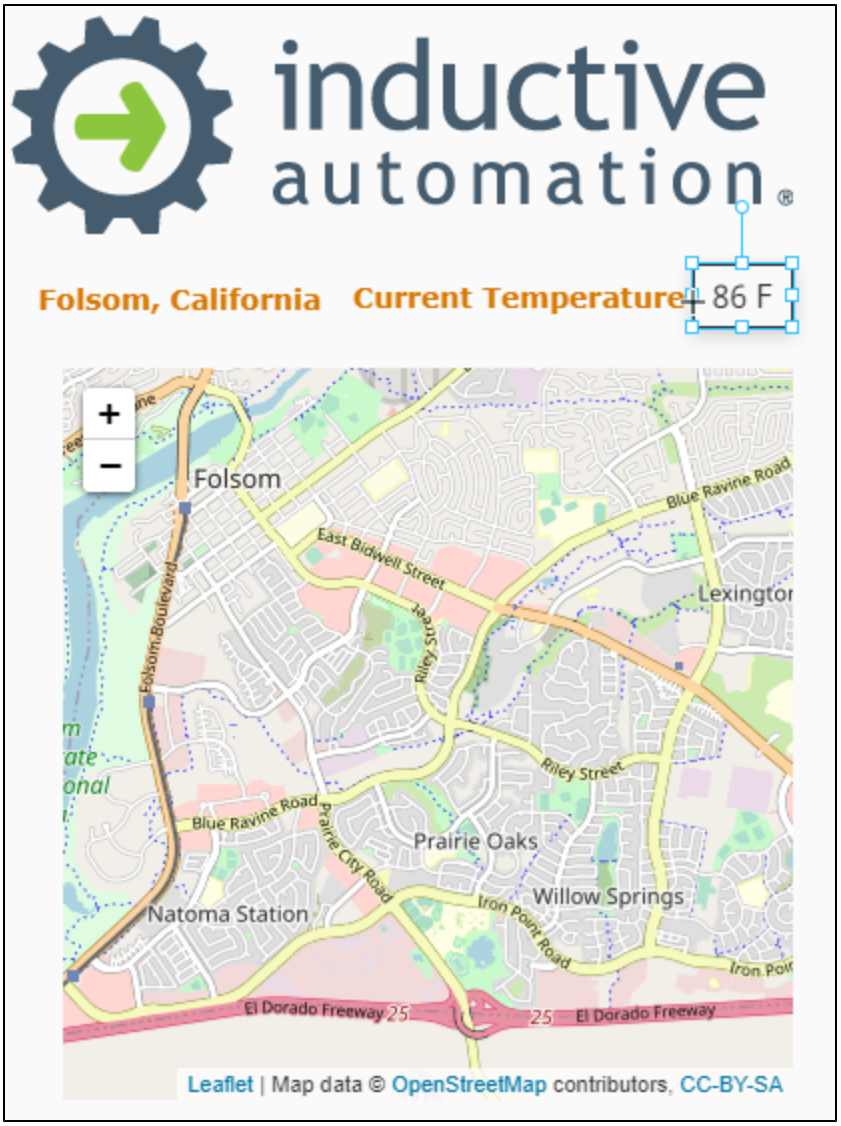


Weather Data Example

The following is an example showing weather data for Inductive Automation headquarters. The API contains detailed forecast information. In this example we use an Expression transform to capture just the current temperature.



The value is bound to a Label component on a view with some other information about Inductive Automation.



In This Section ...

Tag Bindings in Perspective

A Tag binding is a straight-forward binding type. It simply binds a property directly to a Tag Property (typically the value). This sets up a subscription for that Tag. Every time the chosen Tag changes, the binding is evaluated and pushes the new value into the bound property. If you choose a Tag in the tree, and not a specific property of that Tag, the Value property is assumed.

On this page

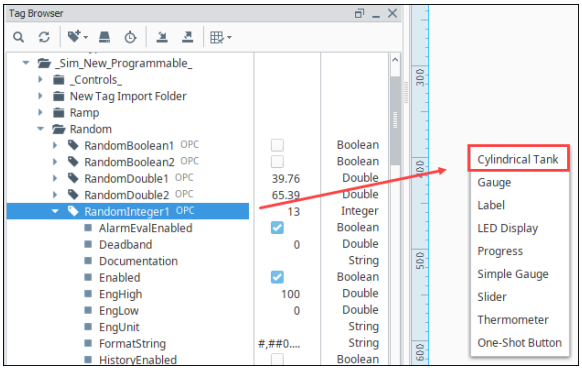
...

- Drag and Drop
- Direct Tag Binding
- Indirect Tag Binding
  - Indirect Tag Binding - Bidirectional
  - Indirect Tag Binding Example
- Tag Expression Binding
  - Tag Expression Binding - Example

Drag and Drop

Ignition automatically creates the Tag bindings to several of the component properties when you choose to bind a Tag to a component by dragging and dropping. This is true for both creating a component by dragging and dropping a Tag onto empty space on a window at the same time Igniti on prompts you for what type of component you want to create, and by dragging and dropping a dr opping a Tag directly onto a component that already exists on a window. In both cases, Ignition aut omatically creates the Tag bindings on the component.

You can drag a Tag onto a container or view. 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.

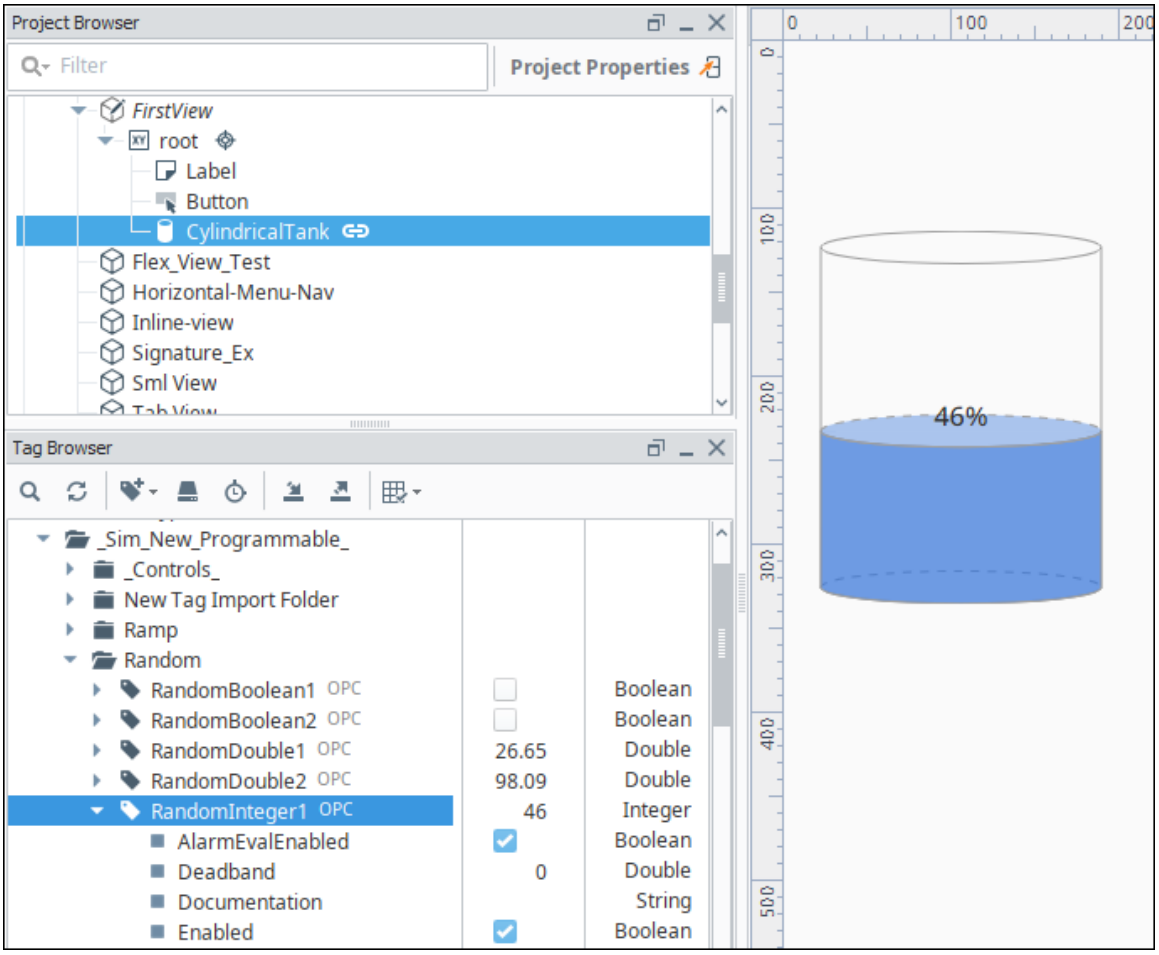


A Cylindrical Tank component is placed on the view. Notice that the value for the component and the value for the Tag match. The component's displayed value will update as the Tag updates.

INDUCTIVE UNIVERSITY

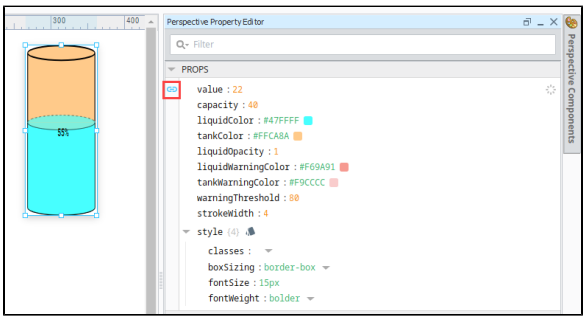
Tag Drop

Watch the Video



Direct Tag Binding

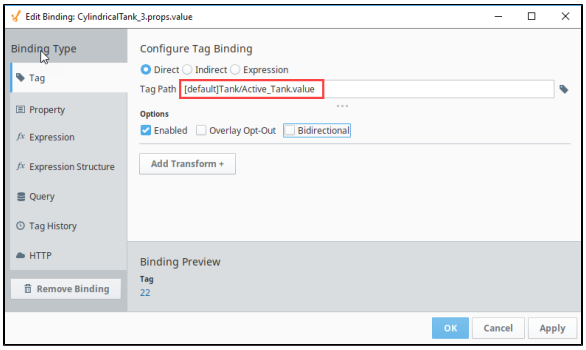
A direct binding binds a component property to a Tag path. Every time the Tag's value changes, the binding is evaluated and the new value is sent to the bound property. In the example below, the value of the Active_Tank Tag is displayed as a percentage of the capacity of the Cylindrical Tank component.



IU INDUCTIVE UNIVERSITY

Tag Binding

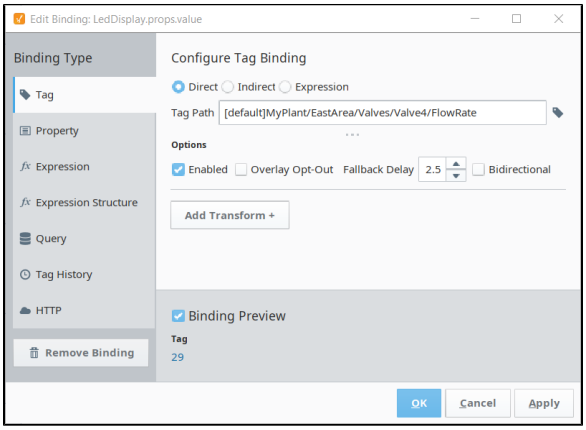
Watch the Video



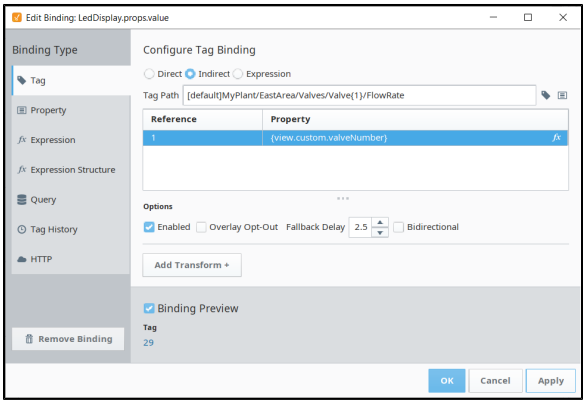
Indirect Tag Binding

An Indirect Tag binding is very much like a standard Tag binding, except that you may introduce any number of indirection parameters to build a Tag path dynamically in the session. These parameters are numbered starting at one, and denoted by braces, for example, {1}. The binding will be linked to the Tag represented by the Tag path after the indirection parameters have been replaced by the literal values they are bound to.

For example, instead of binding directly to the FlowRate tag inside the Valve4 folder, as show in the following example:



You can build an indirect tagpath that will point to the FlowRate tag for all valves between Valve0 to ValveN where N is any valve number. Below, the valve number is replaced by {1} where {1} 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:

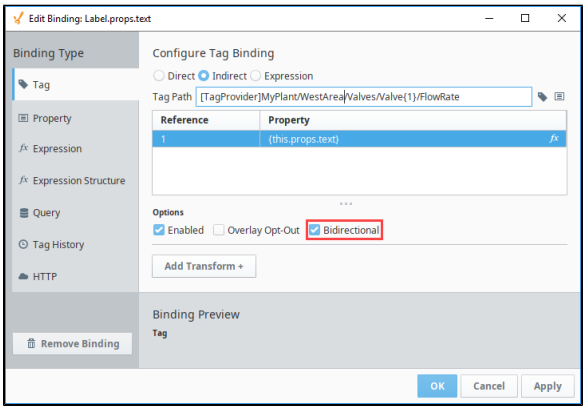


The {1} dynamic **Reference** must be pointed to a **Property**. Your property can pointed to any component property value or tag value to build an Indirect Tag Binding. In the capture above,

the {1} dynamic reference is pointing to a view custom property named valveNumber which 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.



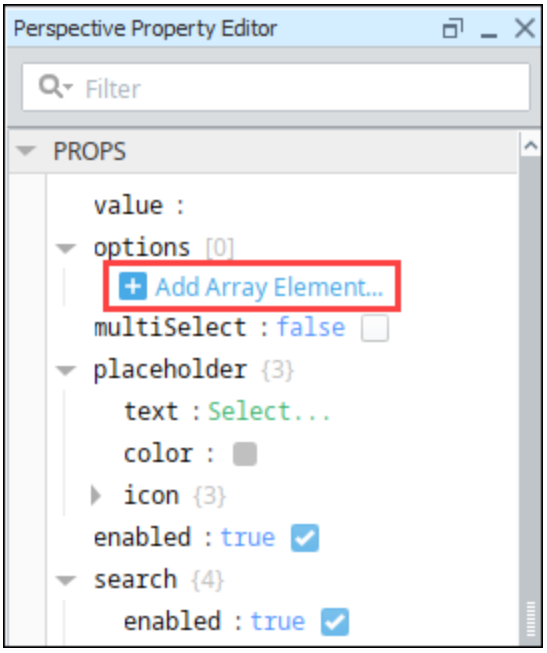
Indirect Tag Binding Example

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

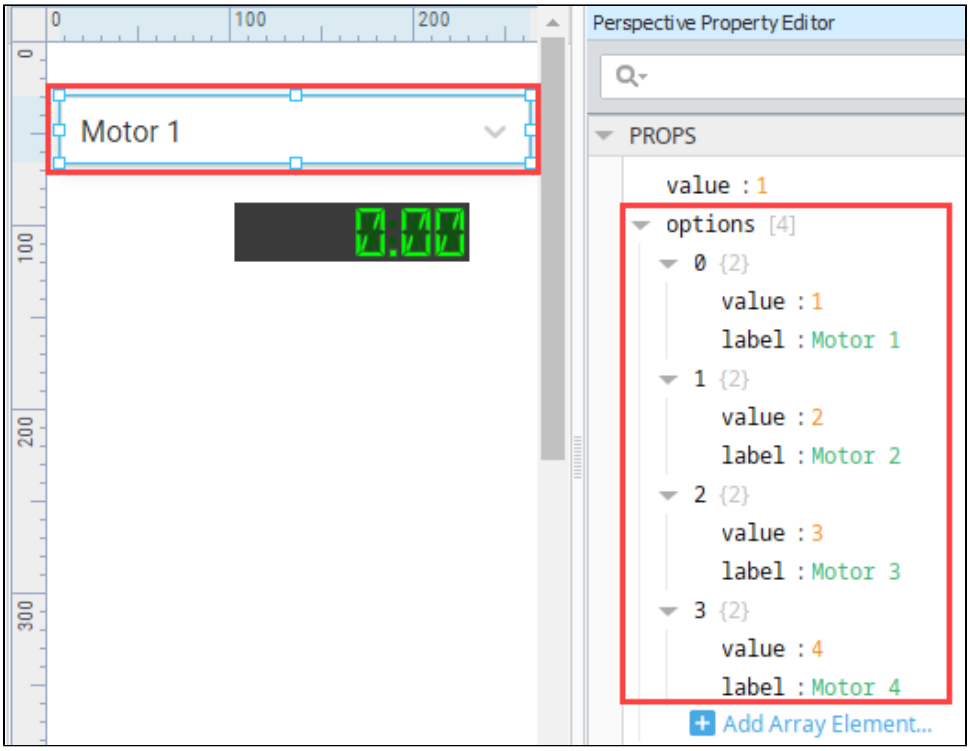
```
Motors/Motor 1/amps
Motors/Motor 2/amps
Motors/Motor 3/amps
Motors/Motor 4/amps
```

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

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

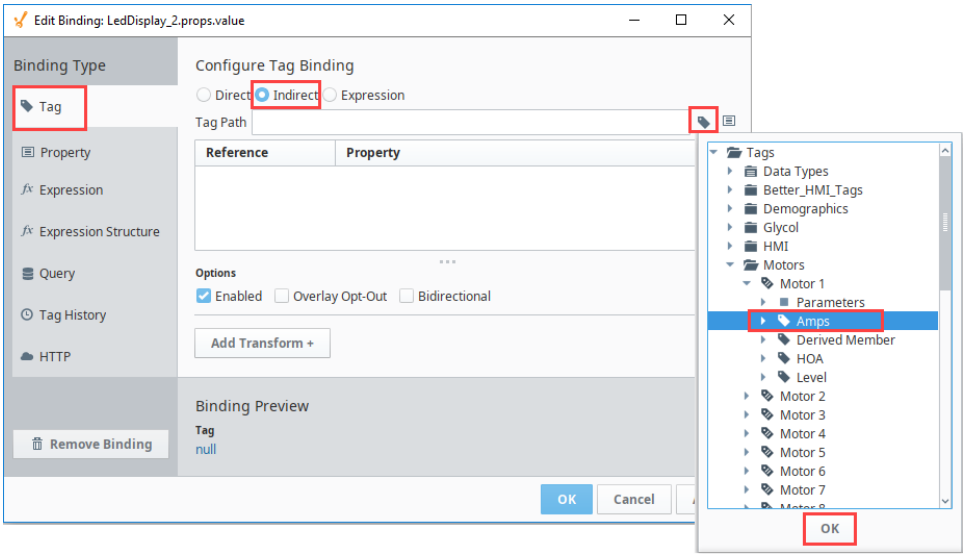


4. Set the value to 1 and the label to Motor 1.
5. Repeat steps 3 and 4 to add array elements for Motors 2, 3, and 4.

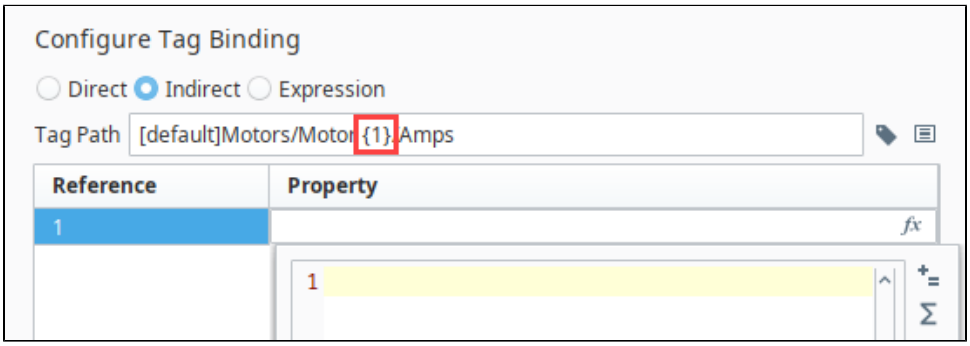


6. Click on the LED Display component. Select the **Binding** icon in the value property.
7. Click on the Tag binding type then click the **Indirect** button.

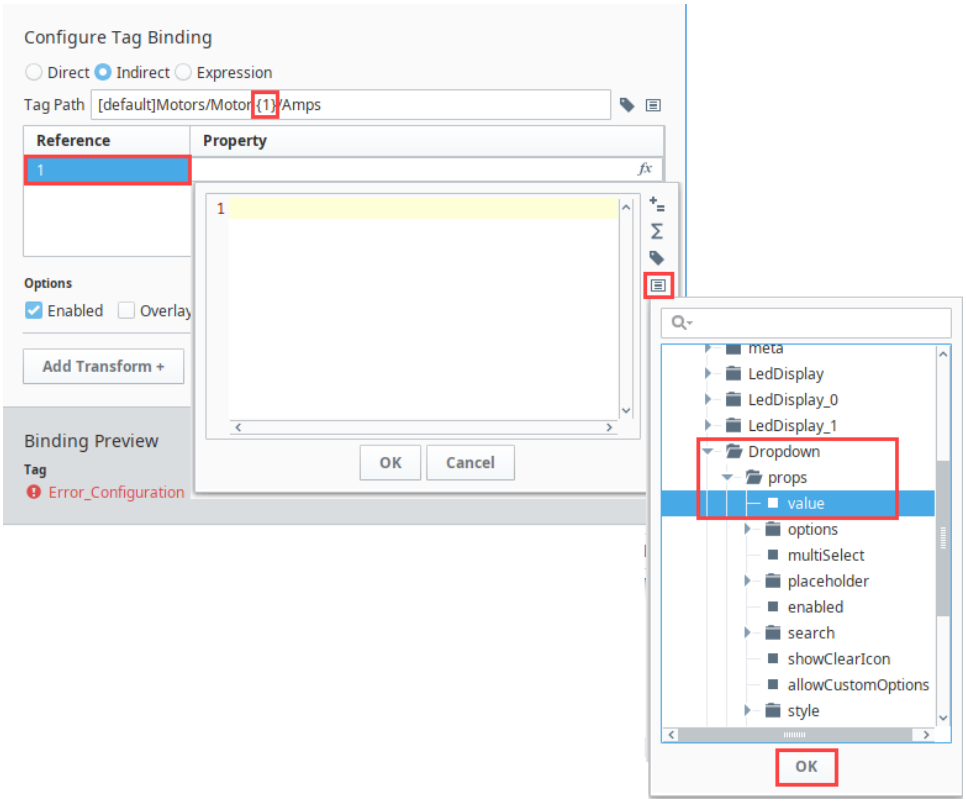
8. Click the **Tag** icon and scroll down to the Motors/Motor 1/Amps Tag. Click **OK**.



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" in "Motor" and "{1}".

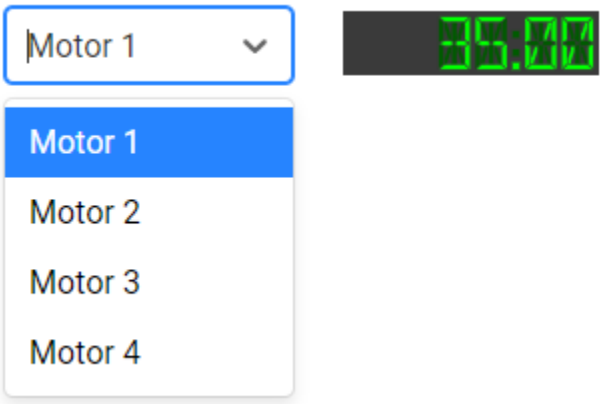


10. In the References list, select the row. Click the **Properties** icon.
11. Scroll to the Dropdown component and select the **value** property of the Dropdown. Click **OK**.



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

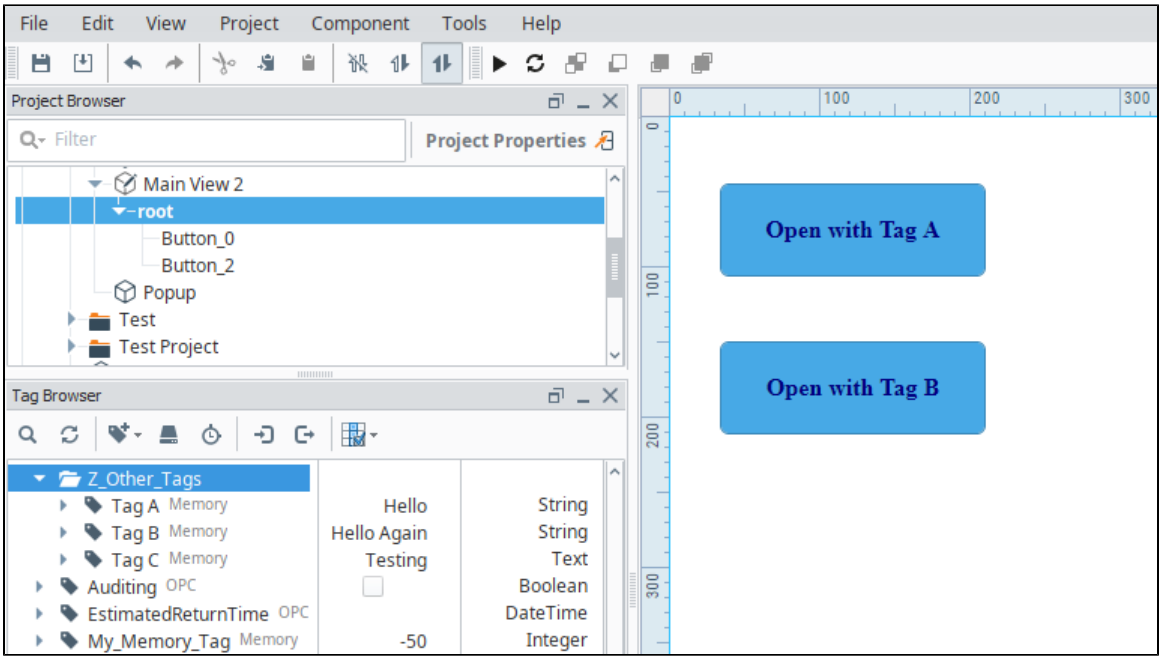
The Tag Expression binding uses the Expression language to specify an entire Tag path. This mode allows the bound property (Tag) to be bidirectional. The Tag path in the Expression is expected to be a string. Note that is different and not to be confused with an Expression Binding.

Tag Binding - Expression

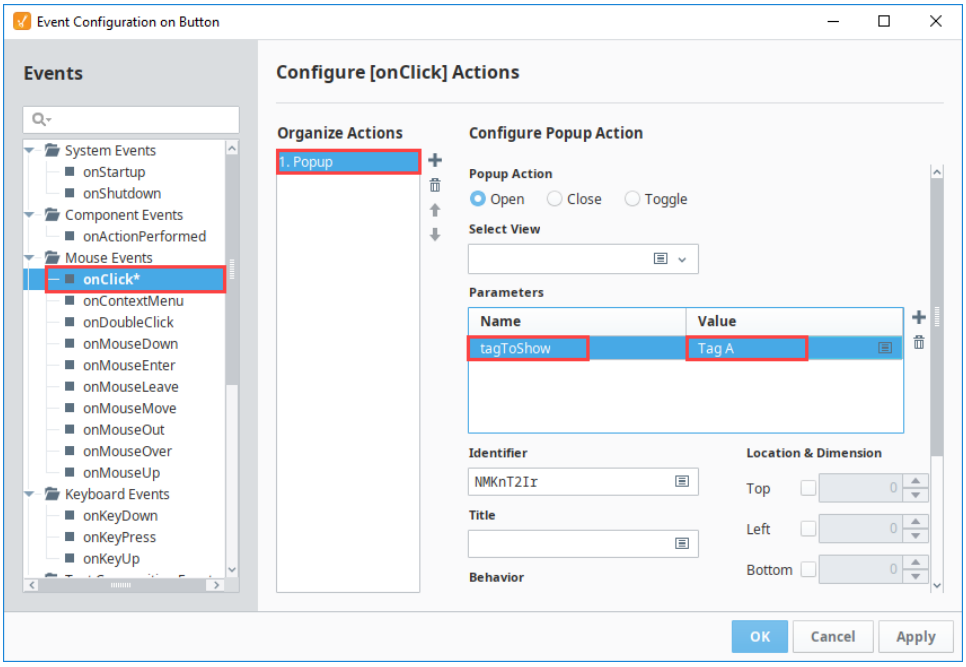
Watch the Video

Tag Expression Binding - Example

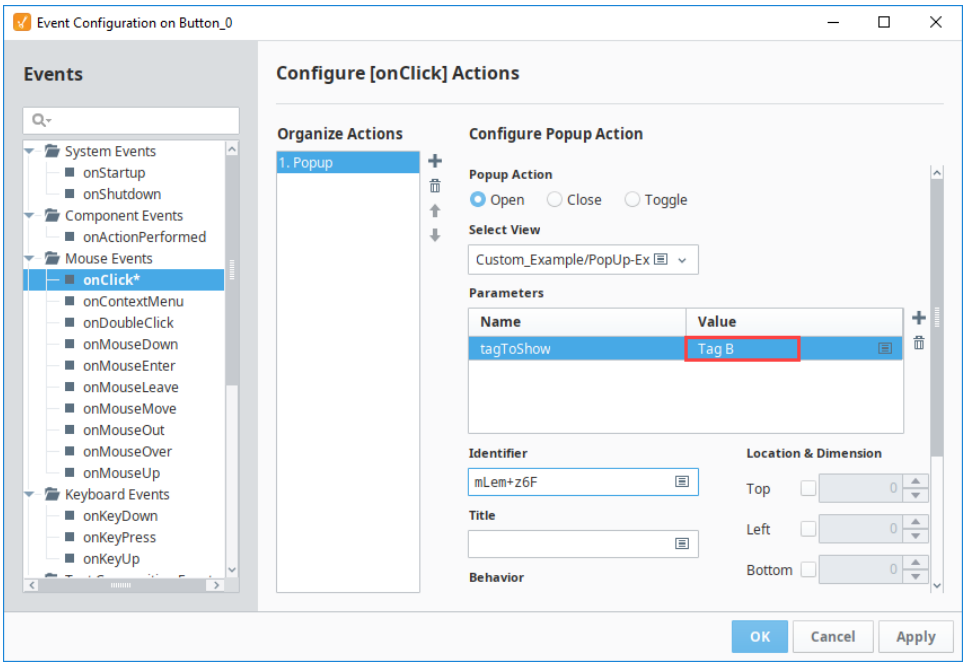
For this example, we start with two Tags: Tag A and Tag B and two buttons on a view.



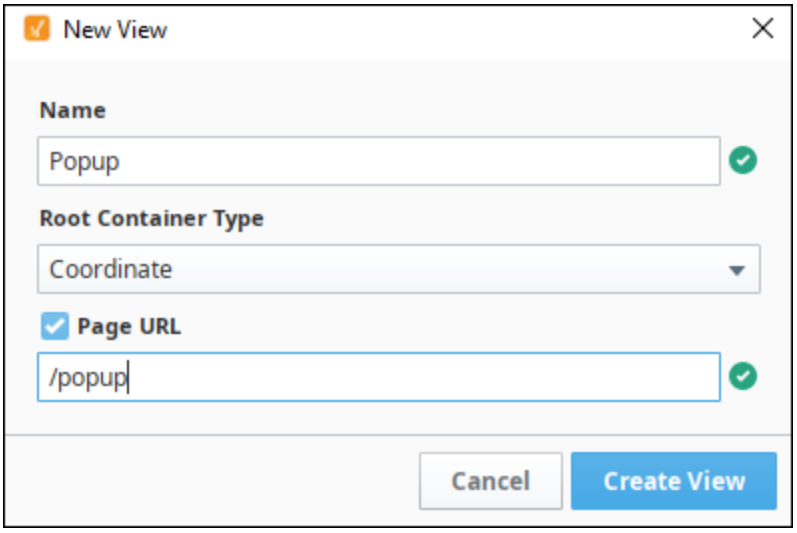
1. Right click on the first button and select **Configure Events**.
2. Select **onClick**.
3. Under Organize Actions, click the Add ✚ icon and select **Popup**.
4. Under Parameters, click the **Add** ✚ icon to add a new parameter.
5. Enter "tagToShow" in the Name field and "Tag A" in the Value field.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



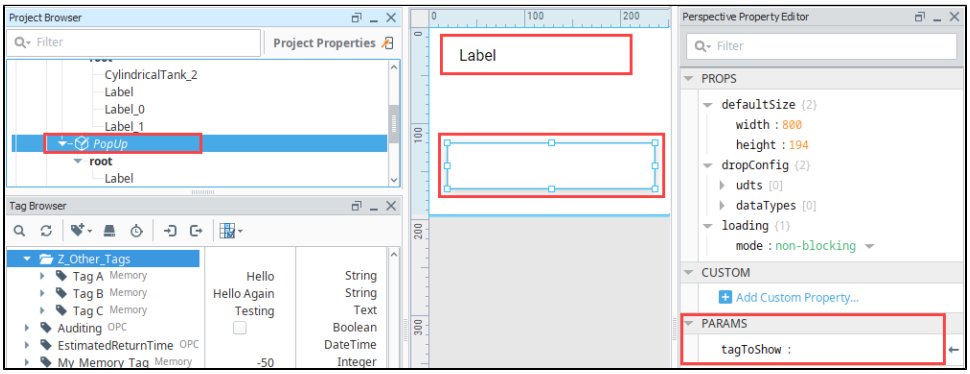
6. Repeat steps 1 through 4 for the second button, but in the Value field, enter **Tag B**.



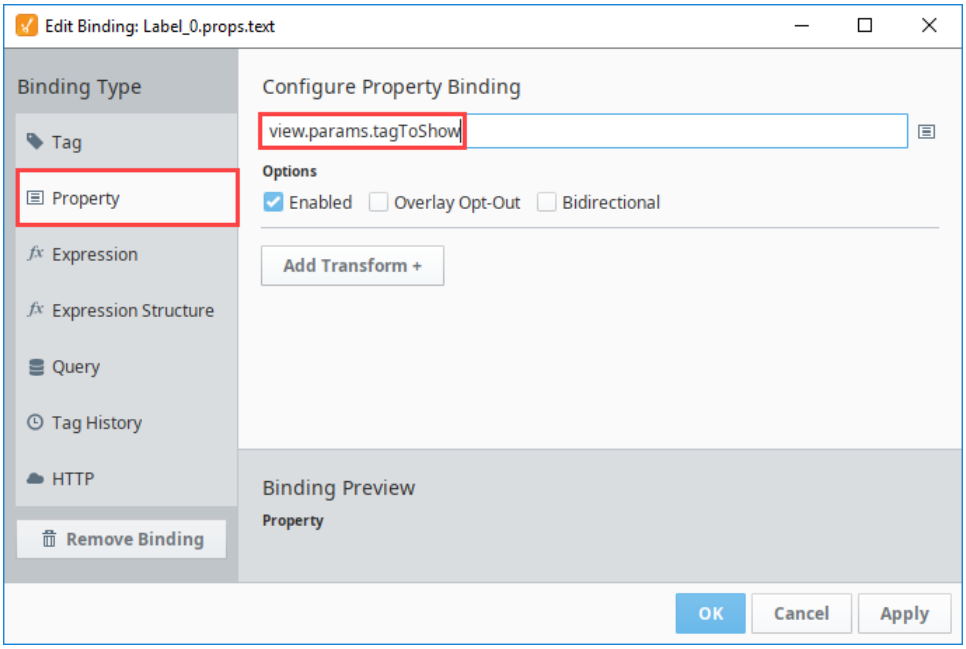
7. Create a new view named **Popup**.



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

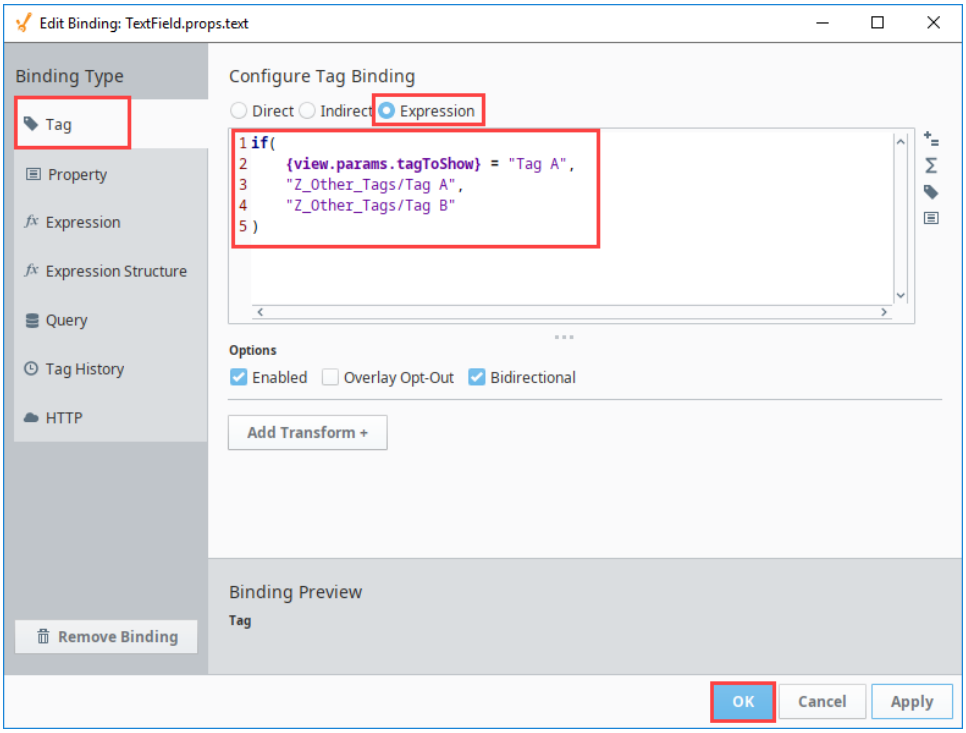


11. Select the Label component. Click on the **Binding** icon next to the text property.
12. On the Edit Binding screen, select the Property binding type. Enter **view.params.tagToShow** in the Configure Property Binding field.
13. Click **OK**.

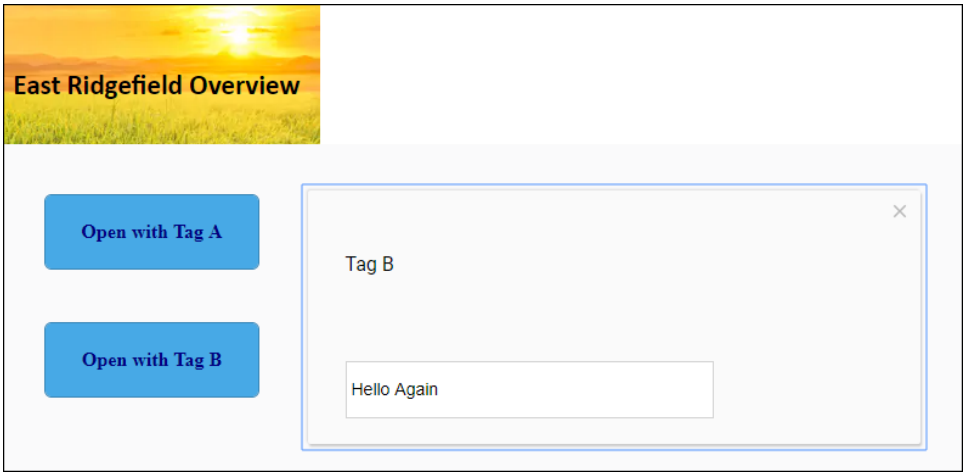


14. Next set a binding on the Text Field component. Select the component then click on the **Binding** icon next to the text property.
15. On the Edit Binding screen, select the Tag binding type and select the **Expression** button.
16. Enter the following expression. This expression in the example tells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



Property Bindings in Perspective

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

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

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

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

On this page

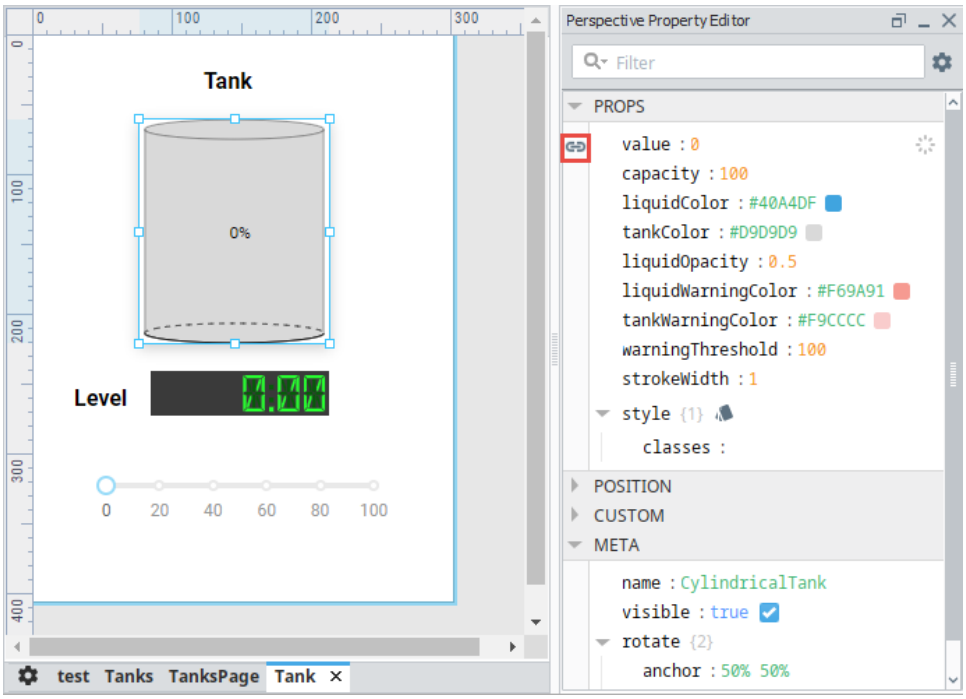
...

- Property to Property Binding
- Pass a Property into an Embedded View Using a View Parameter
  - Use a Property Binding
  - Using a Tag Binding
- Tag dropConfig

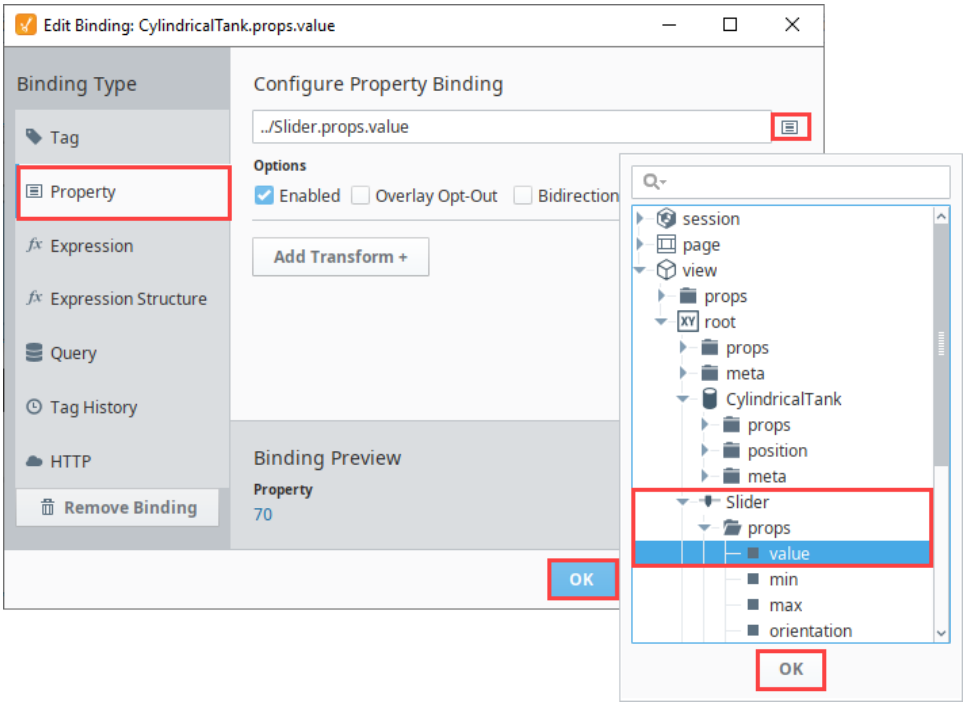
Property to Property Binding

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 setup 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.

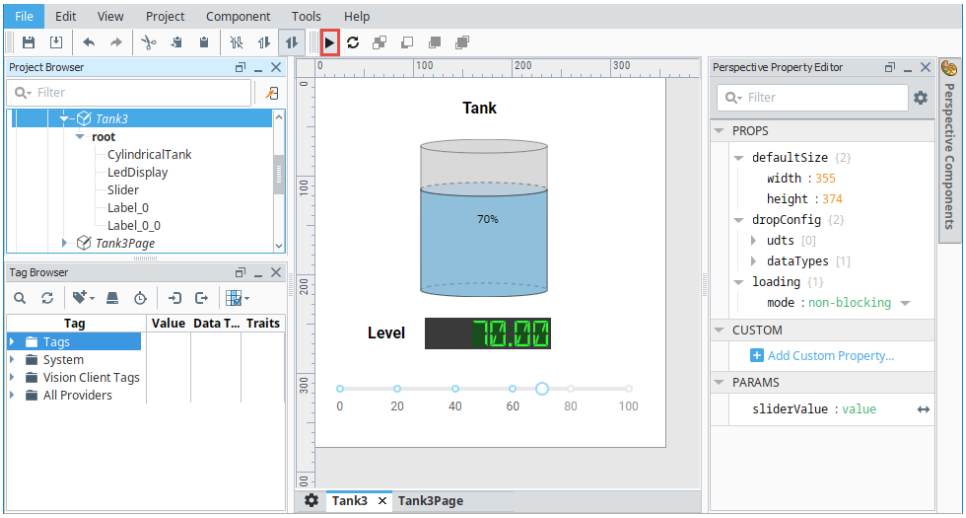


4. This opens the **Edit Binding** window. Configure the following settings:
   a. Select the **Property** binding type.
   b. Under **Configure Property Binding**, click the **Insert Property** icon to open the **Property** popup box. Expand the folders and select the Slider **'value'** property. Click **OK**.
   c. Click **OK** to save the binding settings.



5. Select the **LED Display**. In the **Property Editor**, click on the **Binding** icon for the **'value'** property and repeat Step 4.
6. To see the labels on the Slider, select the Slider and set the **show** property to '**true**'.
7. Save your project. Put the Designer in **Preview Mode** ▶.

8. Move the slider and you'll see both the tank level and LED display change to the value of the Slider.



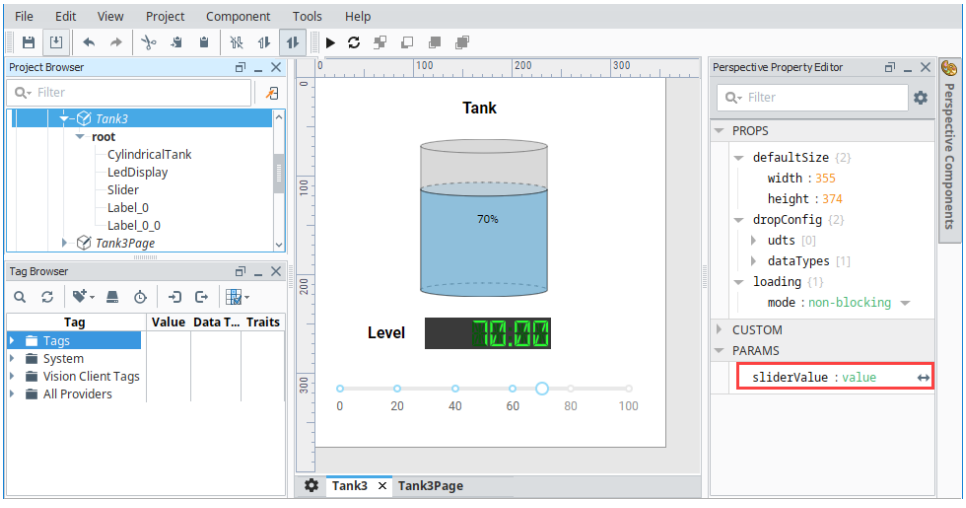
Pass a Property into an Embedded View Using a View Parameter

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

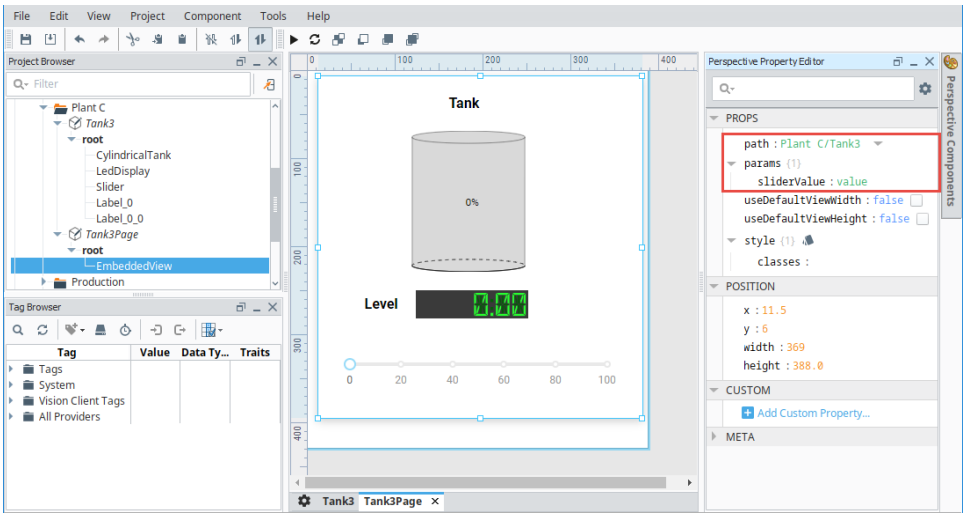
Use a Property Binding

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

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



3. In the Project Browser, create a new view (i.e., Tank3Page) that will contain the embedded view.
4. Drag a **Embedded View** from the Component Palette to the Designer workspace.
    a. With the Embedded View selected, set the 'path' property to your original view (i.e., Tank3) from the dropdown list.
    b. In the **Property Editor**, create a parameter called 'sliderValue'. (Hover under the **params** property and you'll see a **plus** + icon to add a new param).

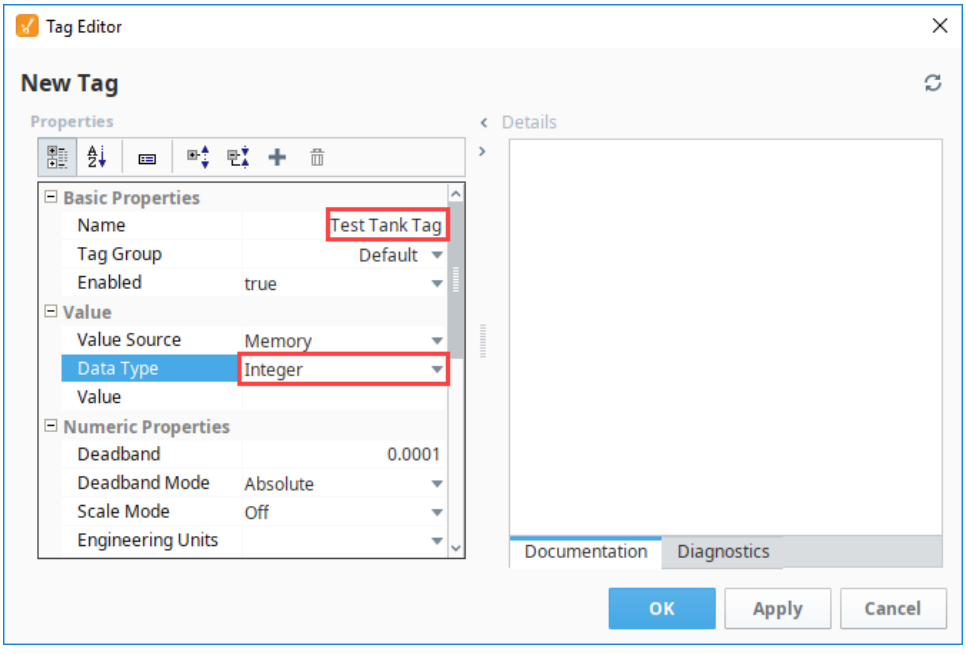


5. **Save** your project.
6. From this new view, put the Designer in **Preview Mode** ▶. Drag the slider to a value to change the value on the Tank and LED Display.
7. You can see from the Embedded View (i.e., Tank3Page), that your sliderValue reflects the same values as your slider.

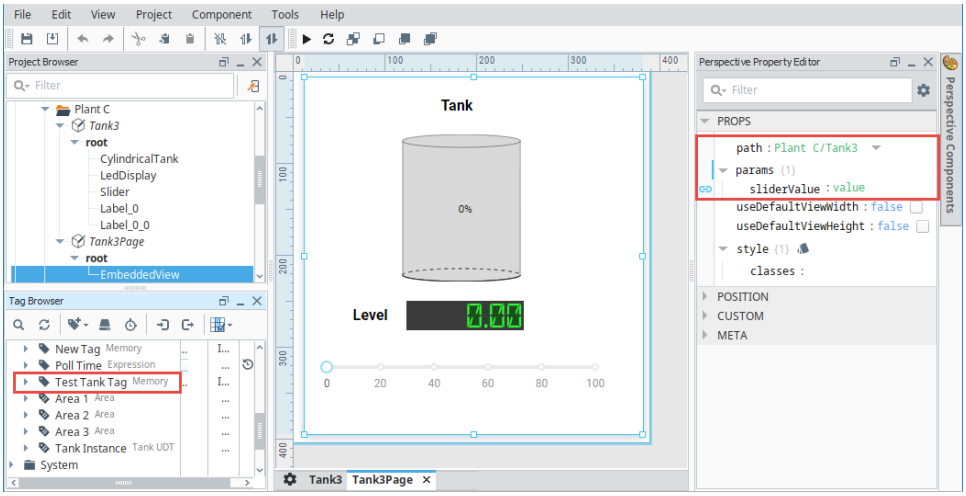
Using a Tag Binding

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 session and store values in a database in the event you want to collect history. Let's continue with the example above.

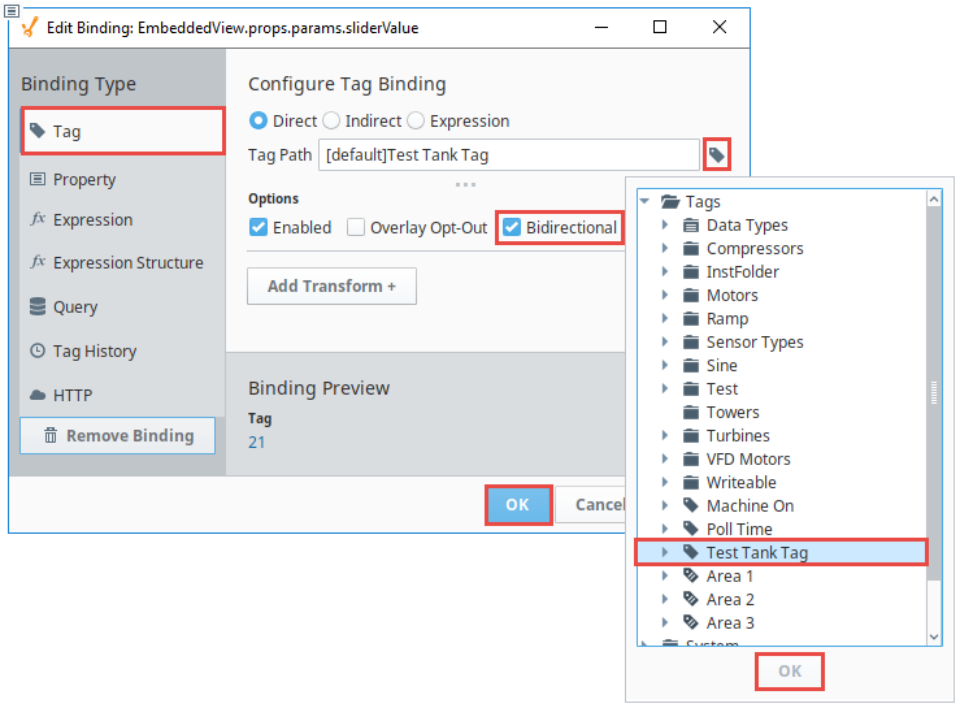
1. In the **Tag Browser**, create a Memory Tag with a data type of Integer, and assign a name (i.e.,Test Tank Tag).



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



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.

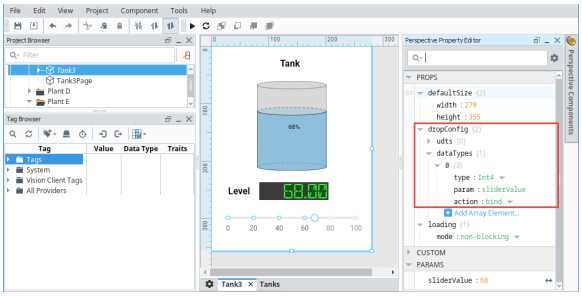
Tag dropConfig

In the previous example, we set up a Tag binding to pass a property into an embedded view. Now let's use a Tag to create an embedded view with all the property bindings configured by simply dropping the Tag in a view. Ignition will prompt you for the type of component or view you want to create.

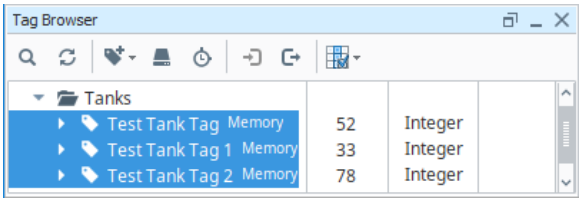
Tag Drop

Watch the Video

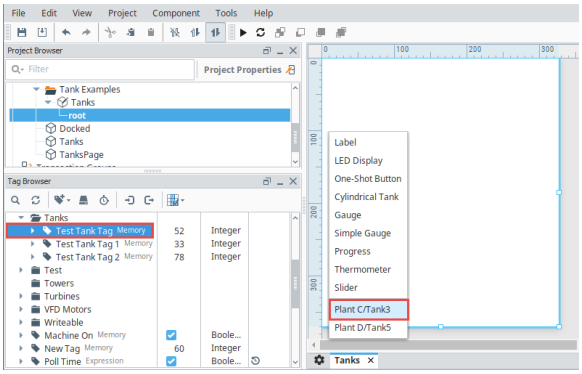
1. In the **Project Browser**, select your original view that contains the Tank, LED Display, Slider, and Labels (i.e., Tanks3).
2. In the **Property Editor**, expand the **dropConfig** property, and then expand the **dataTypes** property. Configure the following properties:
    a. Select the data type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



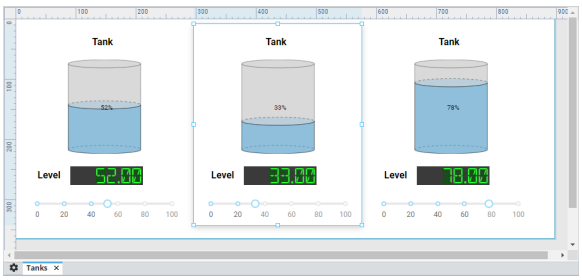
3. Let's create a couple of Tags so we can use the **'dropConfig'** feature to create multiple embedded views. Copy the Tag you created in the last example (i.e., Test Tank Tag) to make two more Tags, and change their values.



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).

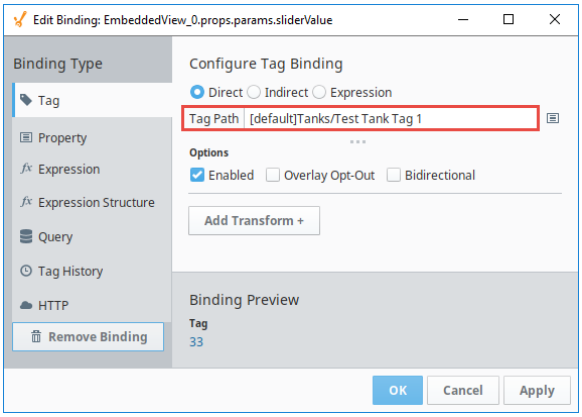


6. Once you've dragged all three of your Tags into the large view, change the Tag values, it should look something like the following:



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

Expression Bindings in Perspective

Binding Properties to the Outcome of an Expression

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

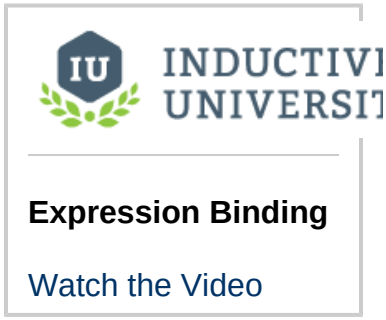
Event Based and Polling

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

On this page

...

- Binding Properties to the Outcome of an Expression
  - Event Based and Polling
- Using Expression Bindings
  - Example 1
  - Example 2



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.

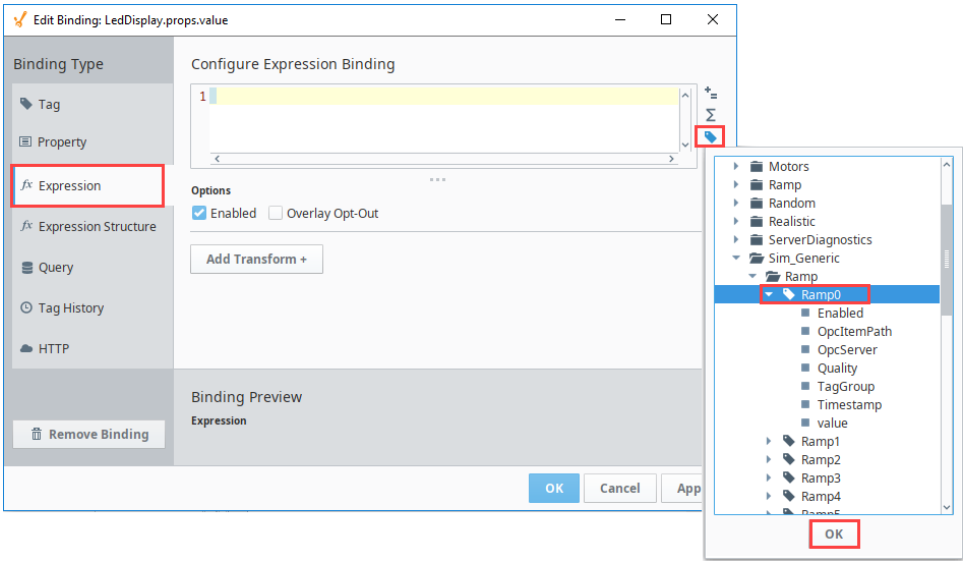
Icon	Function	Description
	Operators	Places the operator into the expression at the cursor. Mostly used as a reference to what operators are available for use.
	Functions	Places the function into the expression at the cursor. Can be used as a reference for what functions are available, as well as the 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

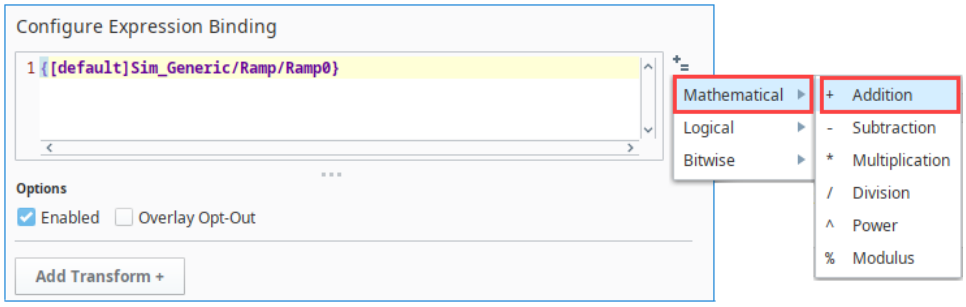
In this first example we'll use an Expression binding to combine and display the value of two Tags.

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**.

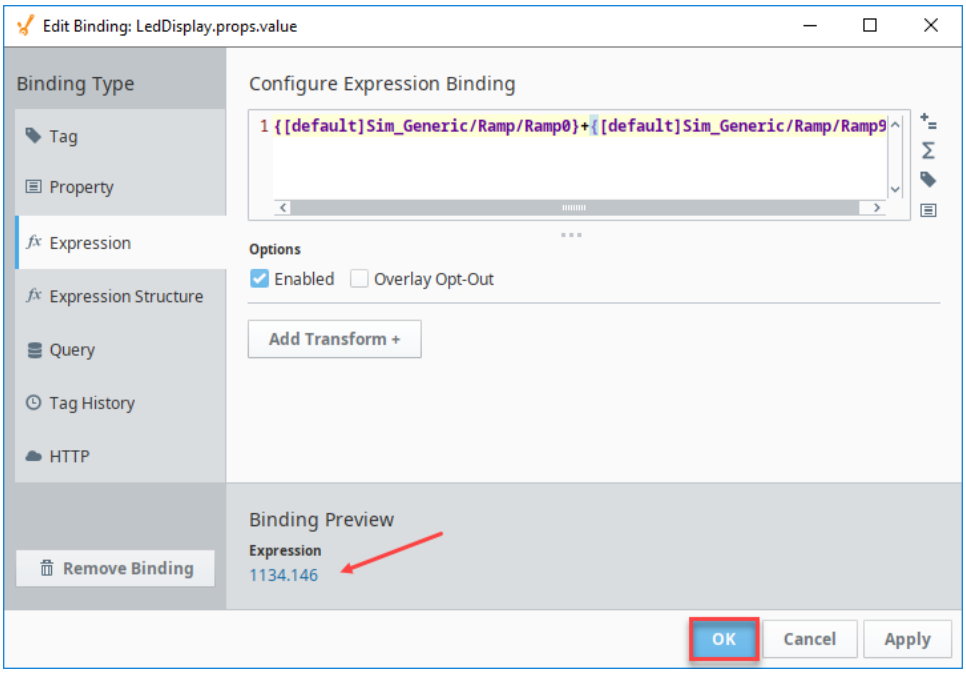


6. Next click the **Operators** icon. Select **Mathematical**, and then choose the **Addition** option.



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

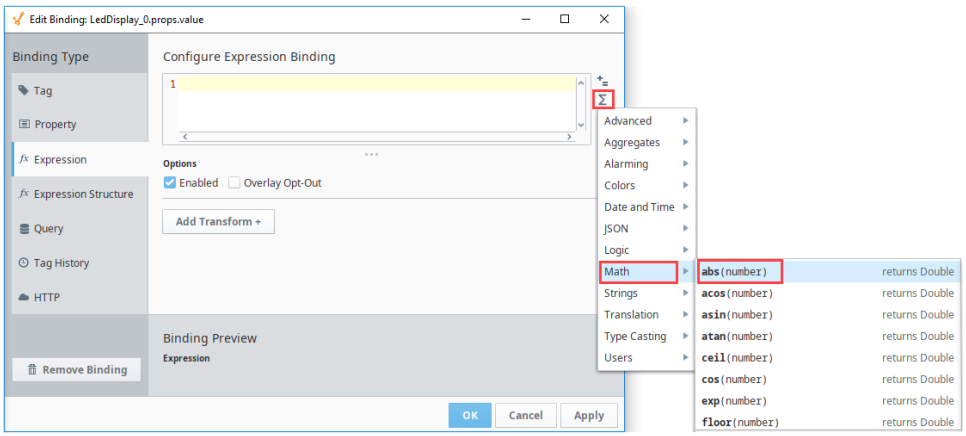
   Now when this Expression is run, the value of this Tag will be added to the first Tag. Note that a preview of the Expression binding value is shown on the lower left.



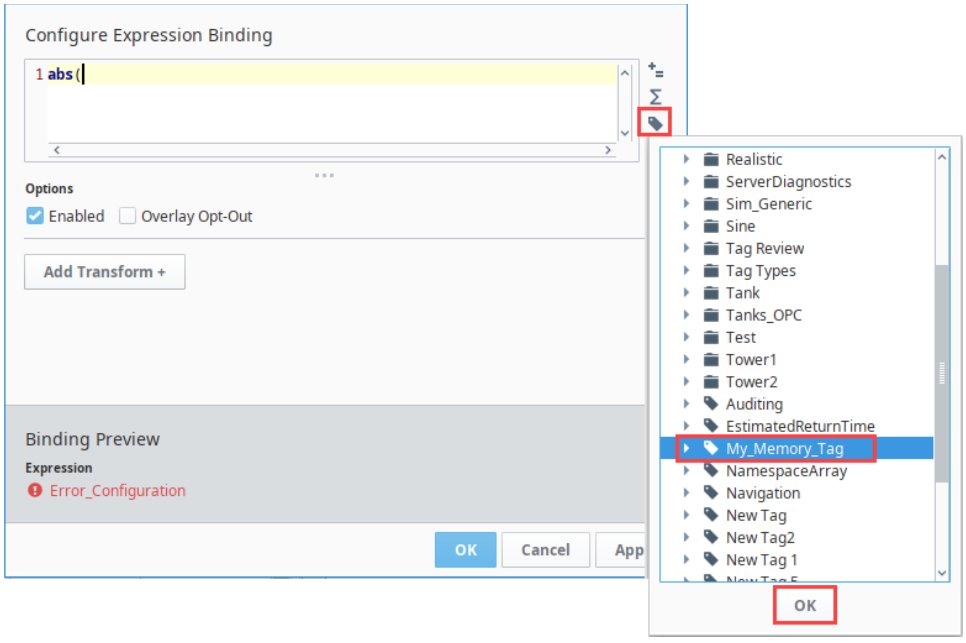
Example 2

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

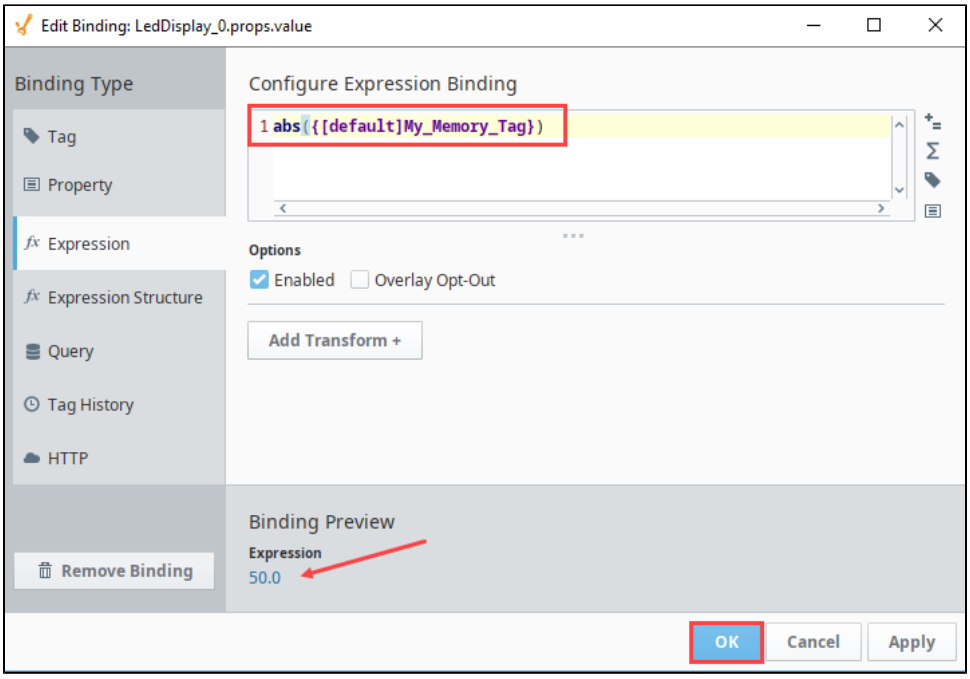
1. Select the second LED component and click on the **Binding** icon for the value property.
2. Select Expression the binding type.
3. In the Configure Expression Binding section, click on the **Functions** Σ icon. Scroll down to the **Math** functions and then select the **abs** (number) function.



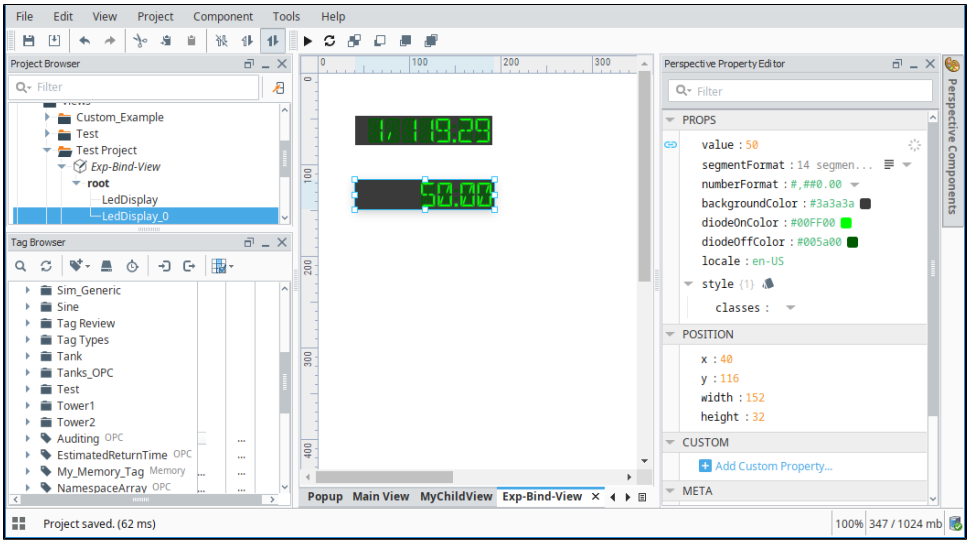
4. Next click the **Tag** icon and select the Memory Tag. Click **OK**.



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.


File
Edit
View
Project
Component
Tools
Help
Project Browser
Filter
Custom_Example
Test
Test Project
Exp-Bind-View
root
LedDisplay
LedDisplay_0
Tag Browser
Sim_Generic
Sine
Tag Review
Tag Types
Tank
Tanks_OPC
Test
Tower1
Tower2
Auditing OPC
EstimatedReturnTime OPC
My_Memory_Tag Memory
NamespaceArray OPC
Popup
Main View
MyChildView
Exp-Bind-View
Perspective Property Editor
Filter
PROPS
value : 50
segmentFormat : 14 segmen...
numberFormat : #,##0.00
backgroundColor : #3a3a3a
diodeOnColor : #00FF00
diodeOffColor : #005a00
locale : en-US
style (1)
classes :
POSITION
x : 40
y : 116
width : 152
height : 32
CUSTOM
Add Custom Property...
META
Perspective Components
Project saved. (62 ms)
100%
347 / 1024 mb

Expression Structure Bindings in Perspective

What Is an Expression Structure?

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

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

Binding Interface

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

Binding Properties

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

Example

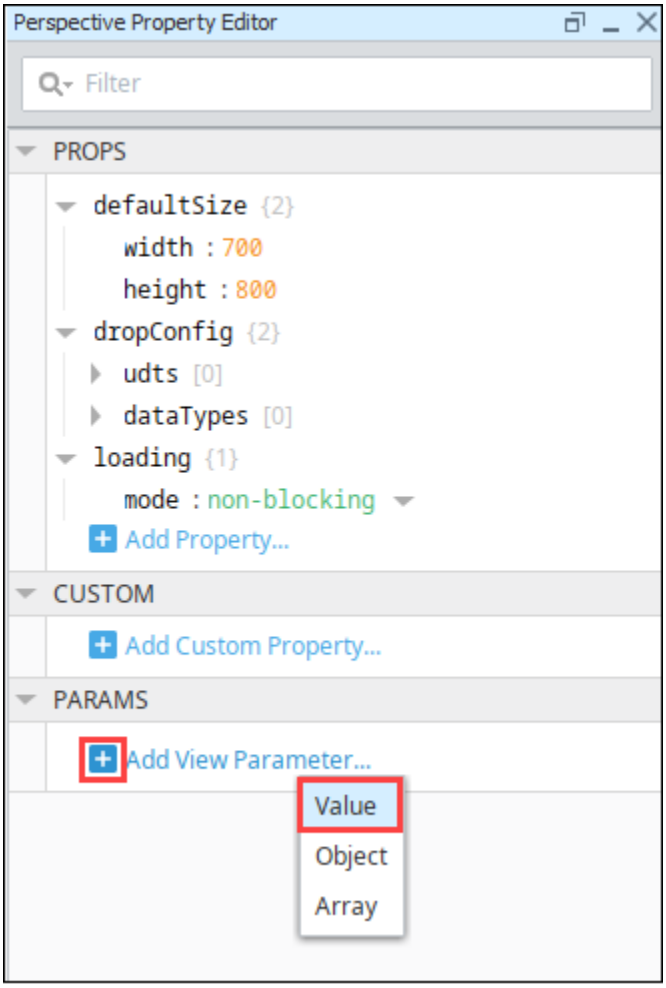
1. Create a new memory Tag called NewTag1. Set the following:
   Data Type: **String**
   Value: **It Works!**
2. Click **OK** to save the Tag.
3. Create View called MyParentView.
   a. Place a Carousel component and a Label component on the view.
   b. Set the label text as "Parent View."
4. Create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

On this page

...

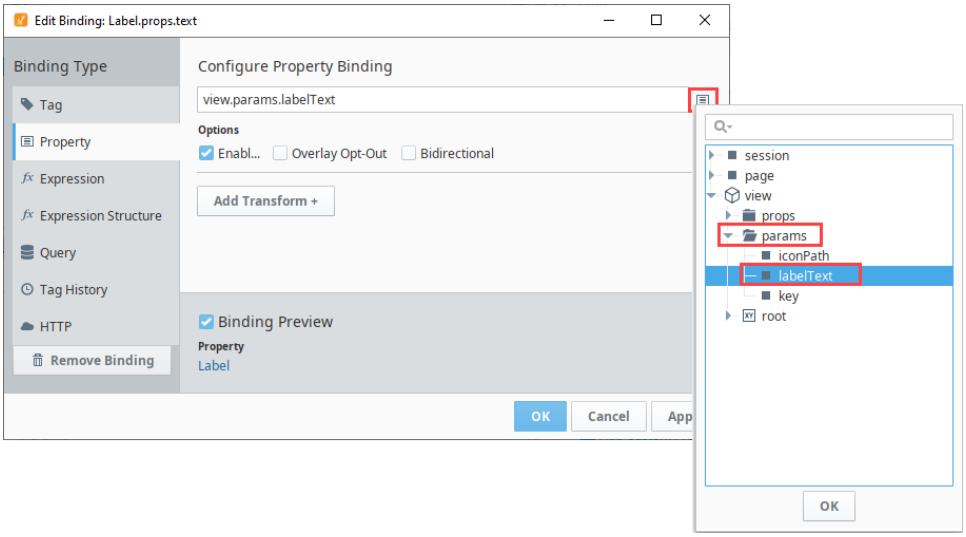
- What Is an Expression Structure?
- Binding Interface
  - Binding Properties
- Example

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



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

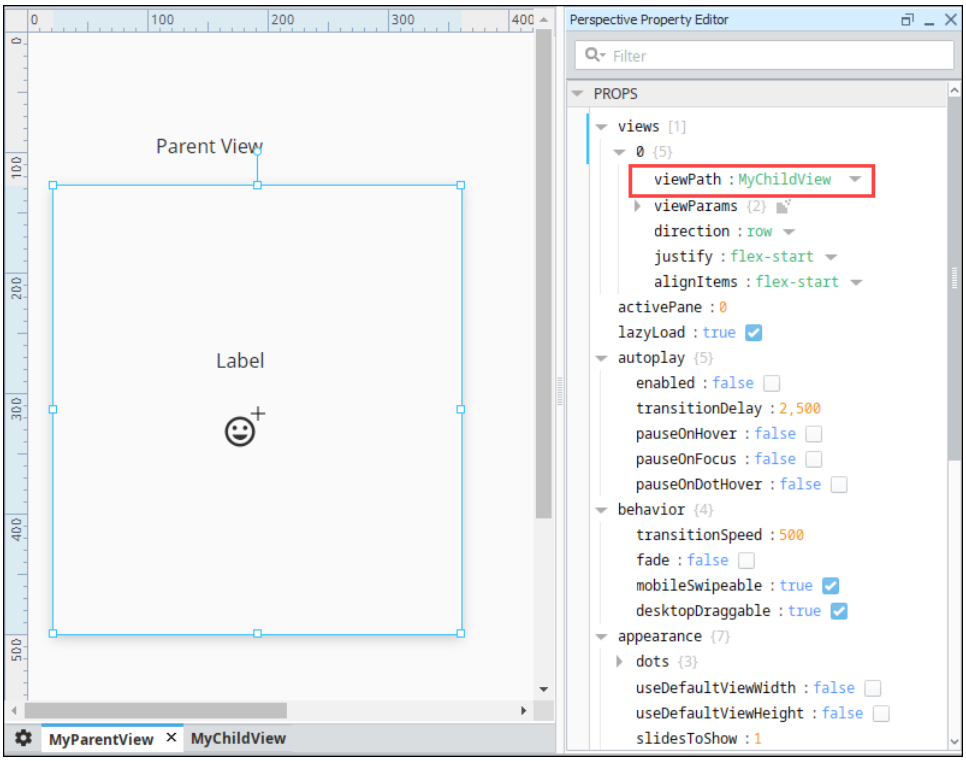
7. Next we'll bind the components to the view parameters. On **MyChildView**, select the Label component.
   a. Click the **Binding** icon next to the text property.
   b. Select the Property binding type.
   c. Click the **Property Editor** icon.
   d. Scroll down to the labelText view parameter. Click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



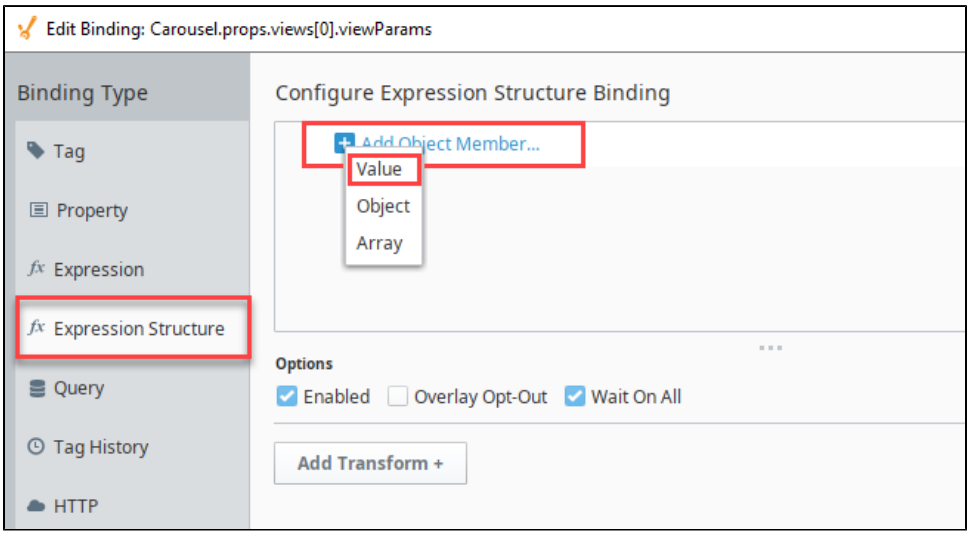
e. Select the Icon component.

f. Click the **Binding** icon next to the path property.
g. Select the Property binding type.
h. Click the **Property Editor** icon.
i. Scroll down to the iconPath view parameter. Click **OK**. Click **OK** again to save the binding.

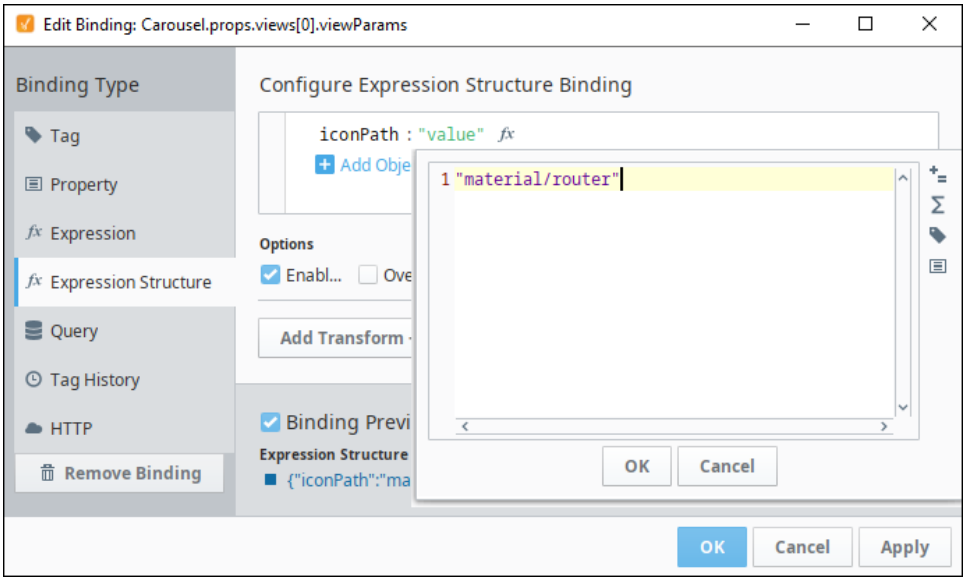
8. Back on MyParentView, select the Carousel component and set the viewPath property to **MyChildView**.



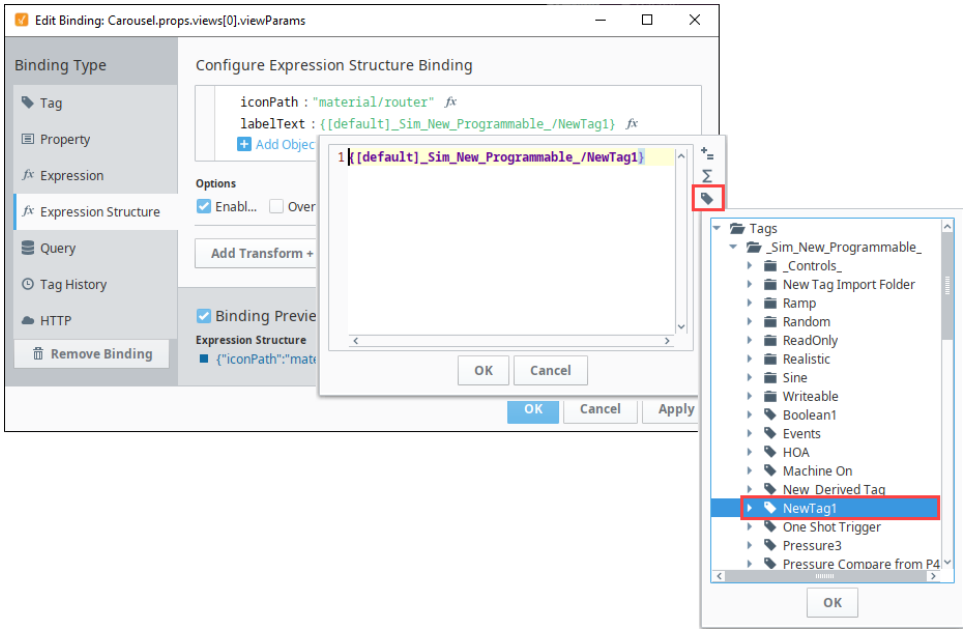
9. Next we need to add two view parameters. Still on the MyParentView, click the **Binding** icon next to the **viewParams**. The Edit Binding screen is displayed.
10. Choose Expression Structure as the binding type. Click **Add Object Member...** and select **Value**.



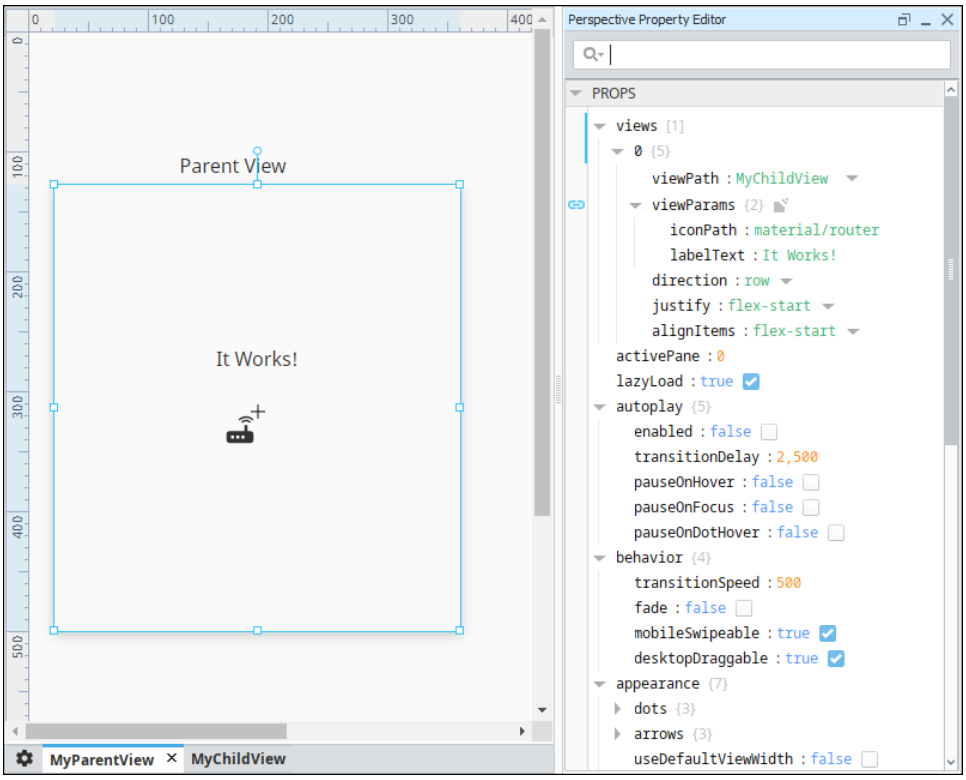
11. Name this parameter **iconPath**.
12. Click the **Expression** icon, then enter "material/insert_emoticon" as the expression. Click **OK**.



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

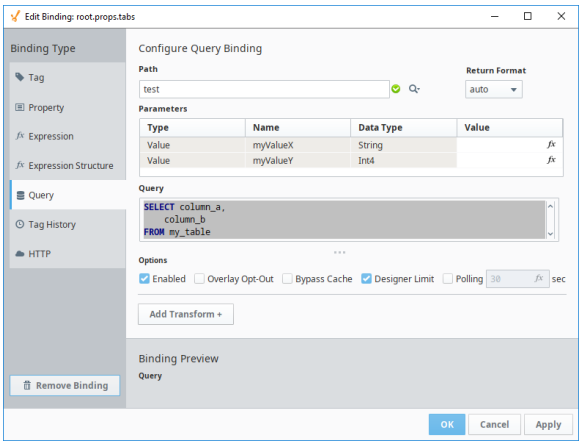


16. Select the **Wait On All** option. This will ensure that all expressions provide a value before this binding will publish its initial value.
17. Click **OK** to save the binding. Now the label text we're using in MyChildView will be populated by this tag (NewTag1).


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

Query Bindings in Perspective

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



On this page

...

- Binding Properties

Binding Properties

Property Name	Description
Path	Here you can enter in the path to the Named Query. Click on the **Search** icon to get a list of all available Named Queries.
Return Format	The Return format specifies how the query results are returned. Options are auto, json, dataset, or scalar. • **auto**: Query results are returned in the format native to the database (typically dataset). • **jSon**: Query results are returned in jSon format. This format is recommended for XY Charts. • **dataset**: Query results are returned in dataset format. This format is recommended for tables. • **scalar**: Returns the first element from the query result. This format is best when a single value is expected.
Parameters	Here you can see a table of all defined Named Query parameters. You can pass in property or Tag values to the parameters by first highlighting the parameter and then selecting either the Property icon or the Tag icon. **Note:** The fields under the Value column are evaluated as expressions, so strings will require quotations marks.
Query	In the query section, there is nothing to configure, but you can see what the Named Query you have selected looks like.
Options	
Enabled	Allows the component to be active/in use /interactive on the screen.
Overlay Opt-Out	Indicates whether the component should reflect a bad quality binding via a Tag overlay.
Bypass cache	This will cause the query to bypass/Ignore any cached values from the Named Query and run every time it is called.

Designer Limit	This setting will force the results of the query to be limited to a few rows when run in the Designer.
Polling	Here you can set the Polling Mode of the Named Query binding based on the Polling rate.

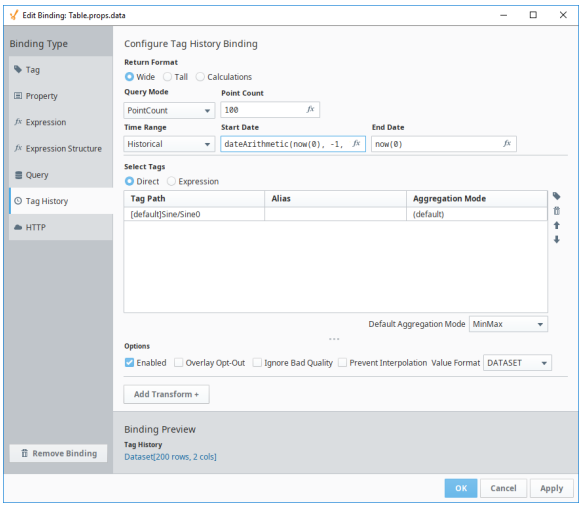
Related Topics ...

- Named Queries
- Transforms

Tag History Bindings in Perspective

Tag History Binding

Tag History Bindings allow you to pull Tag History data that is stored in the database into a component through a binding. The binding type, which is only available for Dataset type properties, runs a query against the Tag Historian.



On this page

...

- Tag History Binding
- Tag History Binding Example
- Using Dynamic Tag Paths

Tag History Binding

Watch the Video

Tag History Binding - Expression

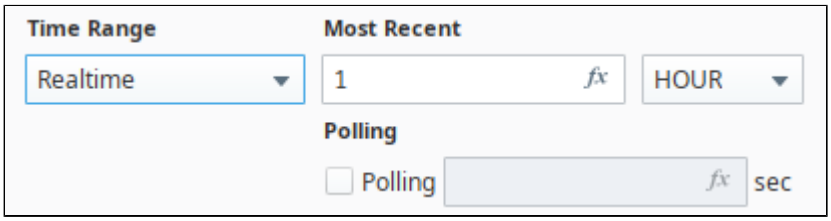
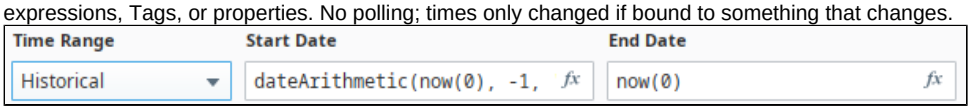
Watch the Video

Configuration Property	Description
Return Format	Allows you to select the return format of the data. Possible options are: **Property** / **Description** 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	How you want to query out the data. Possible options are: **Property** / **Description**

PointCount	Will return the number of records defined in the Point Count property.
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. Possible options are:

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 fx icon to use operators, expressions, Tags, or properties. 
Historical	You can specify the Start and End Date in an expression. Click the Functions fx icon to use operators, expressions, Tags, or properties. No polling; times only changed if bound to something that changes. 

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.

Aggregation Mode

The aggregation mode that will be used, unless a more specific aggregation mode is defined on a Tag Path.

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 in the interval.
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 interval, only one row will be returned.
LastValue	The value closest to the ending time of the interval will be returned.
SimpleAverage	The values are summed together and divided by the number of values.
Sum	The values in the interval are summed together.
Minimum	The minimum value in the interval.
Maximum	The maximum value in the interval.
DurationOn	Returns the number of seconds that the value was recorded as non-zero.
DurationOff	Returns the number of seconds that the value 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 changed from a non-zero value to zero.

Count	Returns the number of times a value was recorded
Range	Returns the range between the highest and lowest value for the period.
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 the values are close to the mean, and high standard deviation shows that the data points are spread out over a large range of values. Only good quality values are used when calculating
PctGood	Time-weighted percentage of good values over the date range.
PctBad	Time-weighted percentage of bad values over the date range.

Default Aggregation Mode

Aggregation mode to use as a default if the Select Tags are set to Default Aggregation mode.

Options

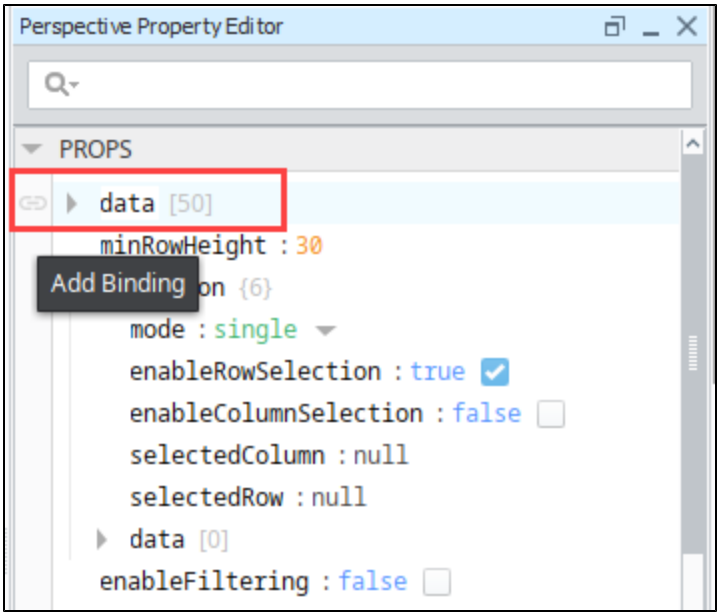
Allows you to specify various options that will apply to the binding.

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.
Value Format	Can be Dataset or Document.

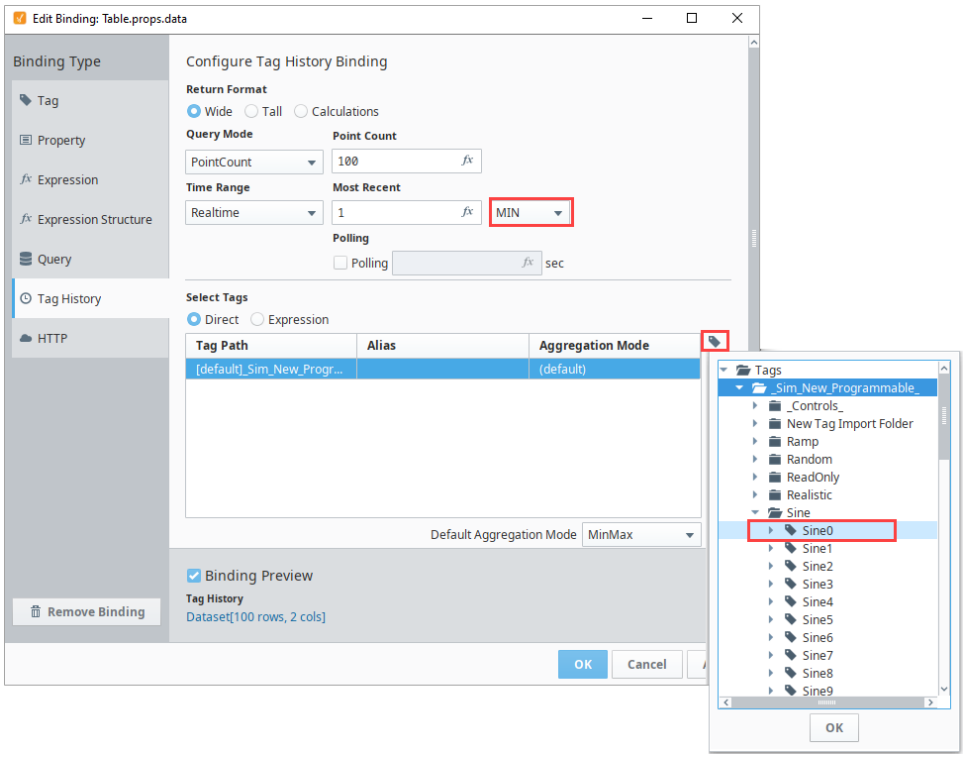
Tag History Binding Example

In this example, we'll use a Table component to show the records from some Tags that have Tag History enabled.

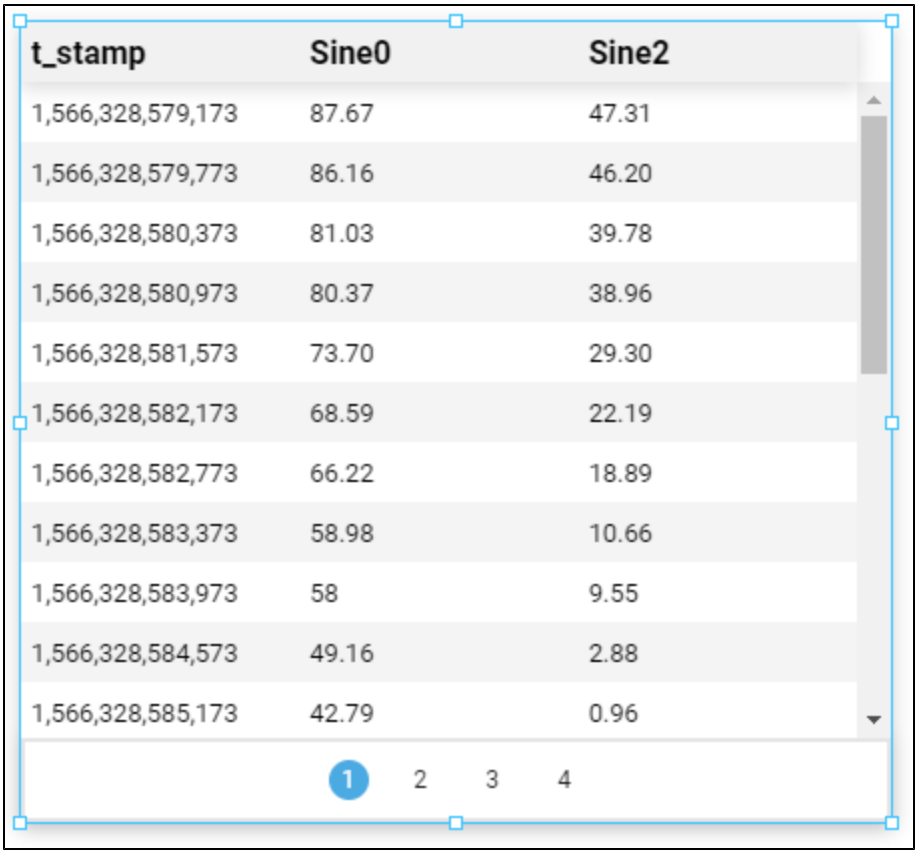
1. Drag a Table component onto a view. Select the Table component.
2. In the Property Editor, click on the **Binding** icon next to the **data** property. The Edit Binding screen is displayed.



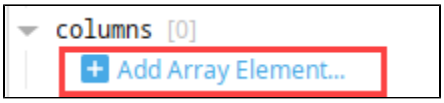
3. Select **Tag History** as the binding type.
4. In the **Time Range** section, select and the last 1 minute of data.
5. Under the **Select Tags** section, click on the Tag icon. Navigate to the first Tag and click **OK**. Repeat for additional Tags. In our example, we chose the Sine0 and Sine2 Tags.



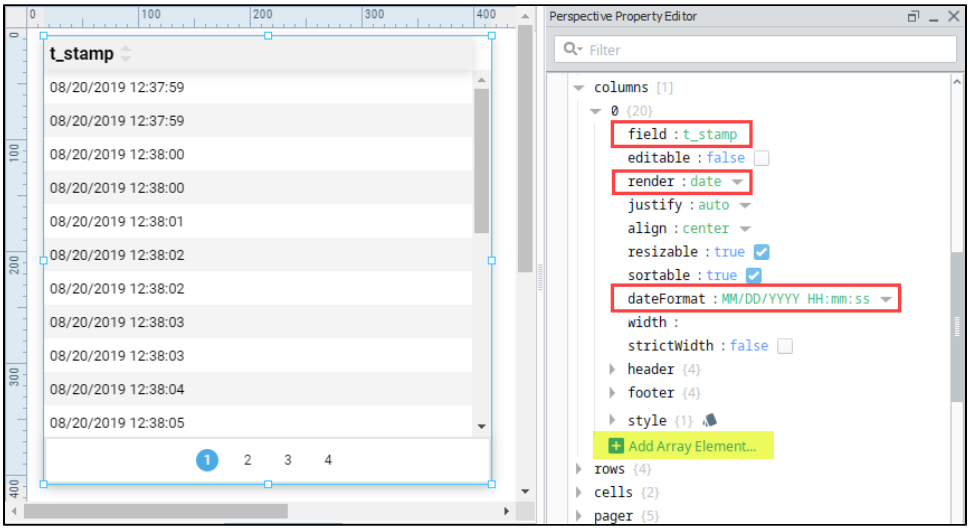
6. Notice the Binding Preview at the bottom of the screen. Click **OK** to save the Tag History binding.
7. Back in the view, the Table component now contains a column for the timestamp and one for each Tag. Notice how the timestamp does not have a date format.



8. In order to change the t-stamp values into a date format, in the Property Editor under columns, click **Add Array Element...**



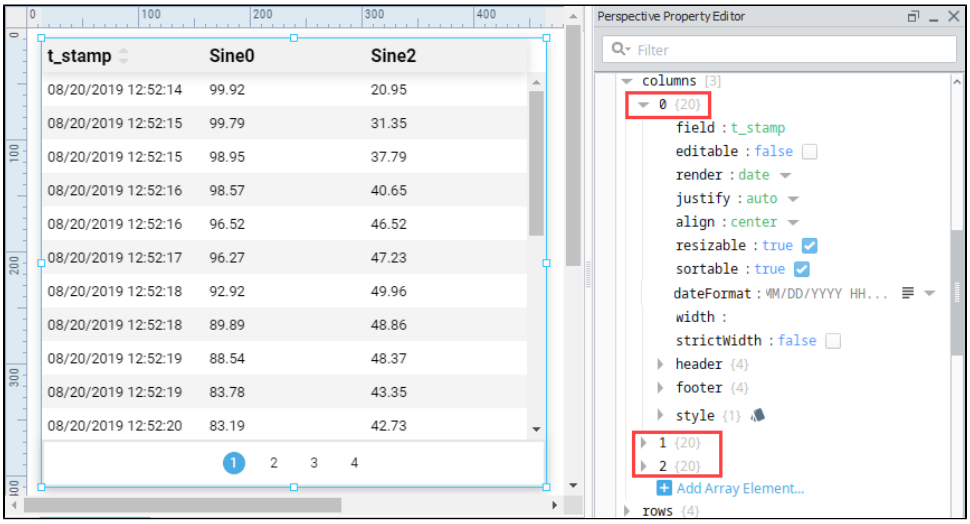
9. You'll notice that once the Array Element is added, the other two columns (Sine0 and Sine2) disappeared. We'll add them back in Step 10.
    a. In the **field** property, enter **t_stamp** (which is the column name).
    b. For the **render** property, select **date** from the dropdown.
    c. Select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



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.



Editor notes are only visible to logged in users
Tag History Binding - Expression

This is supplemental video. Not sure if we need to put a full example in here (complicated). It can possibly be lower priority for now.

Using Dynamic Tag Paths

Tag History bindings have the option to list out Tag paths, or to use an expression to build a Tag path. It is common to create a dynamic path or set of paths as a component property, that you then reference in other places. You can use as many Tag paths as you want, but they must all follow this format:

- **key** [array]
  - **[0]** {object}
    - **aggregate** value
    - **alias** value
    - **path** value
  - **[1]**{object}
    - **aggregate** value
    - **alias** value
    - **path** value
  - ...

For an example, you can create a custom property on a chart and use it to fuel the historical data.

1. Create a custom property named **Key** on a chart component that is an **Array** type.
2. Copy and paste the JSON below into this Key property.

JSON for Key Array

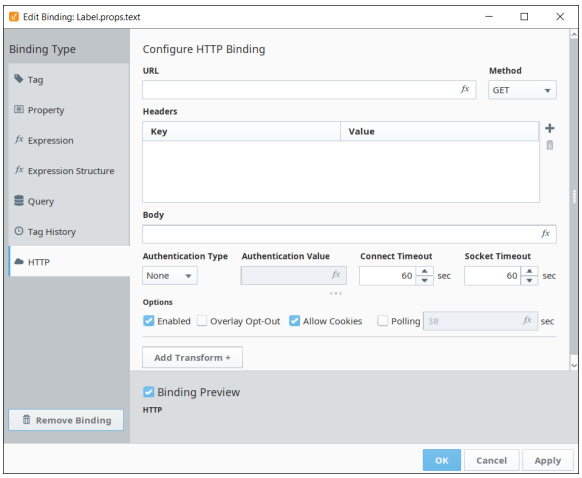
```
[
  {
    "aggregate": "Average",
    "alias": "tank_temp",
    "path": "[default]Tank/03/Temperature"
  },
  {
    "aggregate": "Average",
    "alias": "setpoint",
    "path": "[default]Tank/03/Setpoint"
  }
]
```

3. Open the property binding on your chart and select the **Tag History** binding type.
4. Select the **Expression** radio button to create your own tag structure.
5. Click on the property selector button on the right and find your new custom property.
6. Click **OK** in the lower right.

HTTP Bindings in Perspective

Perspective enables you to integrate network and internet driven resources seamlessly into a project. An HTTP binding is used to pass data directly to and from a URL.

> ⚠ The HTTP binding is an advanced binding type, and requires a basic understanding of web development and the HTTP protocol.



On this page

...

- JSON Support
- HTTP Binding Configuration

HTTP Binding

Watch the Video

JSON Support

One perk of the HTTP binding is the ability to fetch JSON documents from a website or an API. Since the Perspective property tree is also JSON-formatted, this allows you to dynamically create a property structure from a JSON document directly.

HTTP Binding Configuration

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. Options as follows: **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.

<table>
<tr><td>Headers</td><td>Used to pass key/value pairs in the header of our HTTP requests.<table><tr><th>Field</th><th>Definition</th></tr><tr><td>Key</td><td>Allows dropdown selection from common header keys, or the ability to specify a custom one.</td></tr><tr><td>Value</td><td>The Value field is an expression.</td></tr></table></td></tr>
<tr><td>Body</td><td>An expression indicating what to send in the body of our HTTP requests.</td></tr>
<tr><td>Authentication Type</td><td>Indicates what HTTP authentication type to use. Equivalent to specifying the Authorization key in the header. Options are None, Basic, Bearer, or Digest.</td></tr>
<tr><td>Authentication Value</td><td>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:<pre>Authorization: Basic aWduaXRpb246cGFzc3dvcmQ=</pre>Then the Authentication Type should be Basic and the Authentication Value should be aWduaXRpb246cGFzc3dvcmQ=.</td></tr>
<tr><td>Connect Timeout</td><td>Indicates how long the Ignition Gateway should wait for a response to our connect request.</td></tr>
<tr><td>Socket Timeout</td><td>Indicates how long the Ignition Gateway should wait for a response to a given HTTP request.</td></tr>
<tr><td>Enabled</td><td>Indicates whether the binding should be active.</td></tr>
<tr><td>Overlay Opt-Out</td><td>Indicates whether the component should reflect a bad quality binding via a Tag overlay.</td></tr>
<tr><td>Allow Cookies</td><td>Indicates whether to allow the remote web server to store cookies.</td></tr>
<tr><td>Polling</td><td>Controls how frequently HTTP requests should be issued, and therefore how often the binding should be updated.</td></tr>
</table>

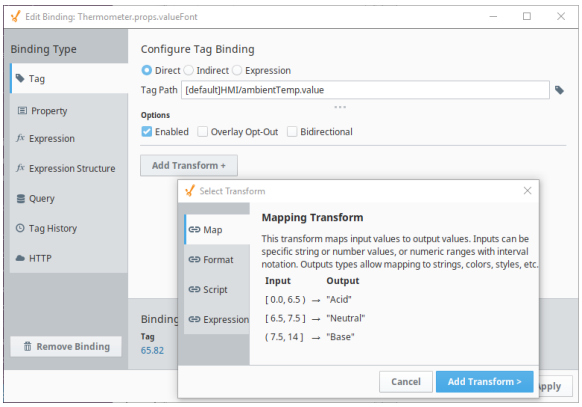
Transforms

Overview

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.



On this page

...

- Overview
- Map
- Format
- Script
- Expression

Map

The Map Transform allows you to setup a map of input values to output values. Inputs can be anything from specific values or strings, to numeric ranges of values. Each input value can then be mapped to an output value, such as a string, integer, color, style, etc. A great example of this is mapping values to colors, so that you can easily show state changes.

Format

The Format Transform applies a format string to the value returned from the binding, allowing you to format the output in any way. The format can be something like a date format or a number format.

Script

The Script Transform will run a script that allows you to manipulate the value returned from the binding using any Python script you want.

Expression

The Expression Transform runs an expression on the original value so you can manipulate it without creating more complex logic. This is most useful when you are indirectly binding to a Tag but still want to apply an expression to it, it can now be done in two distinct steps.

In This Section ...

Map Transform

The Map Transform allows you to setup a map of input values to output values. Inputs can be anything from specific values or strings, to numeric ranges of values. Each input value can then be mapped to an output value, which can be a string, integer, color, style, etc. A great example of this is mapping specific values to specific colors, so that you can easily bind visual properties to tag values.

On this page

...

- Examples
  - Numeric Range to Integer Value
  - Integer to String
  - Expression to Color
  - Numeric Range to Expression

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: 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: 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.

> The following feature is new in Ignition version **8.0.3**
> Click here to check out the other new features

- **Style Class:** A style class to use. Once selected, a dropdown list shows the available style classes.

Use the **Add** icon to add rows to the Map table. The **Delete** icon will delete the selected row. The up/down arrows will sort the order execution for the mapping. The first mapping (from top to bottom) that matches will be the mapping that is assigned.

You can then double-click to set values for each cell except the Fallback cell. The Fallback allows you to create a value to use in case non of your values, expressions, or ranges match to the input.

Examples

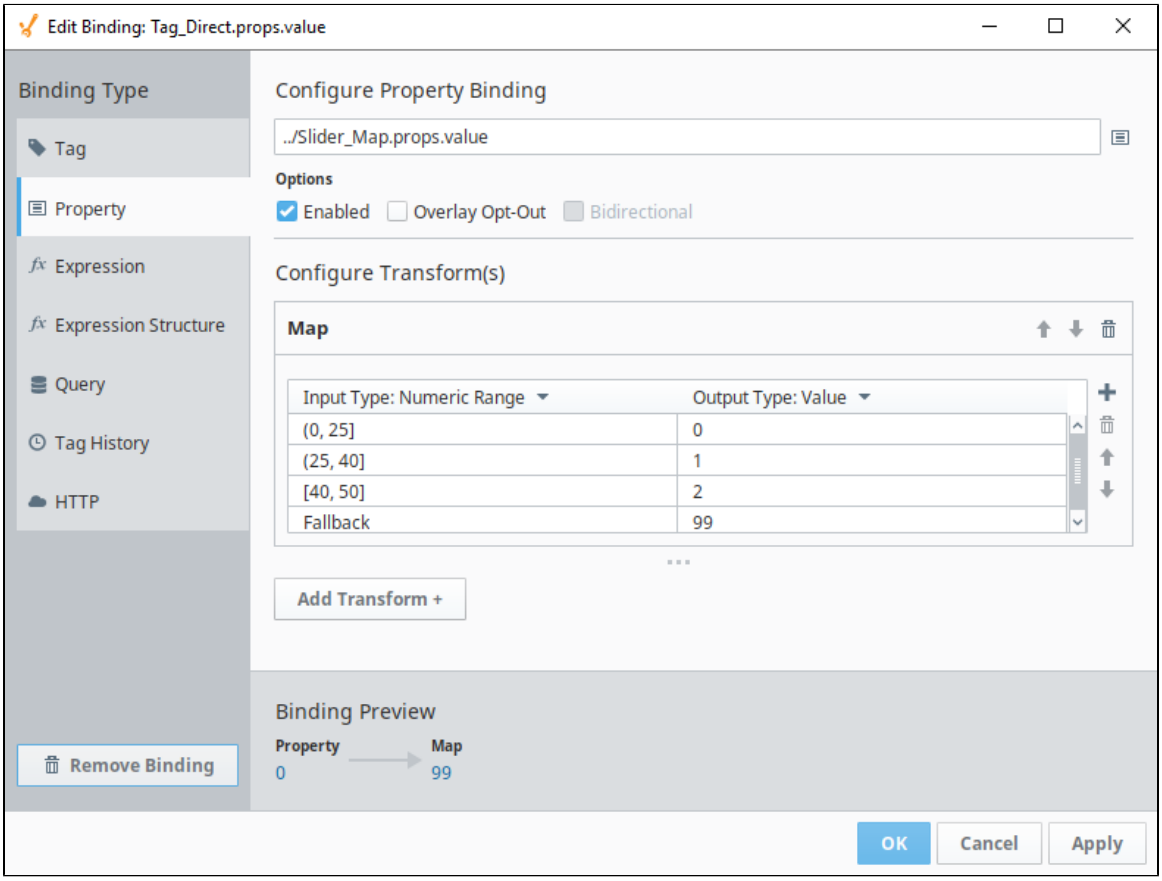
Numeric Range to Integer Value

Input Type: **Numeric Range**

Output Type: **Value**

You can see in the Binding Preview at the bottom that the original value of the property binding is zero. Since the Map Transform input value is 0, we get an output value of 99, the Fallback value. This is because the first mapping uses an exclusive 0 (with a parenthesis) so it is testing for strictly greater than 0, not "greater than or equal to."

Numeric Range to Number Mapping	
(0,25]	0
(25,40]	1
[40,50]	2
Fallback	99



Numeric Range to String Mapping - Range Values Omitted	
(,25]	"Below 25"
(25,)	"Above 25"
Fallback	"Invalid Value"

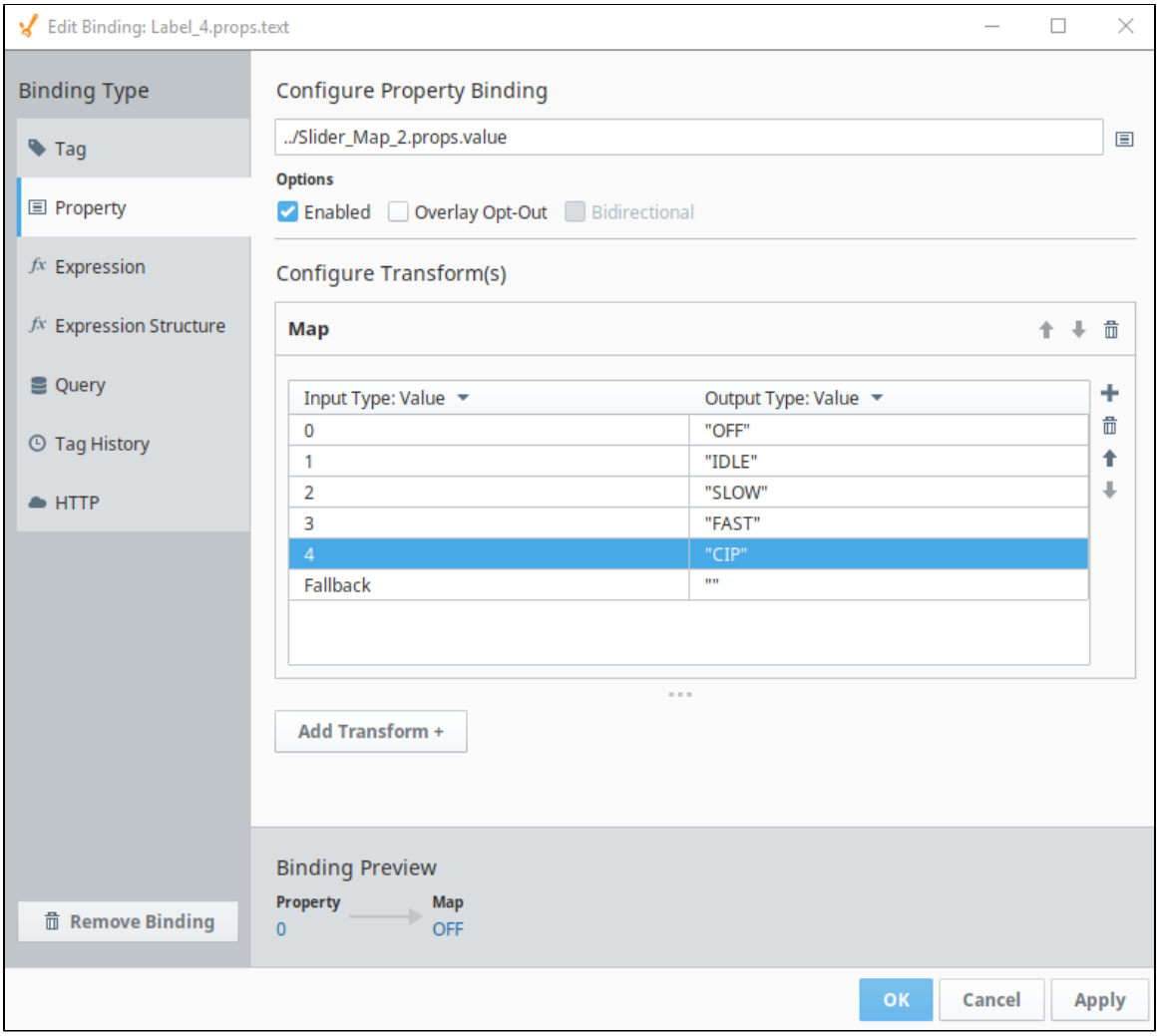
Integer to String

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".



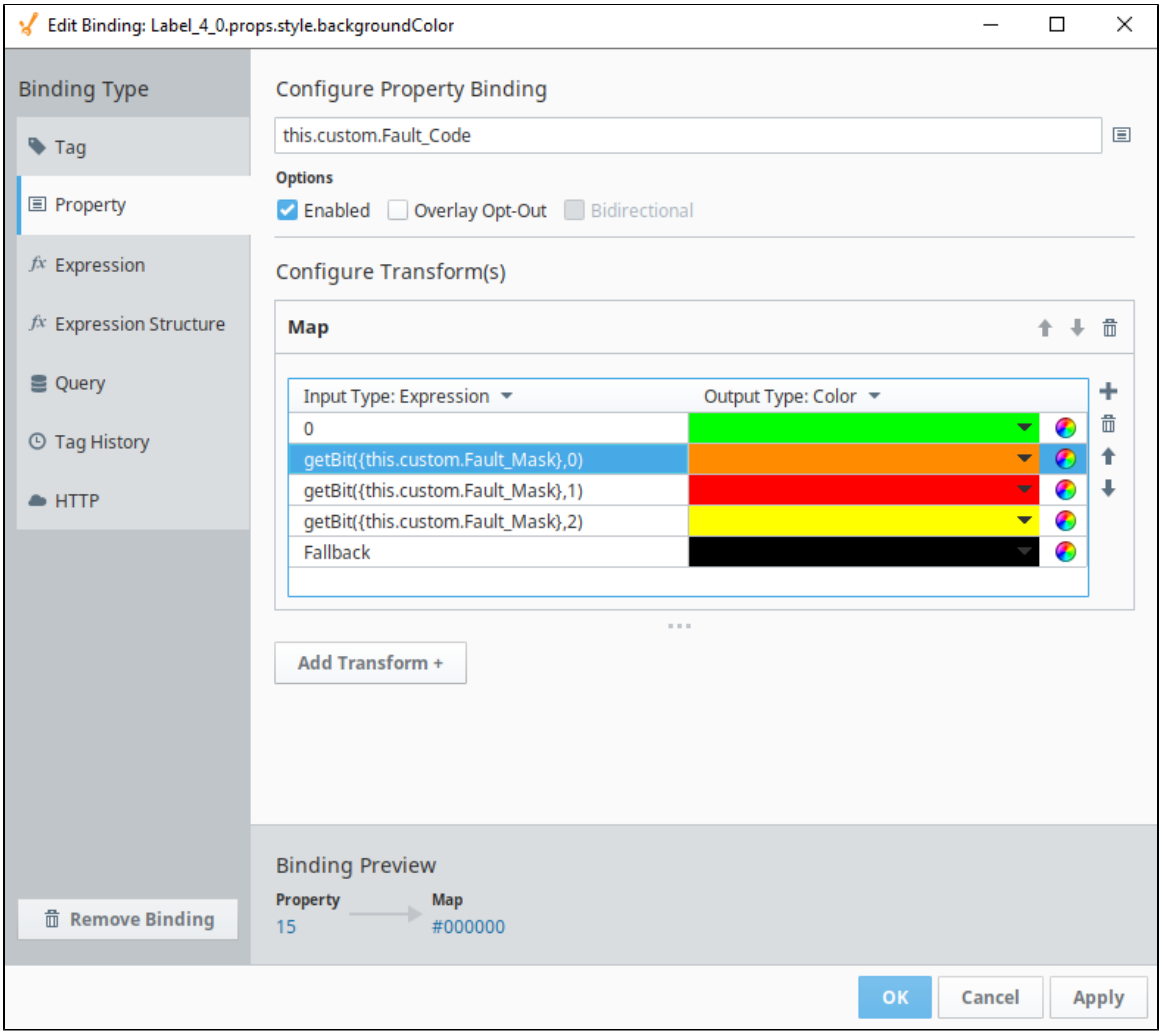
Expression to Color

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.

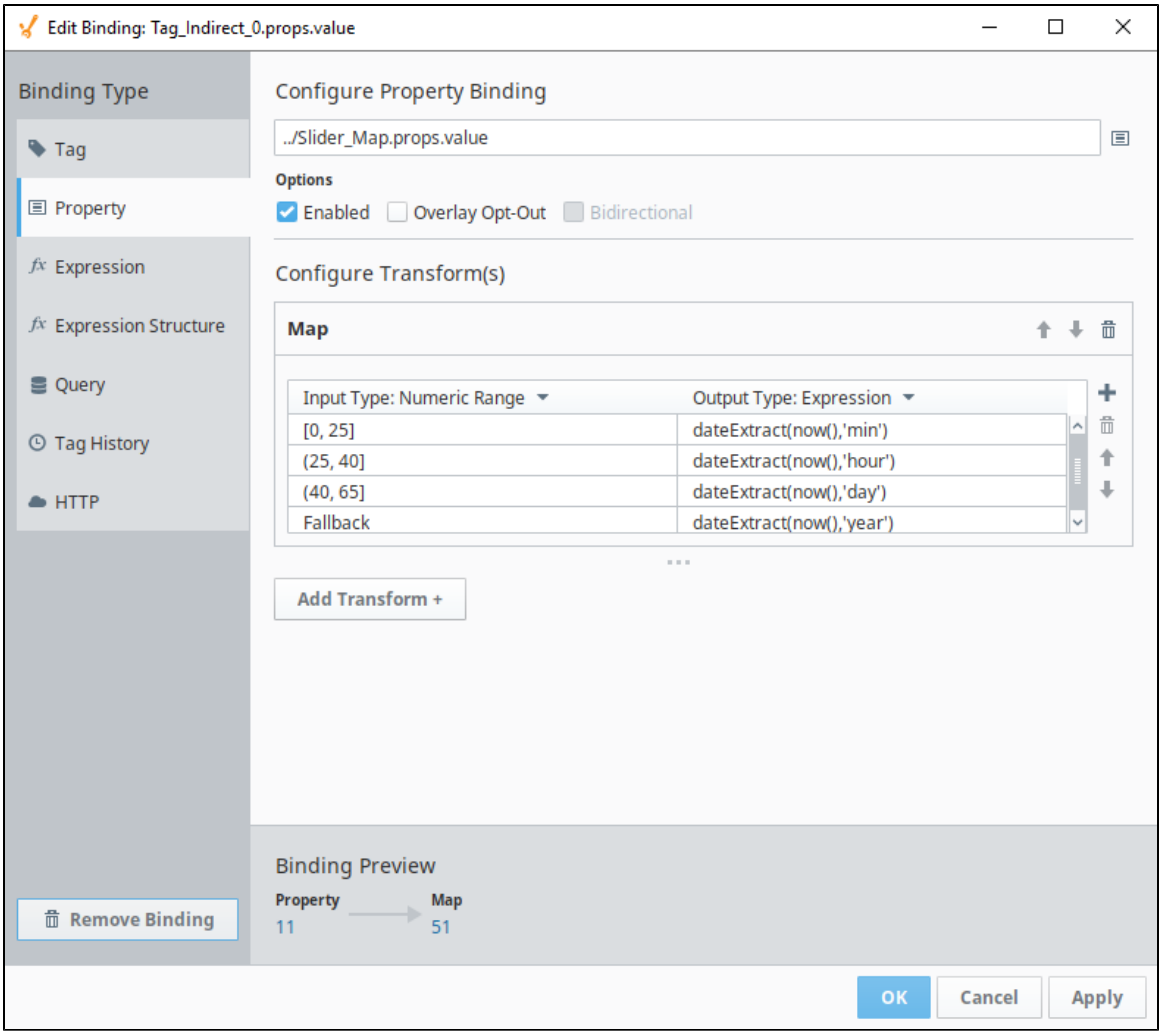


Numeric Range to Expression

Input Type: Range

Output Type: Expression

This mapping maps different ranges of values to different parts of the current time. You can see in the Binding Preview at the bottom that the original value of the property binding is 11 so we get an output value that pulls the minute out of the current time. That was 51 at the time this screenshot was taken.


Edit Binding: Tag_Indirect_0.props.value
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

The Format Transform applies a format string to the value returned from the binding, allowing you to format the way the output is displayed. Typically this applies to a date or number format.

Select Format Type:

- **Datetime:** A date will be formatted to appear a certain way.
- **Numeric:** A number will be formatted to appear a certain way.

Format (Numeric only)

- **Pattern:** Applies a number pattern (using 0 and #) and converts the results to string (good for putting in leading 0's on a display).
- **Integer:** Rounds input (float) to integer.
- **Number:** Formats number based on Language setting (puts in comma or other number separator).
- **Percent:** Converts the input number into percent (0.123 would format to 12.3%).
- **Currency:** Uses the currency specified by the Language setting.

Format (Datetime only)

- **Pattern:** Custom date format string (for example - M/D/YYYY h:m:s a).
- **Date:** Date only will be shown.
- **Time:** Time only will be shown.
- **Datetime:** Date and Time will be shown.

Date (Datetime only)

- **Full:** Example format - Monday, March 18, 2019.
- **Long:** Example format - March 18, 2019.
- **Medium:** Example format - Mar 18, 2019.
- **Short:** Example format - 3/18/19.

Time (Datetime only)

- **Full:** Example Format - 2:02:46 PM Pacific Standard Time.
- **Long:** Example format - 2:02:46 PM PST.
- **Medium:** Example format - 2:02:46 PM.
- **Short:** Example format - 2:02 PM.

Locale

- **auto:** Default time zone of Client.
- ...(Language Selection):

Time Zone (Datetime only)

- **auto:** Default time zone of Client.
- ...(Language Selection):

On this page

...

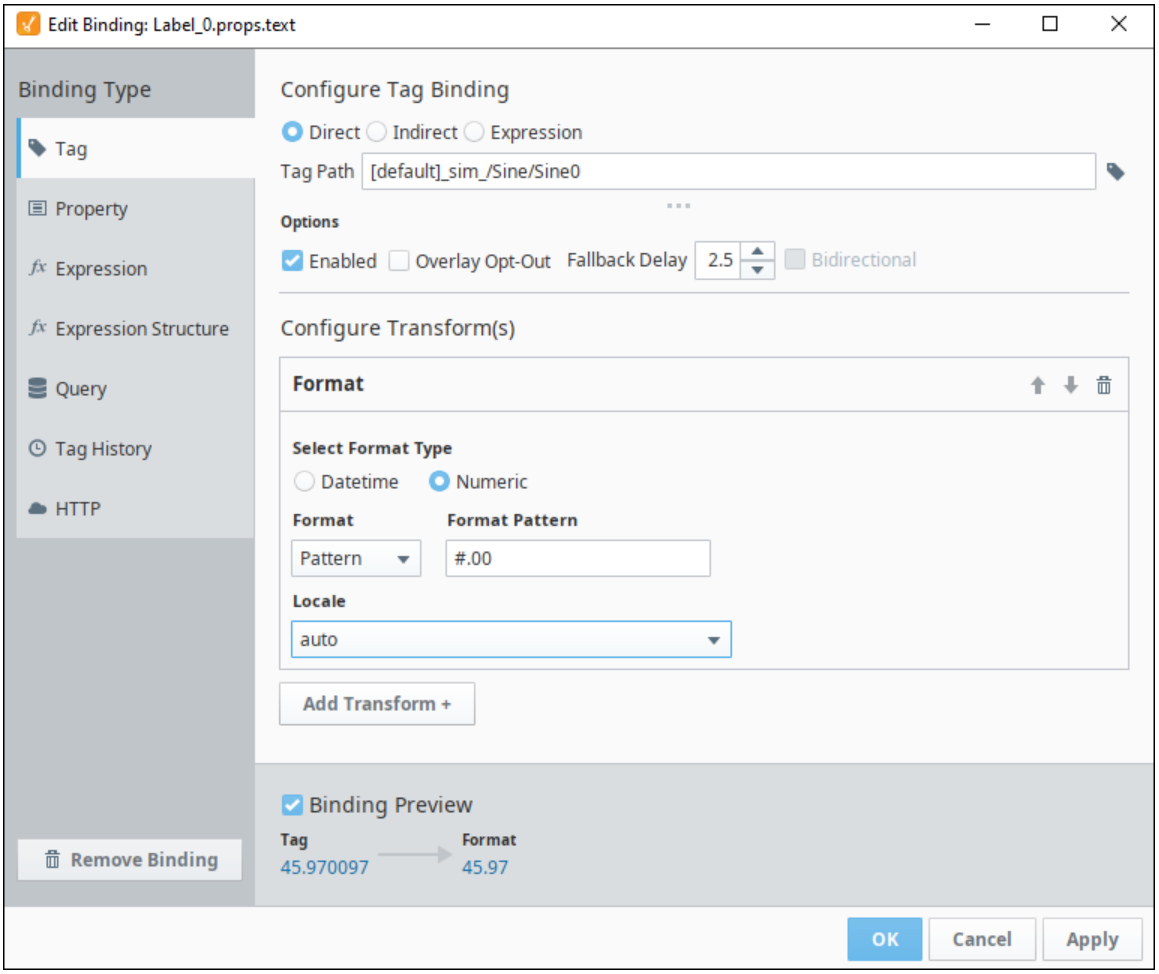
- Examples
  - Numeric Pattern
  - Datetime Short, Short, Time Zone Adjusted

Examples

Numeric Pattern

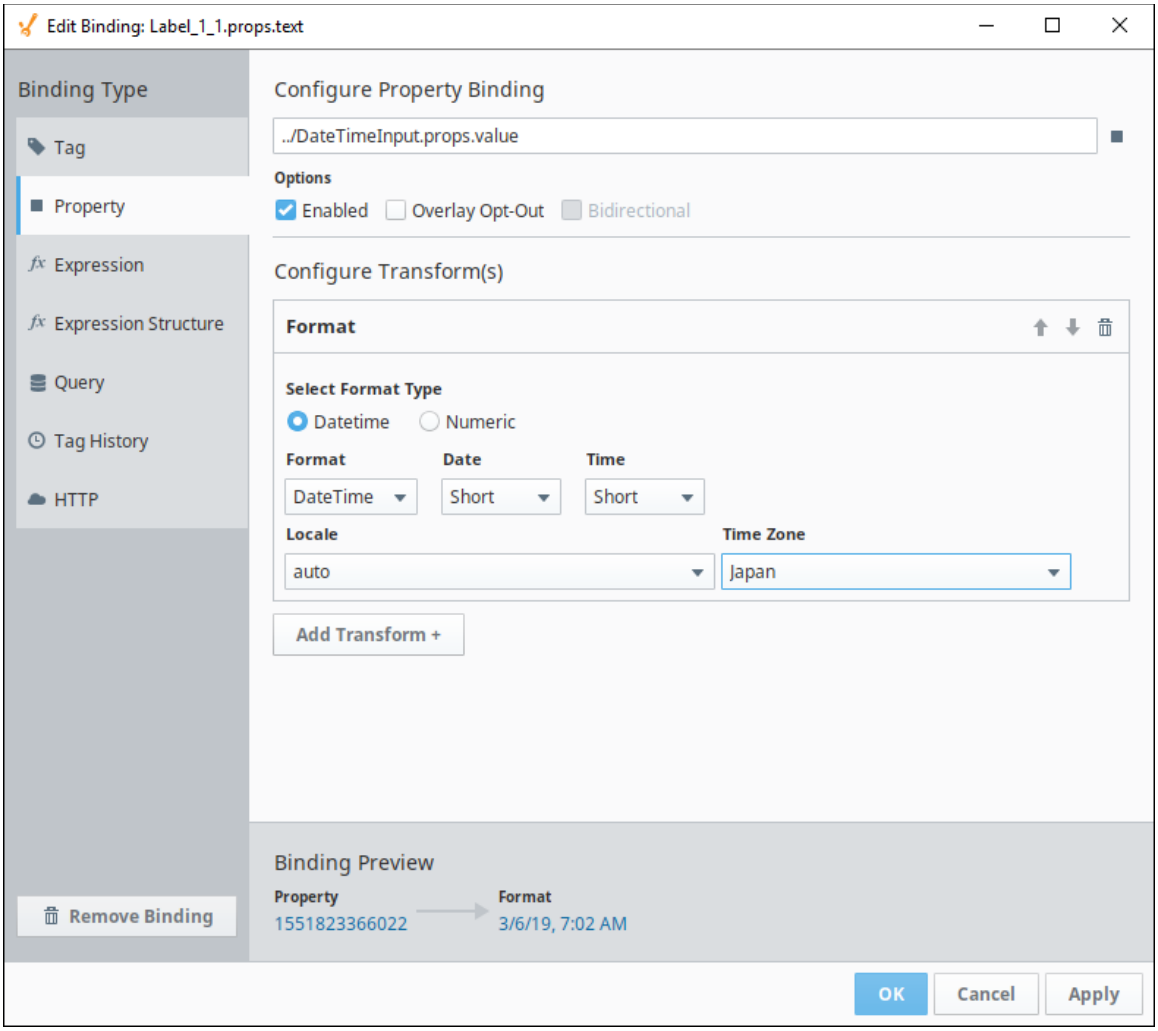
In the image below, an expression on a Tag binding is retrieving the value from a Tag. As shown in the preview, the value on the Tag shows 45.970097.

The Format transform is taking that Tag value, then applying a numeric pattern of **#.00**, which denotes that two digits must always be shown after the decimal point. As a result, the preview shows a Format value of 45.97, since any digits beyond the first two decimal places are ignored by the transform.



Datetime Short, Short, Time Zone Adjusted

The image below is taking a Unix timestamp value (including milliseconds), and turning it into a human readable date, set to Japan's timezone.


Edit Binding: Label_1_1.props.text
Binding Type
Tag
Property
Expression
Expression Structure
Query
Tag History
HTTP
Remove Binding
Configure Property Binding
../DateTimeInput.props.value
Options
Enabled
Overlay Opt-Out
Bidirectional
Configure Transform(s)
Format
Select Format Type
Datetime
Numeric
Format
Date
Time
DateTime
Short
Short
Locale
Time Zone
auto
Japan
Add Transform +
Binding Preview
Property
1551823366022
Format
3/6/19, 7:02 AM
OK
Cancel
Apply

Script Transform

Script Transforms are special functions that are applied only on an existing binding. They take the result of a binding (or transform) as an input and produce a single output. This will allow you to manipulate a binding result using whatever python code you want.

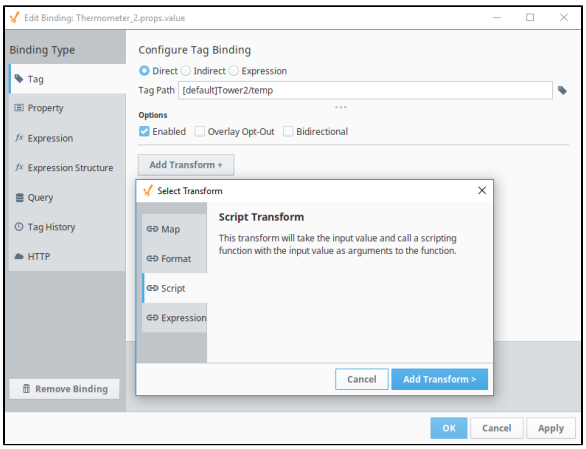
When a script transform is first created it generates a template script with a few assumed input Arguments (see below). From this template you can write your own script and reference an incoming value as well as a few other parameters. It is assumed the custom script will end with a Return comment that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

Arguments:

- **self:** A reference to the component this binding is configured on.
- **value:** The incoming value from the binding or the previous transform.
- **quality:** The quality code of the incoming value.
- **timestamp:** The timestamp of the incoming value as a java.util.Date

Return:

- Any single value of any data type. As with all other transforms, the type of data that is returned will overwrite the data type of the property that is being bound.



On this page

...

- Using Complex Properties in your Transform
- Example Scripts
  - Dataset to Array of Objects
  - Dataset to Array with Renamed Columns
  - Sparkline Chart

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**.

⚠ Don't forget to tab in correctly if you are copying scripts from this page. The Script Transform is a function (starts with "def") and every line of the code below should be tabbed in one from the edge.

These examples are not necessary with Tag History or Query binding types. For both of these, there is a dropdown setting at the bottom of the binding page that allows you to select the return type of DOCUMENT.

Dataset to Array of Objects

This is a script to take a dataset and transform it into a json array. All header names will be included in the resulting structure. This type of array is expected on many Perspective components like the Table component.

Dataset to Array of Objects

```
# convert the incoming value data
pyData = system.dataset.toPyDataSet(value)
# get the header names
header = pyData.getColumnNames()
# create a blank list so we can append later
newList = []

# step through the rows
for row in pyData:
	# create a new blank dictionary for each row of the data
	newDict = {}
	# use an index to step through each column of the data
	for i in range(len(row)):
		# set name/value pairs
		newDict[ header[i] ] = row[i]

	# append the dictionary to list
	newList.append(newDict)

# return the results
return newList
```

Dataset to Array with Renamed Columns

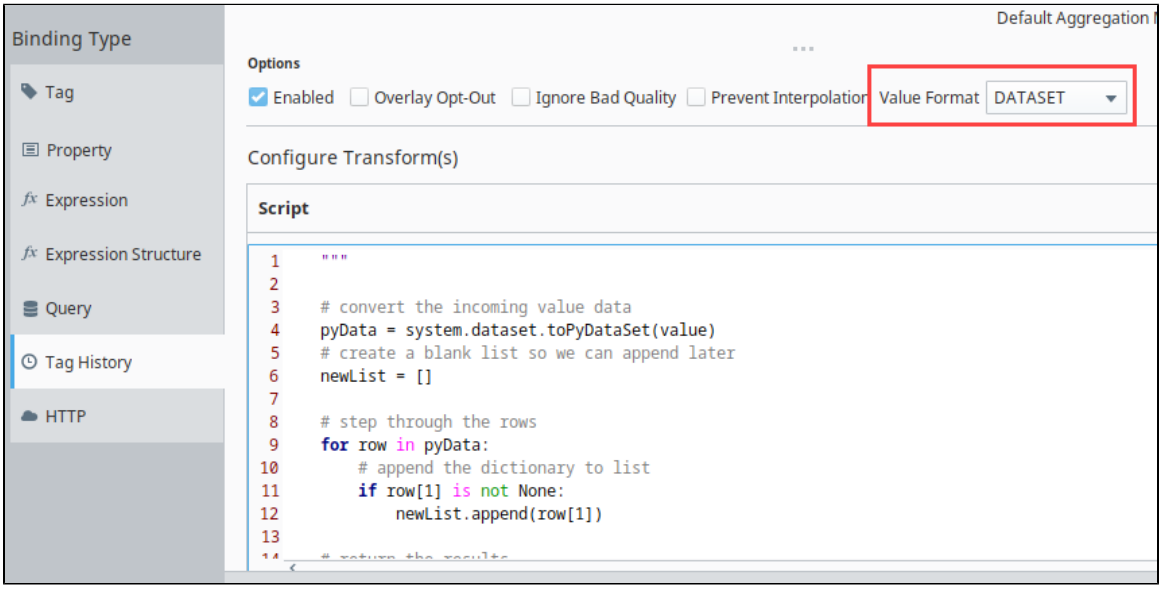
This is another script to take a dataset and transform it into a json array. In this example, you create the header names to be included in the resulting structure. This type of array is expected on many perspective components like the XY chart, Dropdown, etc.

Dataset to Array of Named Objects

```
# set the header names. For the XY chart, these must match the values in the series property.
header = ["Column 1", "Column 2"]

# convert the incoming value data
pyData = system.dataset.toPyDataSet(value)
# create a blank list so we can append later
newList = []

# step through the rows
for row in pyData:
	# create a new blank dictionary for each row of the data
	newDict = {}
	# use an index to step through each column of the data
	for i in range(len(row)):
		# set name/value pairs
		newDict[ header[i] ] = row[i]

	# append the dictionary to list
	newList.append(newDict)

# return the results
return newList
```

Sparkline Chart

The data expected in a Sparkline chart is a bit different than other charts. Instead of having values paired with a timestamp, it just takes an array of values to draw in order. This is a script to take a Tag History binding, apply the Script Transform, and output only the values in an array. Make sure the Value Format dropdown is set to Dataset.



Tag History to list of Data Points

```
# convert the incoming value data
pyData = system.dataset.toPyDataSet(value)
# create a blank list so we can append later
newList = []

# step through the rows
for row in pyData:
        # append the dictionary to list
    if row[1] is not None:
                newList.append(row[1])

# return the results
return newList
```

Expression Transform

The Expression Transform runs an expression that allows you to manipulate the value of the binding using an expression. An Expression transform uses the Ignition Expression binding language and has built in links to several toolsets.

Those expression tools are as follows:

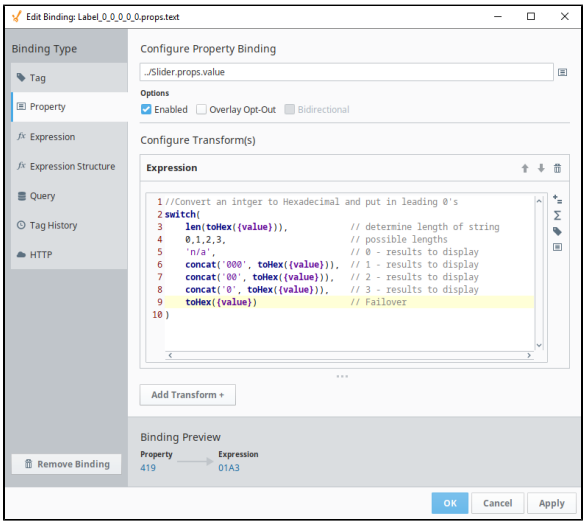
- **Operators:** Mathematical, Logical, Bitwise operators to adjust the incoming value.
- **Functions:** A library of expression functions to adjust the incoming value
- **Browse Tags:** A link to the Tag browser
- **Browse Properties:** A link to Session Properties or other properties in the same View.

On this page

...

- Example - Function

Example - Function



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
)
```

Scripting in Perspective

This section is designed to familiarize you with some of the basics of Python scripting in Perspective. Perspective scripting is particularly powerful, and can be used to control and fine-tune many aspects of project design.

For a more general view of scripting in Ignition, and an introduction to Python, see Scripting in our Ignition Platform section.

On this page

...

- Perspective Scripting Fundamentals
  - Scopes
  - Perspective Data Types
- Object Traversal
- Scripting Transforms

Perspective Scripting Fundamentals

Though Perspective uses the same basic platform (Jython) as other scripting environments in Ignition, interfacing with some unique features in Perspective might make it feel like a new scripting experience. Here are some key details that might be important to Perspective script writers.

Scopes

Perspective does not have a "client" scope, because unlike Vision, Perspective does not have clients. All Perspective scripting is run on the Gateway, although session-specific functions (like navigation) will only affect a single session. Critically, this means that:

- Client-scoped scripting functions (like system.file, system.gui, and system.nav functions) will *not* work in Perspective.
- Other scripting functions, like system.util.getLogger(), will run in a Gateway context.

Perspective Data Types

Component properties in Perspective are structured as JSON. However, interacting with them does not require any kind of specialized knowledge. Critically, every property in Perspective is one of three types:

Type	Description	Example
Value	A value is a simple piece of data, usually a number or string. Assigning a value to a value property is just like assigning a value to an ordinary Python variable.	`self.props.text = "My Text"` `self.props.startAngle = 5`
Object	An object is structured like a Python Dictionary, holding any number of key:value pairs. If you want to pass an object to an object property, you'll need to use the Python Dictionary type. Note that a Perspective Object could contain different kinds of sub datatypes. One of its keys could map to another object, or to an array.	`motorObject = {"motorNum": 1,"motorState":"Running"}` `self.custom.myObject = {"operationNum":15,"motorObject":motorObject}`

Array	An array is structured like a Python List. Unlike an object, where each element in the data type has an associated key, in an array, each element only has a position.
	
	Note that a Perspective Array could contain different kinds of sub datatypes. Arrays can contain objects and other arrays.
		`rowObject1 = {"city":"Folsom","country":"United States","population": 77271}`
		`rowObject2 = {"city":"Helsinki","country":"Finland","population": 625591}`
		`self.props.data = [rowObject1,rowObject2]`

Object Traversal

In scripting, we can use component properties and methods to access related components, and view and session info. See Perspective Component Methods for details.

Scripting Transforms

Any property binding can make use of a Script Transform to apply any python script to the output value of the binding. For more information, see Transforms and Script Transforms.

In This Section ...

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.

On this page

...

- Object Traversal
  - Object Traversal Examples
- Built-In Methods
  - Refreshing Bindings
  - Requesting Focus
- Custom Methods

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	The following feature is new in Ignition version **8.0.3** Click here to check out the other new features Returns all of the component's children.	`self.c hild`
.getChildren()	The following feature is new in Ignition version **8.0.3** Click here to check out the other new features Functionally similar to ".children" above.	`self.getChildren()`
.parent	Calling this property will move up the component hierarchy, accessing the parent container of the	

	preceding object. Root containers will return the view, and views/sessions will return None.	`self.parent`
.getParent()	This is the same as ".parent" above.	`self.getParent()`
.getChild(string)	Method that looks for a child component of a given name. Returns None if not found. String can either be the name of a child object, or a path to an object delimited by a forward slash, allowing you to move through multiple items in the hierarchy in a single call.	`self.getChild('Label_0')self.getChild('Cont`

		ainer/Label_0')
.getSibling(string)	Returns a reference to an object in the same container that the source component is located in. Similar to calling self.parent.getChild('component').	self.getSibling('Label')
.view	Calling this from anywhere within a view will return the parent view of the object.	self.view
.page	Returns a page object associated with the page the current component is on. The following feature is new in Ignition version **8.0.4** Click here to check out the other new features .close() can be called on the page object, and can accept a message string as a parameter.	page=self.pagepageID=

		`page.props.pageId` `page.props.pagePath = page.props.path`
.session	Returns the current perspective session you are in. From this session you can get any of the existing attributes of the Session. Similar to the view.session object. > The following feature is new in Ignition version **8.0.3** > Click here to check out the other new features .close() can be called on the session object, and can accept a message string as a parameter. > The following feature is new in Ignition version **8.0.5** > Click here to check out the other new features .getPages() returns a list of page objects. .getPage(string ID) returns the page associated with the given page ID if it exists. > The following feature is new in Ignition version **8.0.8** > Click here to check out the other new features .getInfo() returns a list of session objects.	`session = self.sessions.sessionName = sess`

			ion.props.gateway.addressesProp=sessionCustom.propertyName
View Objects			
.rootContainer	Returns the root container of the View.		view.rootCo

		n t a i n er
.id	Returns the id of the View. This is a string that is similar to the path of the View. All instances of a View will have the same ID, for example: `Perspective/Views/path/to/view yields path/to/view@C`	v i e w . ic
.session	Returns the current Perspective Session you are in. From this Session you can get any of the existing attributes of the Session. This is similar to the self.session object. > The following feature is new in Ignition version **8.0.3** > Click here to check out the other new features .close() can be called on the session object, and can accept a message string as a parameter. .getPages() returns a list of page objects. .getPage(string ID) returns the page associated with the given page ID if it exists. > The following feature is new in Ignition version **8.0.8** > Click here to check out the other new features .getInfo() returns a list of session objects.	s e s s i o n = v i e w . s e s s i o n s e s N a m e = s e s s i o n . g a t e w a y . a d d r e s s s e s P r o

```
p = session.customm.propertyName
```

Object Traversal Examples

If you want to get other component properties in a view while scripting, you can use the above methods and properties to move around the View.

These examples assume you have the following structure/components in a View:

- View
  - Button 1
  - Text Field 1
  - Container 2
    - Text Field 2
  - Container 3
    - Text Field 3
    - Container 4
      - Button 4
      - Text Field 4

Scripting Example: Get component properties from a Button script

```
# this example code exists on the 'Button 1' in the above hierarchy.

# get to the view that the button is in
view = self.view

# get the text from 'Text Field 1'
text1 = self.getSibling('Text Field 1').props.text

# get the text from 'Text Field 2'
text2 = self.getSibling('Container 2').getChild('Text Field 2').props.text

# get the text from 'Text Field 3'
text3 = self.getSibling('Container 3').getChild('Text Field 3').props.text

# get the text from 'Text Field 4'. Either of these will work the same.
text4 = self.getSibling('Container 3').getChild('Container 4/Text Field 4').props.text
text4 = self.getSibling('Container 3').getChild('Container 4').getChild('Text Field 4').props.text
```

Scripting Example: Get component properties from a Button script

```
# this example code exists on the 'Button 4' in the above hierarchy.

# get to the view that the button is in
view = self.view

# get the text from 'Text Field 1'
text1 = self.parent.parent.getSibling('Text Field 1').props.text

# get the text from 'Text Field 2'
text2 = self.parent.parent.getSibling('Container 2').getChild('Text Field 2').props.text

# get the text from 'Text Field 3'
text3 = self.parent.getSibling('Text Field 3').props.text

# get the text from 'Text Field 4'
text4 = self.getSibling('Text Field 4').props.text
```

Built-In Methods

Perspective components contain several shared methods. This section details such methods. Note that 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.

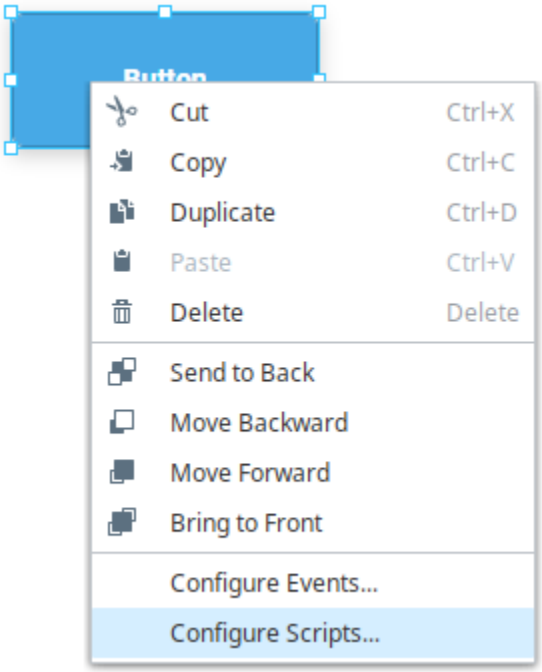
Due to the nature of focus, calling the `focus` method is only effective on components that can have focus. Input components such as the Text Field and Numeric Entry Field components can gain focus, but Display components like Labels and Images can not gain focus.

```
self.focus()
```

Custom Methods

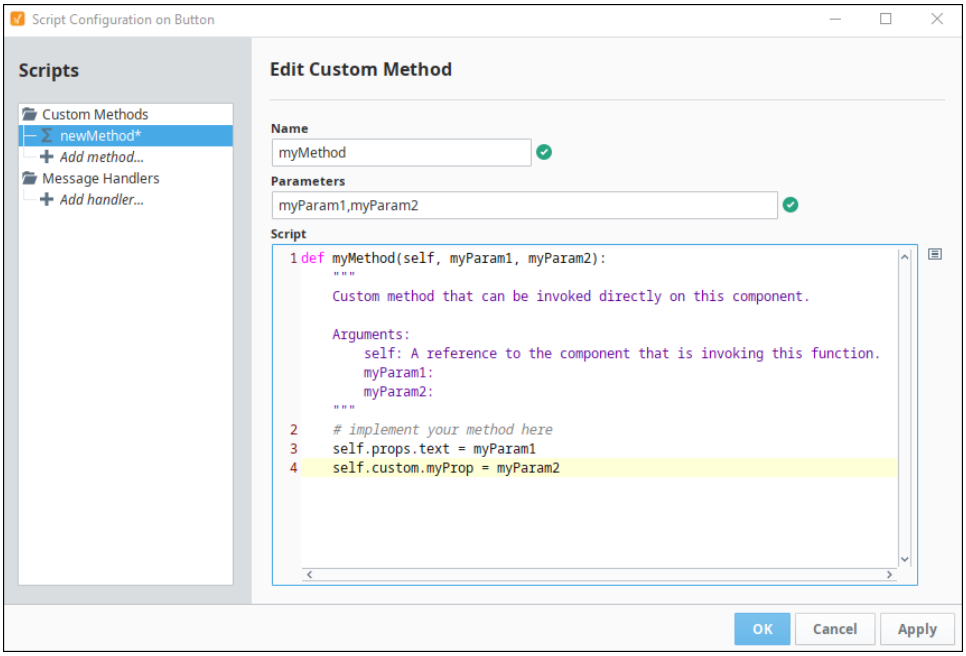
Perspective offers the option of configuring your own methods for a component. To configure a custom method:

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.
4. Enter any number of **Parameters** your method will need, separated by commas.
5. Add code to implement your method.

> 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")
```

Related Topics ...

- Component Message Handlers

Component Message Handlers

In Perspective, Component Message Handlers are the preferred way to pass parameters between components or Views. Doing this involves the system.perspective.sendMessage function. There are typically two steps involved:

1. Creating a message handler on the component that will listen for a particular call.
2. Create a script that will call the message handler.

This page will demonstrate how to prepare both steps. The goal of this example is to create a script that will cause a button to change the text property of a Label.

On this page

...

- Object Traversal
- Message Types
- Message Handler Scope
- Message Handler Example
  - Step 1 - Prepare Perspective Workspace
  - Step 3 - Send A Message
- Passing Parameters Example
  - Step 1 - Update the Button Script Action
  - Step 2 - Update the Message Handler

Object Traversal

When writing a component based script, Object Traversal should be avoided where possible. Object Traversal is the process of declaring hard-coded component paths in a script similar to how the Vision module uses component paths in scripting. While component paths exists in Perspective, they are brittle: changes to the hierarchy in the view (such as placing a component into a new container), or changing a component name will invalidate any paths defined before the change, since the component's relative location has changed.

Consider the following:

Pseudocode - Example Path

```
self.getSibling('Text Field').props.text
```

The path described above only works if there is component named "Text Field" in the same container as the component that is running this script. If the Text Field component is renamed at any point, this reference to the component will fail. Additionally, if the Text Field is placed in a different container, then the getSibling() call will no longer work. Components in other containers (in the same view) are available, but are not siblings.

Additionally, Object Traversal can't be used to reference a property in a separate view: i.e., a script in View A cannot reference something in View B.

We **strongly** suggest you utilize message handlers when a script is trying to interact with a component in another View.

Message Types

When sending a message, the Message Type field represents which Message Handler should respond. If a script sends a message with a type of "foo", then any handler listening for a Type of "foo" will execute. This means multiple components in the same window can all have Message Handlers with the same Type: e.g., if a view has a "reset" button to clear out multiple input components, then each input component can simply have a "reset" message type that clears the field, allowing a single message to trigger multiple handlers. Alternatively, if only a single handler should execute when sending the message, simply give that one handler a unique Type.

Message 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:

- session
- page
- view

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.

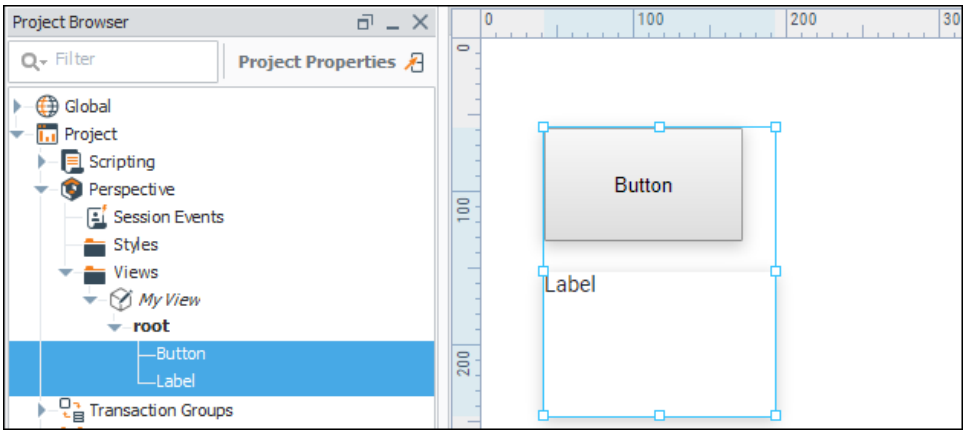
There are two ways to limit the scope of a message:

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.



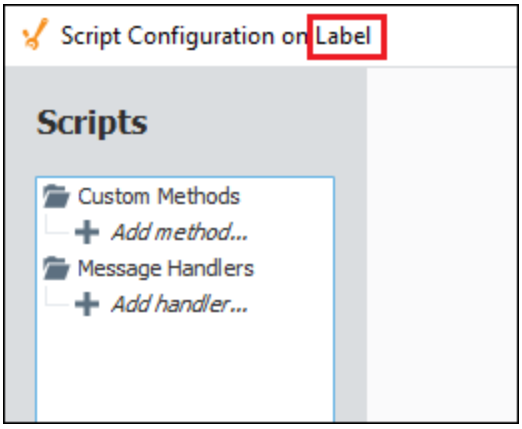
Step 2 - Create A Message Handler

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 don't care about which component broadcast the message: if something sends a message with the correct type, then the Message Handler will execute. Additionally, message can be sent across separate views, pages, or throughout the entire session.

In our example, we want the Label's text property to change when something else in the view happens (in our case, our button is pressed), so it would make sense to configure a Message Handler (listener) on the Label. This way, if we relocate the Label component in the view, the script will still work.

1. Right-click on the Label, and select **Configure Scripts**.
2. The Script Configuration window appears. Make sure the title bar on the window states that it's the **Label** component.

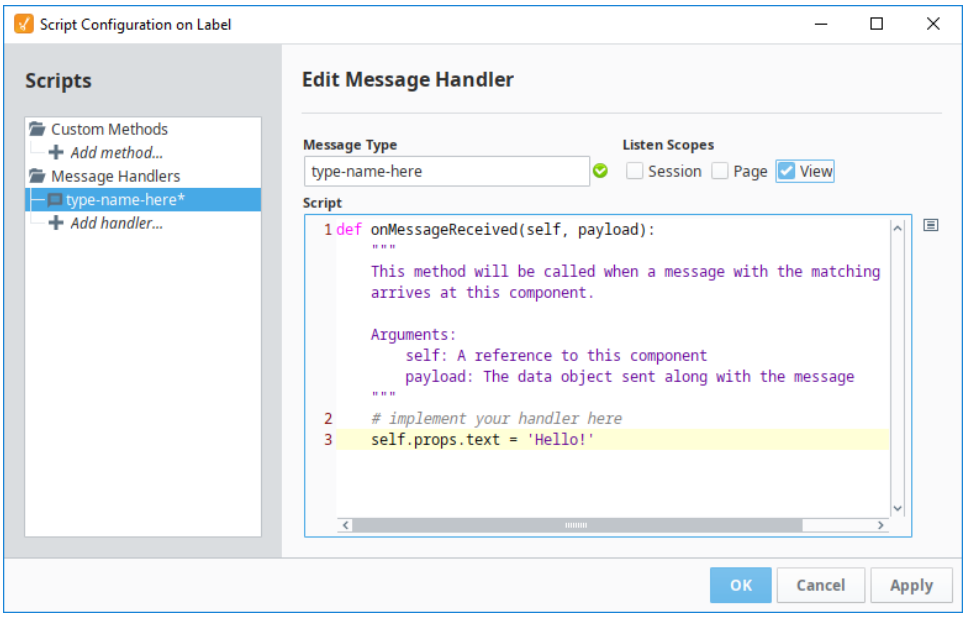


3. On the left side of the **Script Configuration** window, you will see a tree. Double-click on the **Add handler...** item.
4. A new handler named "type-name-here" will appear. Under **Message Type** type `my-handler`. Note that we're using all lowercase characters: **message handlers are case-sensitive**. The Message Type is effectively the name of the message handler. When sending a message, we will specify this message handler's type, which will cause it to respond by executing the script.
5. In our example, let's change the text on the Label to "Hello!". Under the script area, type the following script:

Python - Set the Text

```
self.props.text = 'Hello!'
```

6. This example is fairly limited to this one view. Thus, let's limit the scope so that it will only respond to messages from the same view the Label is on. Under the **Listen Scopes**, uncheck **Page**, and check **View**.
7. The window should look like the following:



8. 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

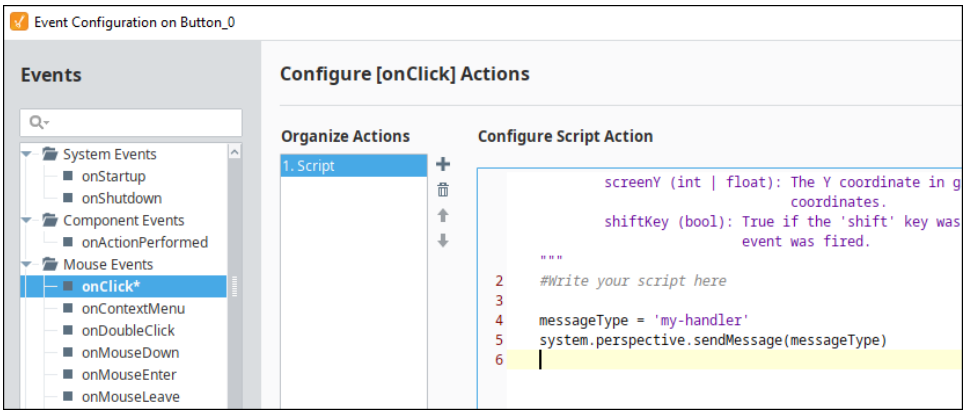
In this step, we place a script on the Button that will call our Message Handler.

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, scope='view')
```

The window should look like the following. Note the tab indentation on lines 4 and 5:



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.

> **Common Issue**
>
> In this simple example, the major issue you may run into is the Message Type. Recall from #4 in Step 1 of the example, that Message Type is case-sensitive, so make sure the script on the Button is correctly referencing the message type, and try again.

Passing Parameters Example

Let's make the previous example more complicated and pass some values with the message. The example above can be modified to determine the timestamp. When passing parameters in a script, the most direct approach is to include any parameters along with the message. Message Handlers have a built-in argument called a **payload**, which is used to transfer values to the handler. The payload is simply a Python Dictionary, so please see the Dictionaries page for more information.

Alternatively, we could create a session property to hold the value, and have the label reference the session property. However this would require that we either create a new property to hold the value or overwrite the value of another property. Thus, our next example will demonstrate how to utilize the payload.

> **Avoid Storing Values in Tags**
>
> Since Tag values are shared by all Perspective sessions, you may not want to write the parameters in Tags. Doing so would result in each session instance potentially trying to overwrite the same value.

Step 1 - Update the Button Script Action

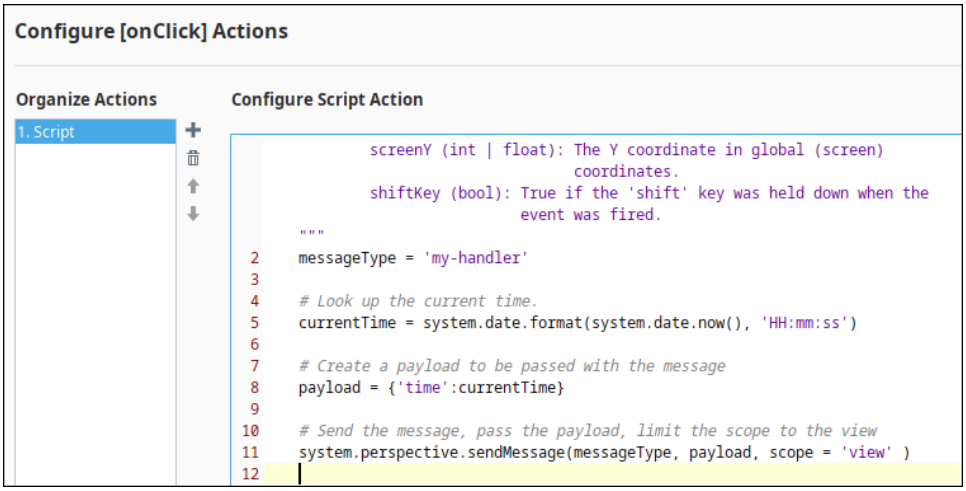
1. Right-click on the Button component and select **Configure events...**
2. Replace the code with the following:

Python - Check the Time, Send a Message

```
messageType = 'my-handler'

# Look up the current time.
currentTime = system.date.format(system.date.now(), 'HH:mm:ss')

# Create a payload to be passed with the message
payload = {'time':currentTime}
```

```
# Send the message, pass the payload, limit the scope to the view
system.perspective.sendMessage(messageType, payload, scope = 'view' )
```

3. Make sure the code is properly indented. It should look like the following:



> ⚠ On line 8, we're creating a dictionary, creating a key called "time", and storing the current time with the "time" key. When our handler receives the payload, it can retrieve the value we passed by referencing the "time" key.

4. Once finished, click the **OK** button.

Step 2 - Update the Message Handler

Now that we're including a payload with the message, we need to modify our handler so that it will extract the time from the payload.

1. Right-click on the Label component, and select **Configure scripts...**
2. Replace the original code with the following:

```
# Access the time by referencing the 'time' key
self.props.text = payload['time']
```

3. Click **OK**.
4. To test it, enable **Preview mode**, and click the Button component. You will see the current time populate in the Label.

Related Topics ...

- Scripting
- Dictionaries

Perspective Property Change Scripts

With Perspective, individual component properties can have a property change script. When a change script is set up on a property, it will run when the property changes its value. Multiple different properties on the same component can each have different scripts configured. In Perspective, you can put a property change script on any component property.

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

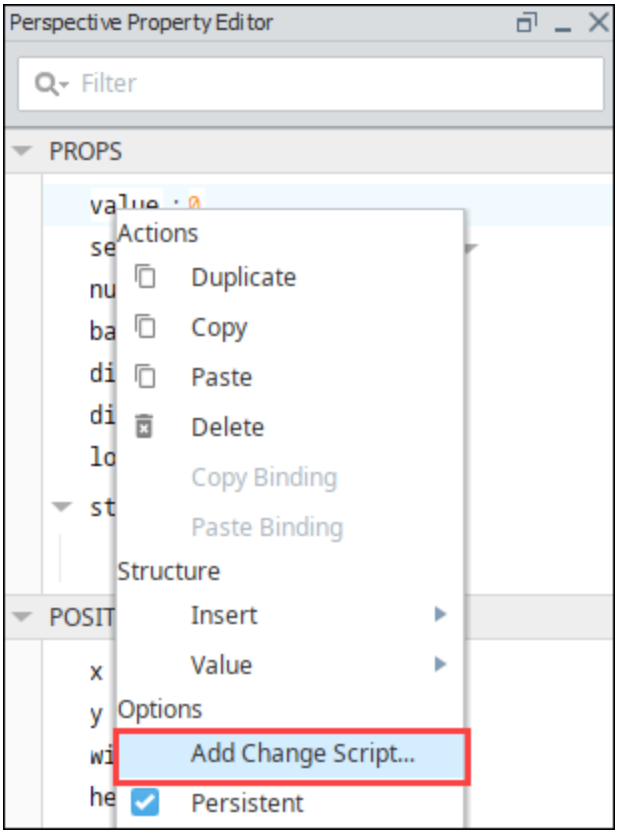
On this page

...

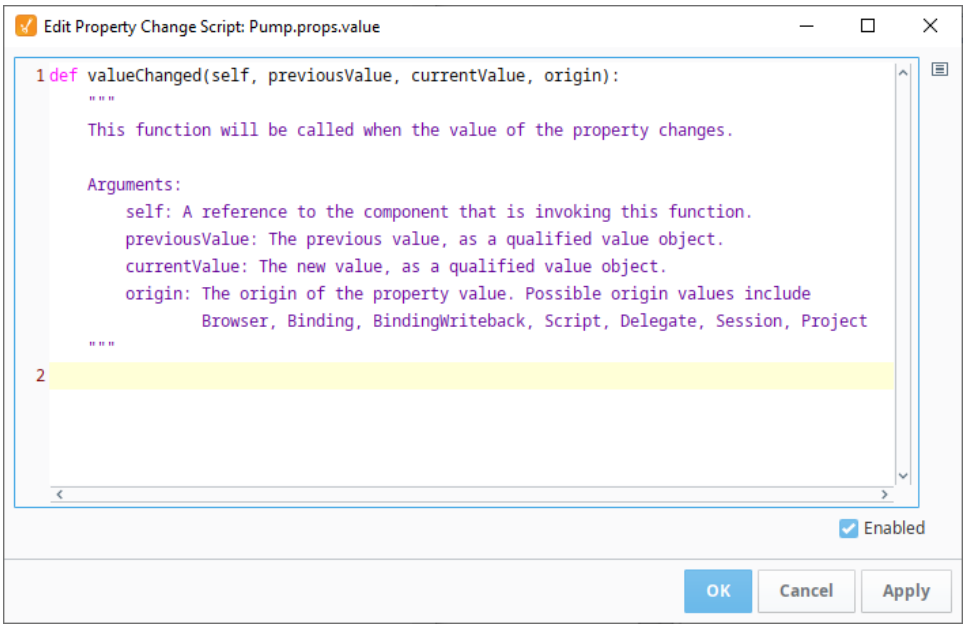
- Add a Property Change Script
- Property Change Arguments
- Change Script Example

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**.

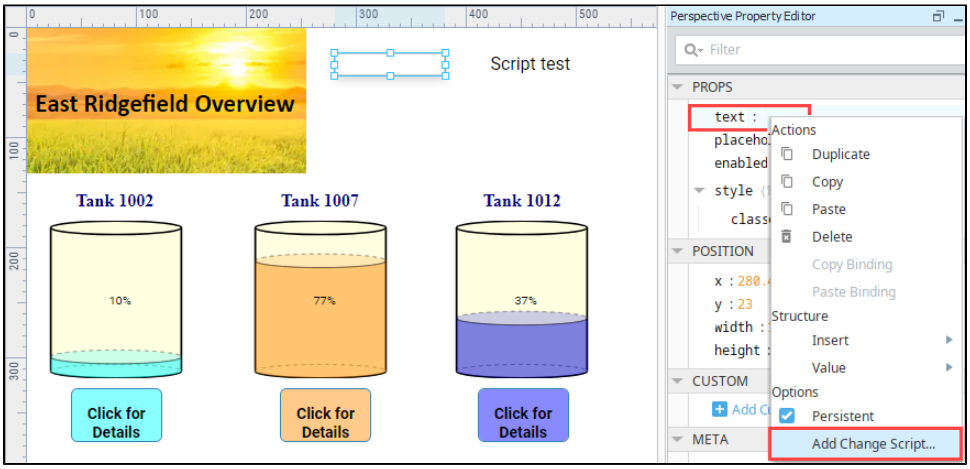
Property Change Arguments

Argument	Description
self	A reference to the component that has the property in question. If the property change script is on a session property, the

<table>
<tr><td></td><td>session object will be passed.</td></tr>
<tr><td>previousvalue</td><td>The previous value, as a qualified value object. Qualified value objects have a value, quality, and timestamp.</td></tr>
<tr><td>currentValue</td><td>The new value, as a qualified value object. Qualified value objects have a value, quality, and timestamp.</td></tr>
<tr><td>origin</td><td>The origin of the property value, as a unicode string. The origin parameter will take on one of six types depending on how the property value is being updated:
<table>
<tr><th>Name</th><th>Description</th><th>Example</th></tr>
<tr><td>Browser</td><td>Used when the change comes from the Browser interface.</td><td>The user changes the text property on a text field by typing a word into the field.</td></tr>
<tr><td>Binding</td><td>Used when the change comes from a binding (or transform) generating a new value.</td><td>A Tag changes value, and a property with a binding to that tag is updated.</td></tr>
<tr><td>BindingWriteback</td><td>Used when the change comes from a bidirectional binding writing back to its source.</td><td>ComponentB's 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.</td></tr>
<tr><td>Script</td><td>Used when the change comes from a script.</td><td>A user presses a button, and a script on the button assigns a new value to a custom property.</td></tr>
<tr><td>Delegate</td><td>Used when a change to a property comes from something intrinsic to the component's design.</td><td>A complex component that automatically fills itself with data, like the alarm status table component.</td></tr>
<tr><td>Session</td><td>Used when the session itself causes the property change.</td><td>A change in user privileges causes access to be revoked, resulting in a change in the auth session property.</td></tr>
<tr><td>Project</td><td>Used when the default property value is changed in the designer and saved.</td><td>A session property was set to a value of "A." The default value of that property was then changed in the designer to "B" and saved. The value for that property changes from "A" to "B" in the running sessions.</td></tr>
</table>
</td></tr>
</table>

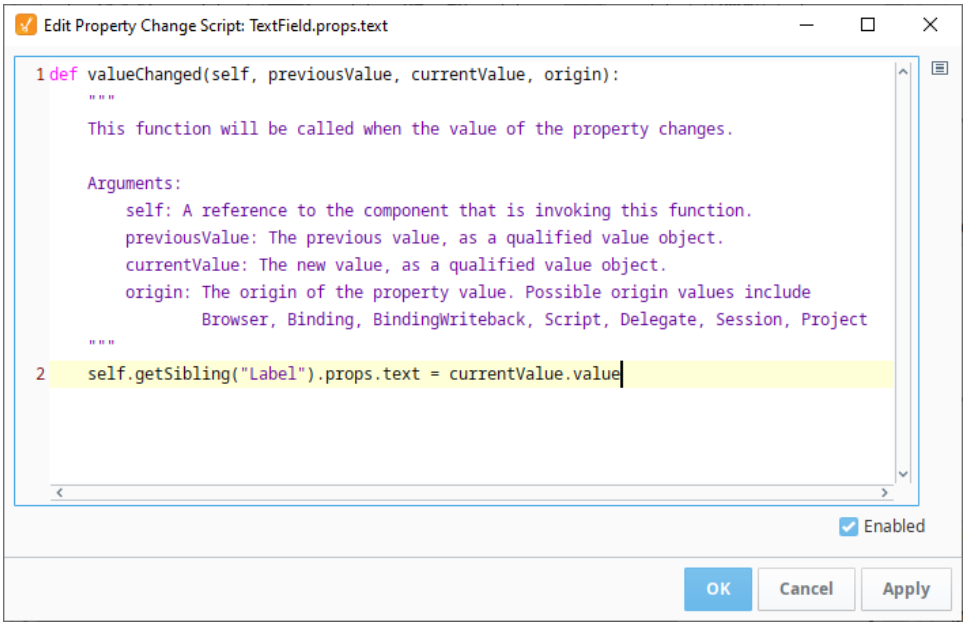
Change Script Example

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. Click on **Add Change Script**.



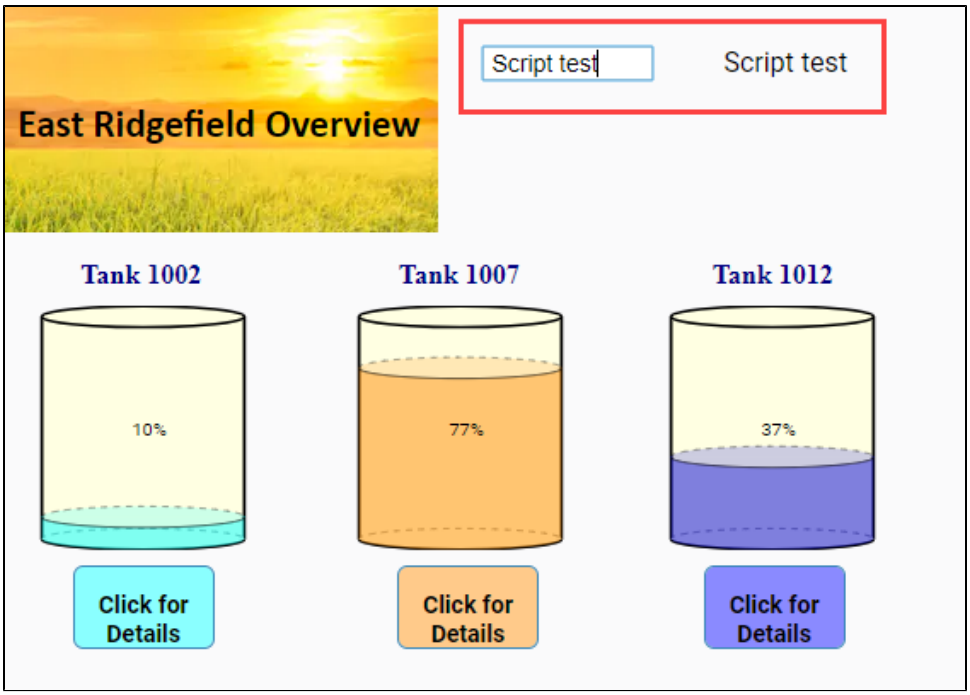
3. The Edit Property Change Script screen is displayed. Enter the following script, which will write the current value of the Text Field component to the Label component.

```
self.getSibling("Label").props.text = currentValue.value
```



4. Click **OK** to commit the script. You'll see that the Property Editor now shows a Change Script icon next to the **text** property.

5. **Save** your project.
6. 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.



Perspective Session Event Scripts

Perspective offers a collection of *Session Events* designed to allow the Gateway to track and interact with the session at critical moments. Specifically, they are scripts that run in the Gateway when a session starts up, shuts down, or runs a Native App Action.

There are five configurable session events in Perspective:

- Startup
- Shutdown
- Barcode Scanned
- Bluetooth Device Data Received (new in 8.0.5)
- NFC NDEF Scanned
- Accelerometer Data Received
- Message (new in 8.0.7)

On this page

...

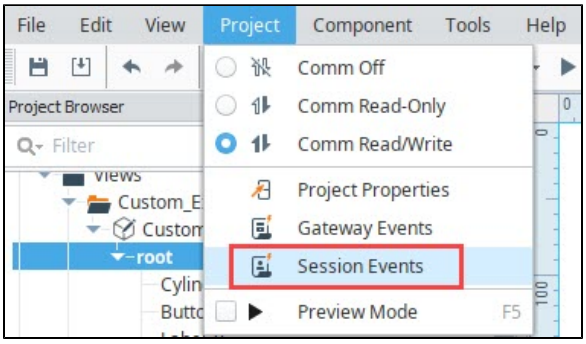
- Configuring Session Events
- Startup and Shutdown Event Scripts
  - Startup and Shutdown Example
- Native App Event Scripts
- Barcode Scanned
  - Barcode Scanned Example
- Bluetooth Data Received
- Accelerometer Data Received
  - Accelerometer Data Received Example
- NFC Ndef Scanned
  - NFC Ndef Scanned Example
- Message

Although they are designed to handle *Session Events*, the scripts that you write will be run in a **Gateway** scope, not a Session scope.

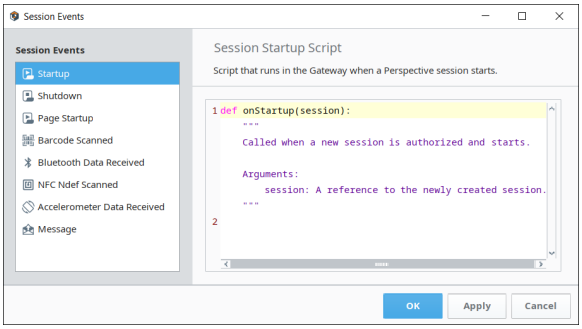
Configuring Session Events

To start working on a session event script:

1. In the Project Browser, double-click on the **Session Events** section:



2. The **Session Events** dialog will appear:



3. Configure a script by selecting one of the events on the left-hand side.

Startup and Shutdown Event Scripts

Startup and **Shutdown** events run, naturally, whenever a session starts or ends. In each case, the gateway will have access to the **session** object associated with the section, complete with all session properties.

When designing a startup or shutdown event script:

- *Custom* session properties can be used to pass any additional information to the gateway, or, in the case of a startup script, to pass information to the newly opening session. You can configure custom session properties from the Page Configuration dialog, by clicking on the **Settings** icon in the Designer.
- A shutdown event script will run *specifically* when a session is ending. This happens specifically when any of the following events occur:
  - The session closes due to a timeout. Session timeout is configurable in the **Perspective > General** section of **Project Properties** in the Designer
  - The user is no longer authorized to run the session.
  - The redundancy system determines that the Gateway is inactive.
  - The licensing system no longer permits the user to run the session.
  - The project is no longer runnable.
  - The project is deleted.

⚠ Closing the browser tab with the session will *not* immediately close the session; the session must first time out.

Startup and Shutdown Example

This example will record the session start and end time to the database.

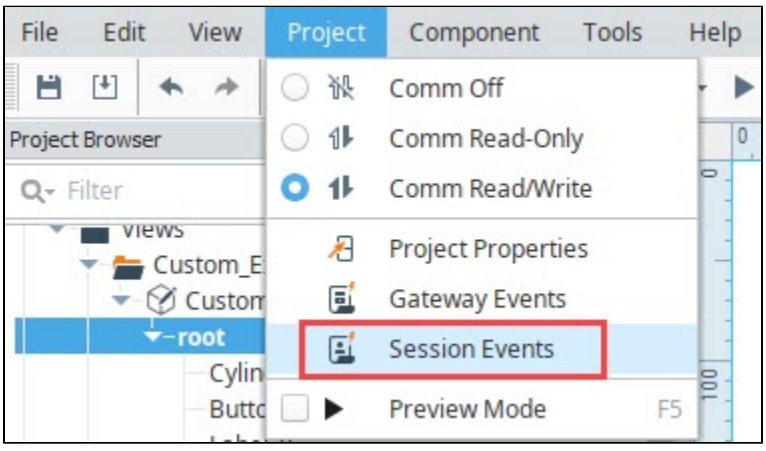
1. For this example, we need to set up a few queries that we can use to write our data to the database.
2. Make a new named query called **Startup Query**.
   a. Set the Query Type to **Update Query**.
   b. Set up a single Value type parameter with a name of **SessionID** and a datatype of string.
   c. Add the query:

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

3. Make a second new named query called **Shutdown Query**.
   a. Set the Query Type to **Update Query**.
   b. Set up a single Value type parameter with a name of **SessionID** and a datatype of string.
   c. Add the query:

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

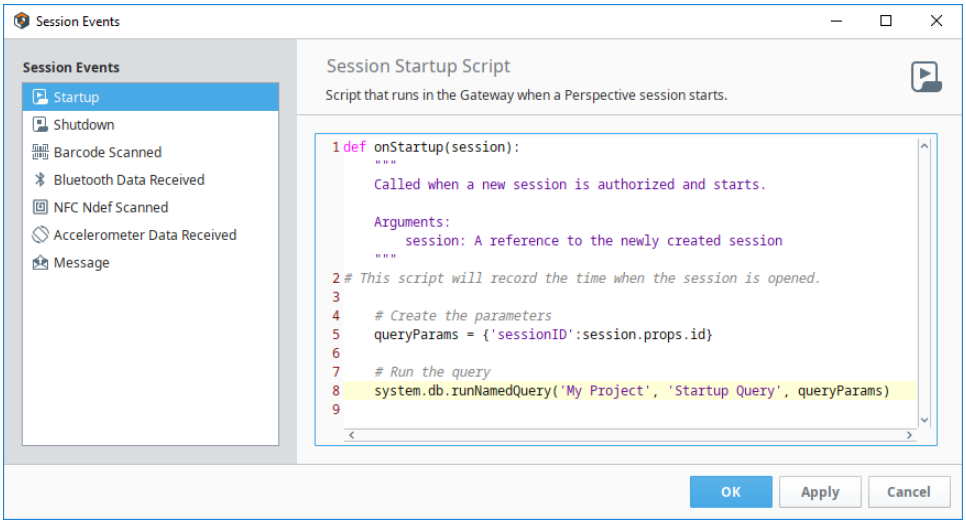
4. Next, we need to add Session Events so that Perspective knows to run those queries on startup and shutdown.
5. Under the **Project** tab, select **Session Events**.



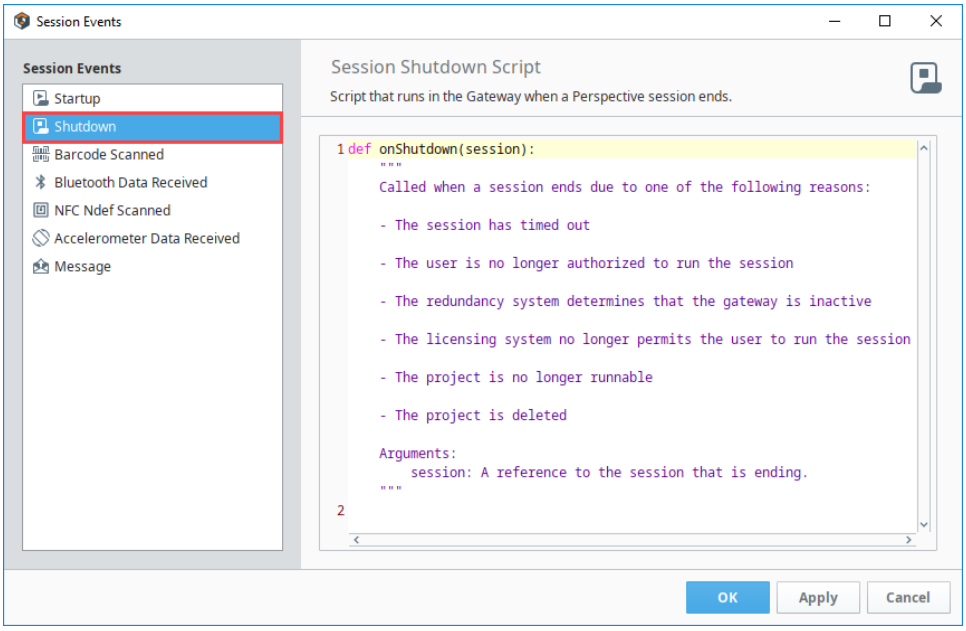
6. On the **Session Events** screen, click the **Startup** icon.
7. Add the following script to the page:

```
# This script will record the time when the session is opened.

        # Create the parameters
```

```
        queryParams = {'sessionID':session.props.id}

        # Run the query
        system.db.runNamedQuery('My Project', 'Startup Query', queryParams)
```



8. Click **Apply** and then click the **Shutdown**  icon.



9. Add the following script to the page:

```
# This script will record the time when the session times out. Note that after the session is
closed,
# the session won't time out until the time out period is reached..

        # Create the parameters
        queryParams = {'sessionID':session.props.id}

        # Run the query
        system.db.runNamedQuery('My Project', 'Shutdown Query', queryParams)
```

10. Click **OK**.
11. Save your project.

Test the Example

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

1. You should see a new entry in the database table with the time the time the session was started as well as the session id.
2. After the session times out (such as after it is closed) you should see the original entry get updated to include the new shutdown time.

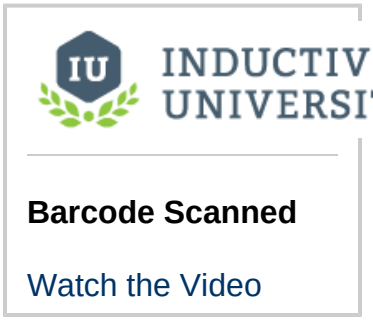
Native App Event Scripts

When the Perspective App is running on a mobile device, it enables users to use tools available on the device, such as GPS location data, the camera, or the accelerometer. The remaining session events are designed specifically to handle the three Native App Actions.

Barcode Scanned

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

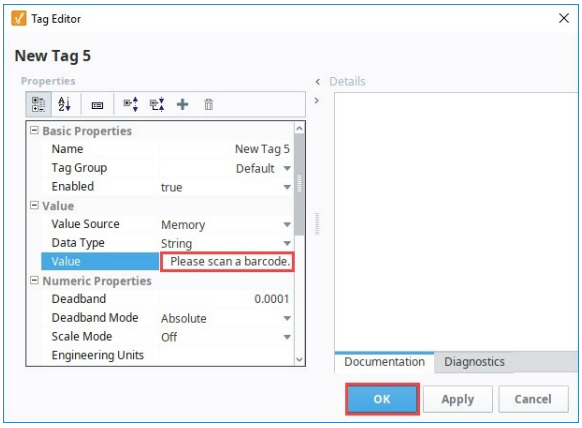
Arguments	Description
session	An object that references the project session that called the Barcode Scanned event. Use this to identify the specific session that scanned the barcode.
data	The data returned from the barcode scan. Access the underlying barcode data using: `data.text`
context	The user defined context object that can be defined on the action.



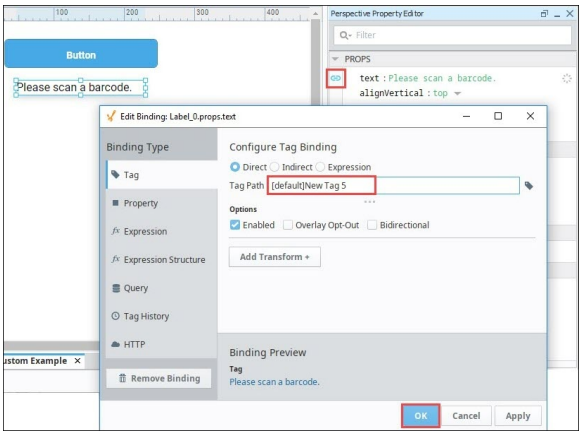
Barcode Scanned Example

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

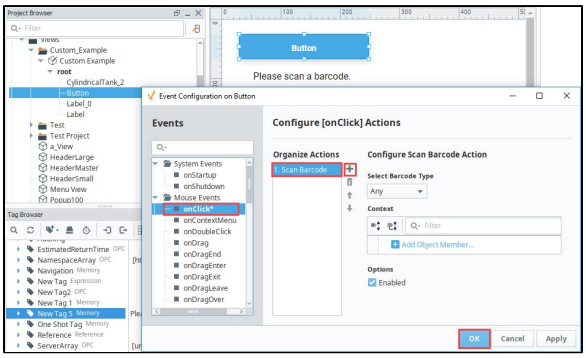
1. For this example, drag a Button component and a Label component onto a view.
2. Create a new memory Tag with a data type of String. Set the value to "Please scan a barcode."



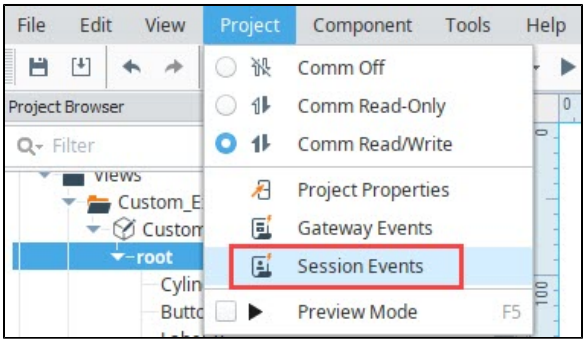
3. Bind the text of the label to the new tag.



4. Next right-click on the Button component and choose **Configure Events**.
5. On the Event Configuration screen, select **Mouse Events > onClick**.
6. Click the **Add** ✚ icon and select **Scan Barcode** action.



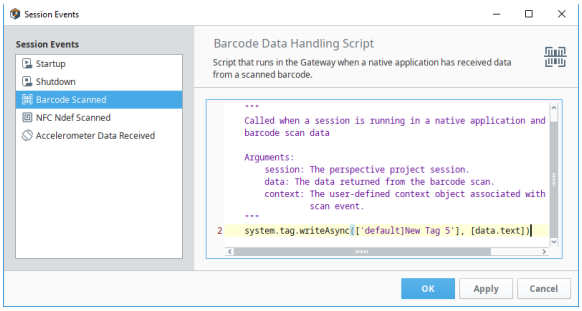
7. Click **OK**. Next we need to set up a Session Event so that Perspective knows how to interpret the scanned barcode data.
8. Under the **Project** tab, select **Session Events**.



9. On the **Session Events** screen, select the **Barcode Scanned** icon.
10. Add the following script to the page:

```
system.tag.writeAsync(['[default]New Tag 5'], [data.text])
```

> ⚠ We used the tag created for this example, New Tag 5. You can enter your own Tag name if different.



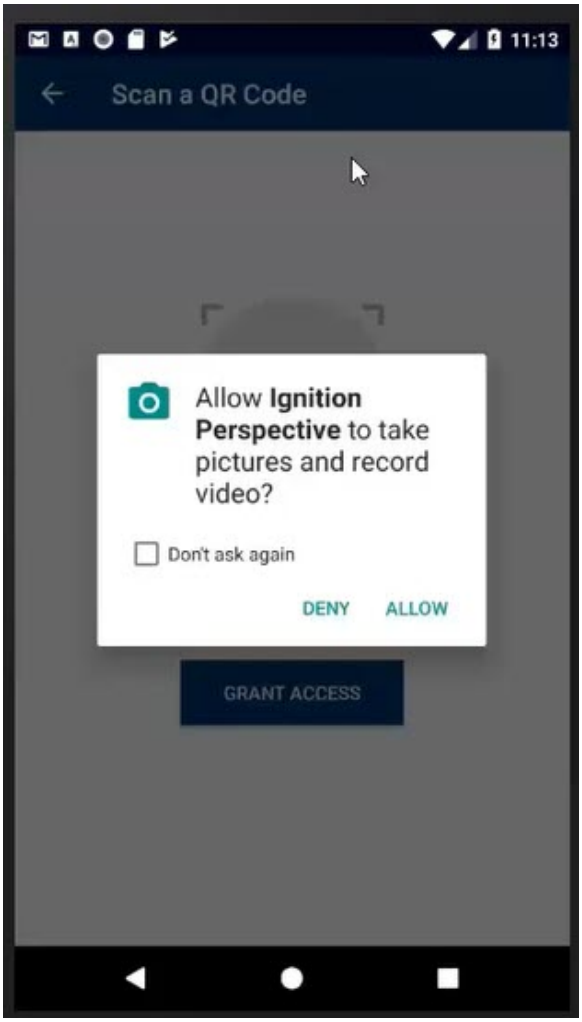
11. Click **OK** and save your project.

Test the Example

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

1. Click the Scan Barcode button.

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.


11:13
Scan a QR Code
Point Camera at QR Code

5. Ignition scans the barcode. Once it recognizes the bar code, the script will run and the text is written to the Tag. The label then shows the new Tag value.

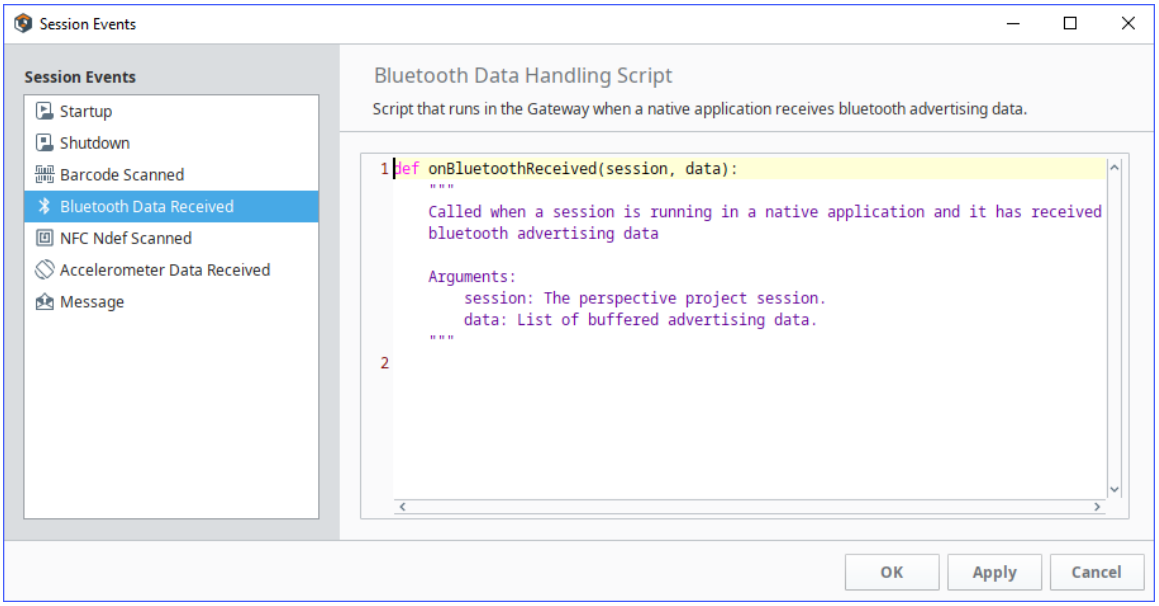
The following feature is new in Ignition version **8.0.5**
Click here to check out the other new features

Bluetooth Data Received

The Bluetooth Data Received event is only used when a session is running in a native application and it has received bluetooth advertising data. This session event script sends Bluetooth advertising data to Perspective. It supports iBeacon and Eddystone formats.

- Eddystone will work on iOS and Android
- iBeacon on iOS requires user to specify the specific region (iBeacon UUID) located on session props bluetooth.config. iBeaconRegion.

⚠ Bluetooth "Advertising Data" is the name of the communication data according to the Bluetooth spec. There is no connection to advertising as an industry.



Arguments	Description
session	An object that references the Project Session
data	List of buffered advertising data. The data comes in as a data object, which has various parts. The following is example output.


```
{
	"values":
		[
			{
				"rssi":-48,
				"timestamp":1570147485167,
				"manufacturerData":{
					"companyId":6,
					"dataBase64Encoded":"AQkgAl_edccdnHClMvecMCiM--
aAkLHAPibd"
				}
			},
			{
				"rssi":-48,
				"timestamp":1570147486012,
				"serviceUUIDs":[
					"FEAA"
					],
				"serviceData":{
					"uuid":"FEAA",
					"dataBase64Encoded":"AOyqqqqqqqqqqqqAAAAAAAA"
					},
				"eddystoneUID":{
					"txPower":-20,
					"namespaceID":"AAAAAAAAAAAAAAAAAAAA",
					"instanceID":"000000000000"
					}
			},
			{
				"rssi":-54,
				"timestamp":1570147485919,
				"manufacturerData":{
					"companyId":76,
					"dataBase64Encoded":"
AhV88isfQjVLY4VnXXfYqpTyAAAAAL8="
					},
				"iBeacon":{
					"uuid":"7CF22B1F-4235-4B63-8567-5D77D8AA94F2",
					"major":0,
					"minor":0,
					"txPower":-65
					}
```

```
                        },
                        {
                                "rssi":-54,
                                "timestamp":1570147485987,
                                "manufacturerData":{
                                        "companyId":65535,
                                        "dataBase64Encoded":"
vqwSNFZ4EjQSNBI0EjRWeJASAAAAAOwA"
                                        },
                                "AltBeacon":{
                                        "manufacturerId":65535,
                                        "uuid":"12345678-1234-1234-1234-123456789012",
                                        "instance":"00000000",
                                        "txPower":-20,
                                        "manufacturerReserved":"00"
                                        }
                        }
        ]
}
```

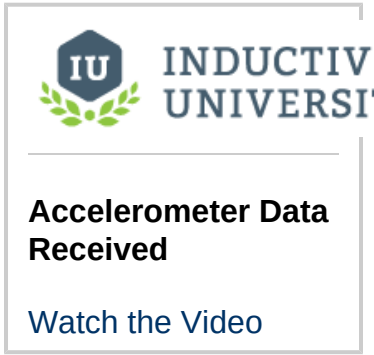

Accelerometer Data Received

The Accelerometer Data Received event is only used when data is coming in from a batched Accelerometer Action.

Arguments	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:
	`logger = system.util.logger('accelerometer')`
	`for row in data.values.data:`
	`  logger.info('X:' + str(row['x']))`
	`  logger.info('Y:' + str(row['y']))`
	`  logger.info('Z:' + str(row['z']))`

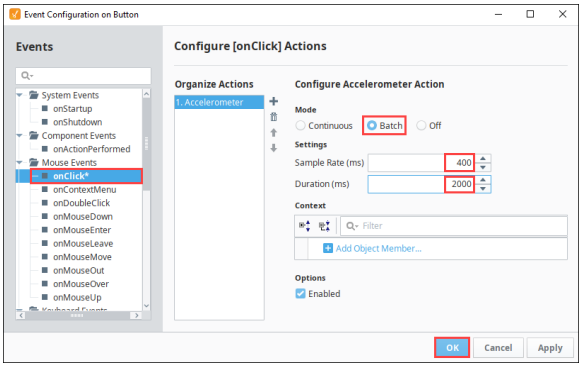
context	The user defined context object that can be defined on the action.



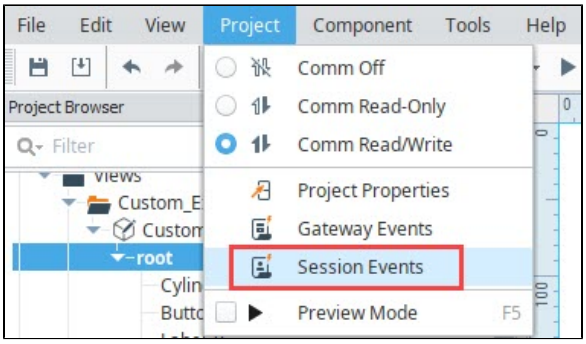
Accelerometer Data Received Example

This example will write accelerometer data to the Gateway logs.

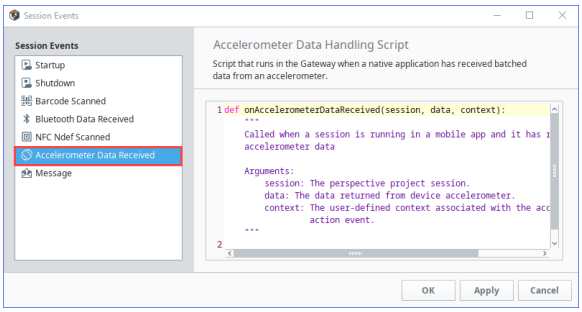
1. For this example, put a Button component onto a view.
2. Next right-click on the Button component and choose **Configure Events**.
3. On the Event Configuration screen, select **Mouse Events > onClick**.
4. Click the **Add** ✚ icon and select **Accelerometer** action.
    a. Select **Batch** mode.
    b. Set a **Sample Rate** of 400.
    c. Set a **Duration** of 2000.



5. Click **OK**. Next we need to set up a Session Event so that Perspective knows how to interpret the accelerometer data.
6. Under the **Project** tab, select **Session Events**.



7. On the **Session Events** screen, click the **Accelerometer Data Received**  icon.



8. Add the following script to the page:

```
# This script will take the accelerometer data and print it to
the Gateway logs.

# Create the logger.
logger = system.util.logger('accelerometer')

# Loop through the list of batched events and pull out the x,
y, and z values to print to the Gateway logs.
for row in data.values.data:
        logger.info('X:' + str(row['x']) + ', Y:' + str(row
['y']) + ', Z:' + str(row['z']))
```

9. Click **OK**.
10. Save your project.

Test the Example

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

1. Click the Accelerometer button.

2. After clicking the button, the script will record accelerometer data for the next two seconds, so try moving the phone around.
3. After the two seconds, you should see the logged information appear in the Gateway logs.

NFC Ndef Scanned

The NFC Ndef Scanned event is used when the NFC Action is used and the mobile device scans an NFC Tag.

Arguments	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.

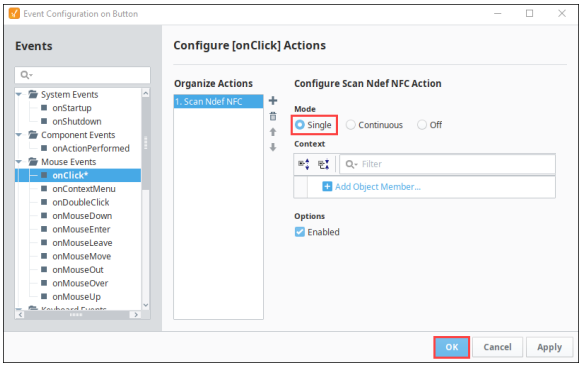
data	The data returned from the NFC Tag scan. The data object is a list which can contain multiple records from a single NFC tag. Access the underlying NFC data using:
	`logger = system.util.logger('NFC')`
	`for row in data:`
	`  logger.info('Type:' + str(row['type']))`
	`  logger.info('Type Name Format:' + str(row['typeNameFormat']))`
	`  logger.info('Payload:' + str(row['payload']))`
	`  logger.info('String:' + str(row['string']))`
	`  logger.info('Bytes:' + str(row['bytes']))`

context	The user defined context object associated with the NFC scan event.

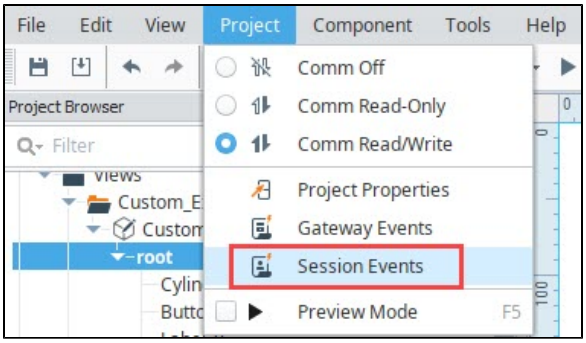
NFC Ndef Scanned Example

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

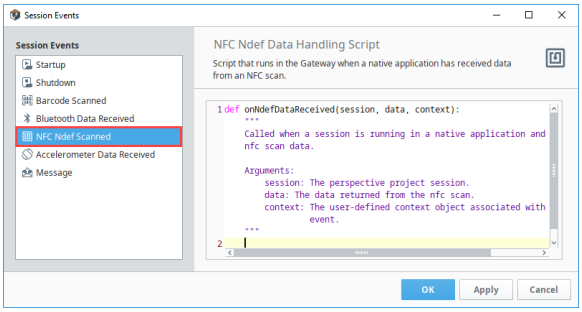
1. For this example, put a Button component onto a view.
2. Next right-click on the Button component and choose **Configure Events**.
3. On the Event Configuration screen, select **Mouse Events > onClick**.
4. Click the **Add** ✚ icon and select **Scan Ndef NFC** action.
5. Select **Single** mode, then click **OK**.



6. Next we need to set up a Session Event so that Perspective knows how to interpret the NFC data. Under the **Project** tab, select **Session Events**.



7. On the **Session Events** screen, click the **NFC Ndef Scanned** icon.



8. Add the following script to the page:

```
# This script will take the NFC data and print it to the
Gateway logs.

# Create the logger.
logger = system.util.logger('NFC')

# Loop through the list of records stored in the NFC tag and
pull out the type, type name format, payload, string data, and
raw byte data from each record and print it to the Gateway
logs.
for row in data:
        logger.info('Type:' + str(row['type']) + ', Type Name
Format:' + str(row['typeNameFormat']) + ', Payload:' + str(row
['payload']) + ', String:' + str(row['string']) + ', Bytes:' +
str(row['bytes']))
```

9. Click **OK**.
10. Save your project.

Test the Example

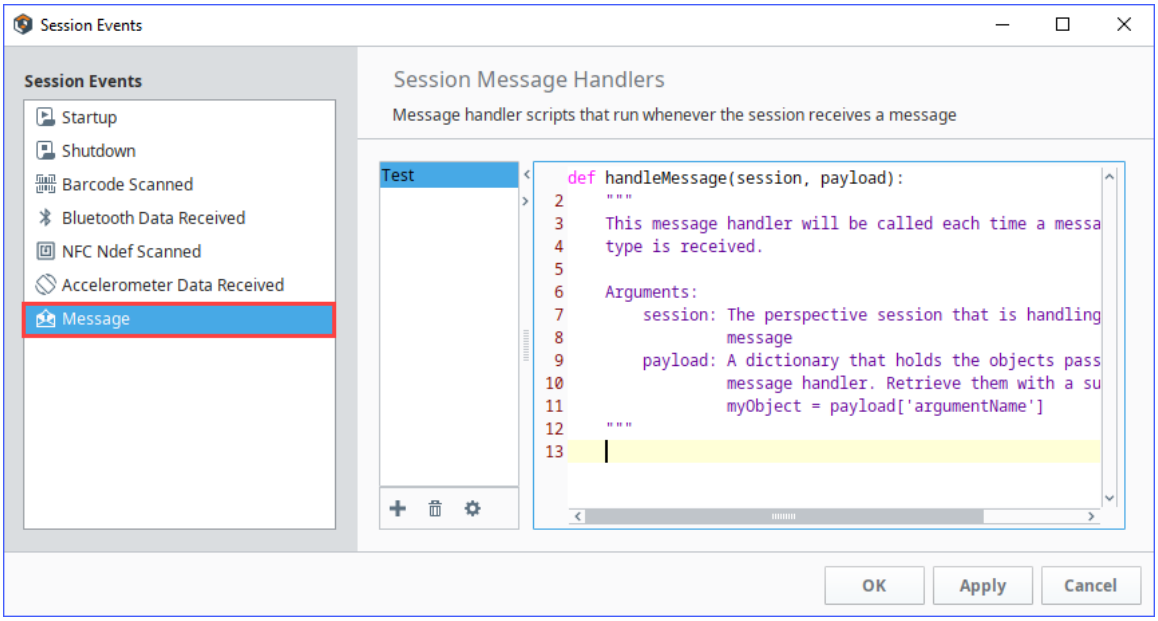
To test the example, open the Perspective App on your mobile device and load the project.

1. Click the NFC button.
2. After clicking the button, the script will pass the next NFC tag scanned to the script to be handled.
3. After scanning an NFC tag, you should see the logged information appear in the Gateway logs.

The following feature is new in Ignition version **8.0.7**
Click here to check out the other new features

Message

The Message Handler scripts will run whenever the session receives a message from system.util.sendMessage or system.util.sendRequest. Note that these types of message handlers are different than component-based message handlers, which are accessed with system.perspective.sendMessage. Session Message Handlers can not be called by system.perspective.sendMessage.



Arguments	Description
session	The Perspective Session that is handling this message.
payload	A dictionary that holds the objects passed to this message handler. Retrieve them with a subscript, e.g., `myObject = payload['argumentName']`

Editor notes are only visible to logged in users
Need tech person to help with an example

Message Example

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.

Permissions can also be set at the Project level in the Designer. This restricts actions such as publishing, viewing, saving, deleting, and editing of project resources to users who have sufficient s ecurity levels to do so.

Once you have an IdP setup as well as Security Levels, Security Level Rules, and User Grants ther e are additional ways to control security for the following:

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

On this page

...

- Perspective Sessions Security
- Perspective Views Security
- Event Actions on Perspective Components

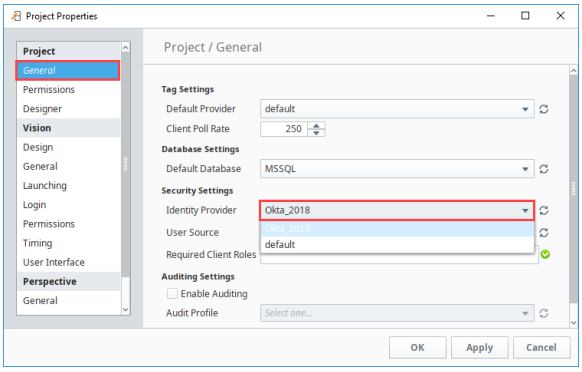
Perspective Sessions Security

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. Select Project > General.
2. In the Identity Provider field, use the dropdown to select the IdP you want to use or to select the default user source.

The following feature is new in Ignition version **8.0.6**
Click here to check out the other new features

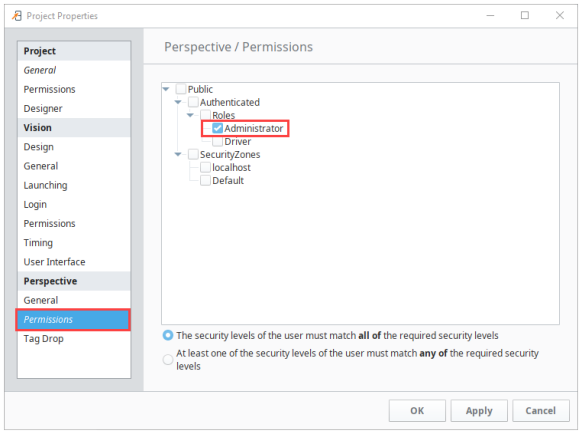
As of release 8.0.6, the Identity Provider setting was moved to this location instead of the Project Properties > Perspective General.



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

Requiring Authentication

Watch the Video



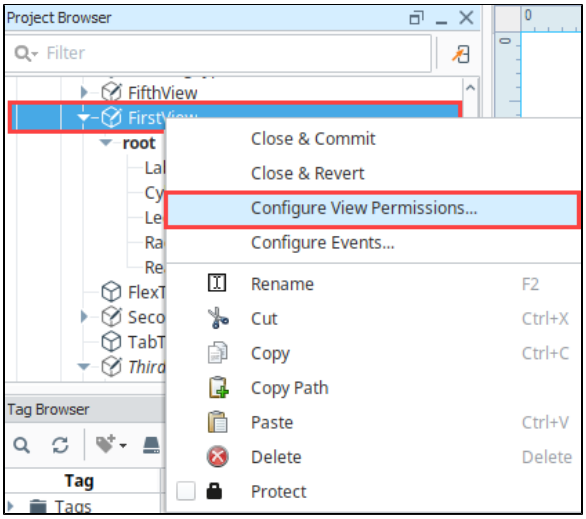
6. Click **OK** to save all of the Project Properties changes.

Perspective Views Security

You can set the security for an individual View in Perspective. When you select the security levels, you are granting any user with that access to the Perspective Session for that Project.

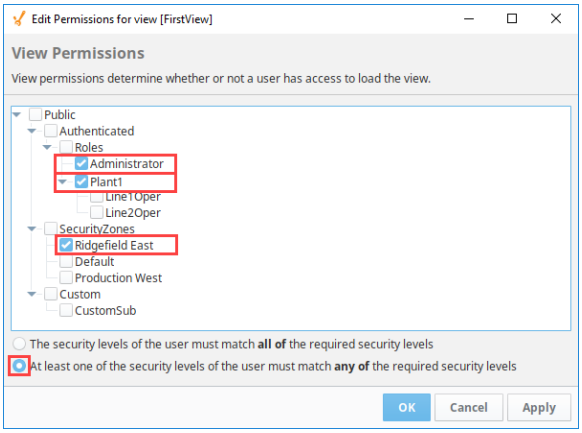
> ⚠ Note that you must have the IdP selected in Project Properties > Project General.

1. In the Project Browser, right click on the view and select **Configure View Permissions...**



2. On the Edit Permissions screen, click the check box next to the security levels that will be able to access this View.
3. Next, click the check box next to the Security Zones that will be able to access this View.
4. Finally, choose one of the radio buttons at the bottom of the screen to indicate whether the user much match **all** of the required security levels you've checked or **any** of them.

   In the example below, a user must have either the Administrator security level or Plant1 security level, or be in Ridgefield East to access this View.



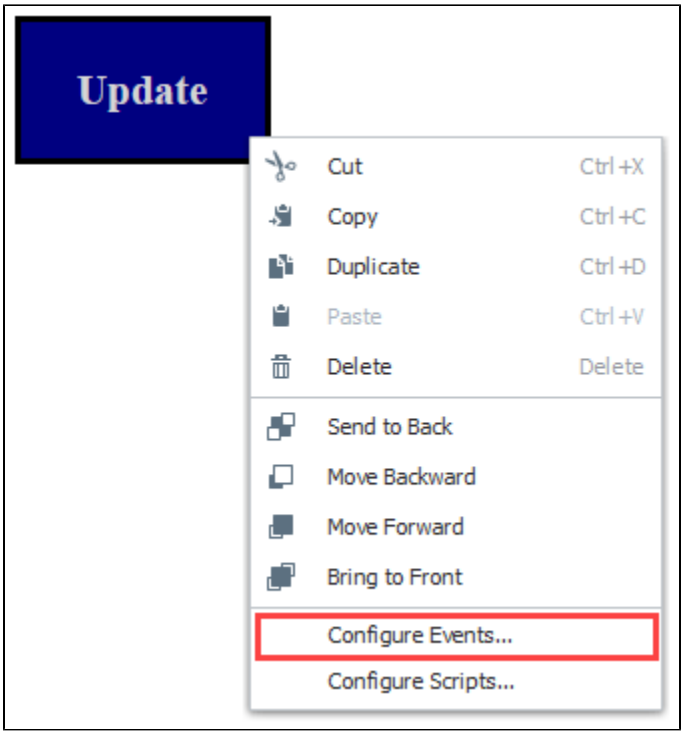
5. Click **OK** to save the permissions for this View.

Event Actions on Perspective Components

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

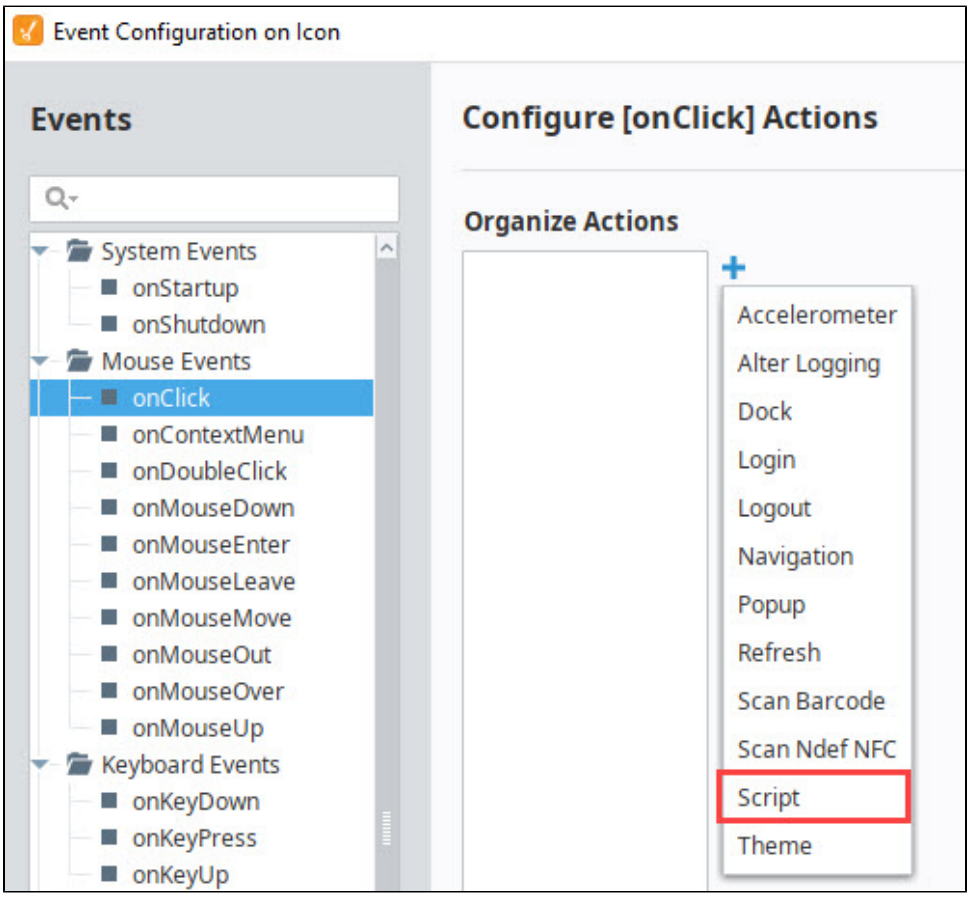
> ⚠ Note that you must have the IdP selected in Project Properties > Project General.

1. To add security to an event on a component, right click on the component then choose **Configure Events...**

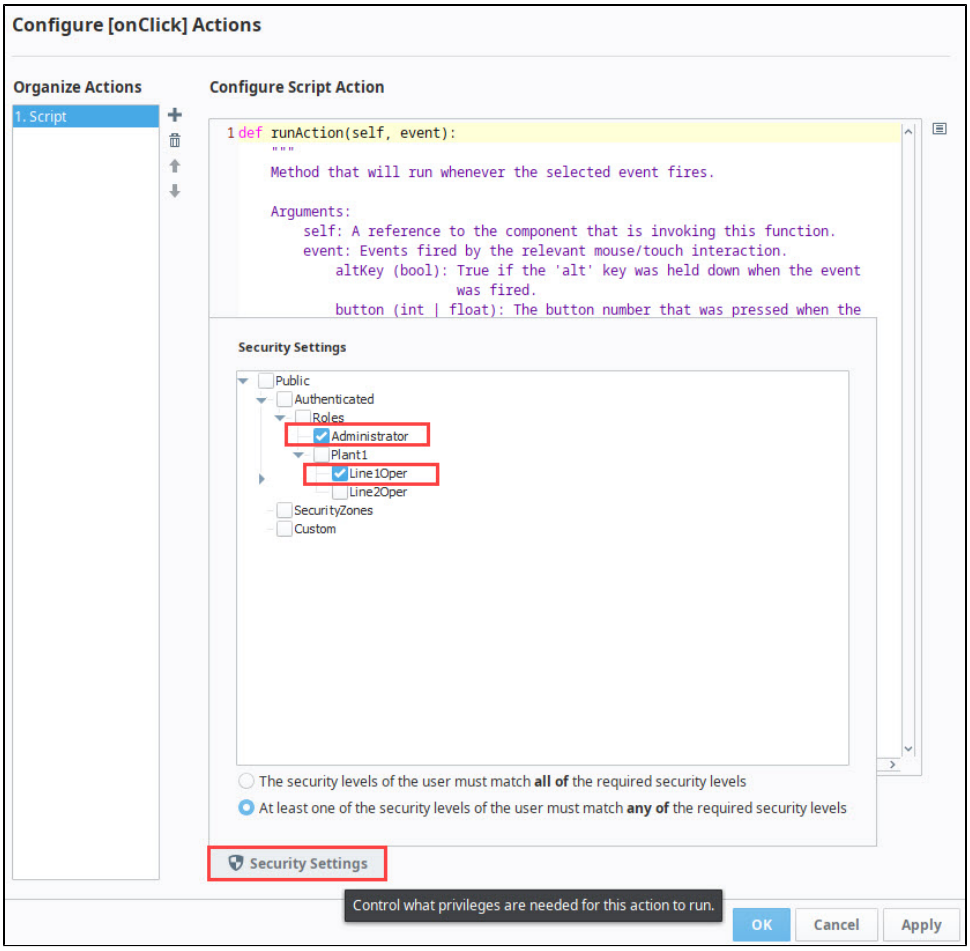


2. The Events Configuration screen is displayed. Many different types of events can be set for a component. For this example, choose **Mouse Events** > **onClick**.

3. Under **Organize Actions**, click the Add ✚ icon, then select **Script** from the list.



4. Click the **Security Settings** icon near the bottom of the screen.
5. Click the check box next to the security levels you want to grant access. In the example, anyone with Administrator or Line1Oper security levels will have permission to run the script associated with the **onClick** event on this button.



6. Click the **Security Settings** icon to close the window, then click **OK**.

Alarming in Perspective

The following feature is new in Ignition version **8.0.3**
Click here to check out the other new features

Alarming in Perspective is made simple with all the features and functions built right into both the Alarm Status Table and Alarm Journal Table. If you used the Alarm Status or the Alarm Journal tables in a Vision Client, you'll be pleased to find that most of the functionality on both tables is built-in to the Perspective Alarm Status and Alarm Journal tables and ready to use in a Perspective Session.

Users can filter on a variety of filter options, display alarm event data on the table by selecting different configuration settings, view Realtime and Historical data within a specified time period, view an alarm's details, sort alarm data to meet their individual needs, and use the search function to refine search results. There are a number of properties enabled by default and some other properties have some default options already selected for you. This allows users to hit the ground running right from the start! In addition, the properties in both tables can be configured in the Designer specifically for your project by your designer or Ignition administrator.

You can interface with Alarm Status and Alarm Journal tables in the Designer, Preview Mode and in a Perspective Session.

On this page

...

- Alarm Status Table
- Alarm Journal Table

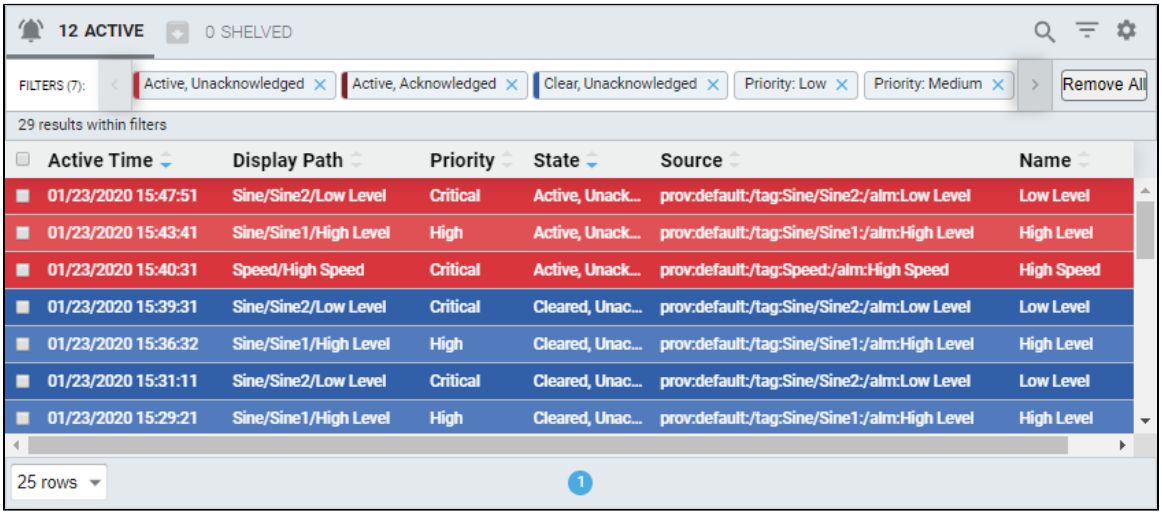
Alarm Status Table

The Alarm Status Table allows you to view currently active alarm events in the system, providing an easy way to inspect the alarm details, shelve alarms, and acknowledge them. At a glance, you will be able to see the current number of active and shelved alarms. The configuration settings, filtering, and search buttons are right on the table for easy access to modifying the alarm data. In a Perspective Session, the table is easily customizable and operators can make on demand changes to configuration settings, filter settings, searching, and sorting of alarm events.

Configure Alarms

You must have alarms configured in your project first, before any alarms will appear in the Alarm Status Table.

When you drag an Alarm Status Table component into the Designer workspace for the first time, the table will automatically populate if you have alarms created and a Alarm Journal Profile created. Your Alarm Status Table will look something like the one below. By default, the Alarm Status Table is configured to show **'Active,Unacked'**, **'Active,Acked'**, and **'Cleared,Unack**' alarm events with priority levels of **Low**, **Medium**, **High** and **Critical**.



For more detailed information using the Alarm Status Table, refer to the Common Tasks section.

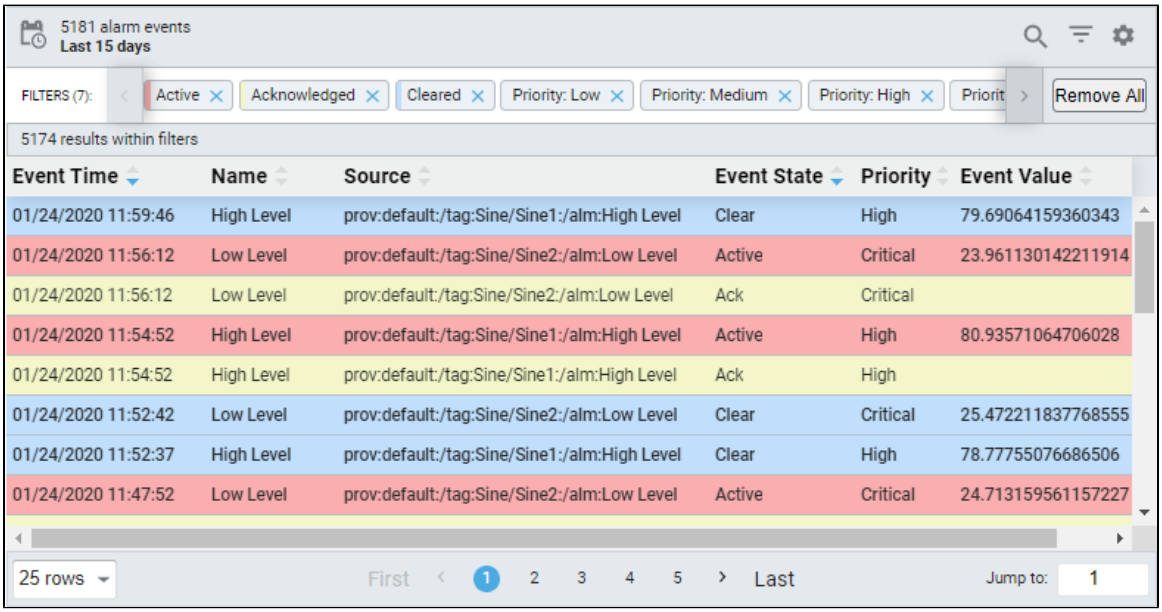
Alarm Journal Table

The Alarm Journal stores historical information about alarms in a database. It stores basic data about alarms that have occurred such as timestamp and values of the alarm's properties at the time the alarm event occurred. The Alarm Journal is used by the Alarm Status Table. You can create a single Alarm Journal Profile to store all of your alarms, or create multiple journals to store alarms across multiple databases. To learn more about alarm journals, refer to the Alarm Journal page.

> **Alarm Journal Profile**
>
> You must have an Alarm Journal Profile created and have a valid database connection to use the Alarm Journal Table.

The Alarm Journal Table looks and behaves very much like the Alarm Status Table. When you drag an Alarm Journal Table component into the Designer workspace for the first time, the table will automatically populate if you have alarms created, an Alarm Journal Profile, and a valid database connection. By default, the Alarm Journal Table is configured to show **'Active'**, **'Acked'**, and **'Cleared'** alarm events with priority levels **Low**, **Medium**, **High** and **Critical**. Your Alarm Journal Table will look something like the one below.

The Perspective Alarm Journal Table has a number of configuration options that can be used to do things like filter on realtime and historical alarm data within a specified time period, filter on alarm event states and priorities, and the search feature will let you refine your alarm event search results. An operator can filter on demand based on their individual needs. You can also change how the component displays those alarm events by simply changing the configuration settings.



For more detailed information using the Alarm Journal Table, refer to the Common Tasks section.

Related Topics ...

- Perspective - Alarm Journal Table
- Perspective Alarm Journal Table - Common Tasks

In This Section ...

Perspective Alarm Status Table - Common Tasks

The Perspective Alarm Status Table has a ton of configuration options that can be used to do things like filter the list of alarms being displayed, acknowledge and shelve alarms, and configure how the Alarm Status Table component displays the alarm events. Each of the pages in this section goes over setting up various aspects of the Alarm Status Table.

On this page ...

- Perspective Alarm Status - User Interaction
- Perspective Alarm Status - Configuring Properties
- Perspective Alarm Status - General Filtering
- Perspective Alarm Status - Acknowledgment
- Perspective Alarm Status - Shelving
- Perspective Alarm Status - Row Styles

Perspective Alarm Status - User Interaction

The User Interaction page introduces you to the Alarm Status Table, its interface and how to navigate around the table. It also briefly describes the Alarm Status Table's features and functions including how to configure the table to get realtime and historical alarm event data.

Perspective Alarm Status - Configuring Properties

The project designer and Ignition administrator have the option to enable or disable properties and setup filtering options in the Property Editor of the Designer based on the project requirements and needs of the client users. This page describes which properties are enabled and which properties have some default filtering options preset. Learn how to configure your own Alarm Status properties in the Designer on the Configuring Properties in the Designer page.

Perspective Alarm Status - General Filtering

The Alarm Status Table has a number of built-in functions right on the table so with a simple click of a button you can filter on alarm states and see alarm details. The Alarm Status Table component has a lot of properties that allow you to filter on various parts of alarms. Learn about all of the different built-in ways that the Alarm Status Table can filter alarms.

Perspective Alarm Status - Acknowledgment

The first step in fixing an alarm is acknowledging that the alarm is happening. Acknowledgement plays a very important part in any alarm system which is why the ability to acknowledge alarms is built right in to the Alarm Status Table component. Learn how to acknowledge alarms and configure acknowledgement options.

Perspective Alarm Status - Shelving

Shelving alarms allows you to temporarily silence an alarm for a fixed period of time while you are working on the issue. This can be useful when doing maintenance if Tags are constantly going in and out of alarm.

Perspective Alarm Status - Row Styles

The Alarm Status Table uses different styled rows to differentiate between alarms in different states. These Row Styles can be completely customized using whatever colors and fonts you want. You can even add an animated style drawing an operator's attention to critical alarms!

In This Section ...

Perspective Alarm Status - User Interaction

Getting Started with the Alarm Status Table

On this page

...

- Getting Started with the Alarm Status Table
  - Anatomy of an Alarm Status Table
  - Alarm Details
  - Acknowledging and Shelving Alarms

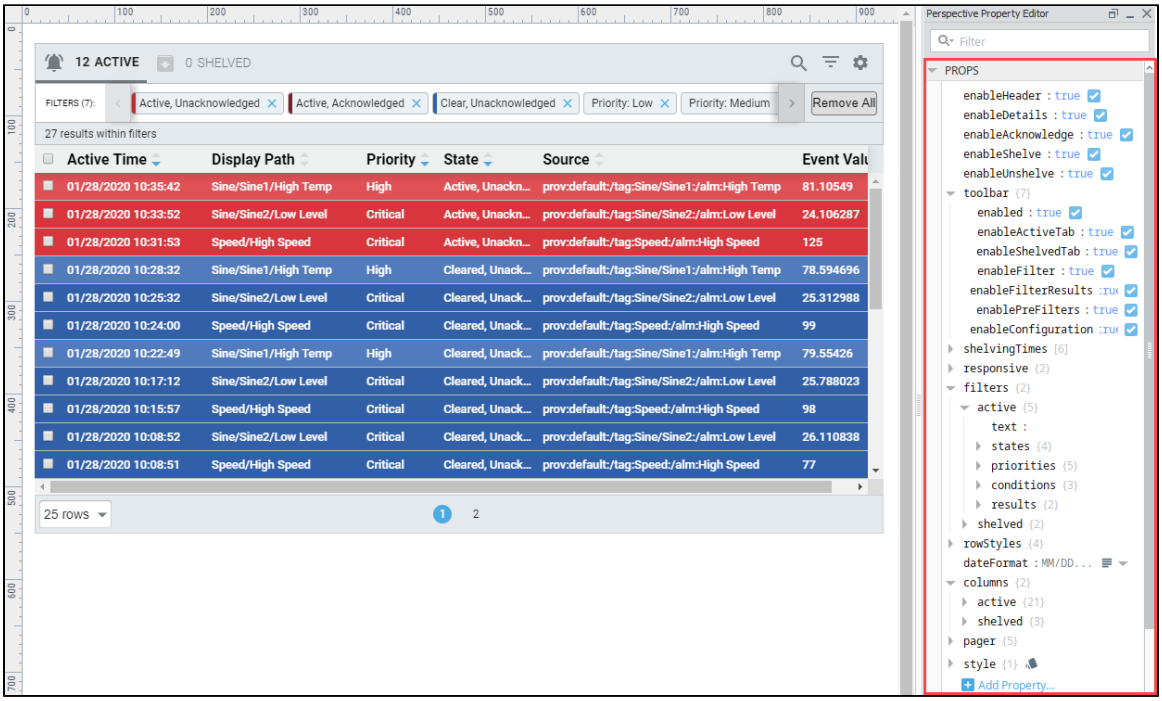
If you're setting up the Alarm Status Table for the first time, you probably will want to take a look at the status table's properties in the Property Editor. There a number of properties that are enabled by default such as the shelve, unshelve, acknowledge, filter, and configuration buttons, as well as the alarm details popup to name a few. You should become familiar with them in the event you would like to change them. Other properties have default options already selected for you, like alarm states, alarm priorities, and row styles.

When you drag your first Alarm Status Table component into the Designer, the Alarm Status Table will automatically populate if you have any alarms configured and an Alarm Journal Profile created. It will look something like the image below.

You can interact with the table in the Designer, in Preview Mode of the Designer, and in a Perspective Session. The Alarm Status Table properties are configured in the Property Editor of the Designer.

When setting up your Alarm Status Table, you will have to toggle between both the Designer and Preview Modes to configure properties and organize the display of your alarm event data and size the data columns in the table.

This page will introduce you to the Alarm Status Table and its features and functions. Other pages in the Perspective Alarm Status Table - Common Tasks section address the details of how to acknowledge, shelve, and filter alarms.



Anatomy of an Alarm Status Table

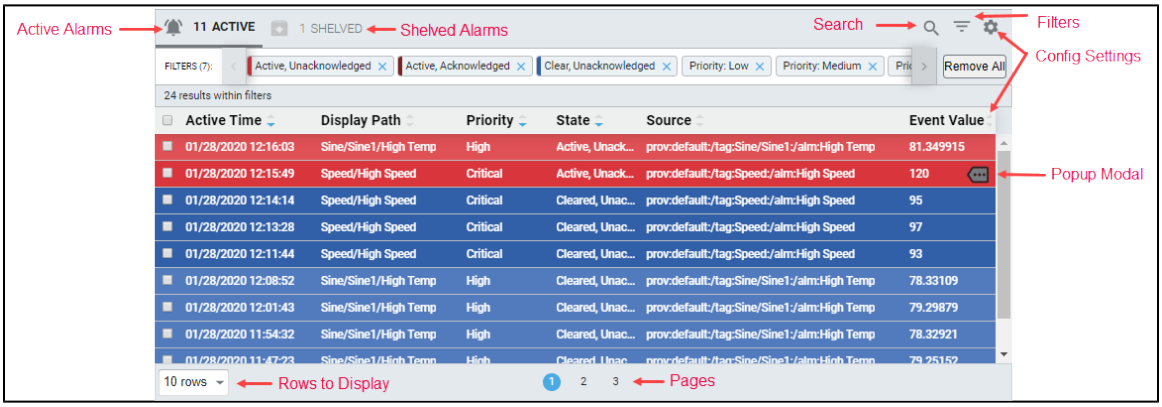
The following image shows the basic anatomy of an Alarm Status Table with alarm event data populated in the rows. At the top of the table, you can immediately see two tabs, one for **Active** alarms and the other for **Shelved** alarms. The tab headers provide a count of the number of Active alarms and Shelved alarms. Click on each of the tabs to view all the currently Active alarms and Shelved alarms.

There are a host of **Configuration Settings**  and **Filtering options**  that you can set to provide relevant information about your alarms events. The **Configuration Settings** contain the available header columns that can be set to display data about the alarm event such as Ack Notes, Ack Pipeline, Ack Time, Ack User, Active Time, Display Path, and Source Path. Filtering options are actually the alarm states and priorities that can be selected to get the most current alarm status in your system. Alarm states are **ActiveUnacknowledged**, **ActiveAcknowledged**, **ClearUnacknowledged** and **ClearAcknowledged**. Alarm Priorities are **Diagnostic**, **Low**, **Medium**, **High** and **Critical**.

The **Filter Bar** shows you exactly what state and priority filter properties are set. The Filtering tab allows you select the alarm state and filter on specific alarm criteria. You can remove individual filter properties by clicking the **'X'** next to each property or remove all the filter properties using the **Remove All** button. To add filter properties, click on the **Filter** icon and select the properties you want to use.

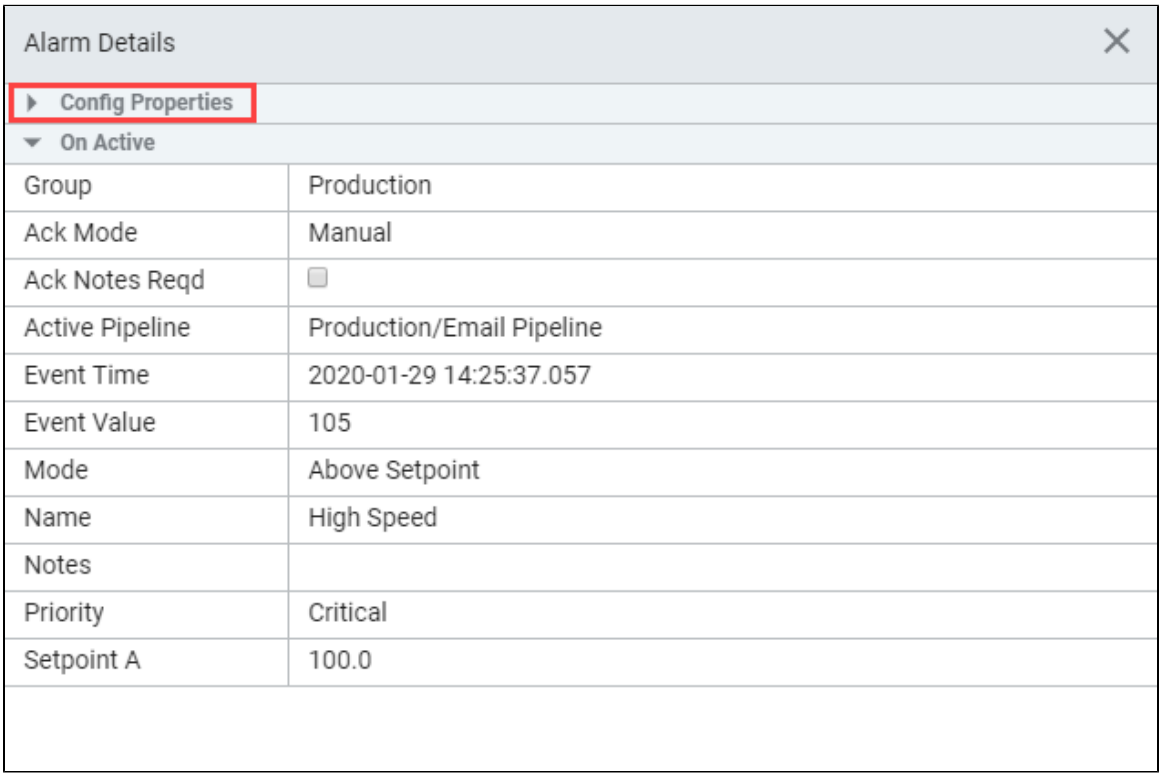
You can also optimize your search results by using the **Search** feature and entering keywords or a string to help further refine your results. When you mouse over an alarm row, a popup modal will appear on the right side of the table. By clicking the **Popup Modal** icon, you can see the alarm's details. You can also set the number of alarm rows to display and scroll through the list of pages at the bottom of the status table. Lastly, the Alarm Status Table comes with a default set of customizable row colors based on the alarm state and priority.

To learn more about using the Alarm Status functions and features such as alarm acknowledgement, shelving, filtering and row styles, refer to the Perspective Alarm Status Table - Common Tasks section. The Alarm Status Table component page describes all the alarm status properties and how to use them.



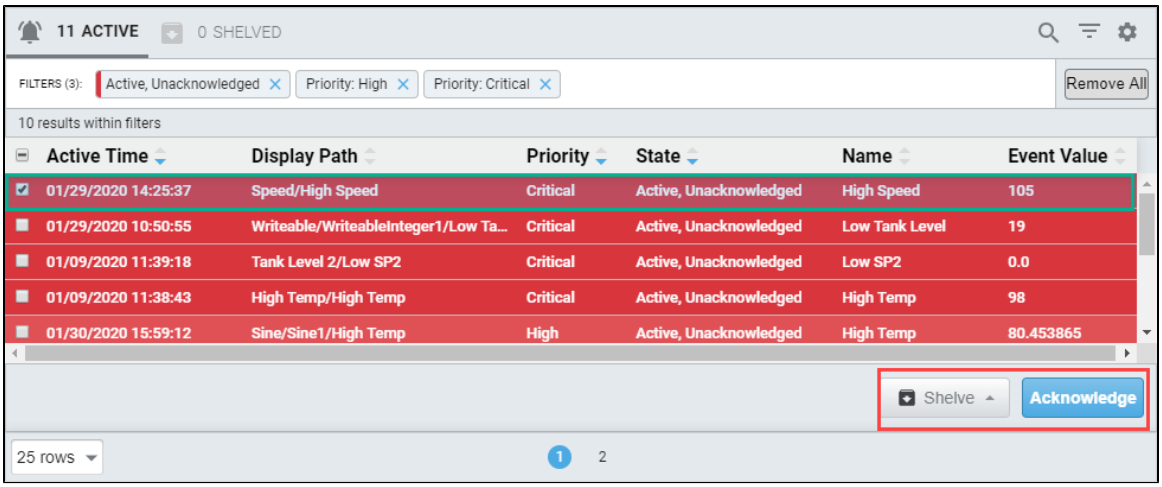
Alarm Details

When you click on the Popup Modal icon, this brings up the Alarms Detail window. Here you can scroll through a list of configuration properties and see the details about the alarm event.



Acknowledging and Shelving Alarms

When an alarm is selected, a footer will slide into view below the table containing two buttons, one to **Shelve** alarms and the other to **Acknow ledge** alarms. Check the box next to the alarm event to either shelve or acknowledge the alarm. With the shelved alarms visible, you'll have the ability to **Unshelve** selected alarms. This footer will also display messages as feedback when performing an action on the selected alarms (i.e., success, failure, etc.). To learn more, refer to the Shelving and Acknowlegement pages in this section.



Related Topics ...

- Perspective Alarm Status Table - Common Tasks
- Perspective - Alarm Status Table

Perspective Alarm Status - Configuring Properties in Designer

The following feature is new in Ignition version **8.0.3**
Click here to check out the other new features

Configure Alarms

Alarms must be set up on Tags for them to show up in the Alarm Status Table.

On this page

...

- Getting Started
- Configuring Properties in the Designer
  - Shelving Times
  - Filtering on Alarm States and Alarm Priorities
  - Filtering on Source Path and Display Path

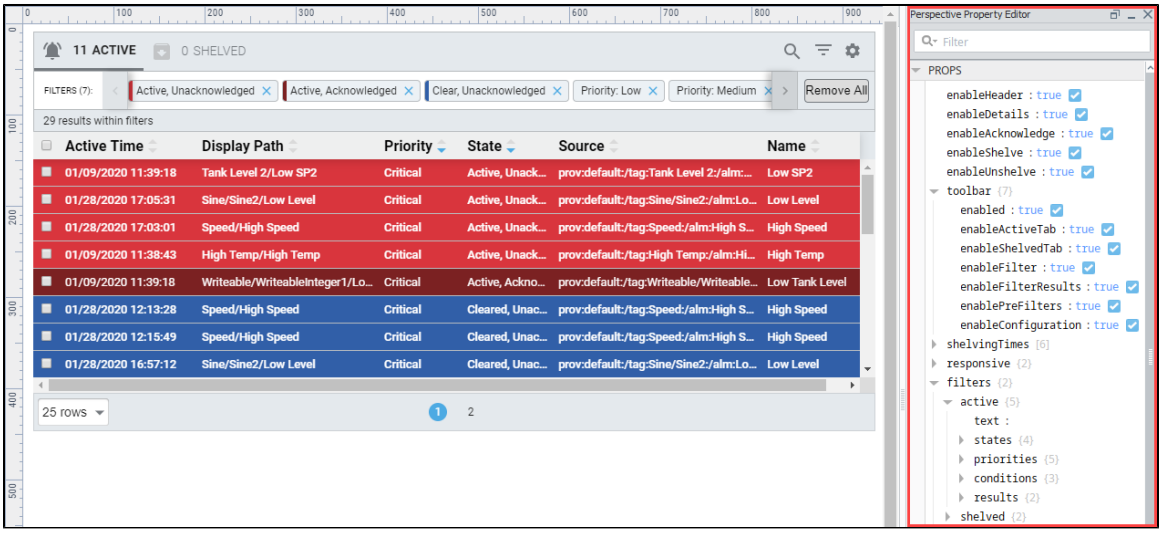
Getting Started

Filtering can be done in both the Designer and in a Perspective Session, but enabling and disabling Alarm Status Table functions and setting up specified filtering options for client users is configured in the Designer. The project designer or Ignition administrator have the permission to enable or disable properties and setup filtering options in the Property Editor of the Designer based on the project requirements and needs of the client users. If you are the project designer or Ignition administrator, you probably want to check out all the Alarm Status Table properties in the Perspective Property Editor. It's a good idea to scroll through all the alarm properties and expand them to see additional properties and see the default property settings and preselected filtering options.

The Alarm Status Table has some properties enabled by default, and some filter options already preset in the Designer to give client users a head start using the table. It's in the Property Editor of the Designer where table properties are enabled and disabled, and filtering options are configured such as the alarm states and priorities to display, setting the shelving times, defining row styles, choosing column headers and column sort options, or setting up a specific display path and source path for operators to view.

The first time an Alarm Status Table component is dragged on to a window in the Designer, by default, the table displays all the alarms that are currently 'Active and Unacknowledged,' 'Active and Acknowledged,' and 'Cleared and Unacknowledged,' with a sort priority ranging from Low to Critical.

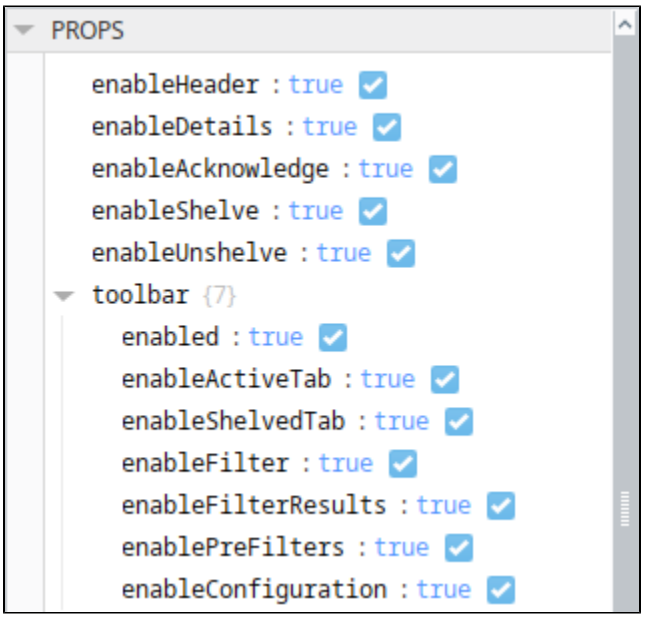
The Alarm Status Table below is similar to what you'll see when you first drag in an Alarm Status Table to a Designer window.



Configuring Properties in the Designer

When you drag a Alarm Status Table component into the Designer for the first time, it displays the alarm results with default properties already configured and some filtering options already preselected for you. Some of the default settings may work for your client users, but others may need to be modified in the Property Editor.

The first thing you might want to look at is the functionality that is enabled on the table component.



While in the Property Editor, expand the table's filters properties to see what filter properties are configured.

Preview Mode

When configuring the Alarm Status Table in the Designer, you have to go to Preview Mode to see your updates, filter, sort and organize the alarm data in the table. When making modfications to the properties in the Property Editor, you will find yourself toggling between the Designer and Preview Modes multiple times to make sure your modifications are displayed the way you want them. Each time you reset your filtering options and column headers in the Designer, the alarm table will refresh with new alarm dat a based on your configuration and filter settings.

Here are a couple of examples of Alarm Status Table properties that you may want to modify.

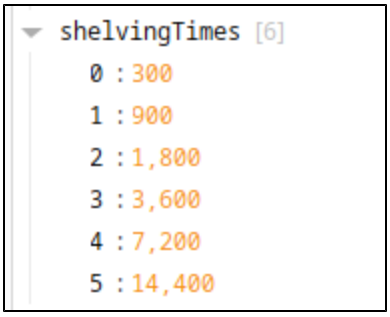
Shelving Times

In this example, let's add a new shelve time.

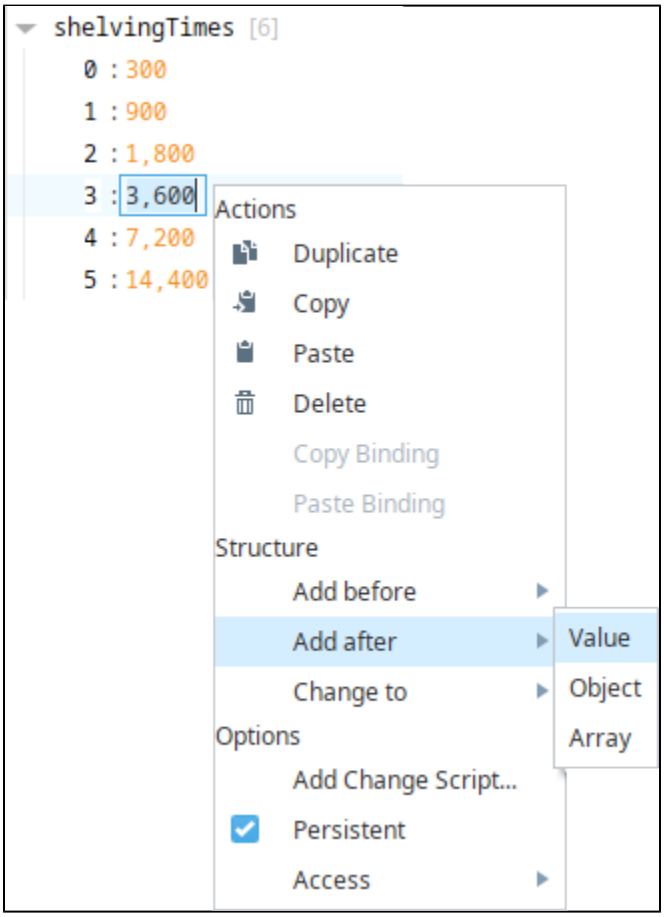
The default shelvng times are as follows:

- 5 minutes
- 15 minutes
- 30 minutes
- 1 hour
- 2 hours
- 4 hours

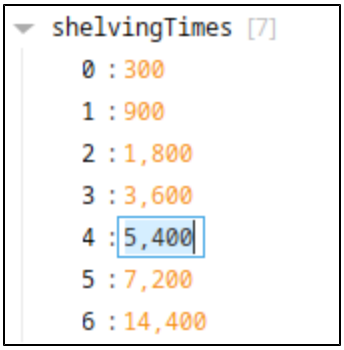
Let's say you wanted to add another shelving time for 90 minutes. The project designer will need to update the **'shelvingTimes'** property to make 90 minutes available to operators. In the Property Editor, shelving times are entered as seconds, not minutes.



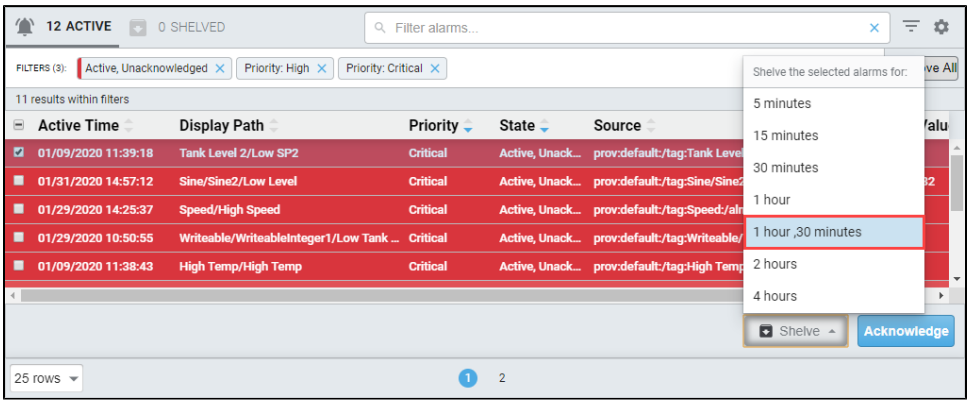
1. To add a shelving time for 90 minutes, mouse over the value 3,600 (1 hour) and right click, select **'Add after'** and click **'Value.'**



2. Enter **5400** (seconds).



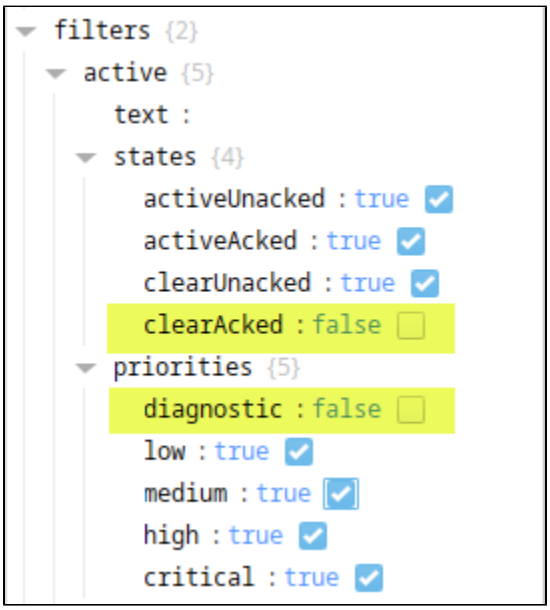
3. Now, open your window in **Preview Mode**, and you'll see your updated shelving times.



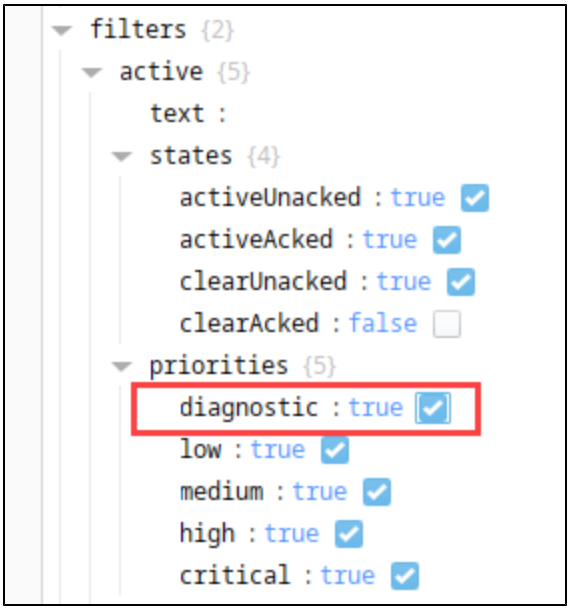
Filtering on Alarm States and Alarm Priorities

By default, the **'clearAcked'** alarm state and the **'diagnostic'** priority are not enabled properties, but say for example the 'diagnostic' property is required by operators. To make this property available, the project designer or Ignition administrator can go to the Property Editor to enable 'diagnostic' for the operators to use in a Perspective Session.

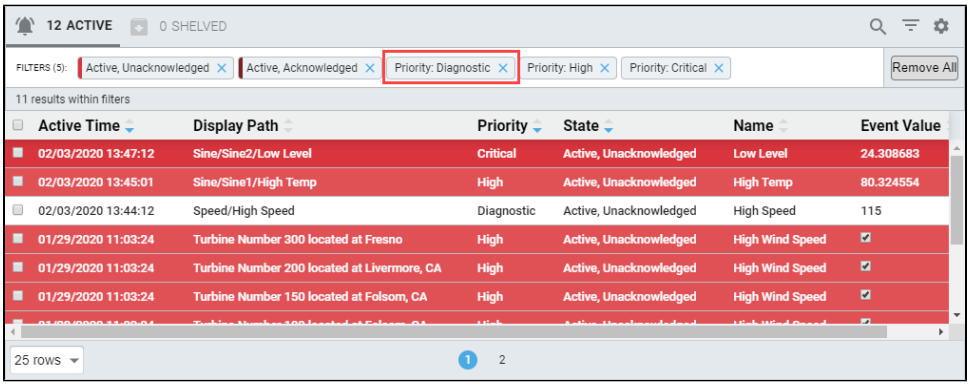
1. Expand the **filters > active > states > priorities** properties. Here you see the default states and priorities.



2. Click the checkbox for the **'diagnostic'** priority property to enable it making it available to the operators in a Perspective Session.



3. Go to **Preview Mode** to verify the 'diagnostic' property was added. You will see it displayed in the Filter Bar.



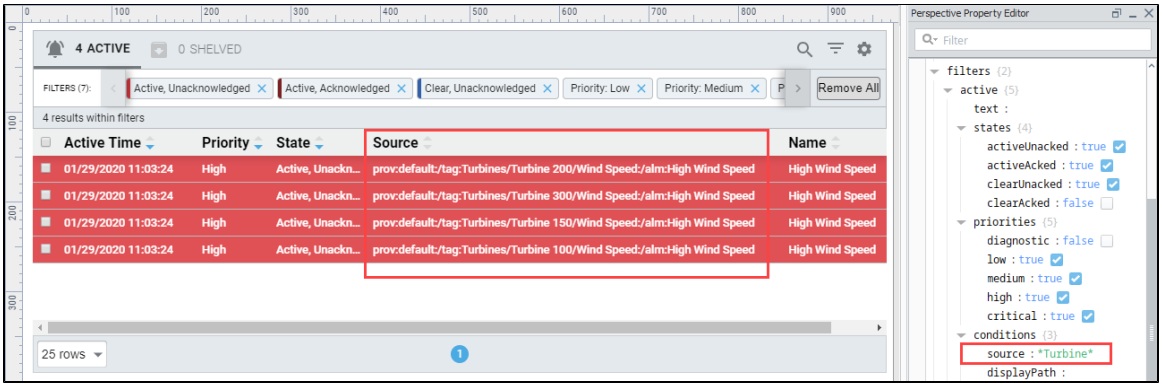
Filtering on Source Path and Display Path

You can filter the alarm list in the table to be a shorter list using the Filter properties instead of scrolling through every single alarm in your system. In the Property Editor of the Designer, there is a group called **Filters** that you can configure to focus on only those alarms you want to see. You can filter on **State, Source Path, Display Path**, and **Tag Provider**. The most helpful properties to an operator are Source Path and Display Path.

Source Path

The **Source** is the actual **Tag path** which means you can also use Tag folders in Ignition to filter for specific alarms. For example, you may want to filter for all Turbine alarms in the Turbines folder. You can enter ***Turbine*** (using ***** as a wildcard) to look for all alarms with the Turbine Tag Path. The **Source** and **Display** properties allow you to restrict the results of the query to one or more paths. Multiple paths may be specified with a comma. Additionally, these properties all use the asterisk (*) as wildcard character to denote any number of leading or trailing characters, depending on placement as shown in the image below.

You can see in this example, all the active alarms have **'Turbine'** in the **Source Path**.



Source Path Examples

Example Filter	Result
`prov:tagProvider:/tag:Inputs /PS_1:/alm:MyAlarm`	Retrieve alarm information from the alarm at precisely the specified path: `prov:tagProvider:/tag:Inputs/PS_1:/alm:MyAlarm`

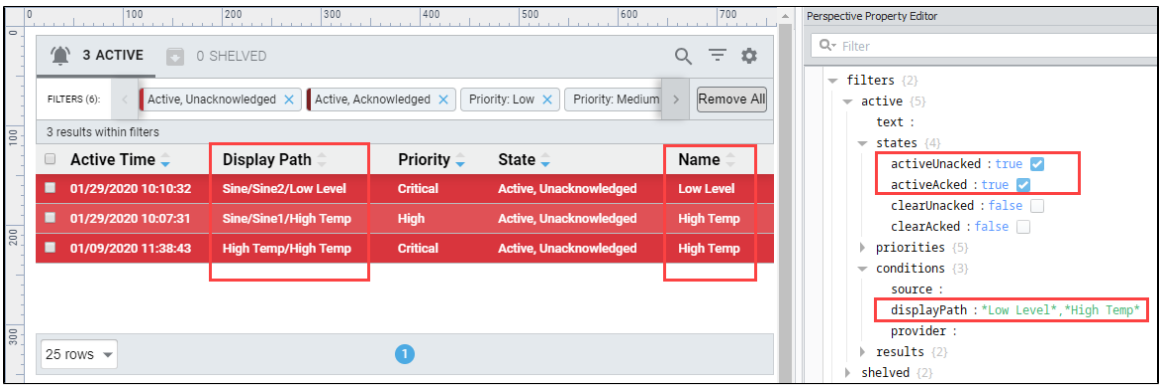
`*PS_1:/alm:MyAlarm`	Retrieves alarm information from any path that ends with `PS_1:/alm:MyAlarm`. Thus the following paths would be returned:
	`prov:tagProvider:/tag:Inputs/PS_1:/alm:MyAlarm`
	`prov:tagProvider:/tag:anotherFolder/different_Path/PS_1:/alm: MyAlarm`

`prov:tagProvider:/tag:PS_*`	Retrieves alarm information from any source path starting with `prov:tagProvider: /tag:PS_`
	`prov:tagProvider:/tag:PS_1/MyAlarm:/alm:MyAlarm`
	`prov:tagProvider:/tag:PS_2/MyAlarm:/alm:MyAlarm`

`*MyAlarm*`	Retrieves any alarm information that has `MyAlarm` somewhere in the path.

Display Path

The **Display Path** is the Tag path that leads to the **Name** of the alarm which can be customized when you configure your alarm. In this example, **'Low Level'** and **'High Level'** were the names that were setup in the Alarm Configuration, and the Display Path is set to the Name you want the operator to see. The Alarm Status Table below is filtered by **Display Path** with a value of **'Low Level'** and **'High Level.'** So in this example, you see results for both **Low Level** and **High Level** alarms in the **Active, Unacknowledged** state.



To learn more about alarm table properties, refer to the Perspective - Alarm Status Table page.

Related Topics ...

- Perspective - Alarm Status Table
- Configuring Alarms

Perspective Alarm Status - Filtering

The following feature is new in Ignition version **8.0.3**
Click here to check out the other new features

Configure Alarms

Alarms must be set up on Tags for them to show up in the Alarm Status Table.

On this page

...

- Filtering in a Perspective Session
  - Using the Search Bar
- Viewing and Sorting on Alarm Data
  - Sorting Alarm Data

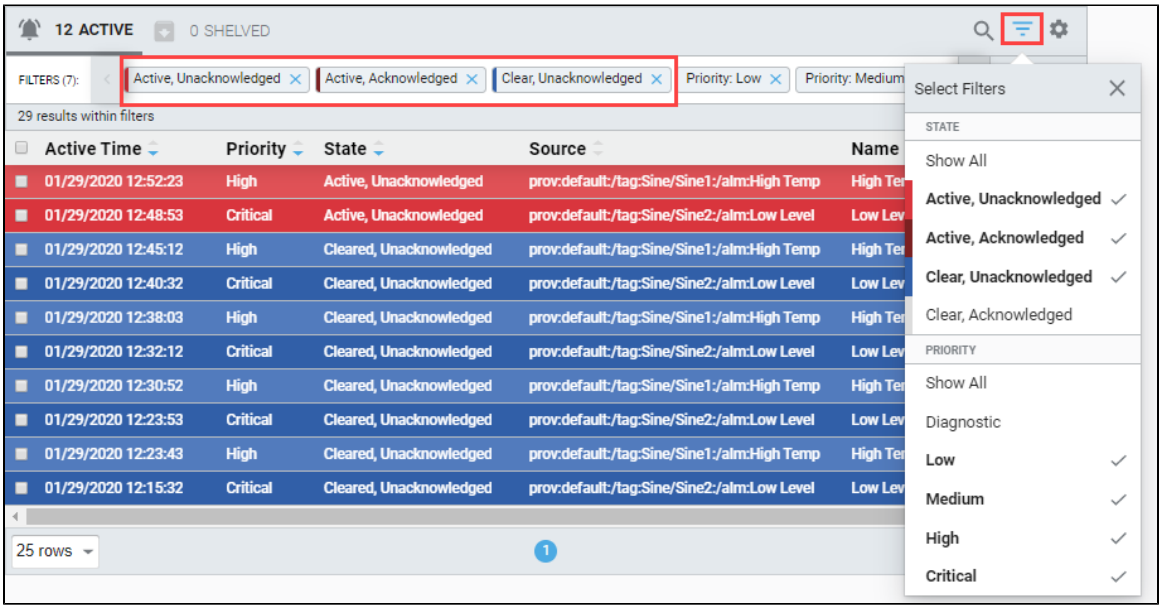
The Alarm Status Table has many built-in properties that allow you to filter on various parts of an alarm. What's super nice about the Perspective Alarm Status Table is everything is right at your fingertips for filtering on alarm events. The table provides a host of built-in filtering options that are immediately available in a Perspective Session and easy to modify to help you get started.

You can choose to filter on alarm states and priorites by clicking on the Filter button. In addition, there is a search bar that lets you further optimized your search results. The Configuration Settings allow operators to view alarm data that is most important to them, as well as organize the alarm data any way they choose.

Filtering in a Perspective Session

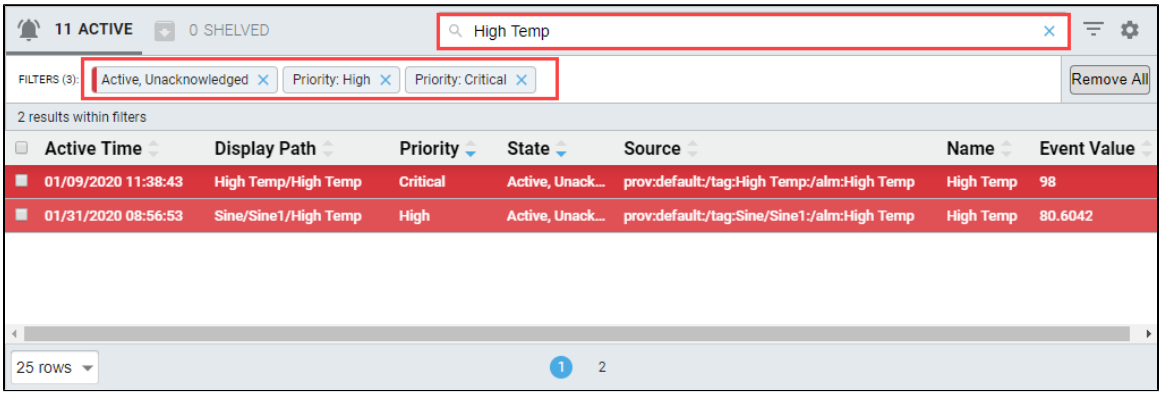
In a Perspective Session, all the filtering tools the operators will need are built in to the Perspective Alarm Status Table. When first using the table in a session, an operator could be using the default alarm status properties, or your project designer or system administrator may have preconfigured some alarm status properties specific to your project. Either way, an operator can easily choose and filter on specific alarm dat a and how they want it displayed.

In a session, you can easily change the filtering options on the table by clicking on the **Filter icon** and adding or removing a filter option from the dropdown by checking or unchecking a filter option. Notice that there is a filter bar at the top of the table that also displays all the filter options that are currently set. You can remove any of these filter options by clicking the **'X'** on the right side of the option, but to add a filter, use the **Filter** dropdown. Notice how the Filter Bar displays the row color on the tab for ActiveUnacknowledged Active, Unacknowledged, ActiveAcknowledged Active, Acknowledged, and ClearUnacknowledged Clear, Unacknowledged states.



Using the Search Bar

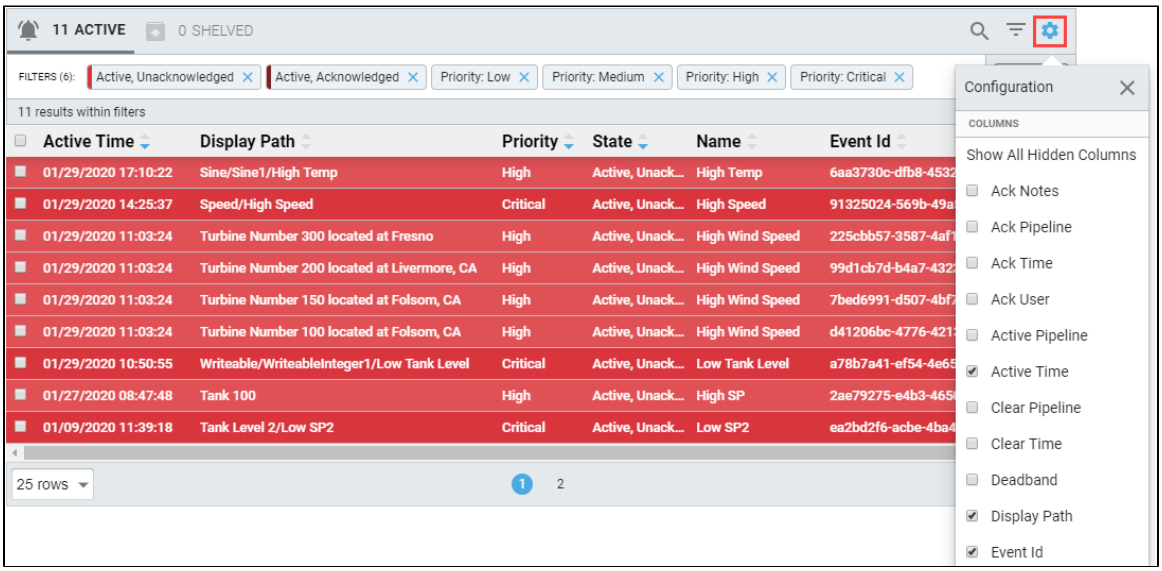
Some operators may want to do some more targeted filtering like only seeing the 'Active, Unacknowledged' alarms for a specific type of alarm. This is easily accomplished by deleting all the alarm state properties except for **'Active, Unacknowledged'** and entering a specific keyword or string in the seach bar to find specific alarms, events, and conditions. In the image below, the search criteria was for **'High Speed.'** It was found under three column headers: Display Path, Source Path, and Name.



Viewing and Sorting on Alarm Data

In a Perspective Session, operators can choose to display or hide alarm data in the table by checking or unchecking any of the **Configuration Settings**  , or right clicking in the header row of the table. You'll find that it's super easy to filter and display alarm event data using the configuration options provided in the column header of the Alarm Status Table. You can do a multi-column sort by holding down **Ctrl + Shift** and clicking on the column headers you want to sort by.

If you need to align the columns, simply put your cursor to the left of the column header and drag left or right. There is a **'strictWidth'** property for each header column that can be configured in the Designer to strictly enforce the column width.



Sorting Alarm Data

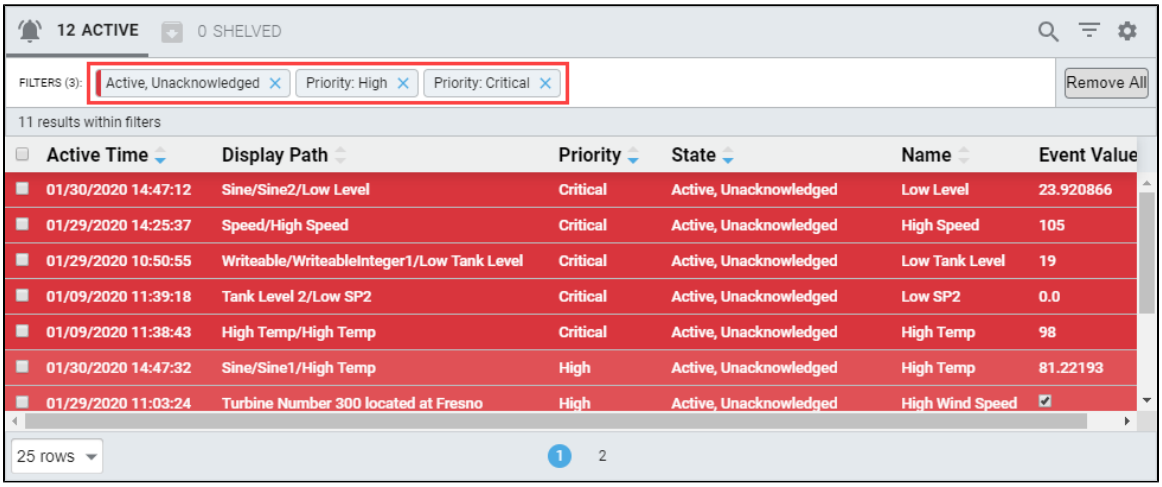
Once your alarm is filtered, operators can also sort table columns in ascending or descending order by simply clicking the up or down arrows next to each column header.

Sorting on alarm State and Priority in the Alarm Status Table, by default, sorts in descending order. All the other columns the sort order is alphanumerical. Sort order for **State** and **Priority** are as follows:

- **Alarm State** - ActiveUnacknowledged, ActiveAcknowledged, ClearUnacknowledged, and Clear Acknowledged.

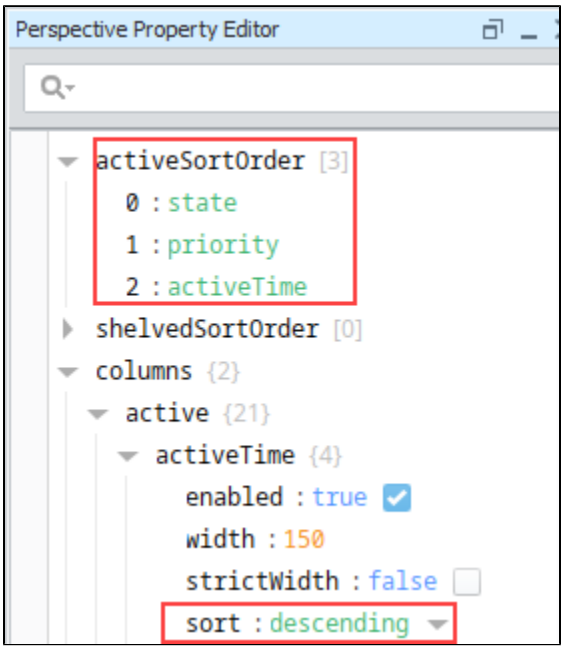
- **Alarm Priority** - Critical, High, Medium, Low, and Diagnostic.

The image below was filtered by ActiveUnacknowledged state, with a priority Critical and High. Once your alarm data is filtered, you can also sort on the data in multiple columns by holding down the **Ctrl + Shift** and clicking on the column headers you want to multisort by.

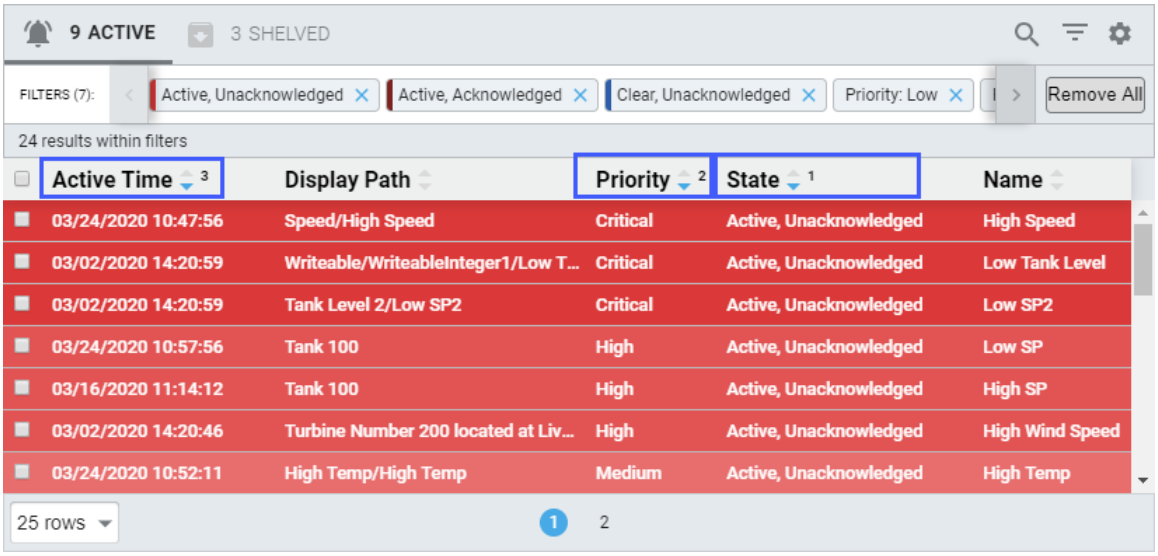


> The following feature is new in Ignition version **8.0.11**
> Click here to check out the other new features

Perspective's Alarm Status Table now supports multiple sort orders. The order is first determined by the sort order properties on the Alarm Status Table. There are two new sort order properties: '`activeSortOrder`' and '`shelvedSortOrder.`' Each column that you add to the sort order property will also need to have a sort defined under the columns property as either ascending or descending.



You also need to make sure the configuration columns you choose for your sort are displayed on your table. One of the most useful sort orders is by **State - Priority - Active Time**, as shown in the following image. Notice how each column is numbered according to your `activeSortOrder` property.



Columns will first sort by anything defined in sortOrder and then fallback to the other sorts defined in the column configuration.

Related Topics ...

- Perspective - Alarm Status Table
- Configuring Alarms

Perspective Alarm Status - Acknowledgement

The following feature is new in Ignition version **8.0.3**
Click here to check out the other new features

On this page

...

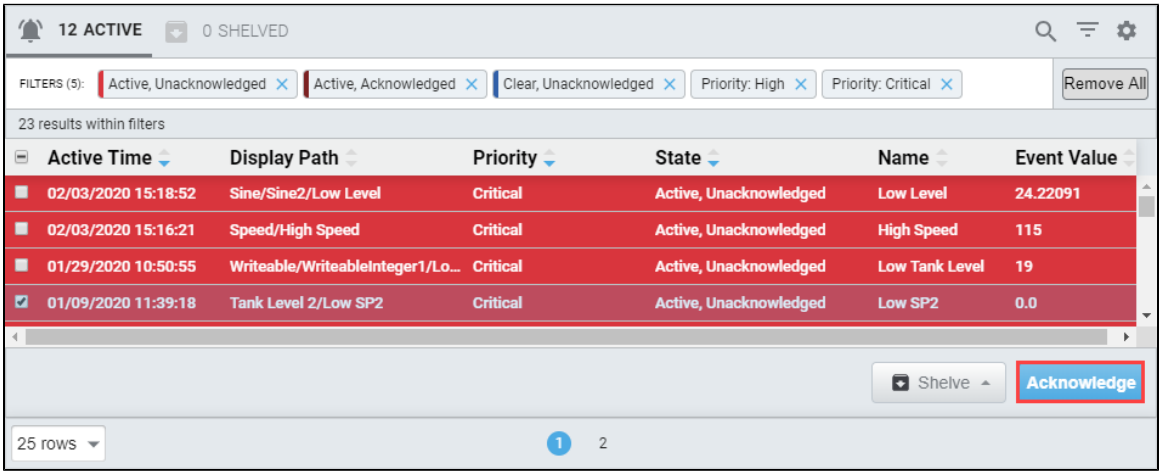
- Acknowledging Alarms
  - Configuring Table Headers
  - Acknowledgement Notes
- Security for Alarm Acknowledgement

Acknowledging Alarms

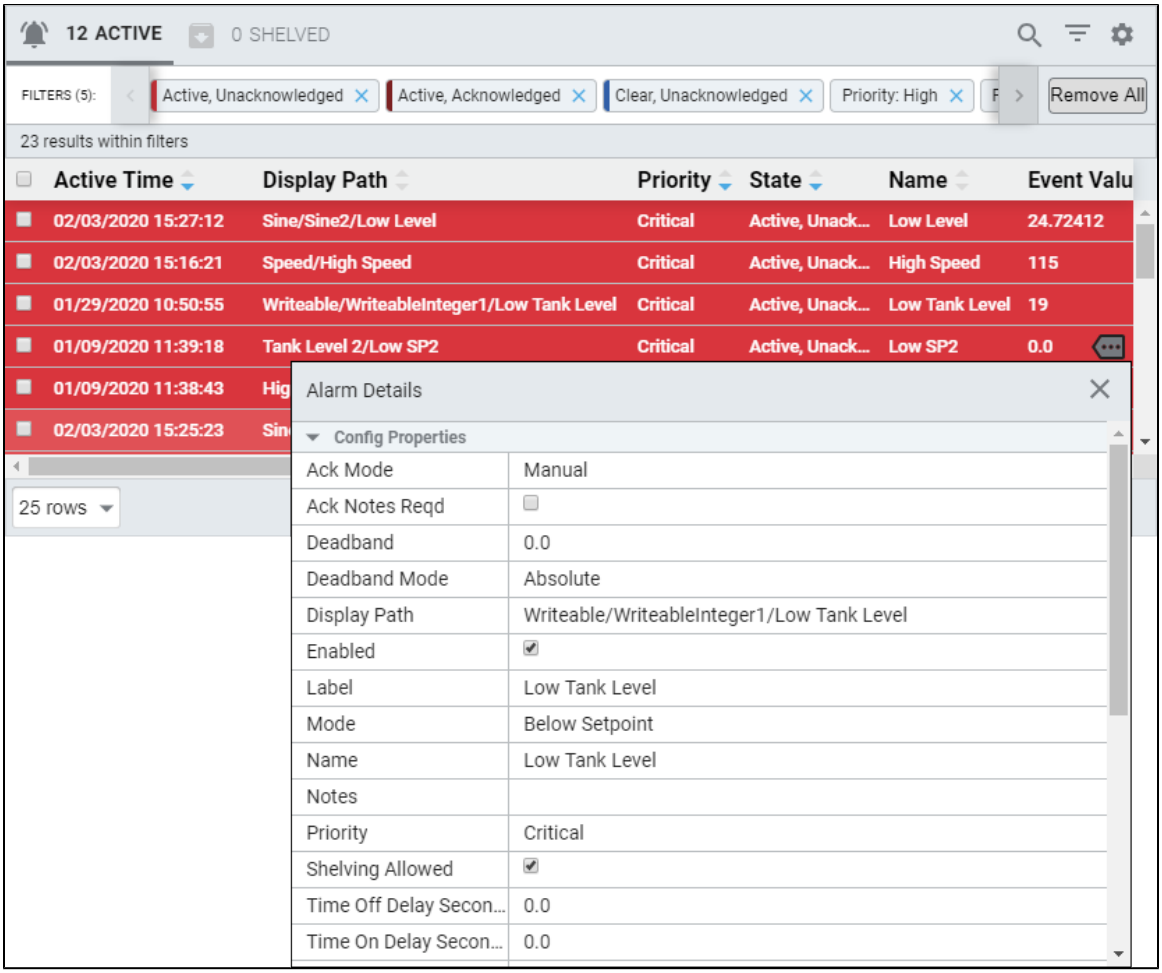
The Alarm Status Table gives operators a ton of information at a glance about alarms that need attention. One of the most important things an operator is going to do in a Perspective Session is acknowledge alarms. Alarm acknowledgement is built in to the Alarm Status Table component. As soon as the operator selects and presses the Acknowledge button, the current state of the alarm will change, and the operator's credentials and the time the alarm was acknowledged will be recorded in the Alarm Status Table.

The Alarm Status Table component allows you to select an individual alarm, multiple alarms or use the **'Select All'** checkbox in the header bar. To Acknowledge alarms, check all the alarms you want to acknowledge, and a footer will open at the bottom of the table to show the **Acknowledge** button. Press the **Acknowledge** button and the Alarm Status Table will record the time the alarm was acknowledged and the user that acknowledged the alarm in the database. (Note, to see this alarm information displayed in the table, the column headers may need to be configured as described later on this page).

In the following example, one alarm was checked and Acknowledged.



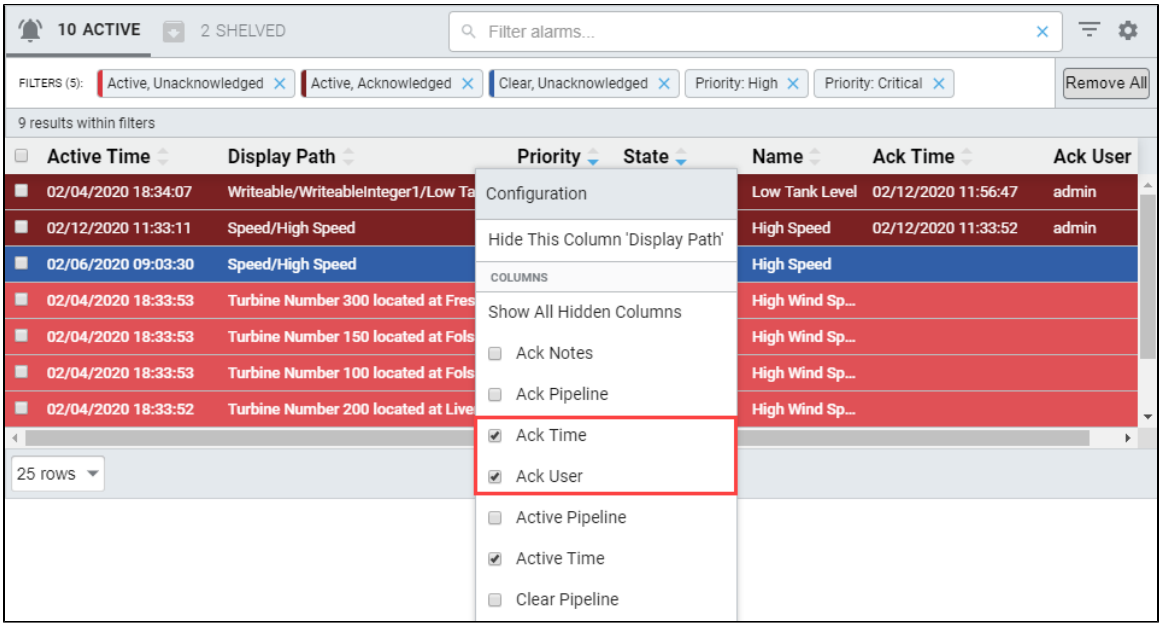
When you mouse-over an alarm row, you'll notice a popup modal ( ) on the far right of the table that allows you to view the alarm details by simply clicking it. This brings up the **Alarms Details** window to view the alarm configuration properties.



Configuring Table Headers

If you don't already have the **'Ack Time'** and **'Ack User'** displayed in your table header, select them from the **Configuration Settings** dr opdown. You can also add any other alarm data you want to display, or remove any alarm data that you don't need to display from the Alarm Status Table.

Notice that the time the alarm was acknowledged and who acknowledged the alarm is now shown in the table.

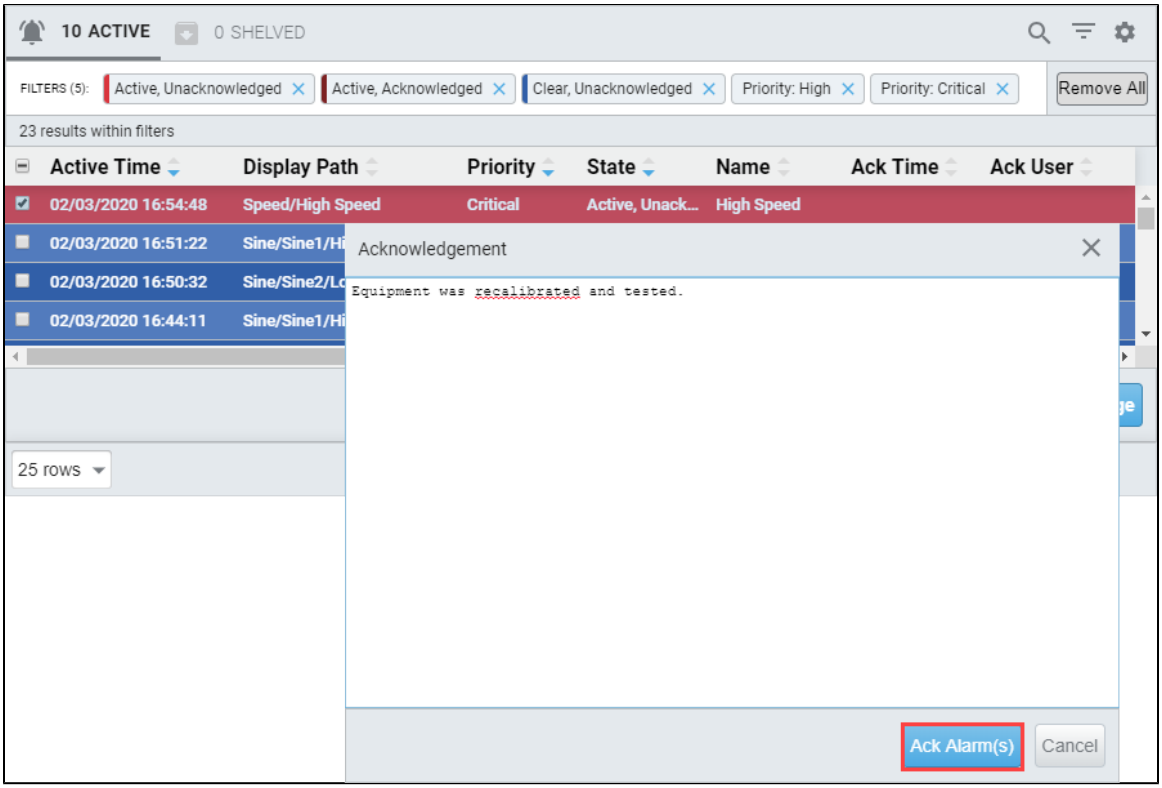


Acknowledgement Notes

If any of the selected alarms require **Acknowledgement Notes**, a small window will appear when the Acknowledgement button is pressed. The operator will be required to add notes, otherwise the alarm cannot be acknowledged. Enter your notes and press **Ack Alarm(s)**. If the operator wants to cancel the Acknowledgement Notes and close the window, click on the **Cancel** button to close the Acknowledgement notes window.

Configuring Acknowledgement Notes

Acknowledgement Notes are set up in the alarm's configuration settings. To set up Acknowledgement Notes, go to your alarm configuration settings and set **'Ack Notes Required'** to **'true.'** For more information, refer to Configuring Alarms.

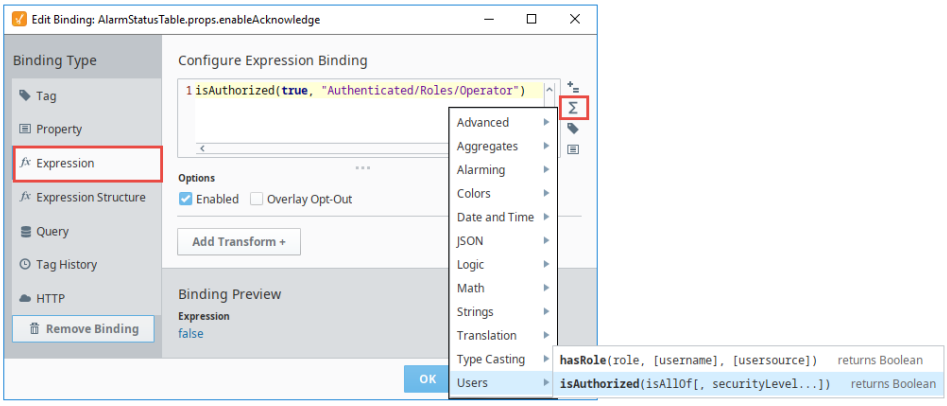


Security for Alarm Acknowledgement

You can restrict specific users or roles from Acknowledging alarms by setting the **enableAcknowledge** property in the Property Editor to **'false.'** This hides the Acknowledge button on the Alarm Status Table for those users who do not have permission. You can setup permissions for any role, user and user source in your system.

For example, if you only want those users with the Operator role to acknowledge alarms, the correct permission must be assigned.

1. Select the Alarm Status Table component, and click the **enableAcknowledge binding** icon to open the Property Binding window.
2. Under **Property Binding Type**, select **Expression**.
3. Click the **Function** icon and scroll down to **Users,** and select '**isAuthorized.'** This enters the function name.
4. Edit the expression to read: **isAuthorized(true, "Authenticated/Roles/Operator")**
5. Click **OK**.



If you currently have the 'Operator' role, you'll notice in the Property Editor of the Designer that the **enableAcknowledge** property is set to **'true,'** and for other roles, it will be set to **'false.'**

Related Topics ...

- Configuring Alarms

Perspective Alarm Status - Shelving

The following feature is new in Ignition version **8.0.3**
Click here to check out the other new features

The capability to shelve alarms is another important feature of the Perspective Alarm Status Table. Shelving alarms allows you to temporarily silence an alarm for a fixed period of time. This feature is extremely handy when an alarm is already active and you want to temporarily surpress the alarm while you're working on the issue. The Perspective Alarm Status Table component will not send any notifications while the alarm is shelved, and will be temporarily dropped from the Alarm Status list so operators don't get confused and think it's active. When the shelved time period is up and if the alarm is still active, it will return into the Alarm Status list.

The top of the Alarm Status Table displays the number of shelved alarms. To view all the shelved alarms, click on this **Shelve** button.

On this page

...

- How to Shelve an Alarm
  - Viewing Shelved Alarms
  - Unshelve Alarms
- Configuring Custom Shelving Duration

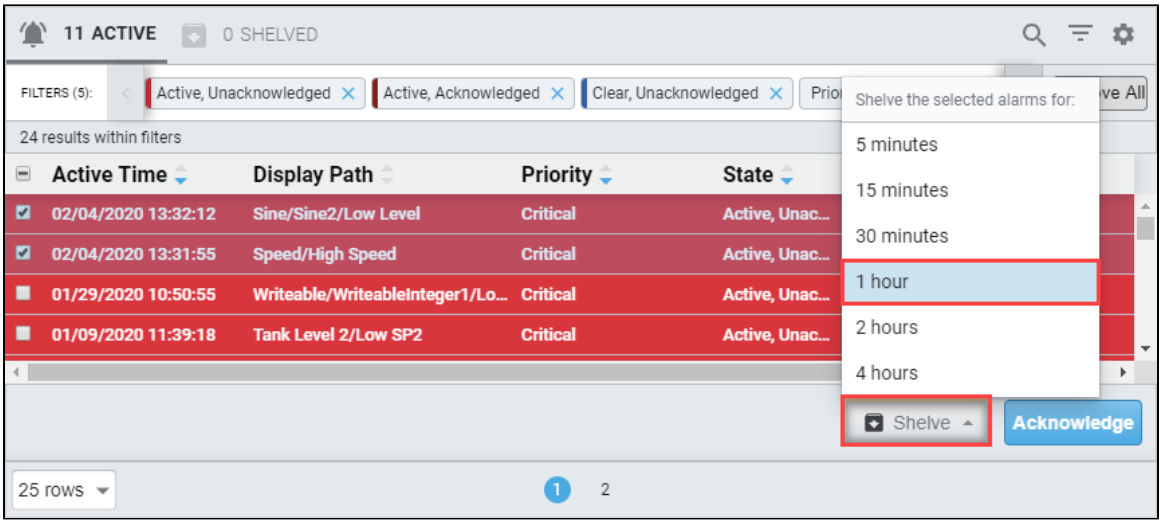
Active Alarms

Active alarms must be present in the Alarm Status Table before you can shelve an alarm.

How to Shelve an Alarm

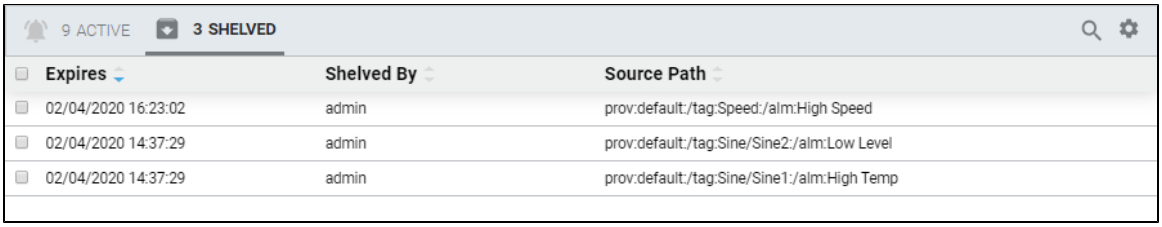
To shelve an alarm, select one alarm or multiple alarms, and a footer will open at the bottom of the table to show the **Shelve** button. Click the **Shelve** button and a dropdown will automatically open so you can set a duration to silence the selected alarms. You can set a duration from 5 minutes to 4 hours to shelve selected alarms. Choose the duration and your alarm will immediately be shelved for your selected alarms.

The example below shows two alarms were shelved for one hour.



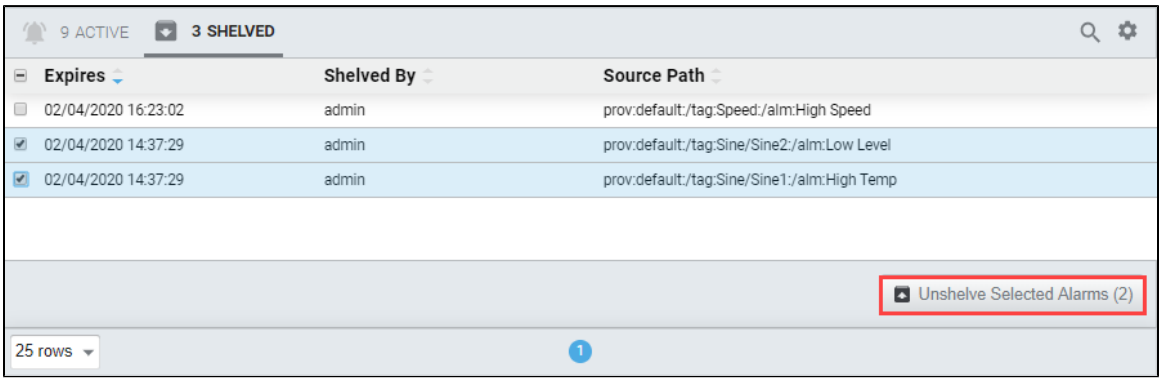
Viewing Shelved Alarms

To see details for the shelved alarms click **Shelved** at the top of the Alarm Status Table. The Shelved tab will display the date/time for when the alarm expires. **Shelved By** and **Source Path** are also displayed for each alarm event. To return to active alarm, click on **Active**.

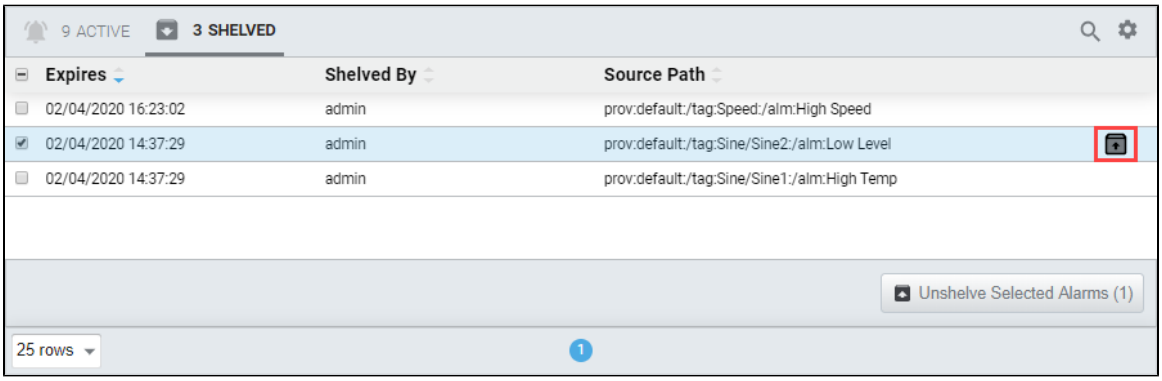


Unshelve Alarms

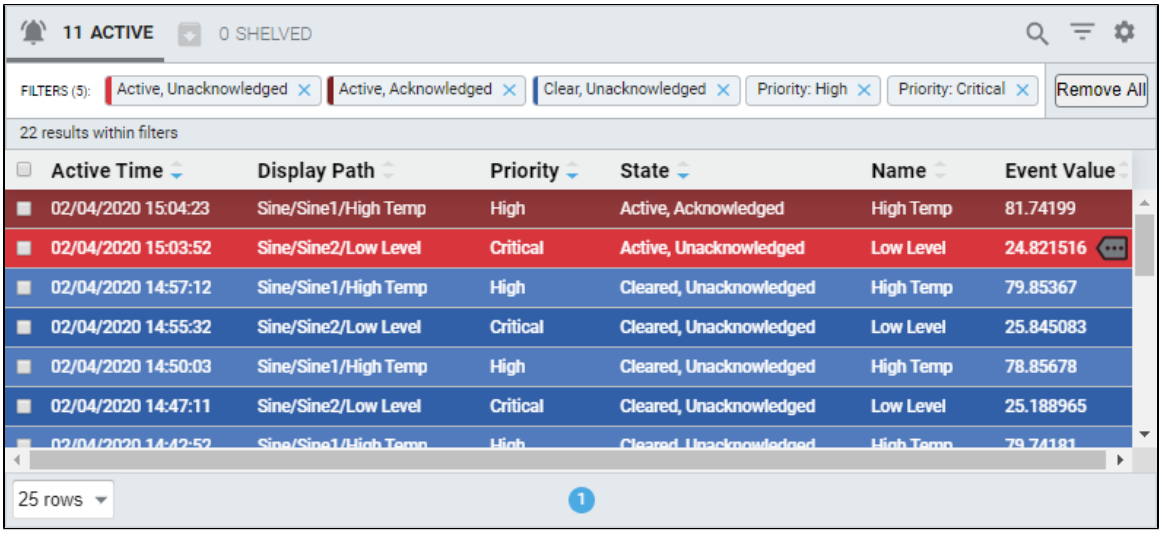
To unshelve alarms, click **Shelved** at the top of the Alarm Status Table. Select one or multiple alarms, and the footer at the bottom of the table will open with the Unshelved Selected Alarms button. Press the **Unshelved Selected Alarms** and the alarm will return back to the Alarm Status list.



Mouse over an alarm event and you'll notice a unshelve icon ( ) on the right side of the alarm. You can delete a single alarm event by clicking the **Unshelve icon** ( ).



After the amount of time expires on a shelved alarm it will be evaluated, and if it is still active, it will automatically return to the Alarm Status list. If the alarm transitions to a cleared state during the time shelved period, the alarm will show up as **'Cleared,Unacknowledged'** in the Alarm Status list instead of **'Active, Unacknowledged'** as shown in the blue rows in the example below.



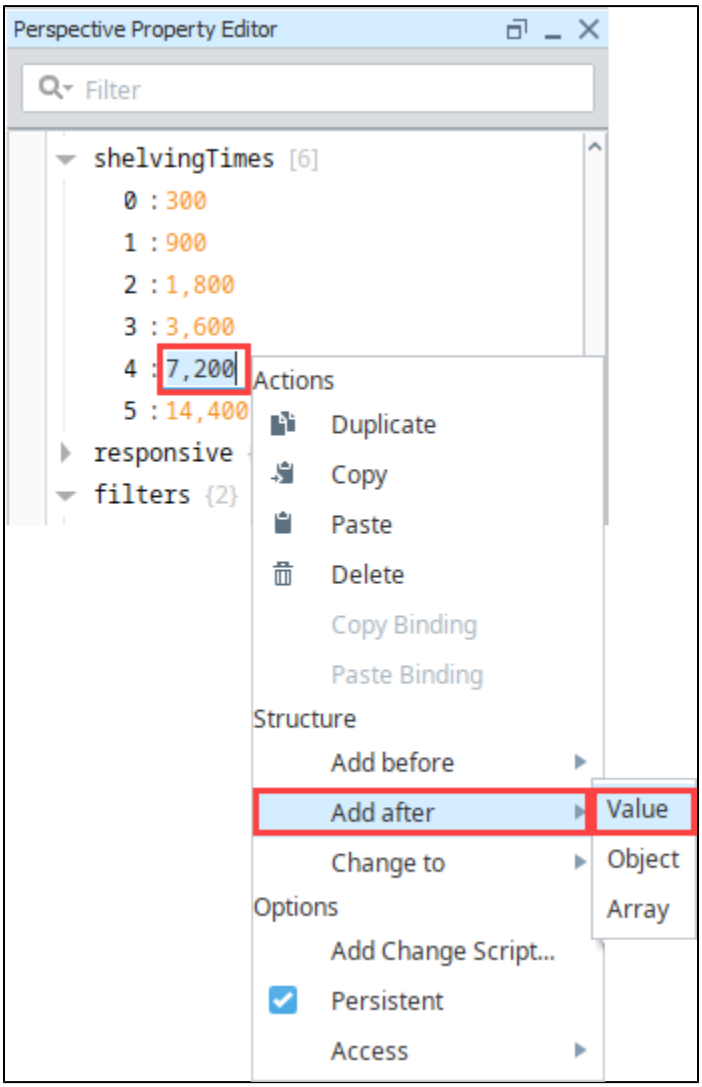
Configuring Custom Shelving Duration

The shelving duration values on the Alarm Status Table component can be customized in the Designer. By default, the Alarm Status Table has several preset shelving duration times, but you can add or remove times based on your requirements. One thing to note when modifying the shelving times is that there is no option to set a shelving time indefinitely. Shelving times are meant to temporarily suppress the alarm while you're working on the issue. They are not meant to be long term.

In the Property Editor, there is a property called **shelvingTimes.** This is where you can add, remove, or change the shelving times for alarms. Shelving time is calculated in seconds.

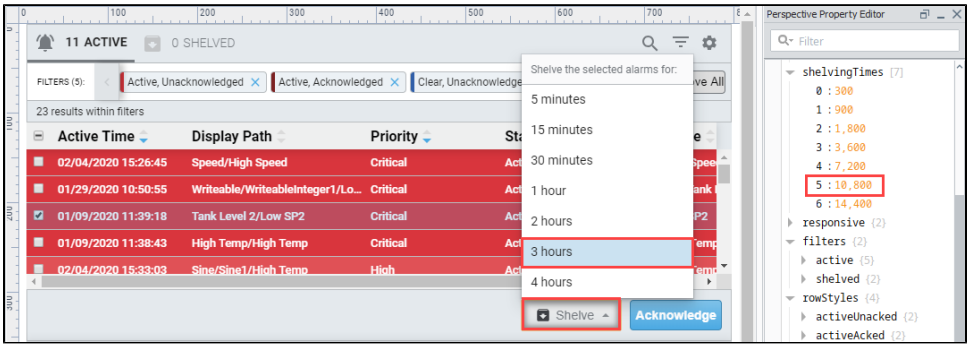
In the following example, let's add **3 hours** to the list of shelving times.

1. Select the Alarm Status Table component. Under the **shelvingTimes** property, insert a **Value** of **10,800** seconds (3 hours).



2. To verify your new time was entered correctly, go to **Preview Mode**, select the alarms you want to shelve, and press the **Shelve** button. A footer will open at the bottom of the table and you should see your new shelve time in the dropdown list (i.e., 3 hours).

3. Click the shelving time to temporarily silence your alarms.



Related Topics ...

- Configuring Alarms

Perspective Alarm Status - Row Styles

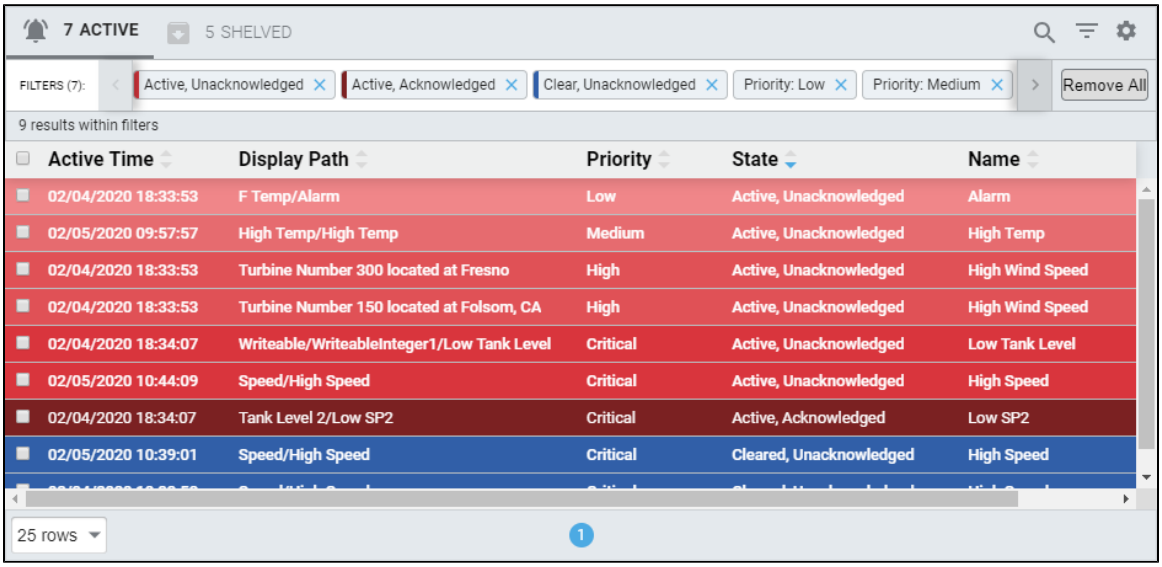
The following feature is new in Ignition version **8.0.3**
Click here to check out the other new features

The Perspective Alarm Status Table comes with a default set of row colors associated with each of the alarm states. Each alarm state has a preset row color, and each of its priorities has a variation of that same color. By default, when you drag a Alarm Status Table into the Designer for the first time, rows are red for Active Unacknowledged alarms events and blue for Cleared Unacknowledged alarm events, as shown in the image below.

On this page

...

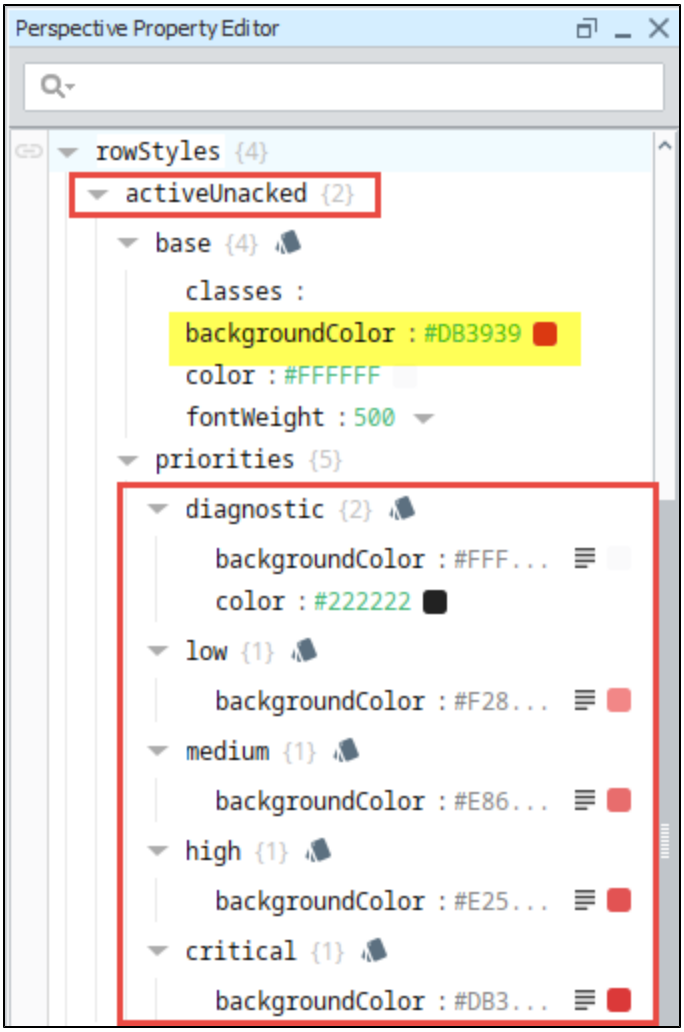
- Customizing Alarm Row Styles
  - Example 1 - Modifying the rowStyle backgroundColor Property for an Alarm State
  - Example 2 - Adding a Style Class to a rowStyle to Make a Critical Priority Alarm Blink



Customizing Alarm Row Styles

In the Property Editor of the Designer you can modify an existing row style, add more styles, or delete a style. You can customize the row styles for any of the alarm states and their priorities. You can even create a style class to set a custom appearance to make a row for a particular state and priority blink.

In the Designer, right click on the Alarm Status Table component, then go to the Property Editor. Expand the **rowStyles** property and also expand each of the priorities for the **activeUnacked** state. You'll notice how each priority has a different shade of red for the **backgroundColor** property, except for diagnostic which is set to white with black text.



Each alarm state has the same default rowStyle properties, but their **backgroundColor** values are different. Here is a list of default properties for each rowStyle alarm state:

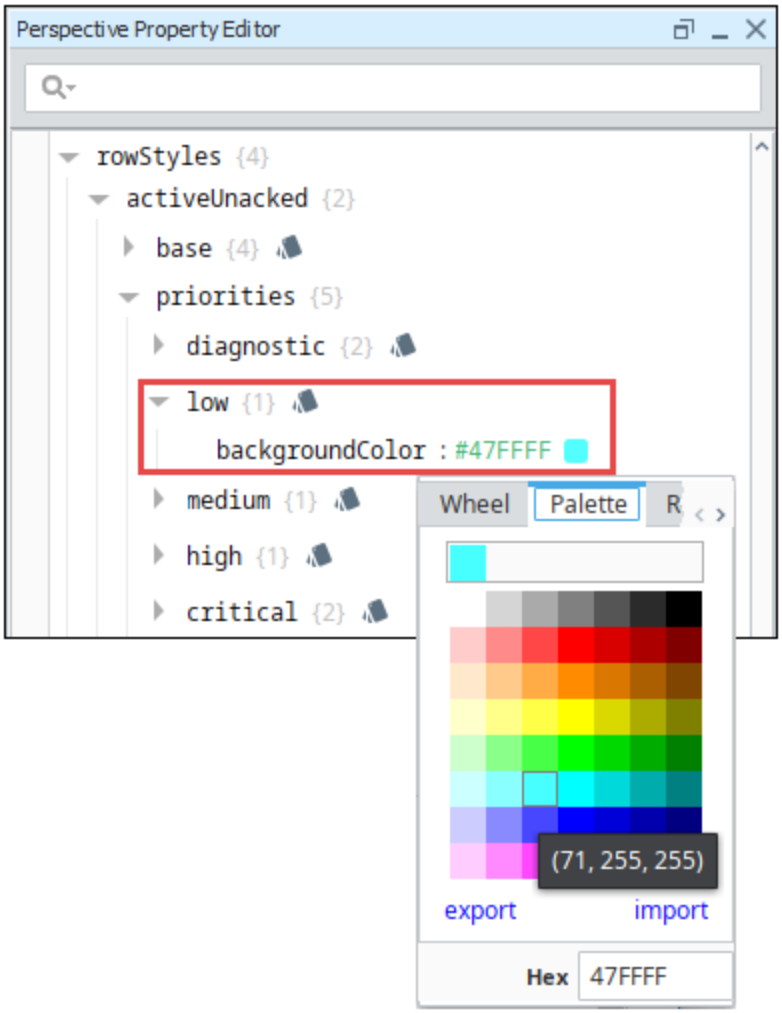
- **base** - Opens a Style Options menu to configure style elements for Text, Background, Margin and Padding, Border, Shape and Misc.
- **classes** - Allows to you set a pre-defined style class.
- **backgroundColor** - Background color for each of the priorites (i.e., diagnostic, low, medium, high, critical) for an alarm state.
- **color** - Color of the Text.
- **fontWeight** - Font weight of the text.

You can modify rowStyles properties in multiple places: using the base (i.e., Style Options menu) or property data types in the Property Editor. When you modify, add, or remove a style property from either Style Options menu or the Property Editor, the change will immediately be visible in both locations, as well as in the table.

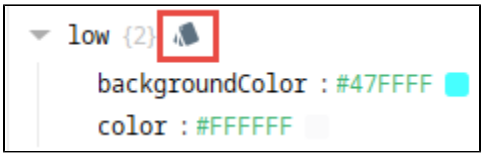
Example 1 - Modifying the rowStyle backgroundColor Property for an Alarm State

Let's change the backgroundColor property for the **Low** priority in the **activeUnacked** state. There are two different ways of changing style properties. Both are documented here.

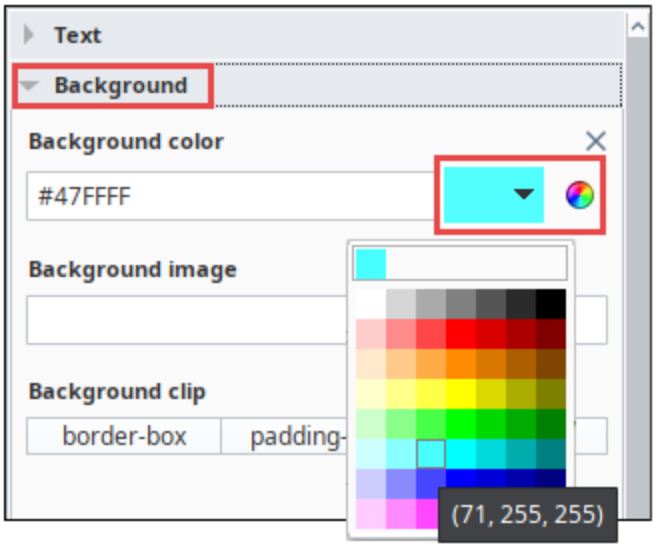
1. In the Designer, select the Alarm Status Table component.
2. Go to the Property Editor, expand **rowStyles** property and the **activeUnacked** state. Also, expand **priorities** and **low** property.
3. The easiest way to change the property is to click on the **backgroundColor** icon. This brings up the Color Selector window. Select one of the four tabs (Wheel, Palette, RGV and HSL) and choose a color.



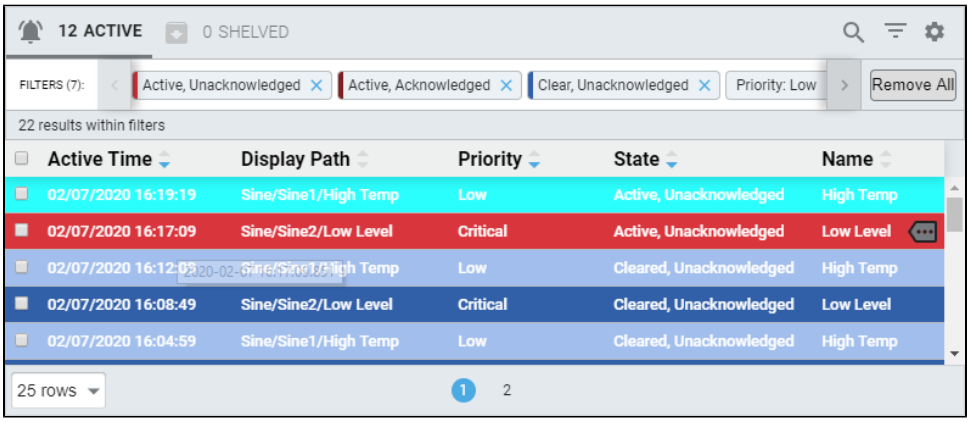
4. Another way to change the backgroundColor in rowStyles is from the Style Options menu.
    a. Simply click on the Modify Style icon next to **low**. This opens the **Style Options** menu.



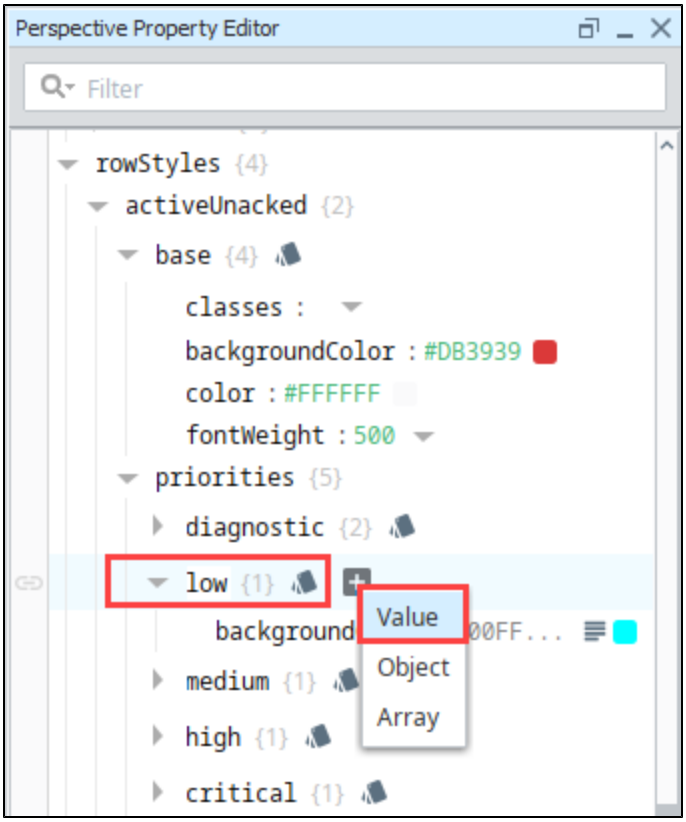
    b. Click the **Background** tab to open a list of background elements.
    c. Click on the dropdown color palette or color wheel. Choose a color.



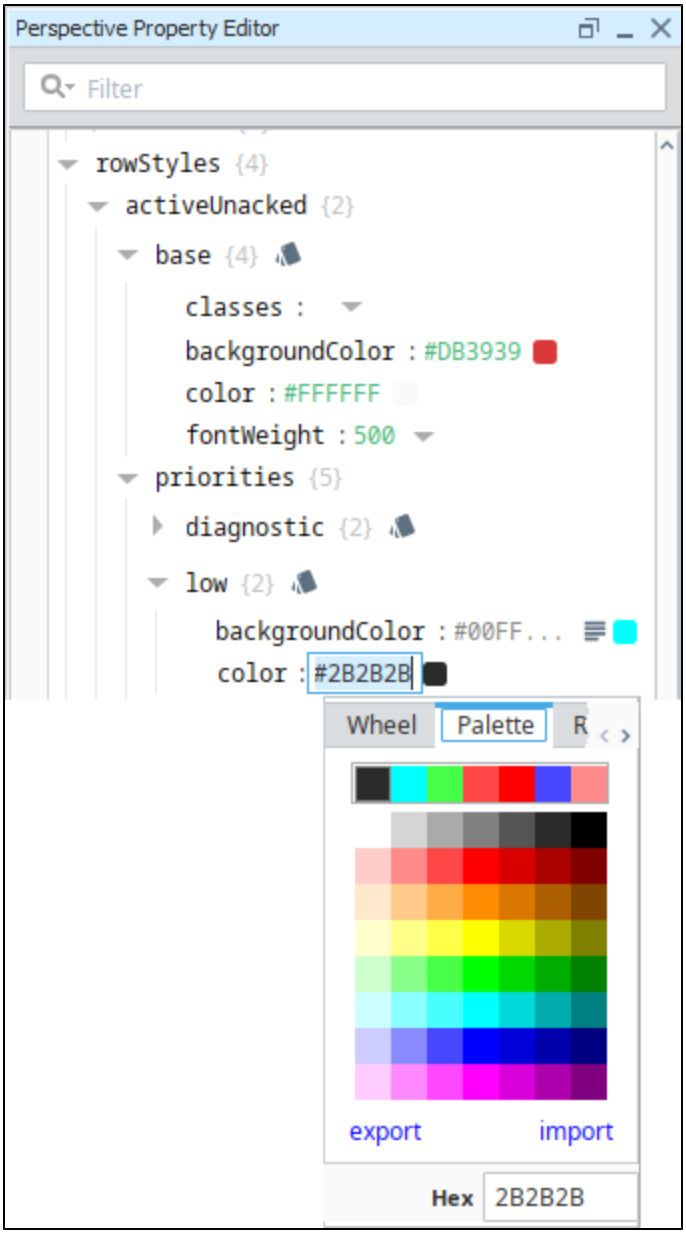
5. The **backgroundColor** for the 'Low' priority is now changed, but it's hard to read. Let's change to the color of the font to black.



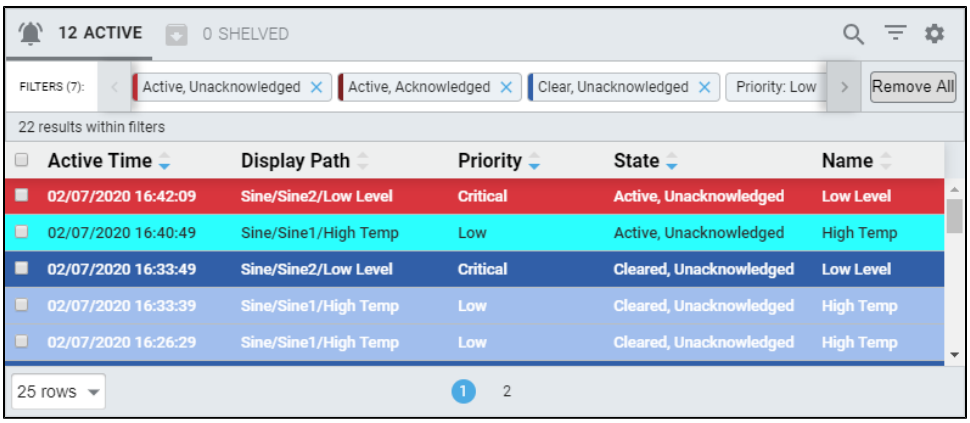
6. Expand **'priorities'**. Mouse over the **'low'** priority, and click the **plus** **icon.**



7. For the **'key'**, enter **'color'** property.
8. Enter a value by clicking the gray square to open the popup color wheel or color palette, then chose the color **'black'**.



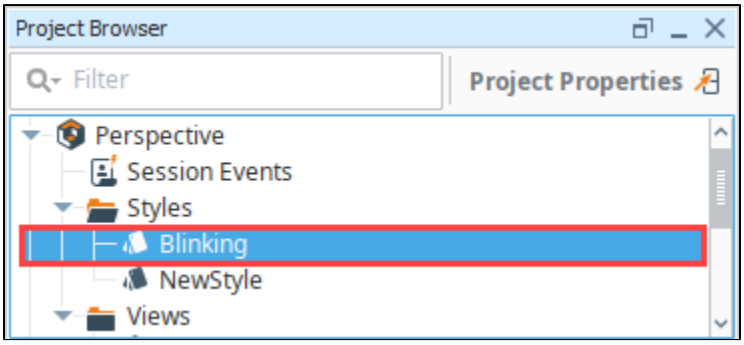
9. The text is now black and readable with the turquoise row style color.



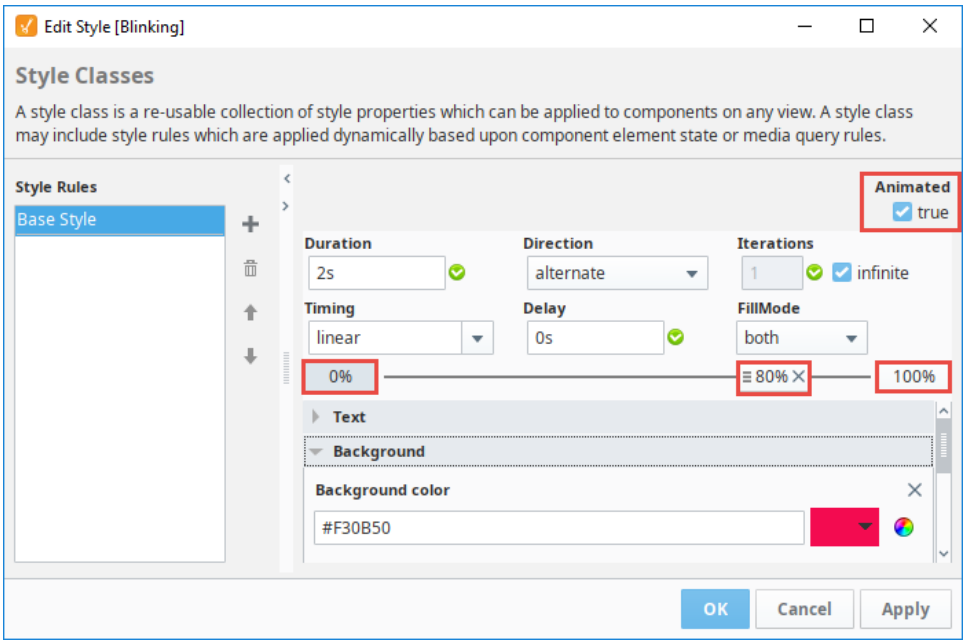
Example 2 - Adding a Style Class to a rowStyle to Make a Critical Priority Alarm Blink

In this example, let's make the activeUnacknowledged Critical property blink between two colors, red and yellow, so it captures the attention of the operator. To do this, we need to create a Style Class.

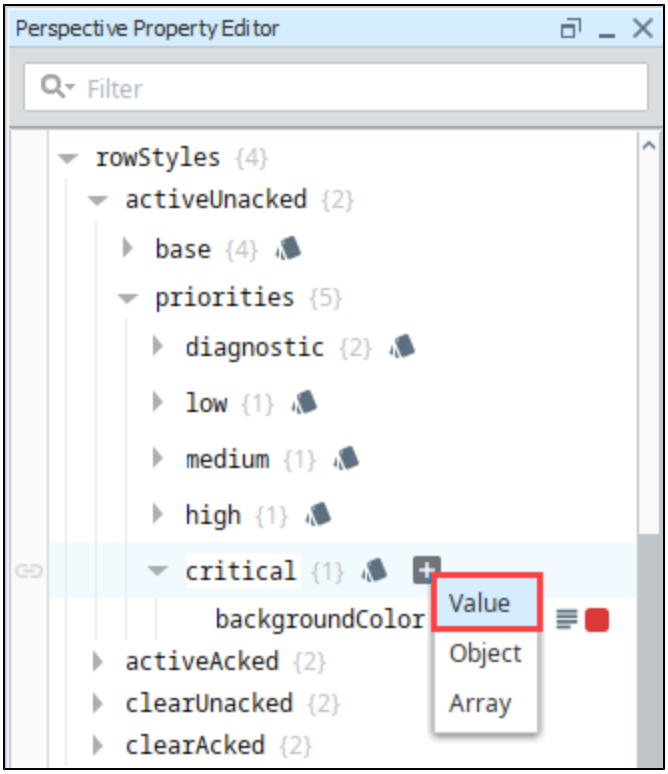
1. In the **Project Browser**, right click on the Styles folder, select **New Style,** and enter a name (i.e, Blinking). Click **OK**.



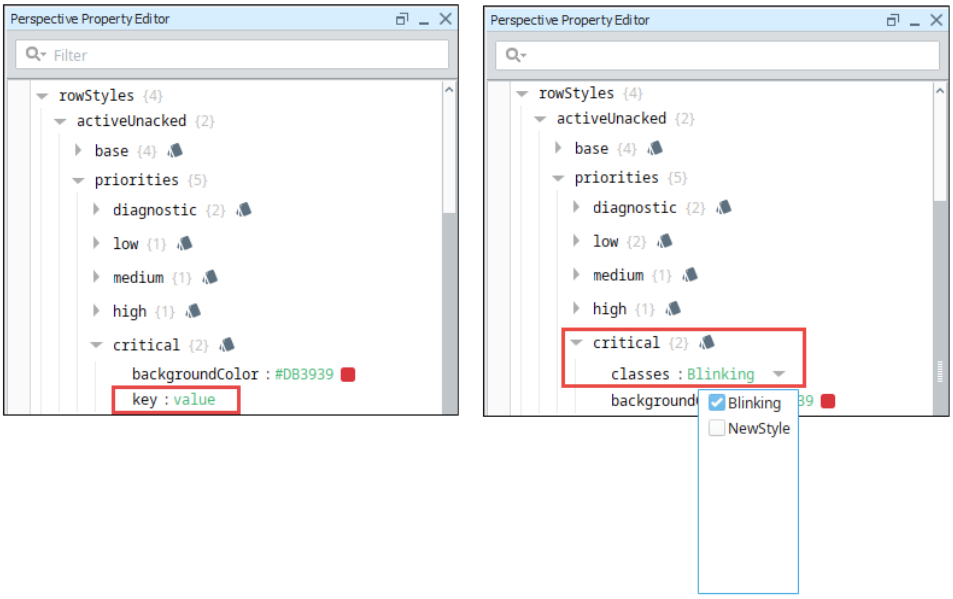
2. Set **Animated** to **'true'** in the upper right corner.
3. Click on **0%** to set the style for the beginning of the animation.
4. Click on **Background** to see Background settings for Critical priority for the activeUnacked state and choose a color.
5. Next, click on **100%** to set the style for the end of the animation.
6. Click on **Background** to expand the Background settings, and choose a color from the either the color palette or color wheel (i.e., yellow). When choosing a second color, make sure it will be readable when it is blinking.
7. Lastly, let's add a stop so that the red rowStyle stays on for a longer period of time than yellow. Right click to add a stop somewhere along the linear line. Drag the stop to about **80%** because red is easier to read than the yellow.
8. Notice the other Animation properties on the Edit Style window. You can set Duration, Direction, number of Iterations, and more. For a detailed description of the Animated properties, refer to Animated Style Classes.



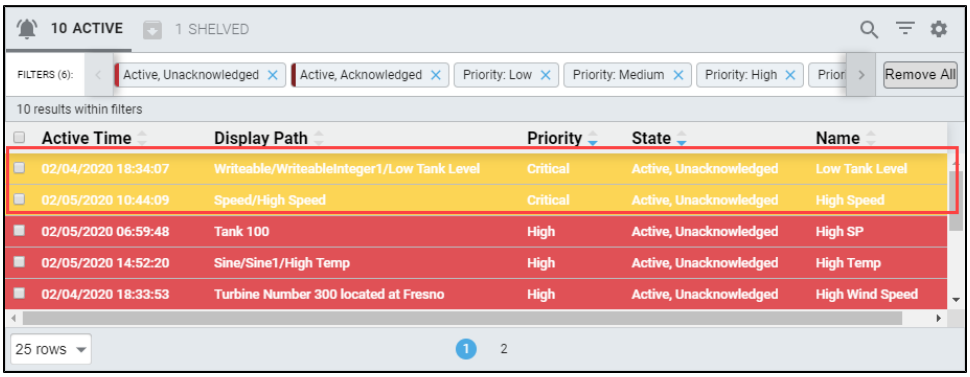
9. Click **OK** to save your new style class.
10. Next, apply the style class to the property. In the Property Editor, expand **rowStyles** and the **activeUnacked** state. Then expand **pri orities** and **critical**.
11. Add a 'value' property data type under the Critical property:
    a. Mouse of the **'critical'** property then click the gray plus sign to add a property.
    b. Choose **value** from the property type popup menu.



c. Click on **key** and change it to **classes** (case sensitive).
d. Click on **value**. Select the **name of your new style class** from the dropdown (i.e., Blinking).



12. Now, all your **critical** alarms for **Active Unacknowledged** will blink red and yellow.



13. When you're finished creating your Style Class and have your rowStyles configured, don't forget to **Save** your project.

Related Topics ...

- Color Selector Reference
- Style Classes
- Style Reference
- Alarm Event Properties Reference

Perspective Alarm Journal Table - Common Tasks

The following feature is new in Ignition version **8.0.8**
Click here to check out the other new features

The Perspective Alarm Journal Table has a number of configuration options that can be used to do things like filter on realtime and historical alarm data, and change how the component displays those alarms. This section describes some of the common tasks for the Alarm Journal Table.

Alarm Journal requires connecting to a database

The Alarm Journal must first be set up with a valid database connection for the Alarm Journal Table to see alarm history from the database.

On this page

...

- Alarm Journal Table - User Interaction
- Perspective Alarm Journal Table - Configuring Properties in the Designer
- Perspective Alarm Journal - Filtering
- Perspective Alarm Journal - Row Styles

Alarm Journal Table - User Interaction

Before you get into the nuts and bolts of how to perform specific tasks outlined on this page, there are a couple things you should know about working with the Alarm Journal Table. The User Interaction page introduces you to the look and feel of the Alarm Journal Table and how to use the basic functionality built in to the table.

Perspective Alarm Journal Table - Configuring Properties in the Designer

The project designer and Ignition administrator have the option to enable or disable properties and setup filtering options in the Property Editor of the Designer based on the project requirements and needs of the client users. This page describes which properties are enabled and which properties have some default filtering options preset. Learn how to configure your own Alarm Journal properties in the Designer on the Configuring Properties in the Designer page.

Perspective Alarm Journal - Filtering

The Perspective Alarm Journal Table component has many built in properties that allow you to filter on various parts of an alarm. Refer to the Perspective Alarm Journal - Filtering page to learn how to use the different built-in filtering options, the search bar to optimized your search results, and the date range feature to view alarm history in the journal table.

Perspective Alarm Journal - Row Styles

The Perspective Alarm Journal Table colors each row a specific color depending on what type of alarm event it is. It uses different styled rows to differentiate between alarms in different state and priorities. These Row Styles can be completely customized using whatever colors and fonts you want for different events. You can also setup new row styles based on other properties and even add a blinking style drawing an operator's attention to critical alarms!

In This Section ...

Perspective Alarm Journal - User Interaction

Getting Started with the Alarm Journal Table

On this page

...

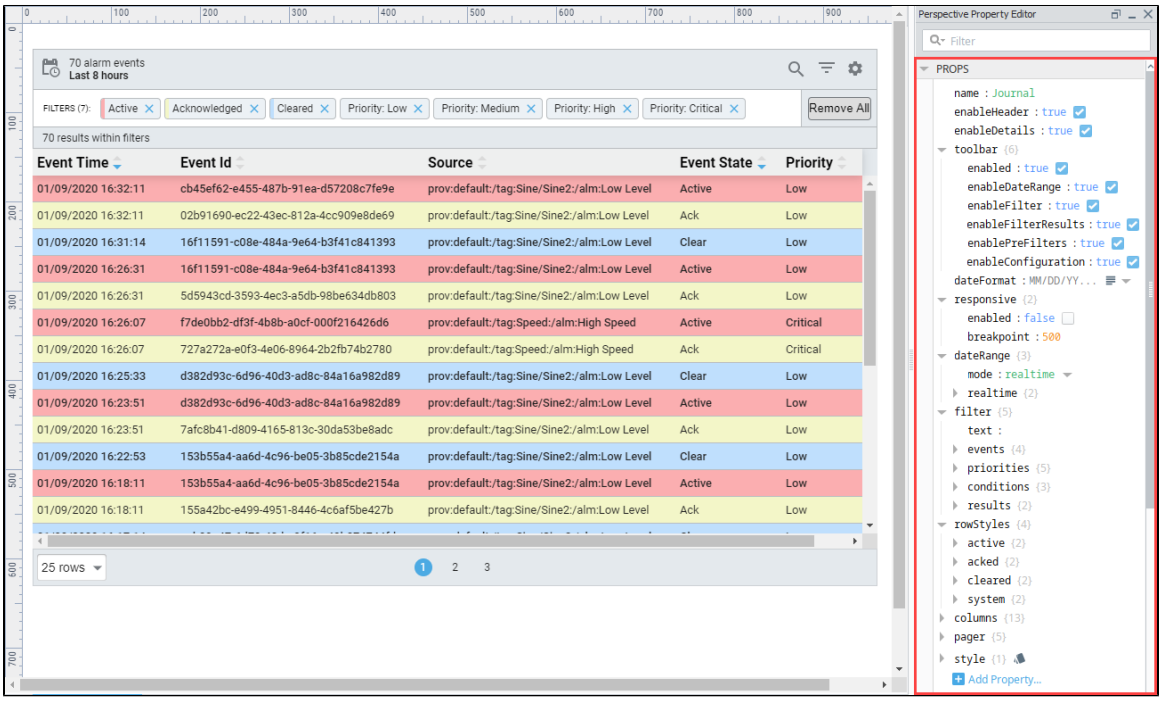
- Getting Started with the Alarm Journal Table
  - Anatomy of an Alarm Journal Table

If you're setting up the Alarm Journal Table for the first time, you probably will want to take a look at the table's properties in the Property Editor. There a number of properties that are enabled by default such as the search toolbar, alarm details popup, and date range to name a few. You should become familiar with them in the event you would like to change them. Other properties have default options already selected for you, like alarm status, alarm priorities and row styles.

When you drag your first Alarm Journal Table component into the Designer workspace, the Alarm Journal Table will automatically populate if you have any alarms configured and an Alarm Journal Profile created. It will look something like the image below. You'll notice it resembles the same layout as the Alarm Status Table and shares the same Configuration settings and filter options. By default, the journal table will show you the last 8 hours of alarm journal data.

You can interface with the journal table in the Designer, in Preview Mode of the Designer, and in a Perspective Session. The Alarm Journal Table properties are configured in the Property Editor of the Designer.

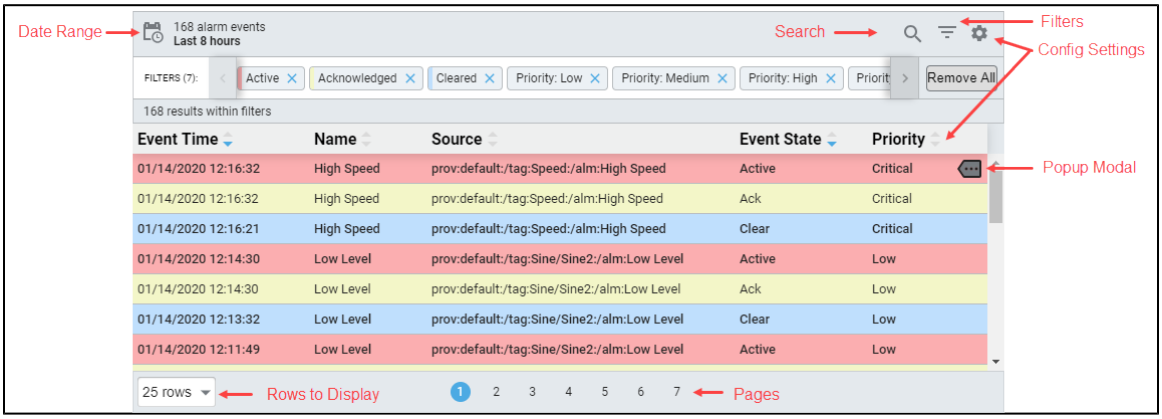
When setting up your Alarm Journal Table for the first time, you will have to toggle between both the Designer and Preview Modes to configure properties and organize how to display your alarm jo urnal data, and how to size the data columns in the table. There is a column property you can set to strictly enforce the column width if you prefer.



Anatomy of an Alarm Journal Table

The Alarm Journal Table may look overwhelming at first, but you will quickly get familiar with it. The following example shows the basic anatomy of an Alarm Journal table with event data populated in the rows. There are a host of Configuration Settings and Filter Settings that you can set to provide relevant information about your alarms events. Use can filter on Realtime or Historical alarms by clicking on the Data Range icon. You can also optimize your search results by using the search feature and entering keywords to help further refine your results. When you hover over a row, a popup modal will appear. By clicking the Popup Modal icon you can see the alarm's details. You can also set the number of alarm rows to display and scroll through the list of pages at the bottom of the journal table. Lastly, the Alarm Journal Table comes with a default set of row colors based on the alarm state and priority which are totally customizable.

To learn more about using the Alarm Journal functions and features, refer to the filtering and row styles pages in this section. The Alarm Journal Table component page describes all the journal properties and how to use them.



Perspective Alarm Journal - Configuring Properties in Designer

The following feature is new in Ignition version **8.0.3**
Click here to check out the other new features

Alarm Journal Profile

To view alarm history, an Alarm Journal Profile and a valid connection to a database must be created first before logging alarms.

On this page

...

- Getting Started
- Configuring Properties in the Designer
  - Filter Properties
  - Condition Properties
  - Row Styles Properties
  - Column Properties

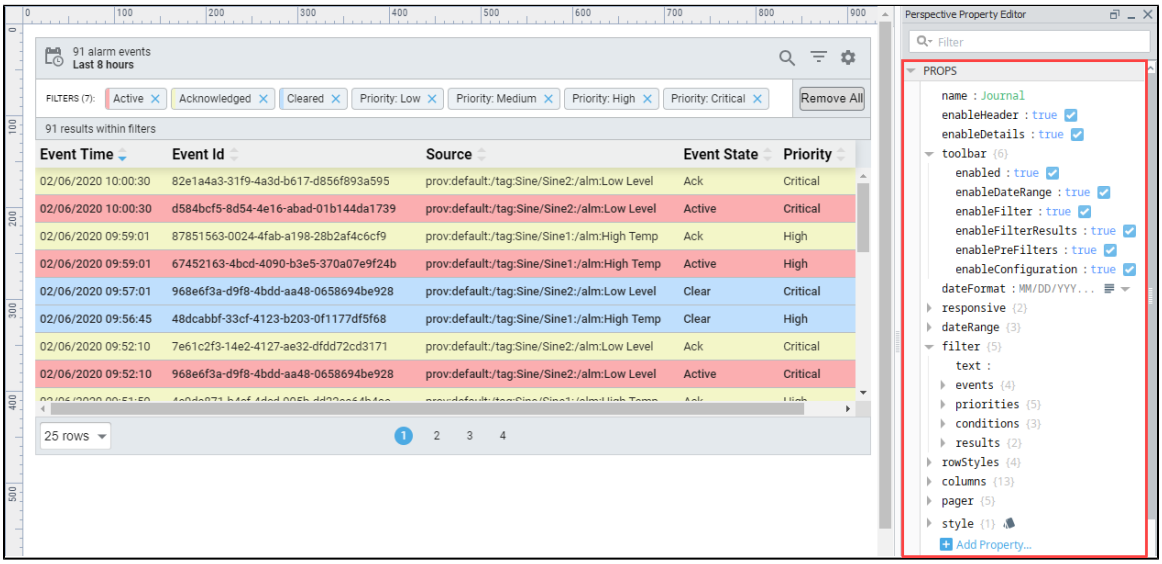
Getting Started

Filtering can be done in both the Designer and in a Perspective Session, but enabling and disabling Alarm Journal Table functions and setting up specified filtering options for client users is configured in the Designer. The project designer and Ignition administrator has the option to enable or disable properties and setup filtering options in the Property Editor of the Designer based on the project requirements and needs of the client users. If you are the project designer or Ignition administrator, it's a good idea to check out all the Alarm JournalTable properties in the Perspective Property Editor. Scroll through all the alarm properties and expand them to see additional properties and see the default property settings and filtering options.

The Alarm Journal Table has some properties enabled by default, and some filter options have already been selected in the Designer to give client users a head start using the table. It's in the Property Editor of the Designer where table properties are enabled and disabled, and filtering options are configured such as the alarm states and priorities to display, setting the shelving times, defining row styles, choosing column headers and column sort options, or setting up a specific display path and source path for the operators to view.

The first time an Alarm Journal Table component is dragged on to a window in the Designer, by default, the table displays all alarm events for the last 8 hours that are in an **Active**, **Acknowledged** and **Cleared** state with a priority of **Low**, **Medium**, **High** and **Critical**.

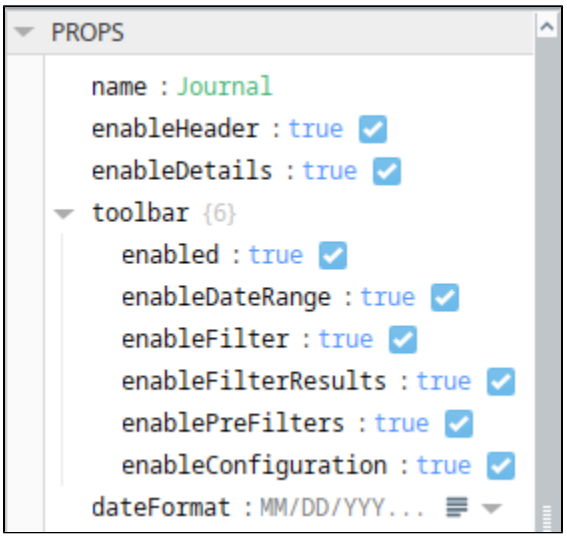
The Alarm Journal Table below is similar to what you'll see when you first drag it into a Designer window.



Configuring Properties in the Designer

When you drag a Alarm Journal Table component into the Designer for the first time, it displays the alarm history with default properties already configured and some filtering options already preselected for you. Some of the default settings may work for your client users, but others may need to be modified in the Property Editor.

As the project designer, the first thing you might want to look at is the functionality that is enabled on the journal table component.

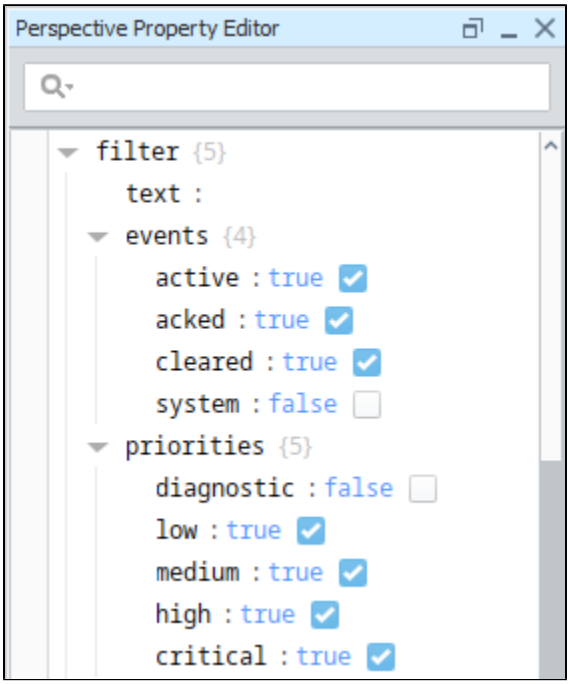


Preview Mode

When configuring the Alarm Journal Table in the Designer, you have to go to Preview Mode to see your updates, filter, sort and organize the alarm data in the table. When making modfications to the properties in the Property Editor, you will find yourself toggling between the Designer and Preview Modes multiple times to make sure your modifications are displayed the way you want them. Each time you reset your filter options and column headers in the Designer, the journal table will refresh with new alarm data based on your configuration and filter settings.

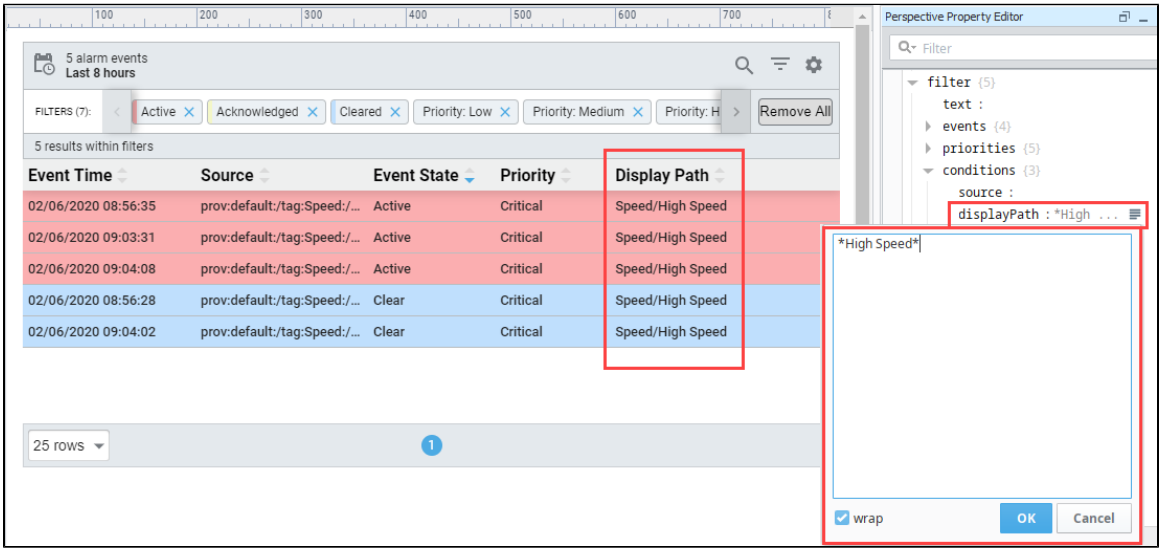
Filter Properties

While in the Property Editor, expand all the table's properties to see what **filter** properties are configured. The first thing you might want to look at is what **events** settings are set. **Active**, **Acked**, and **Cleared** are set by default, but you'll need to decide if these settings work for your operators. Next, look at the alarm priorities and verify if **Low**, **Medium**, **High**, and **Critical** priorities are the alarm events you want your operators to monitor.



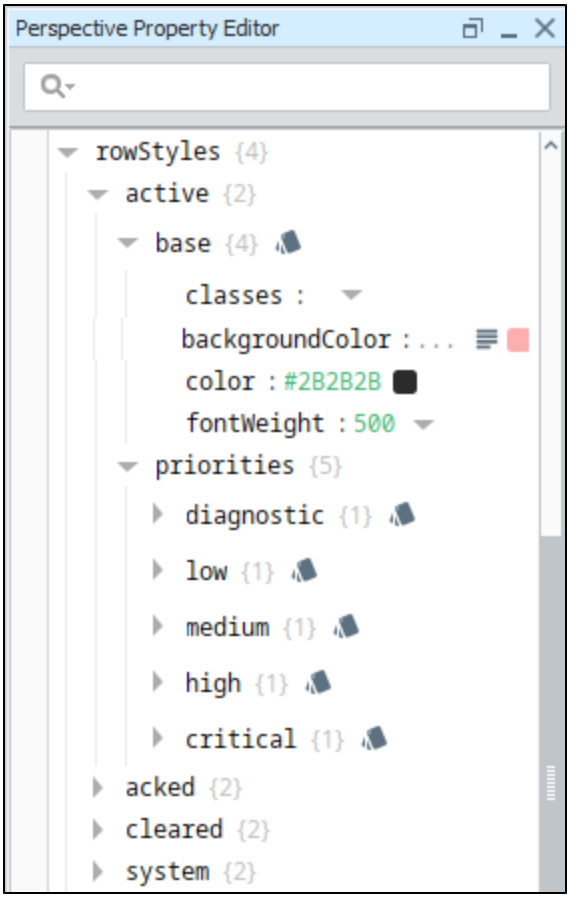
Condition Properties

As the project designer, you might want to control what **source path**, **display path**, and/or **tag provider** you want your operators monitoring. Under the **conditions** property, you can set these values so operators are not navigating to other source paths, display paths or tag providers other than what you configure. In the following example, wildcards were used to search on 'High Speed' in the Display Path property.



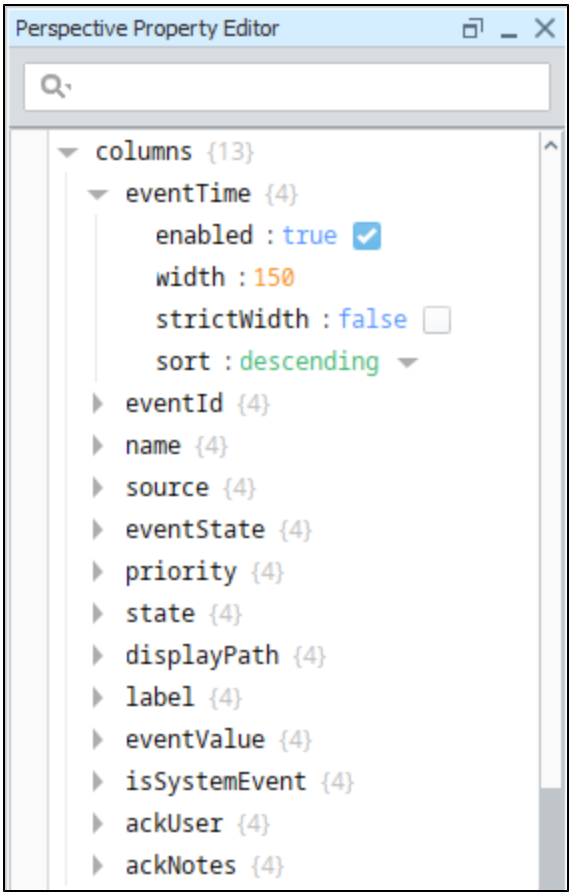
Row Styles Properties

There are some preset row styles background color configured for each alarm state. Open each alarm state property to see its default color. These properties are easy to modify using the color palette. You also have the option to create your own row style for each priority for each alarm state. It's best not too have too many colors and to have a standard set of colors so operators can quickly identify alarms they need to respond to quickly.



Column Properties

The column properties are the Configuration Settings  or column headings on the journal table. By default, all the column headings are enabled so client users can pick and choose which headings or type of alarm data they want to display. Each column heading property has four subproperties: enabled, column width, strict width, and sort order. The designer can configure the column settings for client users, but it's recommended to give client users flexibility to display alarm data in the manner that helps them perform their jobs.



To learn more about alarm journal properties, refer to the Perspective - Alarm Journal Table page.

Related Topics ...

- Perspective - Alarm Status Table
- Configuring Alarms

Perspective Alarm Journal - Filtering

The following feature is new in Ignition version **8.0.8**
Click here to check out the other new features

Alarm Journal Profile

To view alarm history, an Alarm Journal Profile and a valid connection to a database must be created first before logging alarms.

On this page

...

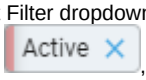
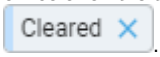
- Overview
- General Filtering
  - Using the Search Bar
  - Filtering on Date Range
- Viewing and Sorting on Alarm Data
- Sorting Alarm Journal Data
- Viewing Alarm Details
  - Viewing All Instances of an Alarm
  - Viewing the Source Path of an Alarm

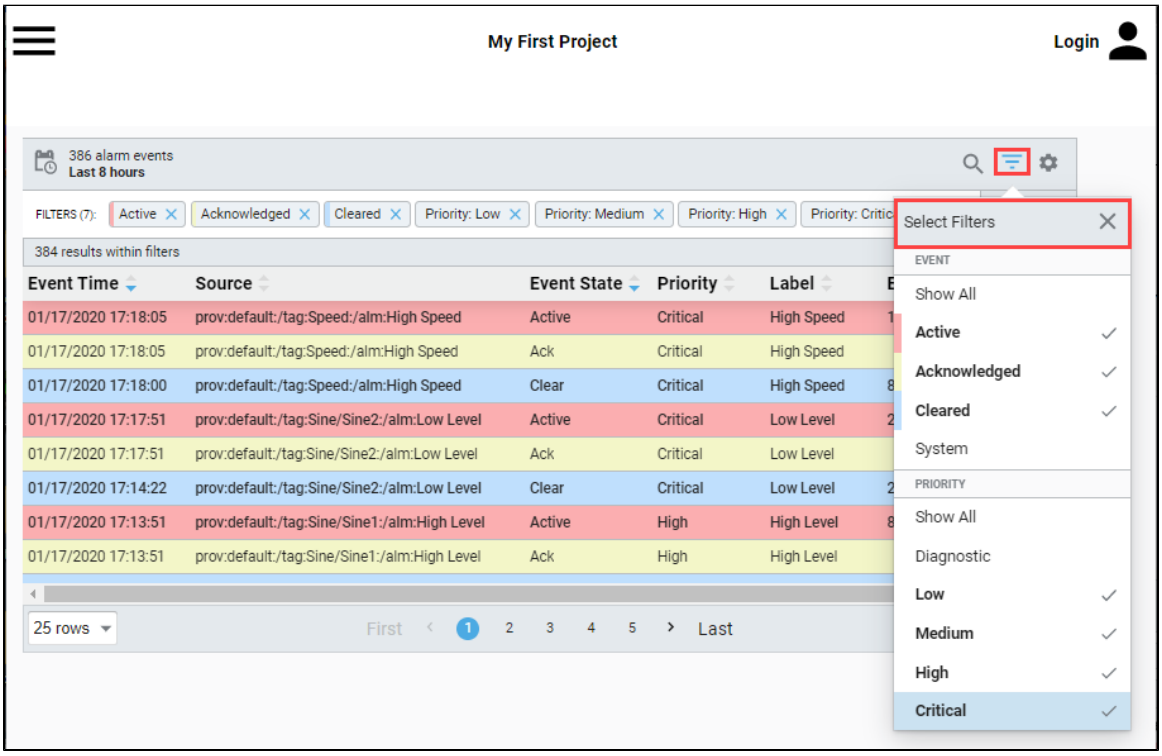
Overview

The Alarm Journal Table has many built-in properties that allow you to filter on various parts of an alarm in a Perspective Session. It provides a host of built-in filter options that are immediately available and easy to modify to help you get started. You can filter on Realtime and Historical alarm events using the Data Range selector  . The Search Bar  lets you further optimized your search results. You can even organize the alarm data to show or hide columns, reorganize and resize columns, and change the sort order to ascending or descending.

General Filtering

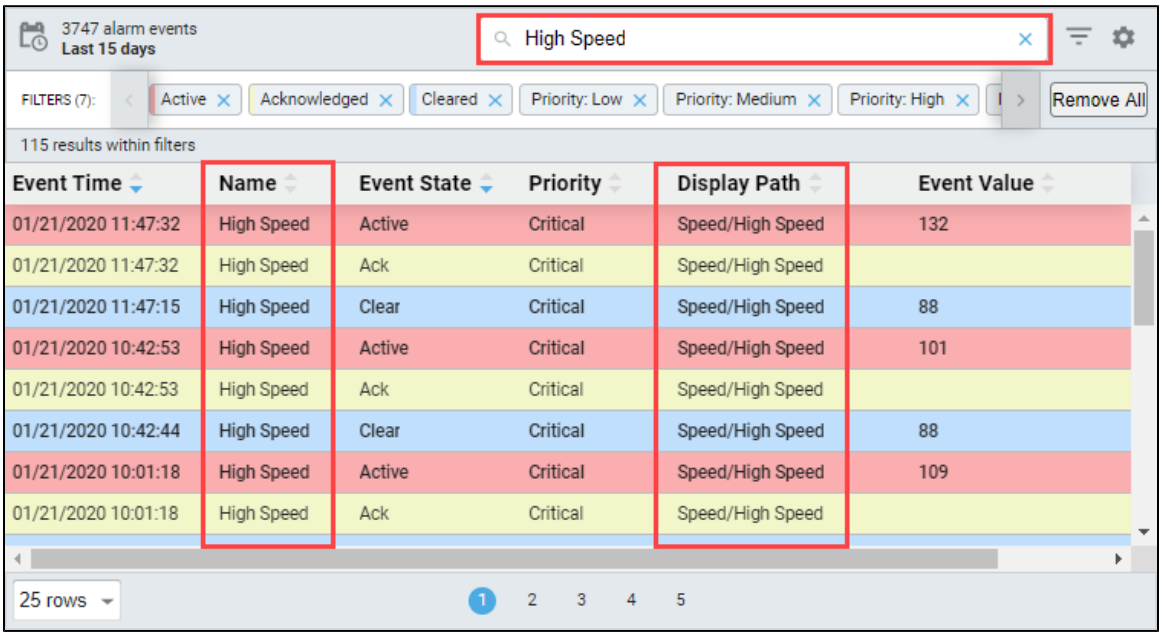
When you use the Alarm Journal Table for the first time in a Perspective Session, you could be using the default journal properties, or your project designer or system administrator may have preconfigured some journal properties specific to your project. Either way, an operator can easily choose and filter the alarm data and how they want it displayed.

By default, the Alarm Journal Table displays all alarm events for the last 8 hour and displays event alarms that are in an **Active**, **Acknowledged** and **Cleared** state with a priority of **Low**, **Medium**, **High** and **Critical**. You can easily change these filter options by clicking on the **Filter icon**  and adding or removing a filter option from the dropdown by checking or unchecking a filter option. Notice that there is a filter bar at the top of the table that also displays all the filter options that are currently set. You can remove any of these filter options by clicking the **'X'** on the right side of the option, but to add a filter, use the Select Filter dropdown  . Notice how the **'state'** filter options show the row color on the tab for Active  , Acknowledged  and Cleared  .



Using the Search Bar

You can optimize your filtered results using the Search Bar. Click on the **Search icon** and create your own search string to find specific alarms, events, and conditions.

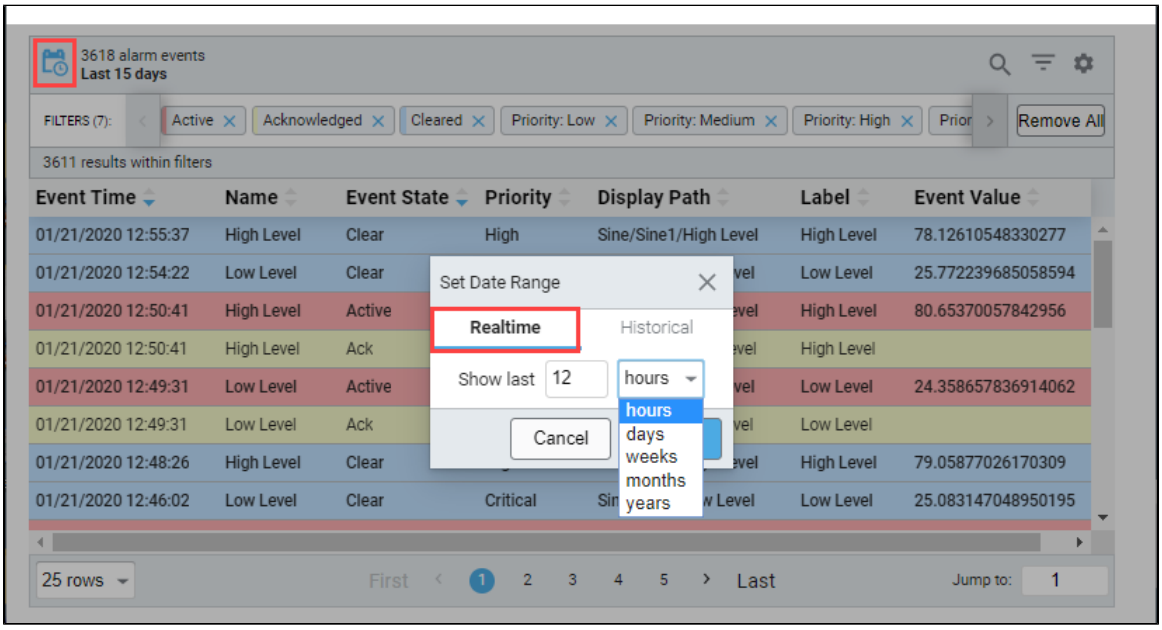


Filtering on Date Range

The Alarm Journal Table can filter for alarm events in either Realtime or Historical. To filter on Realtime or Historical alarm events, select the **Date Range icon** .

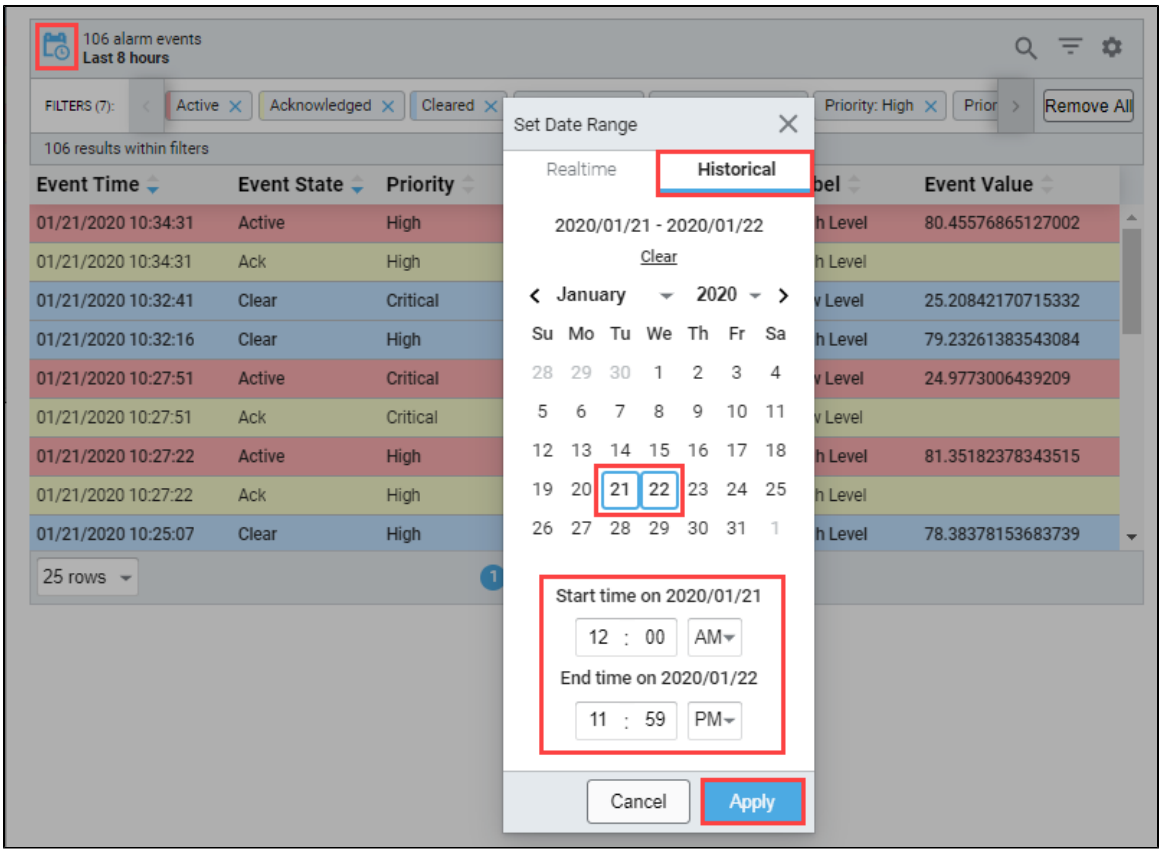
Realtime

You can filter on Realtime alarm events using the **Data Range** feature. Simply enter select Realtime, and enter the amount of time in hours, days, weeks, months, or years of the alarm events to display. Then click **Apply**.



Historical

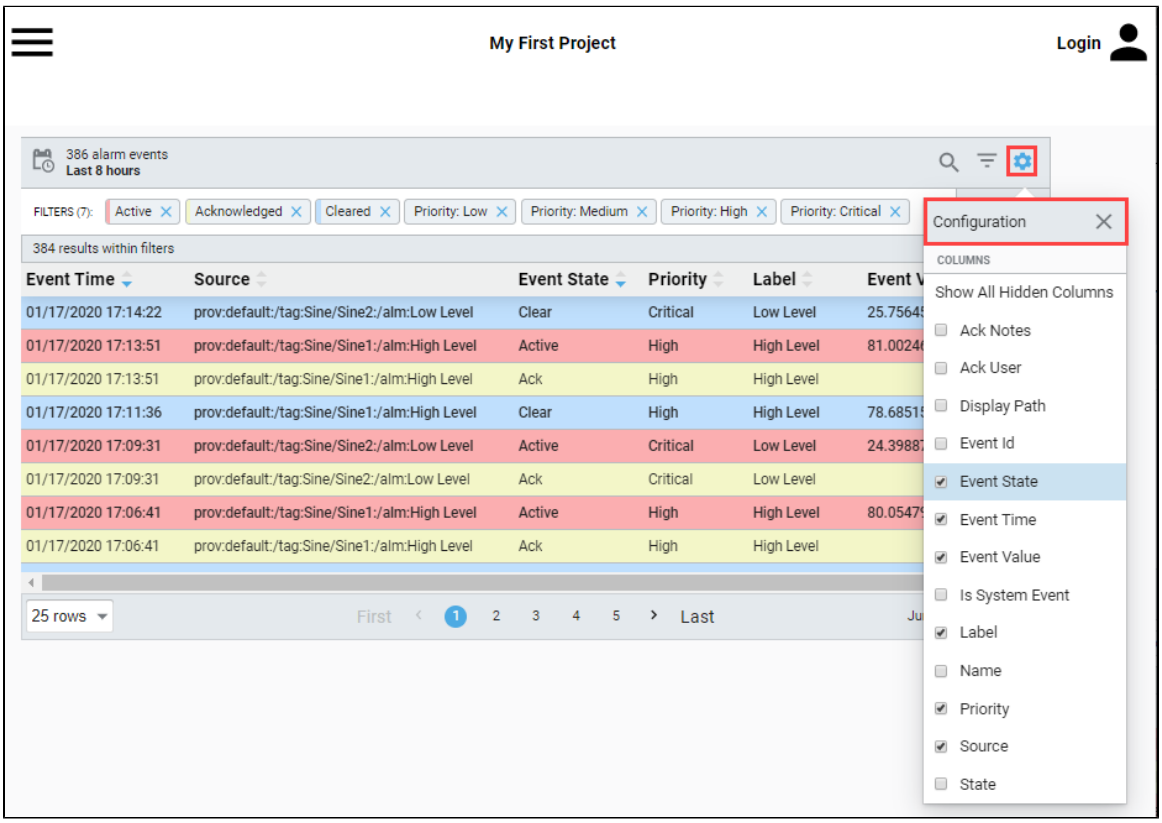
The Alarm Journal Table displays the complete history of all alarms within a specific time period that you set. Typically, operators want to filter alarm events within a specific time period so they can narrow down the number of alarm events that need to be viewed. Select **Historical**, enter the date range by clicking on the **Start Date** and **End Date**, then select the **Start Time** and **End Time**, and click **Apply**.



Viewing and Sorting on Alarm Data

There are a host of configuration settings to choose from in which to display alarm event data. In a Perspective Session, operators can choose to display or hide alarm data in the table by checking or unchecking any of the **Configuration Settings** or by right clicking in the header row of the table. It's super easy to filter and display alarm event data using the configuration options provided in the column header of the Alarm Status Table. You can do a multi-column sort by holding down **Ctrl + Shift** and clicking on the column headers you want to sort by.

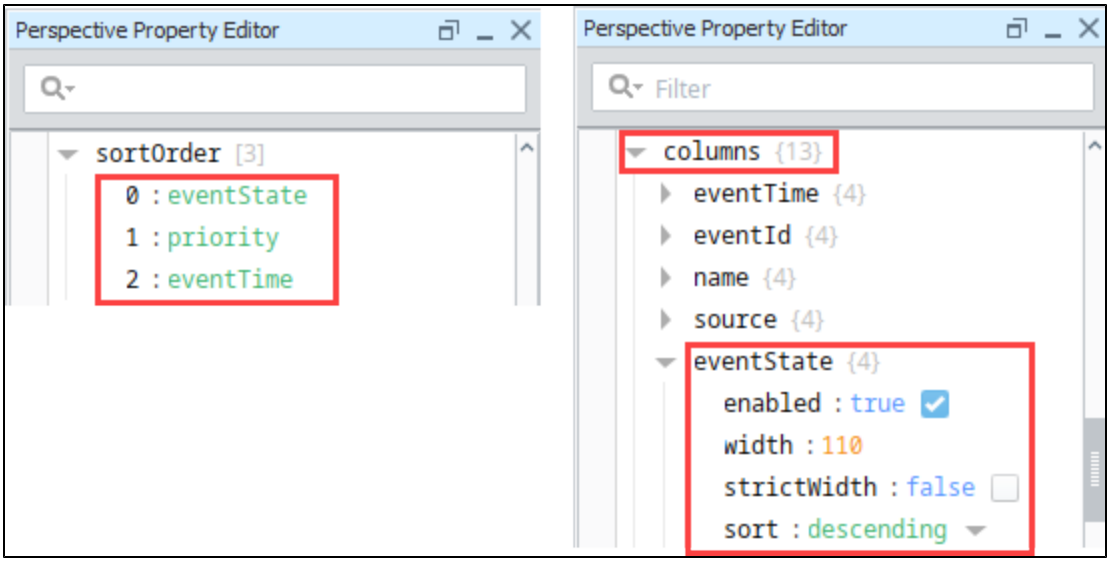
If you need to align the columns, simply put your cursor to the left of the column header and drag left or right. There is a **'strictWidth'** property for each header column that can be configured in the Designer to strictly enforce the column width.



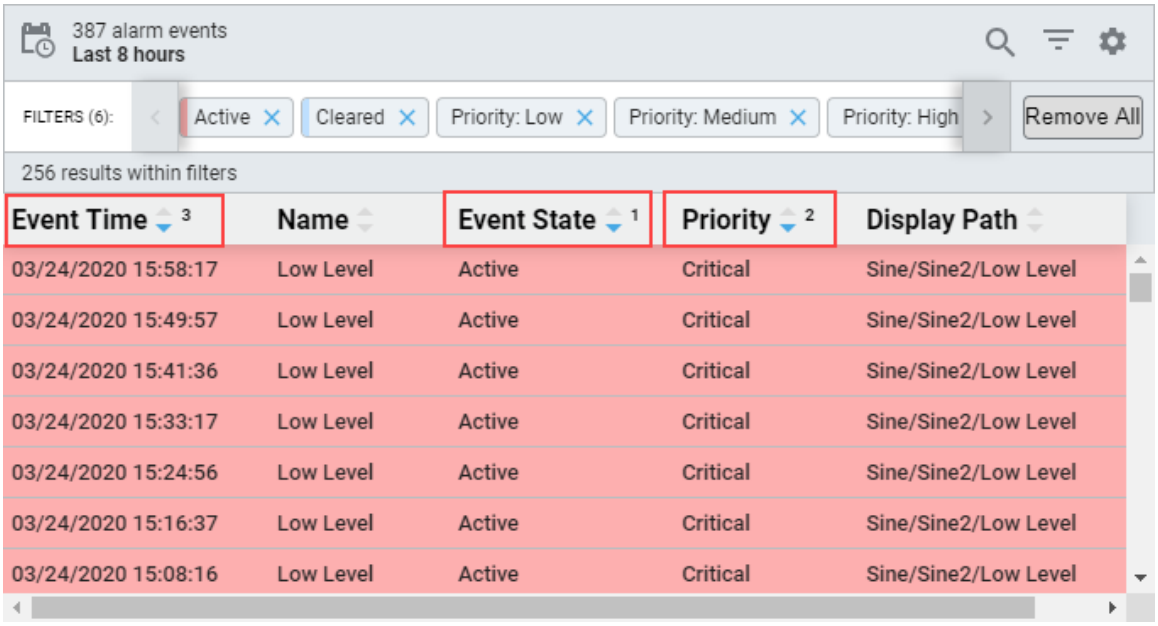
The following feature is new in Ignition version **8.0.11**
Click here to check out the other new features

Sorting Alarm Journal Data

Perspective's Alarm Journal Table now supports multiple sort orders. The order is first determined by the sort order properties on the Alarm Journal Table. The new sort order property is '`activeSortOrder.`' Each column that you add to the sort order property will also need to have a sort defined under the columns property as either ascending or descending.



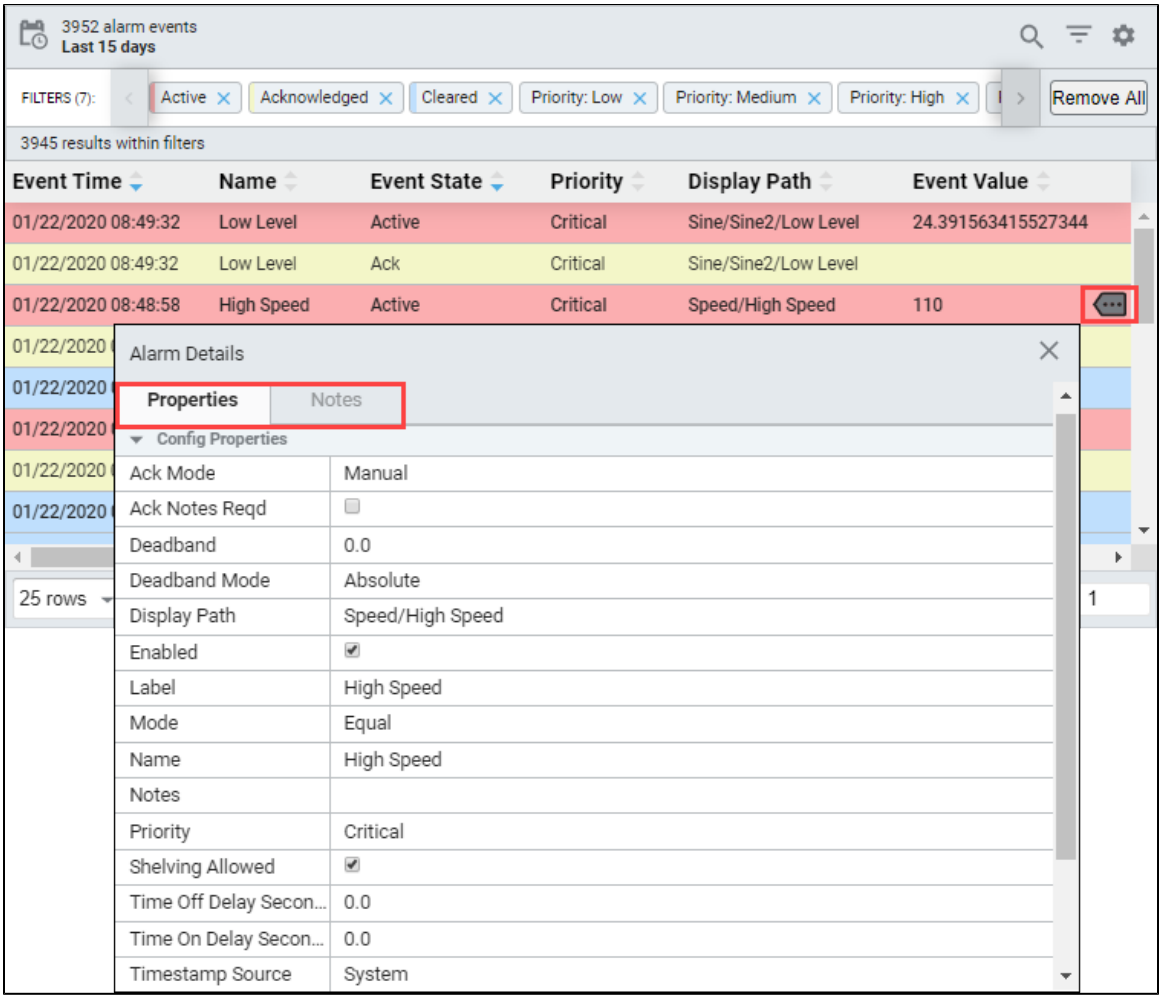
You also need to make sure the configuration columns you choose for your sort are displayed on your table. In the following image, you can see that the sort order is **Event State** - **Priority** - **Event Time**. Notice how each column is numbered according to the order you defined in your `activeSortOrder` property.



Columns will first sort by anything defined in `activeSortOrder` property and then fallback to the other sorts defined in the columns configuration.

Viewing Alarm Details

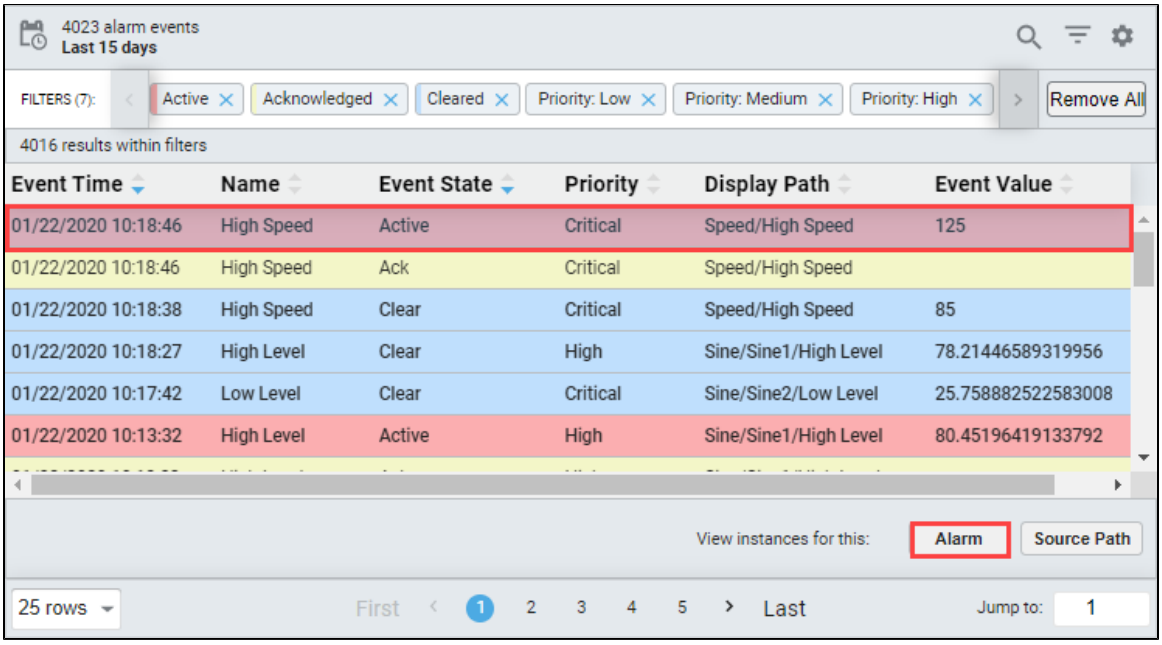
If you want to find out more details about an alarm event, simply mouse over an alarm event and click on the popup modal that will appear on the right side of table. This opens an Alarm Detail window that displays the alarm properties and any notes that were entered by an operator.



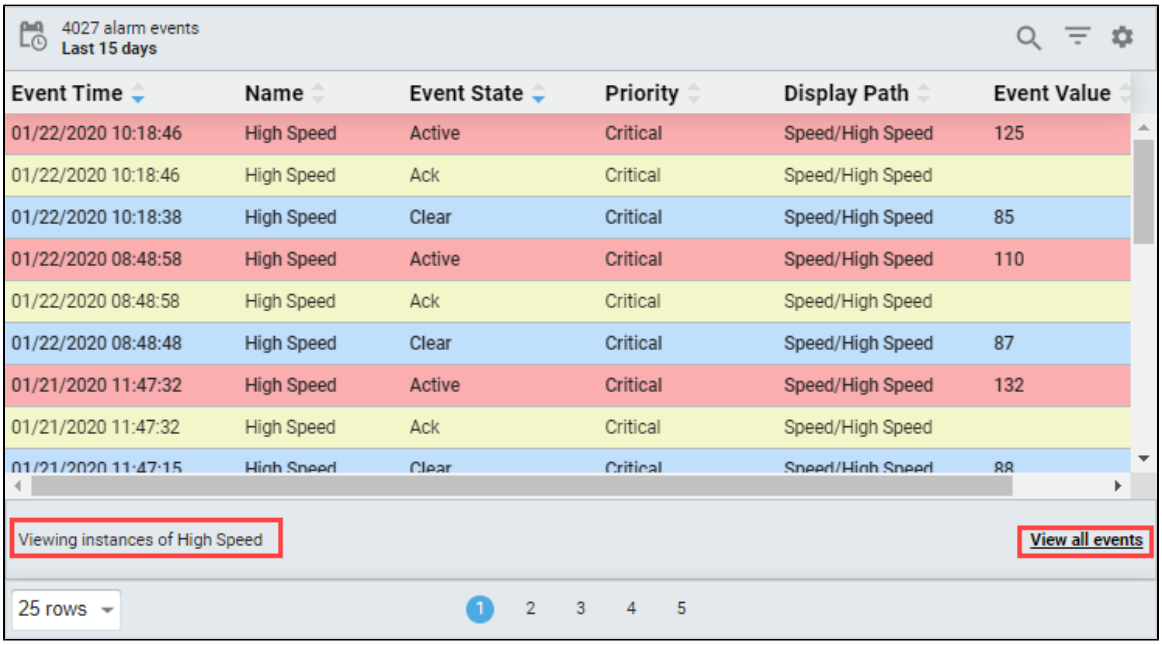
Viewing All Instances of an Alarm

If you select an alarm, a window will appear at the bottom of the journal table where you can choose to view instances of that specific alarm or its source path. When you click on the **Alarm** button, the journal table will refresh with instances of the selected alarm.

In this example, High Speed alarm was selected to view all instances of the alarm.

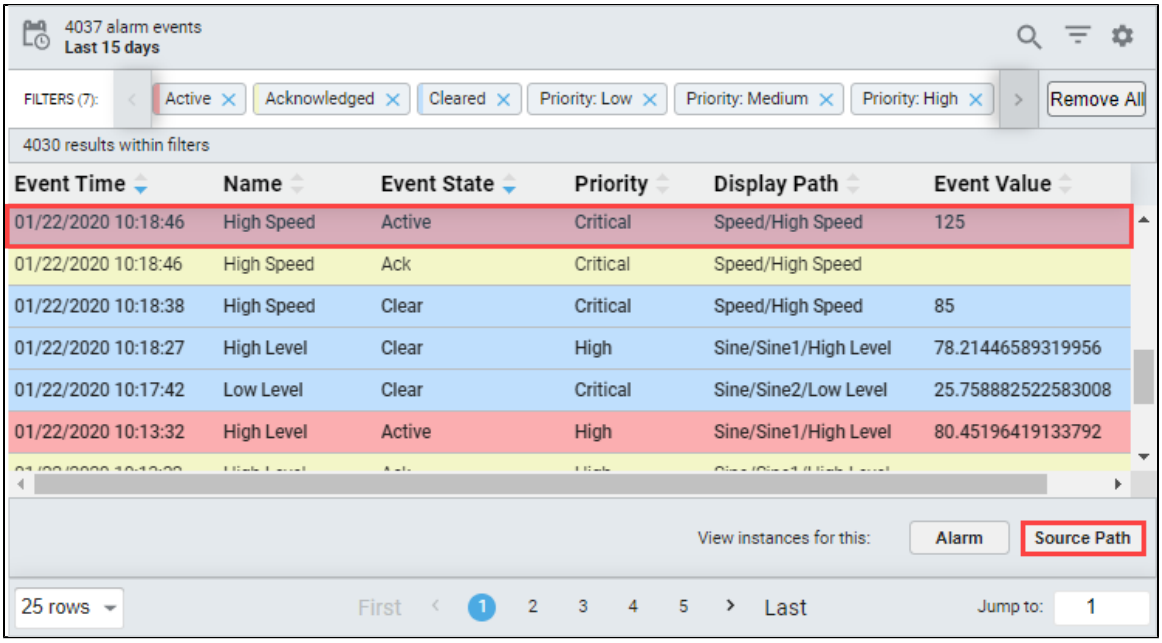


Click the **View all events** link to return to all alarm events on the journal table.

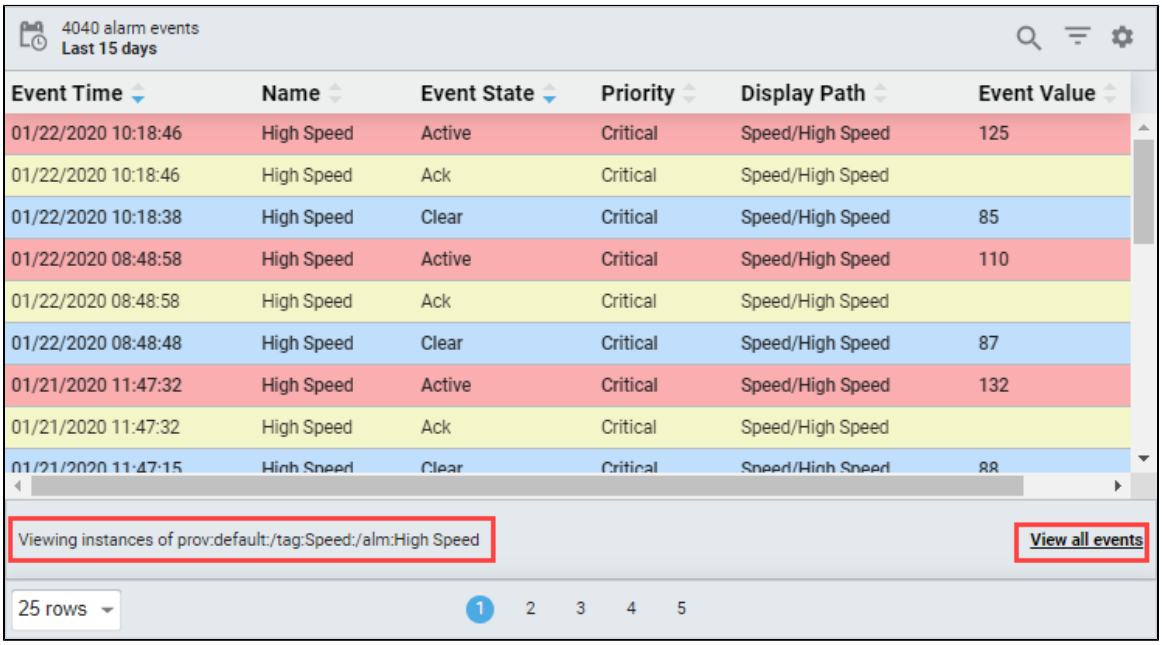


Viewing the Source Path of an Alarm

You can also click the **Source Path** button to display the source path of the selected alarm.



Click the **View all events** link to return to the alarm events on the journal table.



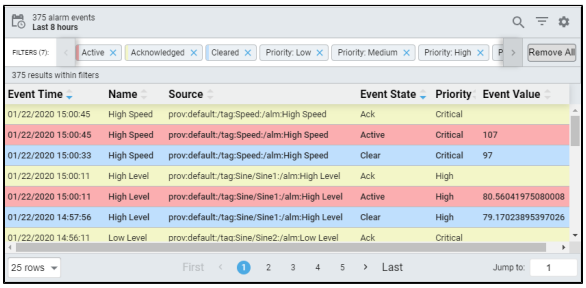
Related Topics ...

- Configuring Alarms
- Perspective - Alarm Journal Table

Perspective Alarm Journal - Row Styles

The following feature is new in Ignition version **8.0.8**
Click here to check out the other new features

The Perspective Alarm Journal Table allows you to customize row styles for different states and priorities of alarm history. Just like the Alarm Status Table, the Alarm Journal Table is preconfigured with a default set of colors under the **rowStyles** property for each of the alarm event states, and immediately ready for you to use. The default row colors are pink for Active, yellow for Acknowledged, blue for Clear, and white for System alarms events.



On this page

...

- Customizing Row Styles for the Different Alarm States
  - Example 1 - Modifying the rowStyle backgroundColor Property for an Alarm State
  - Example 2 - Add a Style Class to a rowStyle

INDUCTIVE UNIVERSITY

Alarm Journal - Row Styles

Watch the Video

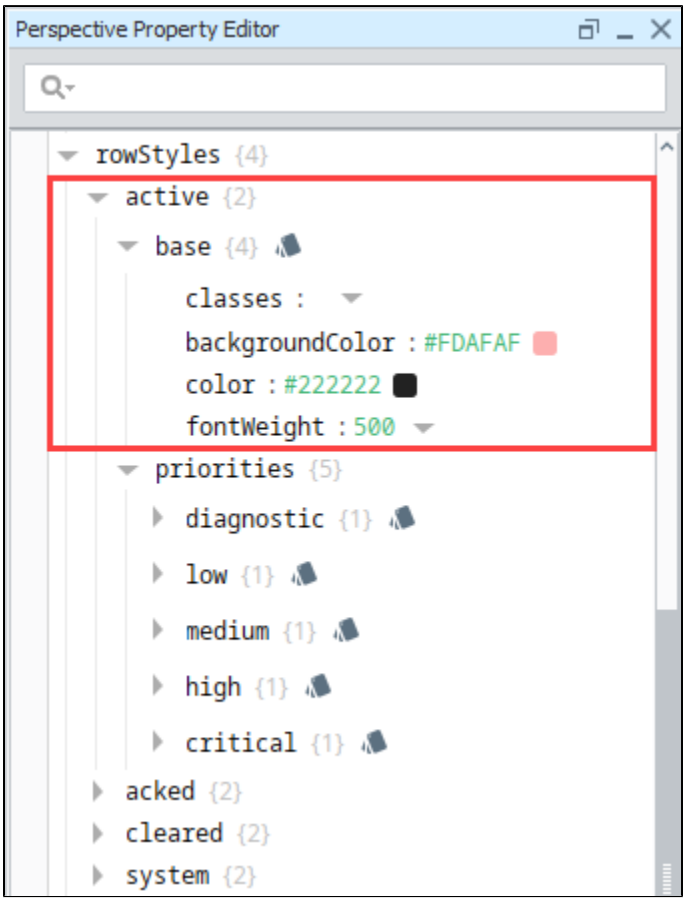
Customizing Row Styles for the Different Alarm States

The Alarm Journal Table comes with a default set of row colors associated with each of the different alarm states. You can modify an existing row style in the Designer by modifying the **'rowStyle'** properties for each of the alarm states. Each alarm state has the same default rowStyle properties, but their **backgroundColor** values are different.

Here is a list of default properties for each row:

- **base** - Opens a Style Options menu to configure style elements for Text, Background, Margin and Padding, Border, Shape and Misc.
- **classes** - Allows to you set a pre-defined style class.
- **backgroundColor** - Background color for each of the priorites (i.e., diagnostic, low, medium, high, critical) for an alarm state.
- **color** - Color of the Text.

- **fontWeight** - Font weight of the text.

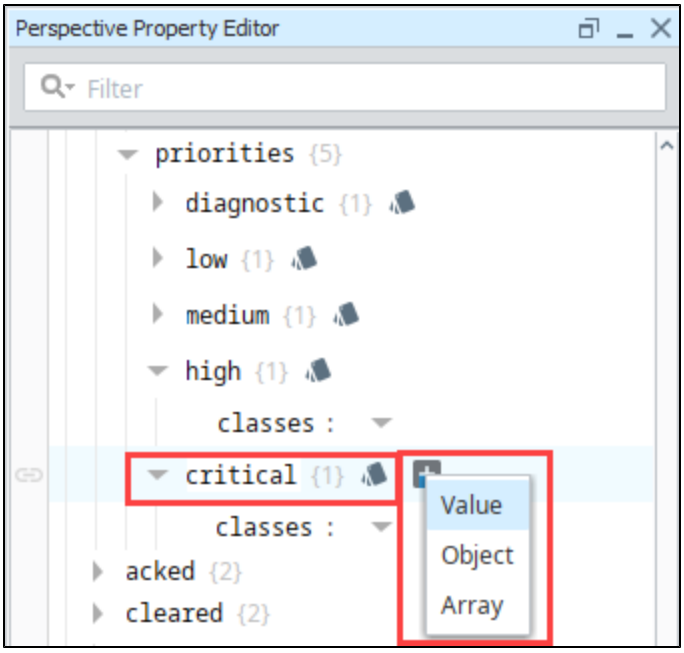


You can modify rowStyles properties in multiple places: using the **'base'** (i.e., Style Options menu) or property data types in the Property Editor. When you modify, add, or remove a style property from either Style Options menu or the Property Editor, the change will immediately be visible in both locations, as well as in the table. Here are two examples of how to modify rowStyles in an Alarm Journal.

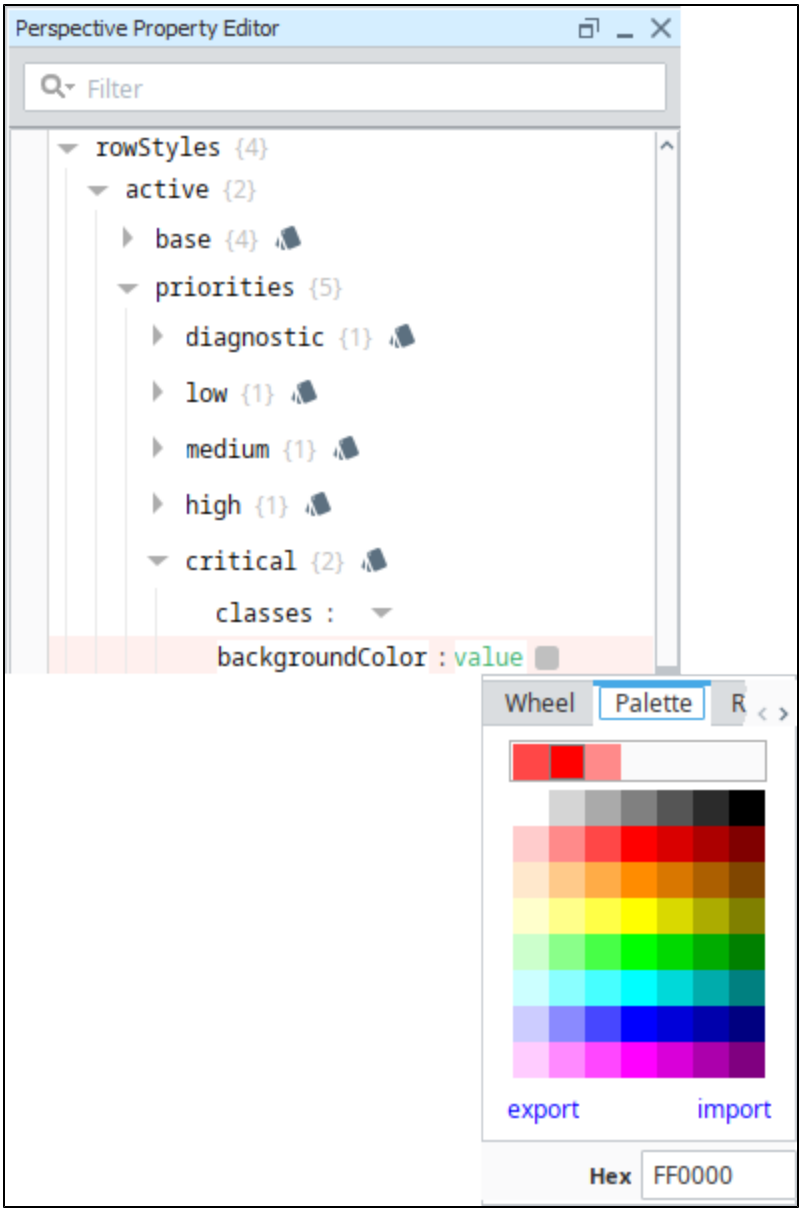
Example 1 - Modifying the rowStyle backgroundColor Property for an Alarm State

Let's change the background row color for the **'active'** state with a **'critical'** priority from pink to red.

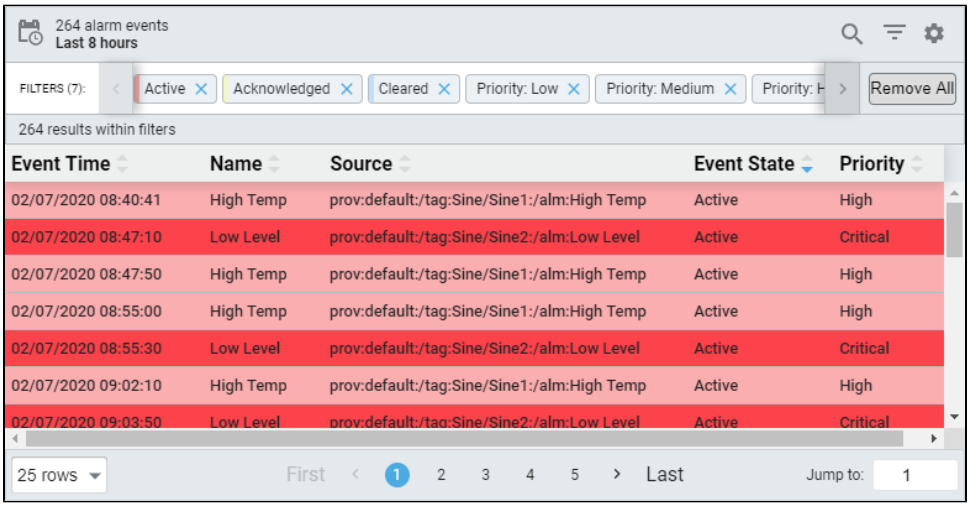
1. In the Designer, select the Alarm Journal Table component.
2. In the Property Editor, expand the **rowStyles** property and the **'active'** state and **'priorities'** properties.
3. Expand the **'critical'** property. Mouse over the **'critical'** priority property and click on the **plus** icon.



4. A dropdown will appear and select **'value.'**
5. The word **'key'** will be highlighted, enter the **'backgroundColor'** property.
6. Enter a value by clicking the gray square to open a popup color wheel or color palette, then choose a color (i.e., red).



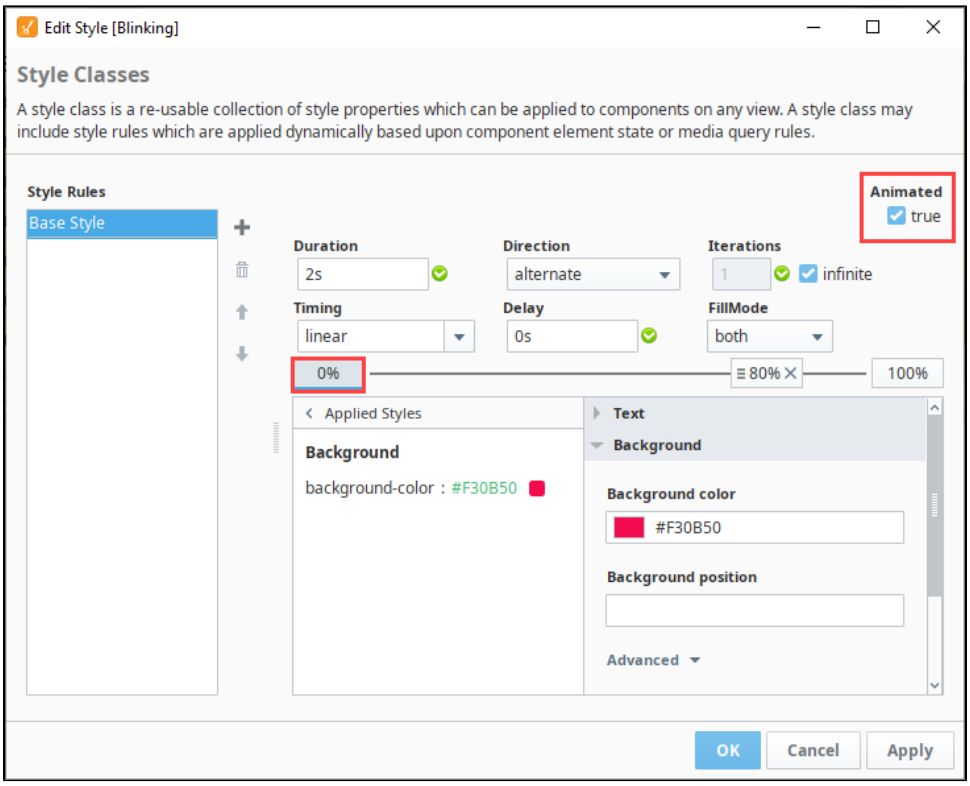
7. This will change the **backgroundColor** property for the **'Active'** state and **'Critical'** priority from pink to red.



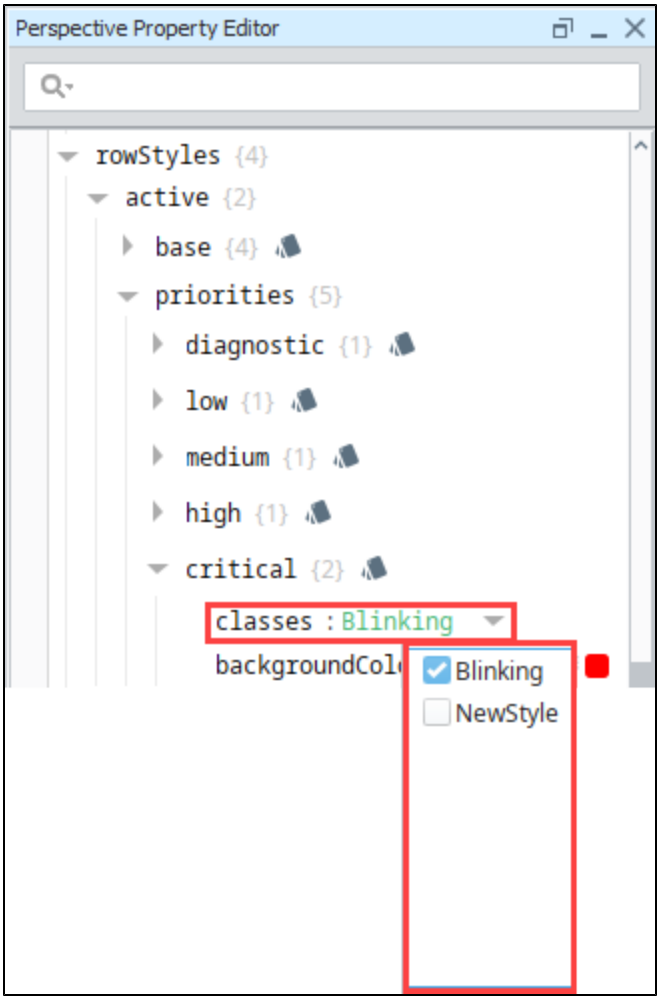
Example 2 - Add a Style Class to a rowStyle

You can create a new style class to make a row style stand out and draw attention to that alarm state. In this example, let's create the **'active'** state with the **'critical'** priority blink between two colors, red and green.

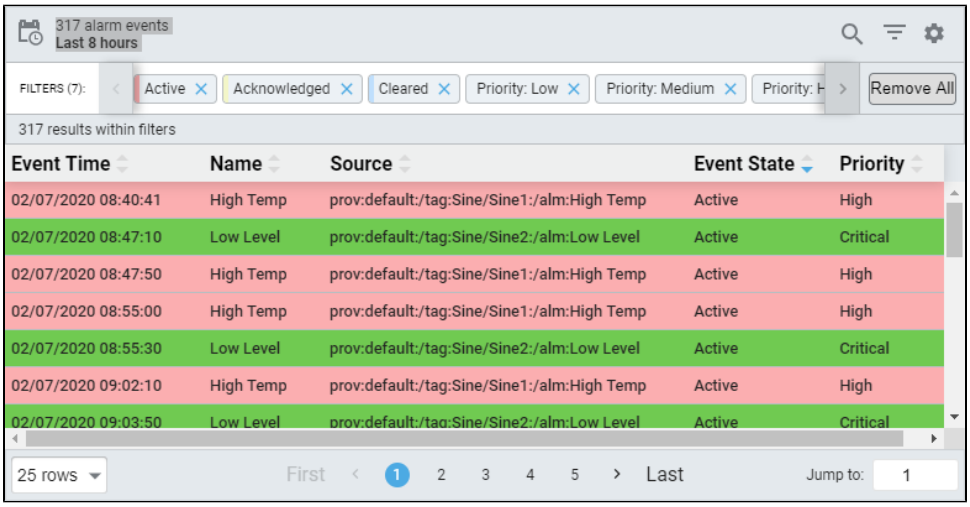
1. To create a new Style Class, go to the **Project Browser,** right click on **Styles Folder,** and click on **New Style.**
2. Enter a name (i.e. Blinking) for your new style, and click **Create Style Class**. The **Edit Style** window will open.
3. Set **Animated** to **'true'** in the upper right corner.
4. Click on **0%** to set the style for the beginning of the animation.
5. Click on **Background** to see Background settings for this style and choose a color. This example uses red.
   Notice the other Animation properties on the Edit Style window. You can set Duration, Direction, and number of Iterations. For a detailed description of the Animated properties, refer to Animated Style Classes.



6. Next, click on **100%** to set the style for the end of the animation.
7. Click on **Background** to expand the Background settings, and choose a color from the either the color palette or color wheel (i.e., green). When choosing a second color, make sure the foreground color will be readable when it's blinking.
8. To make one of the colors stay for a longer part of the duration, we can add another stop to the animation timeline. Right click on the horizontal line between the 0% and 100% boxes. Drag the stop to about **80%** to show the red (0% color) for longer than the green (100% color).
9. Set the Background color on this new 80% stop to match the red background color of the 0% stop.
10. Click **OK** to save your new style class.
11. Lastly, set the style class on the **'cirtical'** priority property to the name of the class (i.e., **Blinking**).



12. Now your **'active'** alarm state with a **'critical'** priority will blink red and green.



Reporting in Perspective

The following feature is new in Ignition version **8.0.10**
Click here to check out the other new features

Report Module

The Report Module must be installed to use the Report Viewer component.

On this page

...

- Configuring the Report Viewer
  - Report Viewer Properties
  - Using the Report Viewer

The Report Viewer component allows you to embed reports from the Reporting Module in a Perspective view, as well as view and print reports in PDF format. The Report Viewer located at the bottom of the Perspective Component palette in the Designer. To configure the Report Viewer, you must first create a report.

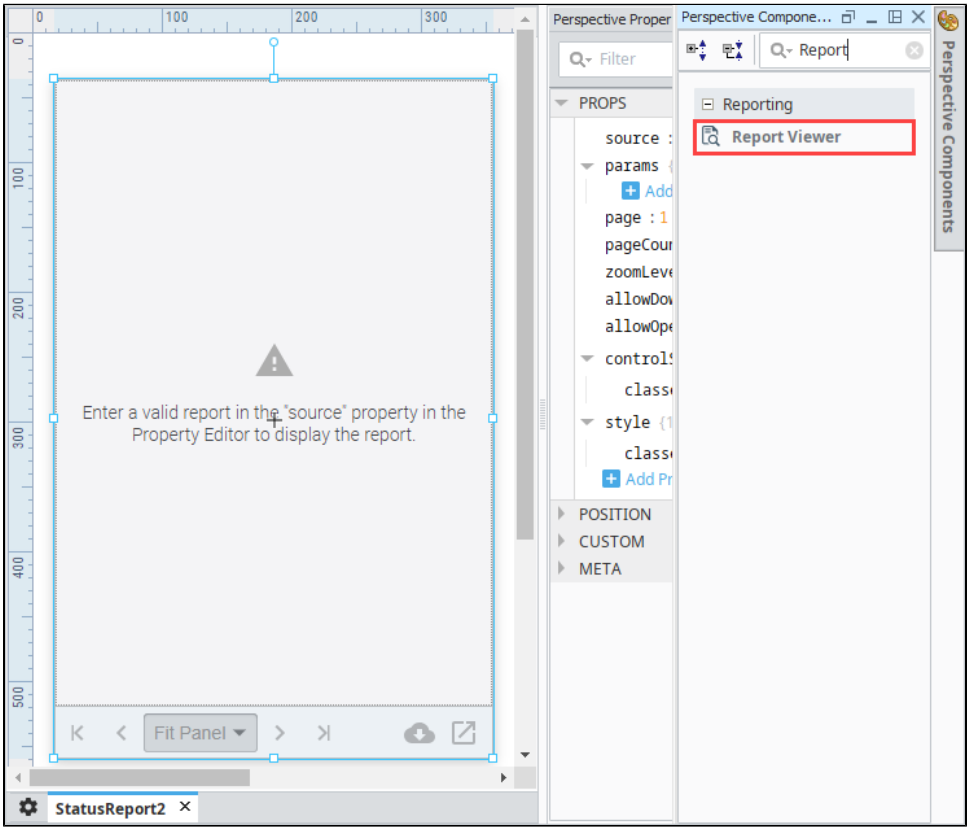
To learn more about creating reports, refer to the Reporting section.

Configuring the Report Viewer

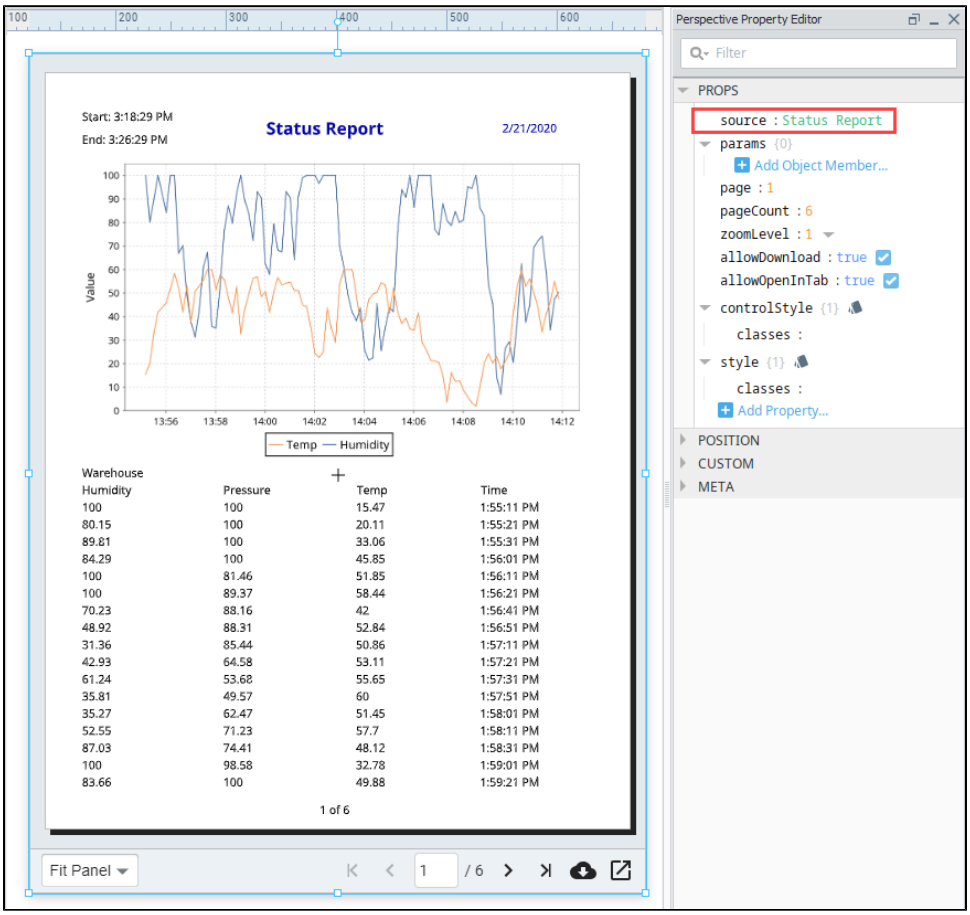
The Report Viewer component provides an easy way to view and print reports in a Perspective View. Let's configure the Report Viewer for a report.

This example requires that you already have a report created in your Gateway. You can learn more about creating reports here.

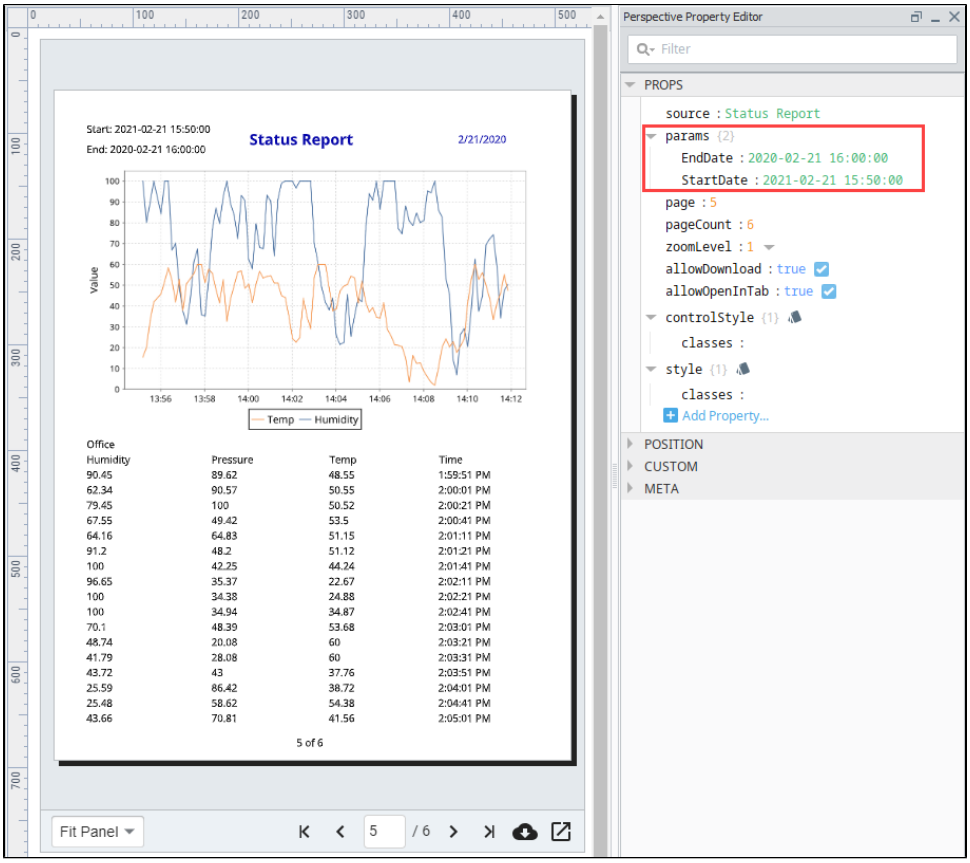
1. Drag a **Report Viewer** component into a view.



2. Enter the Path of the report you want to view in the **'source'** property of the Property Editor. The data from your report will immediately load into the Report Viewer.
   Tip: You can right-click on any report in the designer Project Browser to Copy the full Path of the report.



3. Parameters added during report creation are provided as properties in the Report Viewer. Add them under the **'params'** object. (The example below uses the EndDate and StartDate parameters). The parameter names must match exactly to the parameters in your Report Resource, and will override any default values set in the Report Resource. The Report Viewer also allows you to bind your report parameters in your view.



Report Viewer Properties

Once you are ready to view your report in the Report Viewer, set the **'source'** property to your report. If you have parameters defined in the Report Source, you can configure them under the **'params'** object, but the parameter names must match. The parameter values configured in the Report Viewer will override the values in the Report Source.

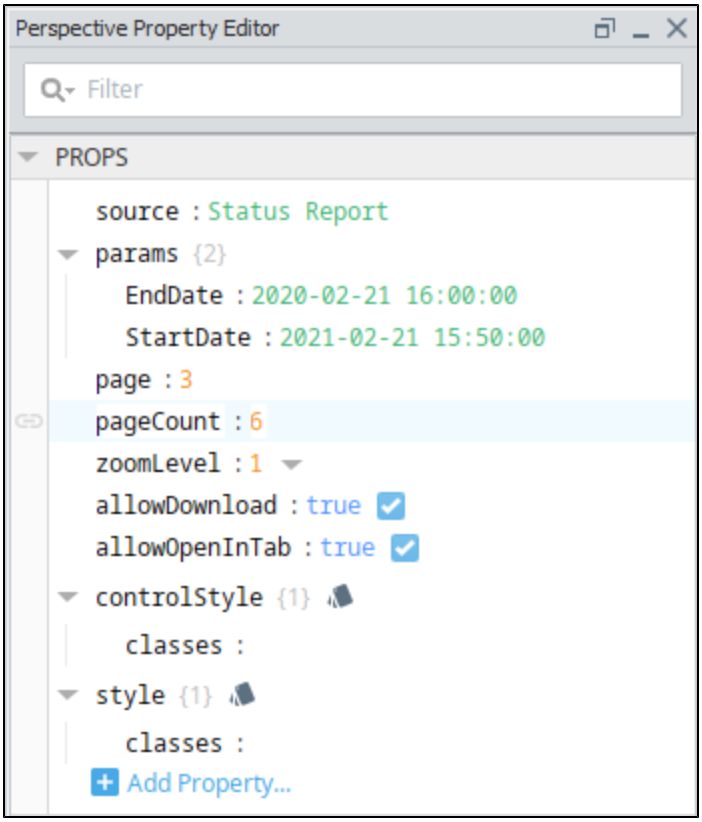
You can pick a starting page using the **'page'** property. Each time another page is viewed, the **'page'** property value will be updated.

You can specify a desired **'zoomLevel'** and every time the zoomLevel changes, the property value will also be updated.

You can customize the visual style of your report by creating a new style for your report such as changing the background colors of the viewer (not the report page), changing to a style class that was already defined in your project, or creating a new style.

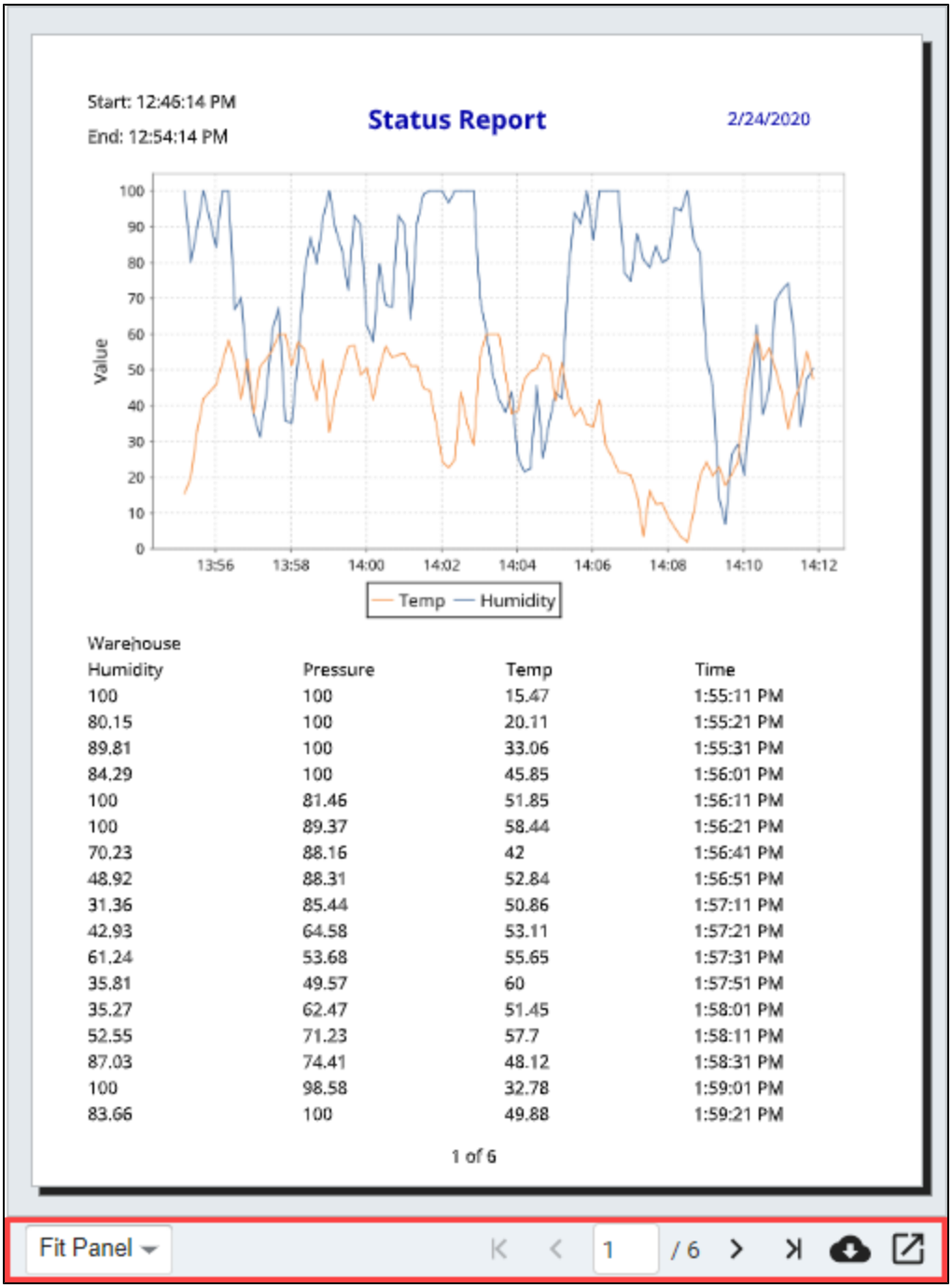
The **'allowDownload'** and **'allowOpenInTab'** properties allow you to view and print your report.

To see the property descriptions, refer to the Perspective - Report Viewer page.



Using the Report Viewer

Once you have your report designed, it can be viewed one page at a time using the navigational controls across the bottom of the component. You have the option to download a report to your local device or print a report to your local printer using these built-in controls.



Related Topics ...

- Reporting
- Perspective - Report Viewer

Common Tasks in Perspective

This section contains examples for items we identified as "common" tasks: methods that many users are looking to utilize when first starting out with the Perspective Module or feature in Ignition. Additionally, this section aims to clarify some of the more complex or abstract tasks that our users may encounter.

The examples in this section are self-contained explanations that may touch upon many other areas of Ignition. While these examples are typically focused on a single goal or end result, they can easily be expanded or modified after the fact. In essence, they serve as a great starting point for users new to Ignition, as well as experienced users that need to get acquainted with a new or unfamiliar feature.

Below is a list of common tasks related to this section of the manual.

On this page

...

- Popup Views
- Navigating with the Horizontal Menu
- Self-Hiding Navigation Drawer
- Configuring a Dashboard
- Displaying a Subview in a Table
- Download and Upload Files
- Table Column Configurations

Popup Views

Popup Views are a great way to enable users to view more detailed information in your HMI. They are also relatively simple to create.

Navigating with the Horizontal Menu

The Horizontal Menu component is a great option for easy, quick navigation between pages. For an example, see Navigating with the Horizontal Menu Component.

Self-Hiding 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. This example walks you through creating a self-hiding navigation drawer that is only displayed when the user needs it.

Configuring a Dashboard

The Dashboard exposes widgets to end users in a Perspective Session so they can customize their dashboard layout for their individual needs. Widgets are views that are pre-configured in the Designer and made available to Perspective Session users. For an example, see Con figuring a Dashboard.

Displaying a Subview in a Table

In a Perspective Table component, you have the option to enable subviews. When a subview is set up, you can click on the Expand icon in the table and have another view be displayed without closing the first view. For an example, see Displaying a View in a Table.

Download and Upload Files

Downloading and uploading files from a Perspective session typically involves storing and retrieving files from a database. A table will store all of the available files, and each row of the table represents a new file. This allows for long term storage that is accessible from any project. For examples, see Download and Upload Files.

Table Column Configurations

In a Perspective Table component, you have the ability to replace a cell with something other than just text or a number. Instead, you can have the column render other objects altogether such as progress bars or a view. For an example, see Table Column Configurations.

In This Section ...

Popup Views

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

On this page

...

- Title Bars on Popups
- Configuring a Popup View

Popup Views

Watch the Video

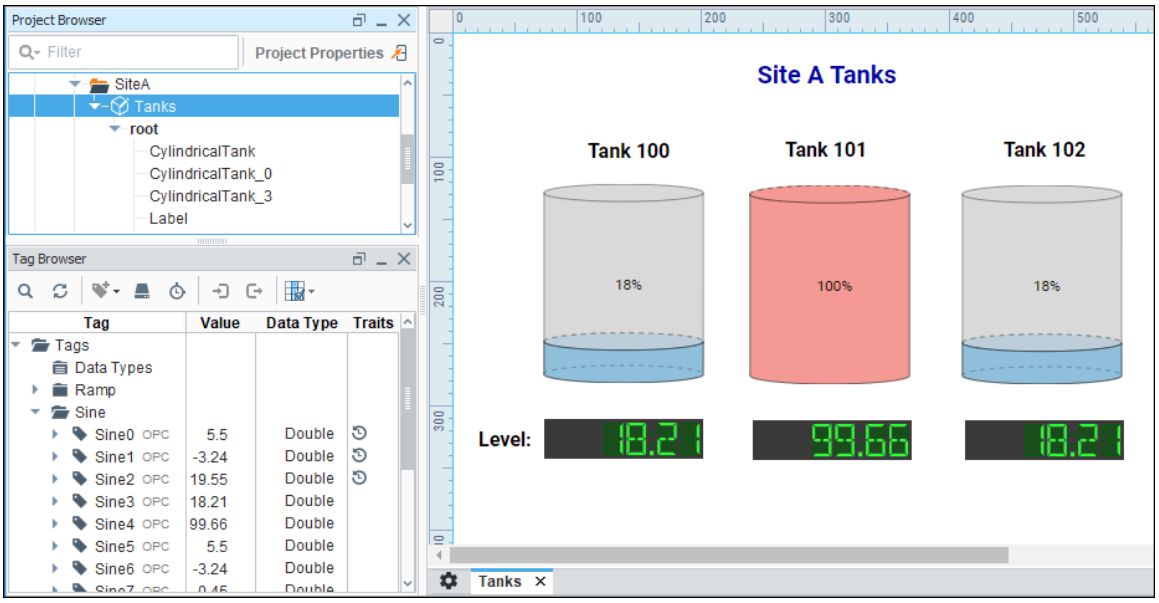
Title Bars on Popups

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

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

Configuring a Popup View

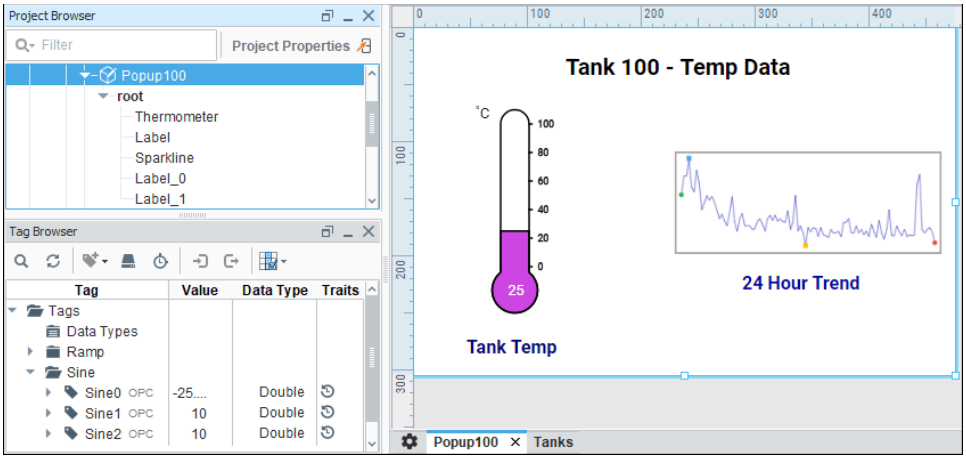
Let's assume you already have a primary view that contains some Tanks. At a glance, an operator can view some basic information on all the Tanks at a particular site, but they cannot see the additional information that is being collected that is unique to each Tank. The perfect way to display that unique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



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

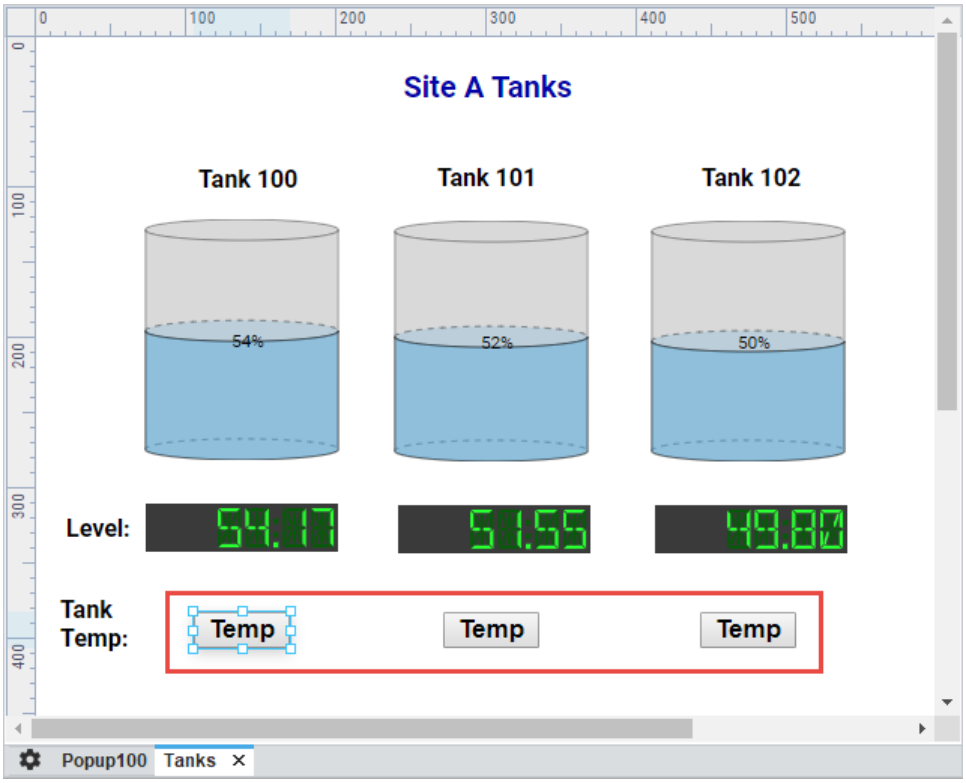
1. First create a new view, and give it a name. You many want to assign a name that you can easily recognize as a Popup View and the Tank ID number (i.e., Popup100).

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

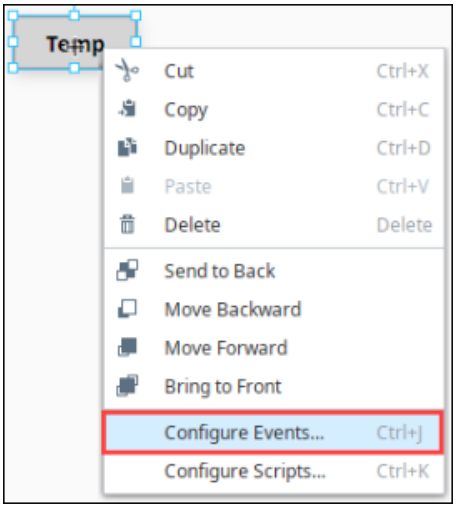


> **Multiple Popup Views**
>
> To display the additional data for all Tanks, you will need to create a Popup View for each Tank. You can use the Flex Repeater component to easily create multiple instances of components for display in another view each having the same look, feel, and functionality of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

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



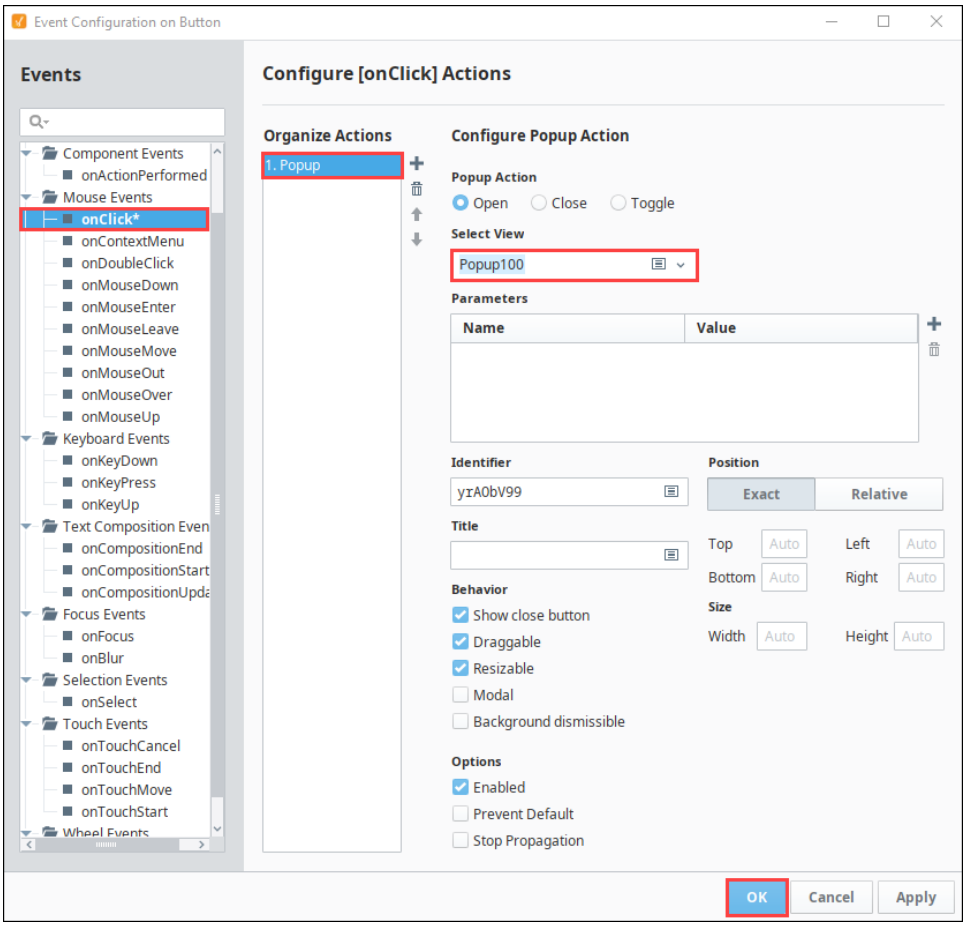
4. Right click on the Temp button for Tank 100, and select **Configure Events**.



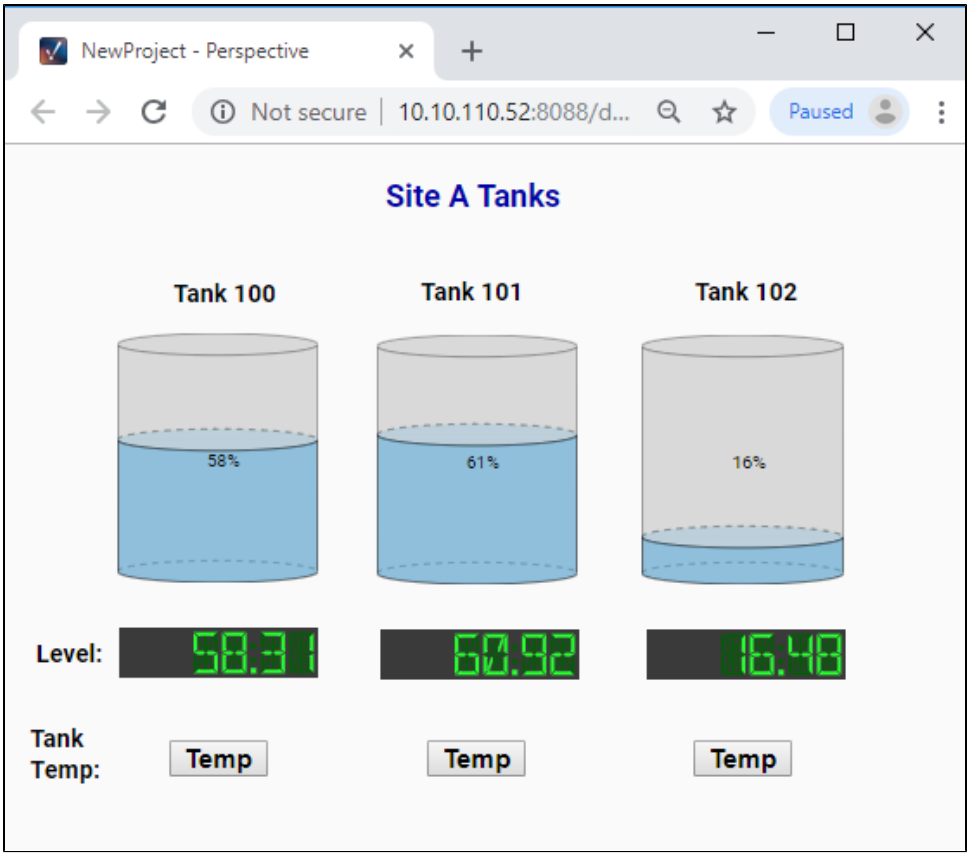
5. The Event Configuration window opens. Under Events, select **onClick**.
6. Under Organize Actions, click the **Add** icon to choose the appropriate action for the Temp button. Since we are opening a Popup View, select **Popup** from the dropdown list.

7. You have several Popup Actions to choose from.
    - **Open** - Opens the Popup View.
    - **Close** - Closes the Popup View.
    - **Toggle** - Opens and closes the Popup View. Select **Toggle** so you can use the Temp button to both open and close the Popup View for the specified tank.

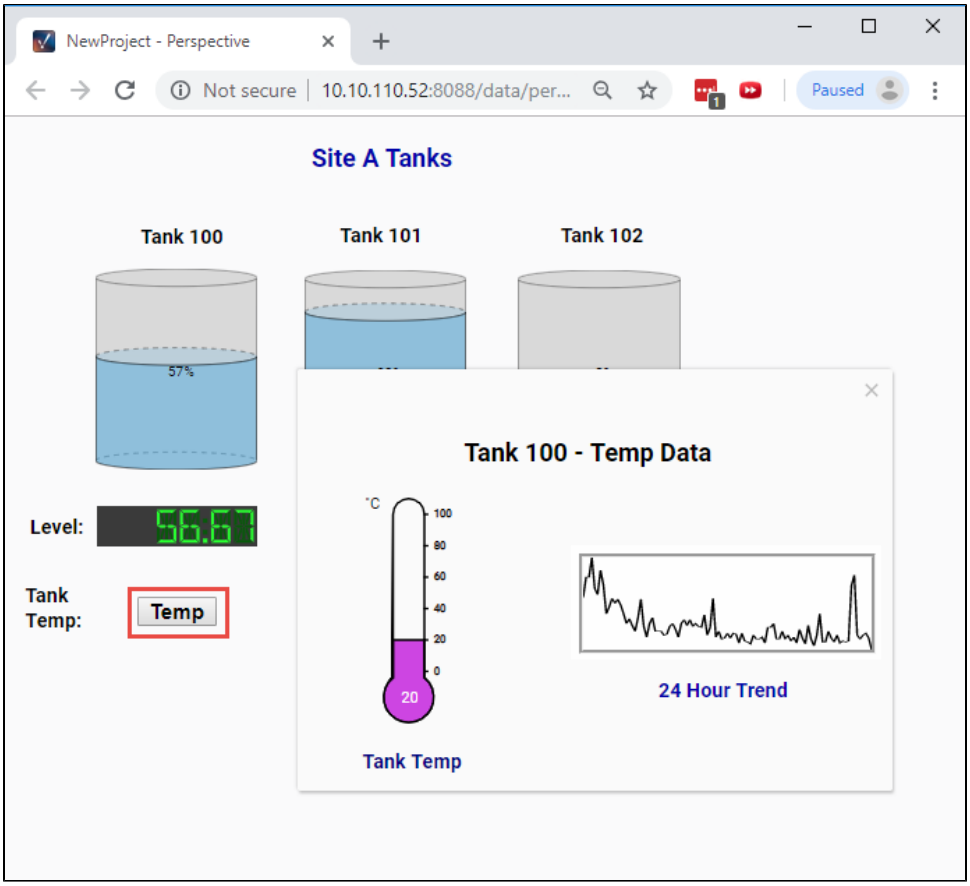
8. In the Configure Popup Action area under Select View, select your Popup View that you created in Step 1. This example uses Tank 100 and the Popup View is **'Popup100'**.
9. You also have some additional settings for customizing the behavior and appearance of your popup. Some of these options are set by default, and others you can customize. When you're finished, click **OK**.
10. Repeat Steps 4 through 9 for Tank 101 and Tank 102.



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



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



Related Topics ...

- Views and Containers in Perspective
- Pages in Perspective

Navigating with the Horizontal Menu Component

The Horizontal Menu component enables you to build a menu structure by setting up multiple links to different page URLs from the component. Our example has a menu with links to three internal pages and one external page on the Internet.

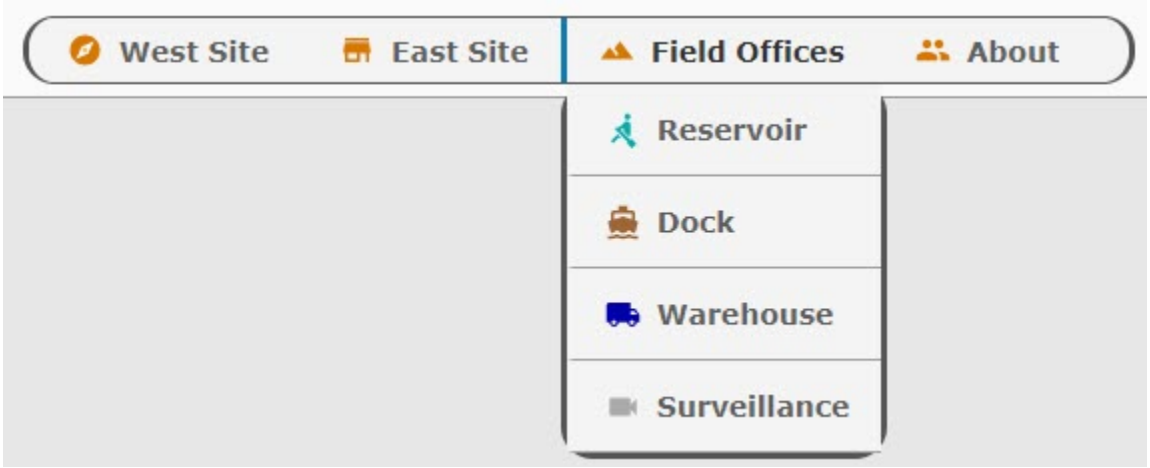
This example shows a Horizontal Menu with four items. Three items are linked to views within the Perspective project and the fourth is linked to a website.

The third item in the list, Field Offices, has four subitems. Each item links to a view for a different field office. Icons are taken from the Material Design icons that can be found here: https://material.io/tools/icons/.

On this page

...

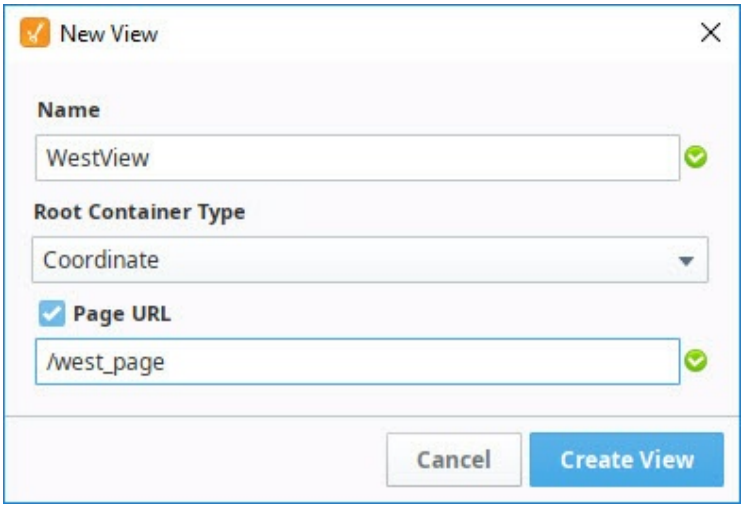
- Initial Project Setup
- Set Up a Header View
- Configure the Tabs in the Horizontal Menu
  - Configure the Field Offices Tab in the Horizontal Menu



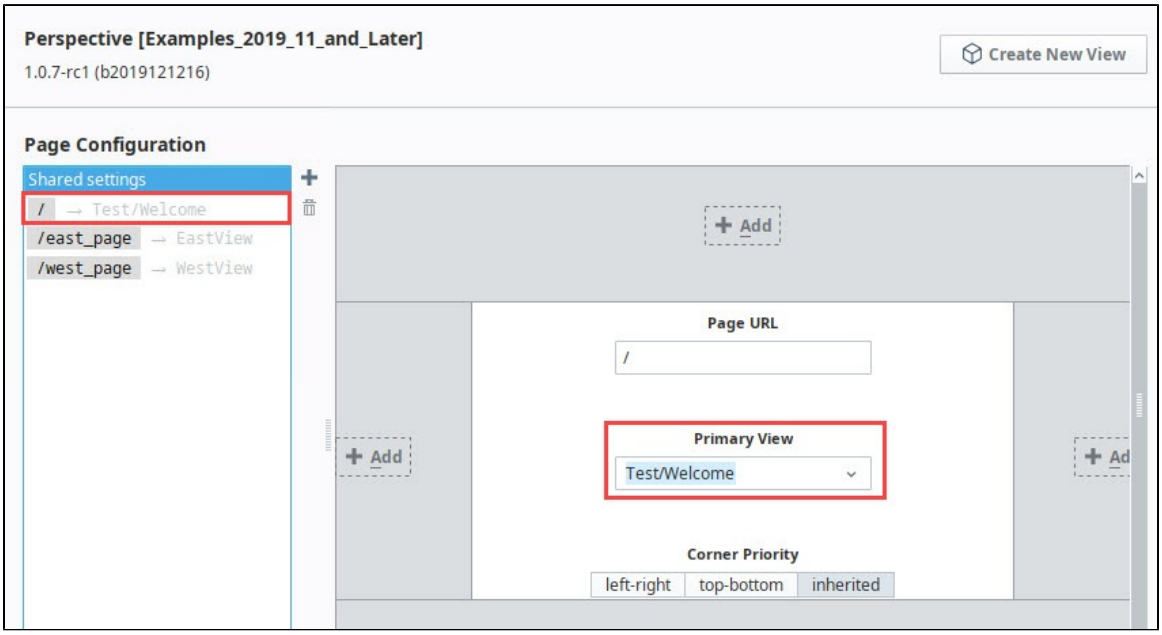
Initial Project Setup

To begin with we have created three views: **WestView**, **EastView**, and **Welcome**. They are each coordinate type views. As we create each new view, we checked the Page URL property and added a page name.

View Name	Page URL
WestView	/west_page
EastView	/east_page
Welcome	/



On Page Configuration, you'll note that each page is already set up with a Primary view.



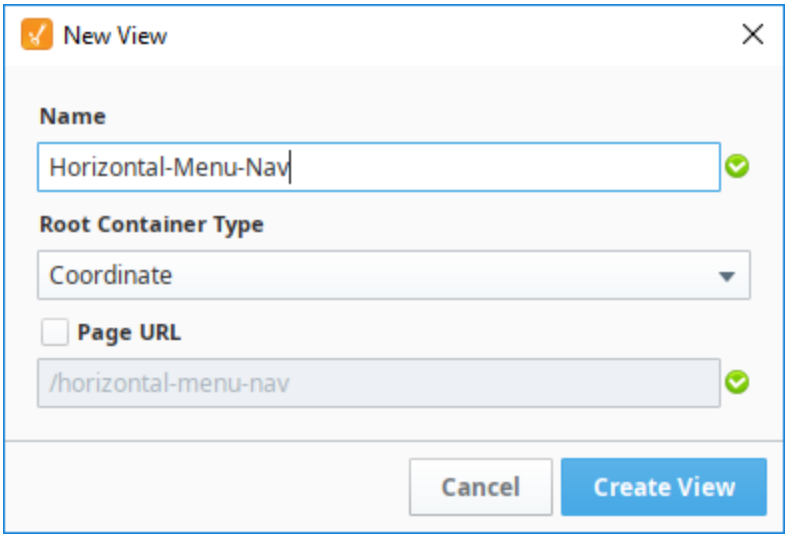
Now we're ready to start building the navigation.

Set Up a Header View

The first thing we'll set up is a view that will hold the Horizontal Menu component. We'll use this view as a header for our pages within this project.

1. In the Project Browser, right click on Views and select the **NewFolder** option. Name the folder "Header".
2. Right click on the Header folder and select the **NewView** option.

   Name: **Horizontal-Menu-Nav**
   Layout: **Coordinate**
   Page URL: unchecked



3. Click **Create View**.
4. In the Property Editor for the view, set the **defaultSize** property as follows:

   width: **800**
   height: **50**

5. Drag a Horizontal Menu component onto the view.
6. In the Property Editor, set the Position Properties as follows:

Property	Value
position.x	15
position.y	10
position.width	550
position.height	30

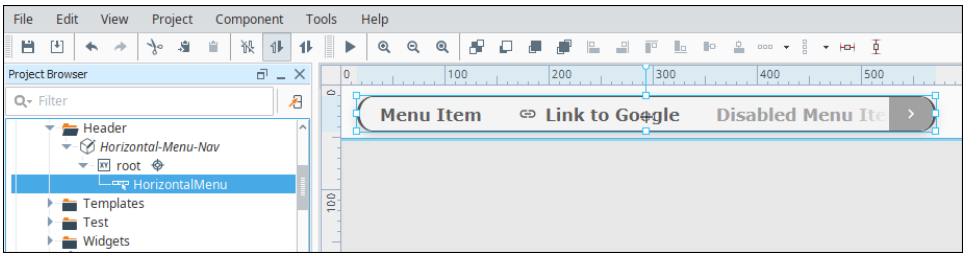
7. In the Property Editor, scroll down to style and click the **ModifyStyle** icon.
   a. Expand the Text section and set the style options as follows:

Property	Value
props.style.fontWeight	bolder
props.style.fontFamily	Verdana
props.style.fontSize	14px

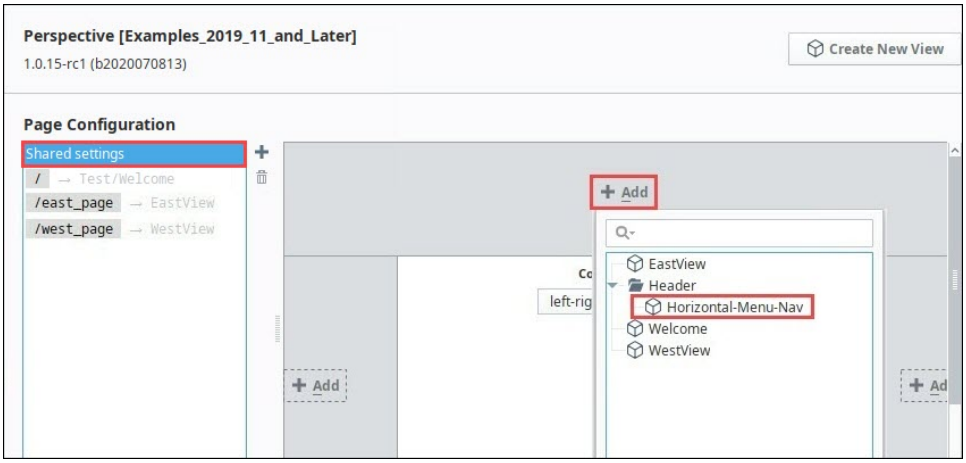
   b. Expand the Border section and set the style options as follows:

Property	Value
props.style.borderStyle	solid
props.style.borderColor	#555555
props.style.borderRadius	16px
All Corners	(selected)

   Your Designer will look like this at this point:



8. Open Page Configuration by clicking on the **Settings** icon at the bottom left of the Designer window.
9. Under Page Configuration, click on **Shared Settings**.
10. In the header part of the page mockup, click on the **Add** icon.
11. Select the Horizontal-Menu-Nav view from the dropdown.



12. Click **OK**. The Horizontal-Menu-Nav view will now appear at the top of all pages in the project.

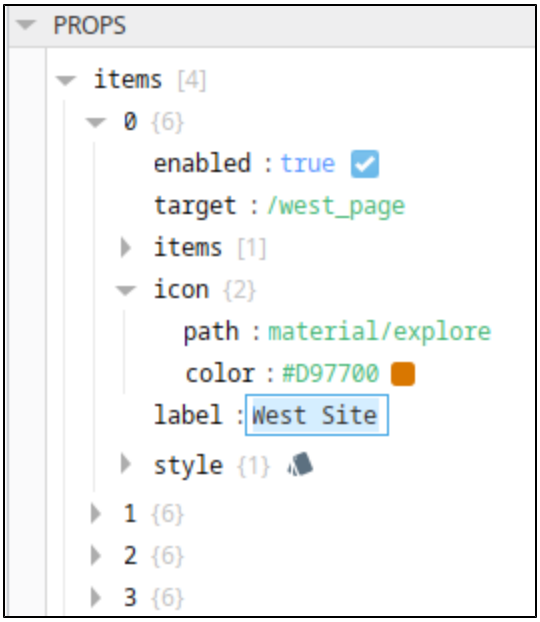
Configure the Tabs in the Horizontal Menu

Now let's set the properties for each of the four tabs in the Horizontal Menu. Each tab will have a display a name and an icon and will have a target page or website to open when clicked.

1. In the Property Editor, expand the items property and set the following for item 0:

Property	Value
props.items.0.enabled	true
props.items.0.target	/west_page
props.items.0.icon.path	material/explore
props.items.0.icon.color	#D97700
props.items.0.label	West Site

The Property Editor will look like this:



2. In the Property Editor, set the following for item 1:

Property	Value
props.items.1.enabled	true
props.items.1.target	/east_page
props.items.1.icon	material/store
props.items.1.color	#D97700
props.items.1.label	East Site

3. In the Property Editor, set the following for item 2:

> ⚠ Do not set a props.items.2.target property value for this tab because we will set up dropdown tabs in the next section..

Property	Value
props.items.2.enabled	true
props.items.2.icon.path	material/landscape
props.items.2.icon.color	#D97700
props.items.2.label	Field Offices

4. In the Property Editor, set the following for item 3:

> ⚠ This tab uses a website as its target, therefore it does not need to target a page within Perspective.

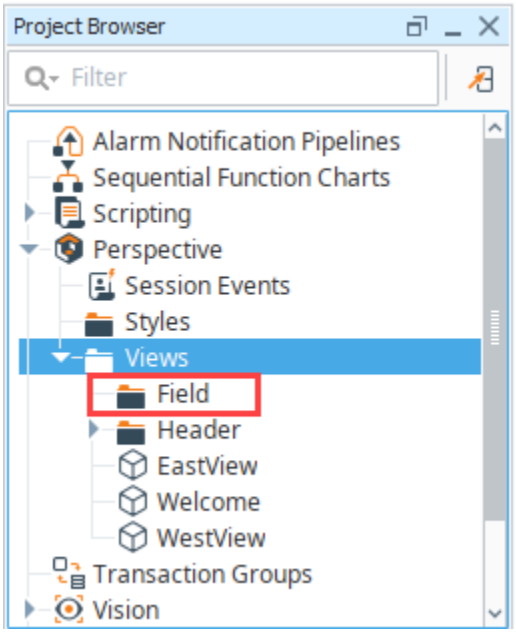
Property	Value
props.items.3.enabled	true
props.items.3.target	http://inductiveautomation.com/about/
props.items.3.icon.path	material/people

props.items.3.icon.color	#D97700
props.items.3.label	About

Configure the Field Offices Tab in the Horizontal Menu

The third tab in the Horizontal Menu is titled "Field Offices." Instead of navigating to one page, this tab has a dropdown menu with four options on it: Reservoir, Dock, Warehouse, and Surveillance.

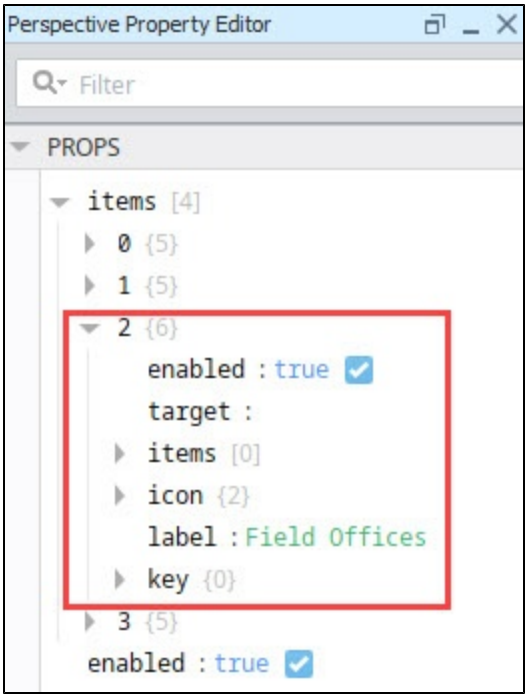
1. To start, we created a new folder in Views called **Field**.



2. Within the Field folder, we create four views: **Reservoir**, **Dock**, **Warehouse**, and **Surveillance.** Make them Coordinate types and set up a page URL for each.

View Name	Page URL
Reservoir	/reservoir_page
Dock	/dock_page
Warehouse	/warehouse_page
Surveillance	/surveillance_page

3. Open the Horizontal-Menu-Nav view and select the Horizontal Menu component.
4. In the Property Editor, expand the properties for Item 2.



5. Select the props.items.2.items property.
6. Click the Add Add Array Element... icon to add four items.
7. Set the properties for Item 0 as follows:

Property	Value
props.items.2.items.0.enabled	true
props.items.2.items.0.target	/reservoir_page
props.items.2.items.0.icon.path	material/rowing
props.items.2.items.0.icon.color	#00ACAC
props.items.2.items.0.label	Reservoir

The Property Editor will look like this:

8. Now we'll do the same for the other three items. Set the properties for Item 1 as follows:

Property	Value
props.items.2.items.1.enabled	true
props.items.2.items.1.target	/dock_page
props.items.2.items.1.icon.path	material/directions_boat
props.items.2.items.1.icon.color	#9E6635
props.items.2.items.1.label	Dock

9. Set the properties for Item 2 as follows:

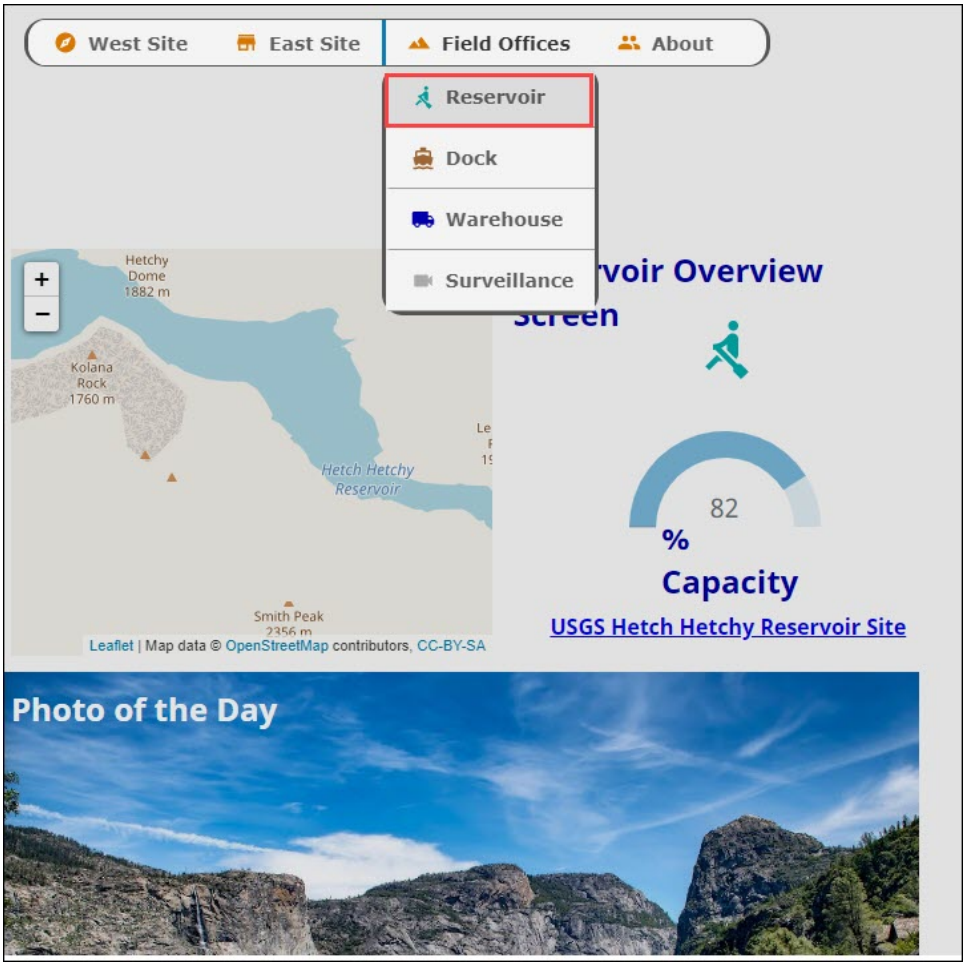
Property	Value
props.items.2.items.2.enabled	true
props.items.2.items.2.target	/warehouse_page
props.items.2.items.2.icon.path	material/local_shipping

props.items.2.items.2.icon.color	#0000AC
props.items.2.items.2.label	Warehouse

10. Set the properties for item 3 as follows:

Property	Value
props.items.2.items.3.enabled	true
props.items.2.items.3.target	/surveillance_page
props.items.2.items.3.icon.path	material/videocam
props.items.2.items.3.icon.color	#AAAAAA
props.items.2.items.3.label	Surveillance

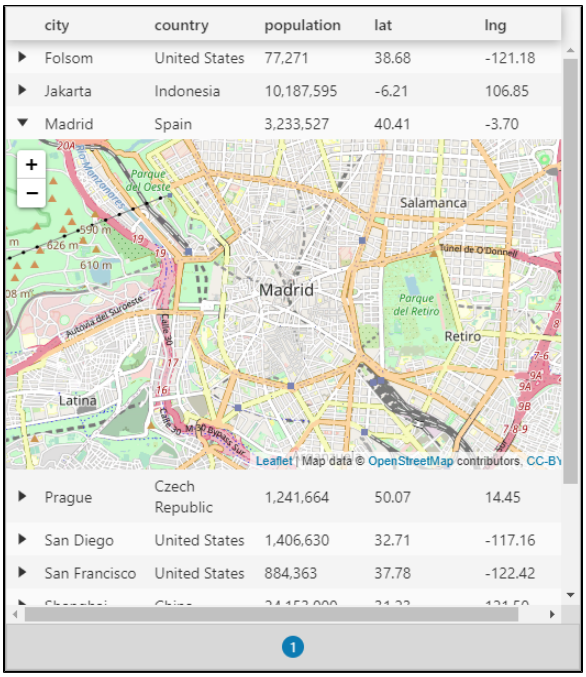
11. Save your project.
12. Click **Tools > Launch Perspective > Launch Session**.
13. Click on the tabs in the header to view different pages. For our example, we have put a few components on each view. Here is an example of what the Field Offices > Reservoir page might look like:



Displaying a SubView in a Table

In a Perspective Table component, you have the option to enable subviews. When a subview is set up, you can click on the Expand icon in the table and have another view be displayed without closing the first view. This example sets up a table with several cities and statistics. When the Expand icon is selected for a city, a Map component will be displayed showing the location of the city on the map.

This example focuses on using a map component in a subview on the table, but the larger implication here is that subviews in table rows can receive values from each row in the table, and utlizie them with property bindings, allowing each subview to contain data unique to from the row. The image below shows what our finished view will look like.



On this page

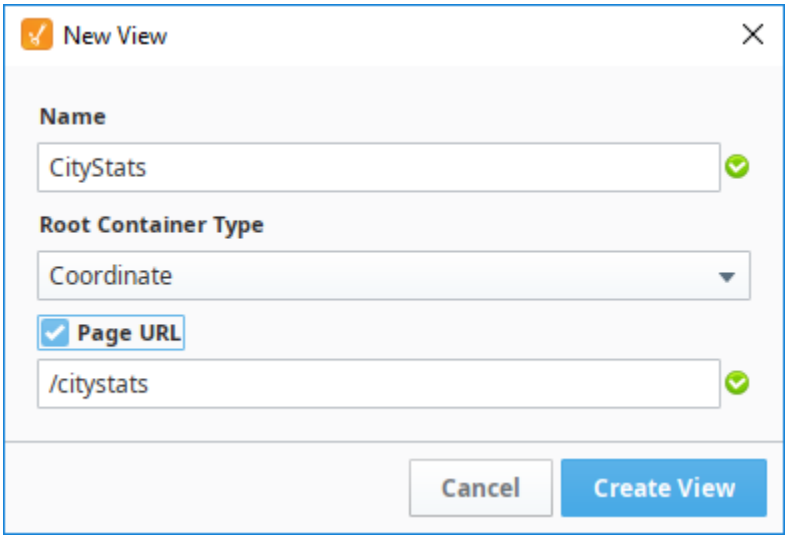
...

- Create a View for the Table Data
- Create a View for Displaying the Map
- Use the Maps View as a Subview for the Table

Create a View for the Table Data

We'll start by creating a view that will contain the table.

1. Right click on Views to create a view. In the example, we named ours **CityStats**. Set it as a Coordinate layout and check the Page URL option if you want to create a page for this view (you can always add a page later if you want to).

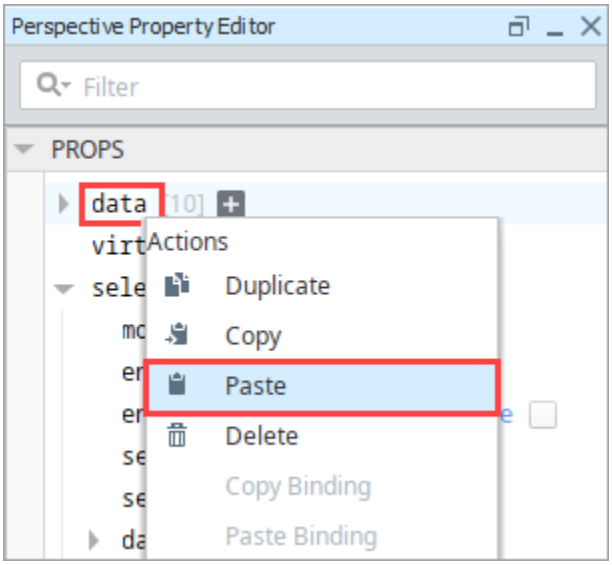


2. Drag a Table component onto the view.
3. The table needs to have Latitude and Longitude data for the map to show that location. Highlight and copy the following data:

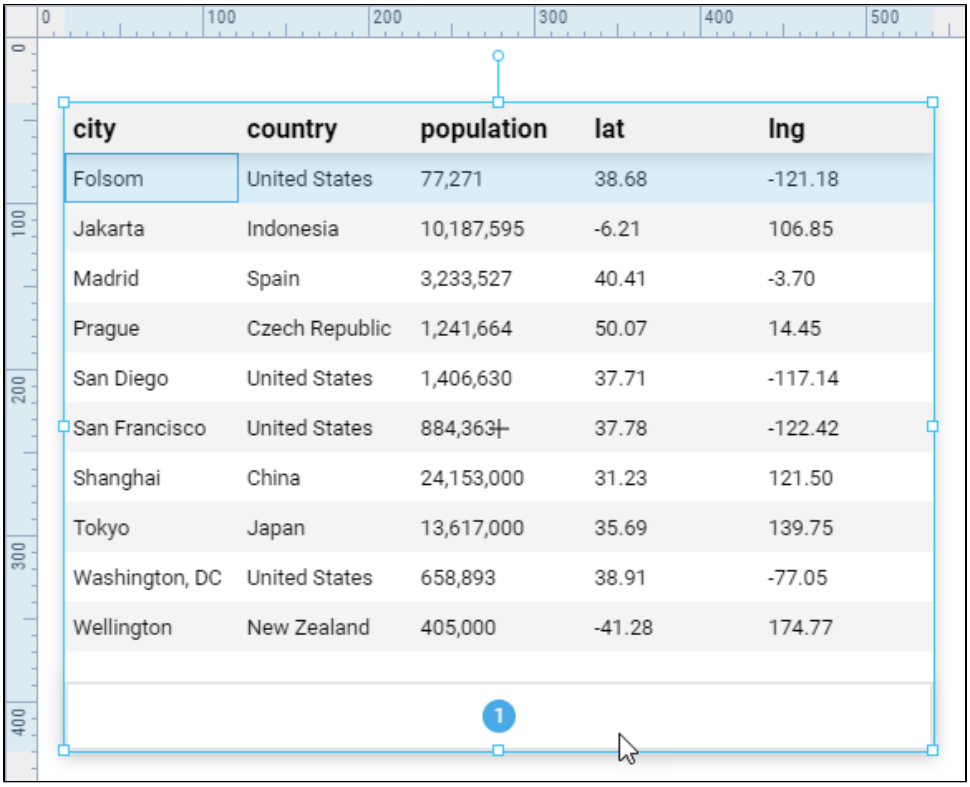

```
[
  {
    "city": "Folsom",
    "country": "United States",
    "population": 77271,
    "lat": 38.678287,
    "lng": -121.177318
  },
  {
    "city": "Jakarta",
    "country": "Indonesia",
    "population": 10187595,
    "lat": -6.208404,
    "lng": 106.849087
  },
  {
    "city": "Madrid",
    "country": "Spain",
    "population": 3233527,
    "lat": 40.41498,
    "lng": -3.702002
  },
  {
    "city": "Prague",
    "country": "Czech Republic",
    "population": 1241664,
    "lat": 50.073453,
    "lng": 14.450091
  },
  {
    "city": "San Diego",
    "country": "United States",
    "population": 1406630,
    "lat": 32.713832,
    "lng": -117.158616
  },
  {
    "city": "San Francisco",
    "country": "United States",
    "population": 884363,
    "lat": 37.776379,
    "lng": -122.423501
  },
  {
    "city": "Shanghai",
    "country": "China",
    "population": 24153000,
```

```
    "lat": 31.227167,
    "lng": 121.498839
  },
  {
    "city": "Tokyo",
    "country": "Japan",
    "population": 13617000,
    "lat": 35.69042,
    "lng": 139.746457
  },
  {
    "city": "Washington, DC",
    "country": "United States",
    "population": 658893,
    "lat": 38.90598,
    "lng": -77.04882
  },
  {
    "city": "Wellington",
    "country": "New Zealand",
    "population": 405000,
    "lat": -41.284336,
    "lng": 174.770488
  }
]
```

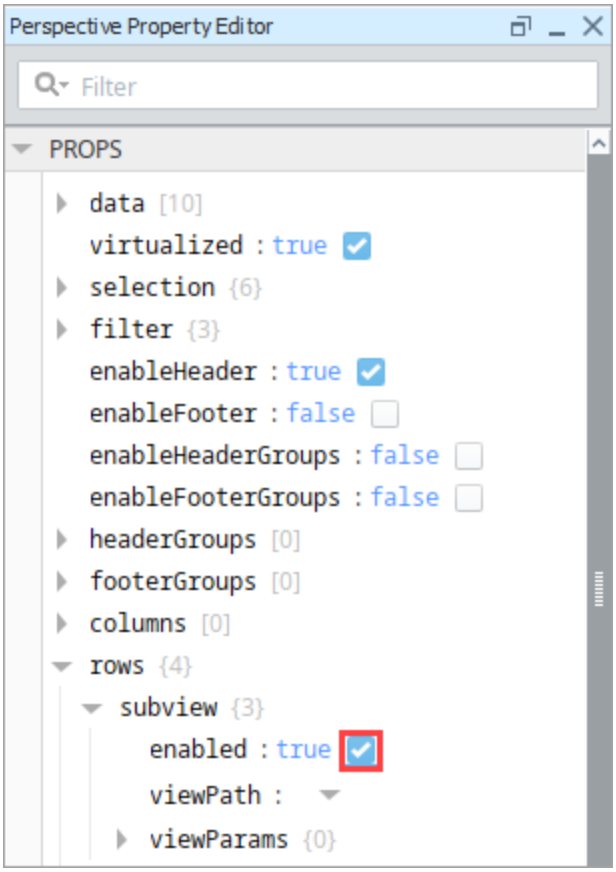

4. Right click on the data property of the Table component and select **Paste**.



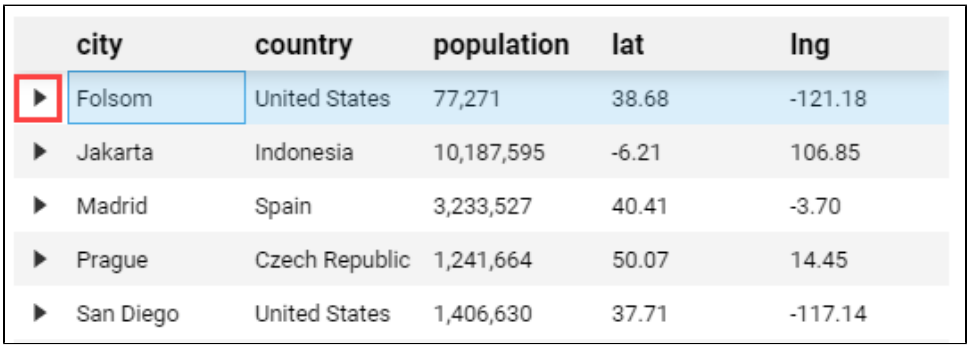
5. Your table will now display the data for 10 rows and 5 columns.



6. Next, enable the **subview** property under props.rows.



7. The table now has **Expand** ▶ icons for each row.

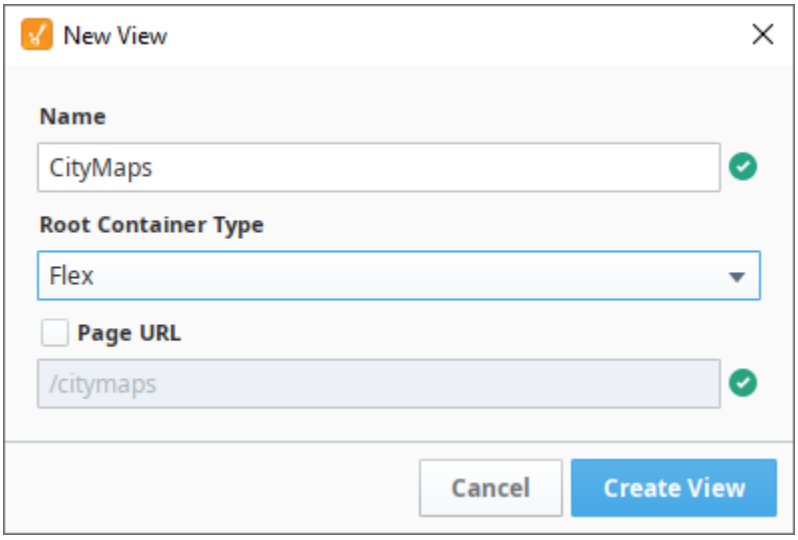


	city	country	population	lat	lng
▶	Folsom	United States	77,271	38.68	-121.18
▶	Jakarta	Indonesia	10,187,595	-6.21	106.85
▶	Madrid	Spain	3,233,527	40.41	-3.70
▶	Prague	Czech Republic	1,241,664	50.07	14.45
▶	San Diego	United States	1,406,630	37.71	-117.14

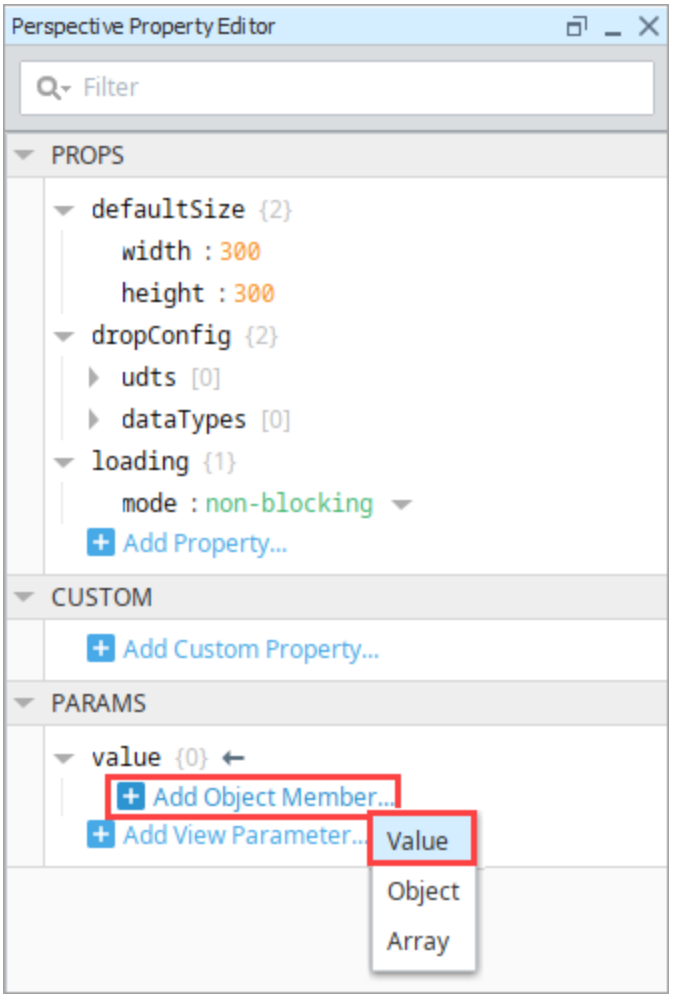
Create a View for Displaying the Map

Next we'll make the view that will be display a map of the cities in our table.

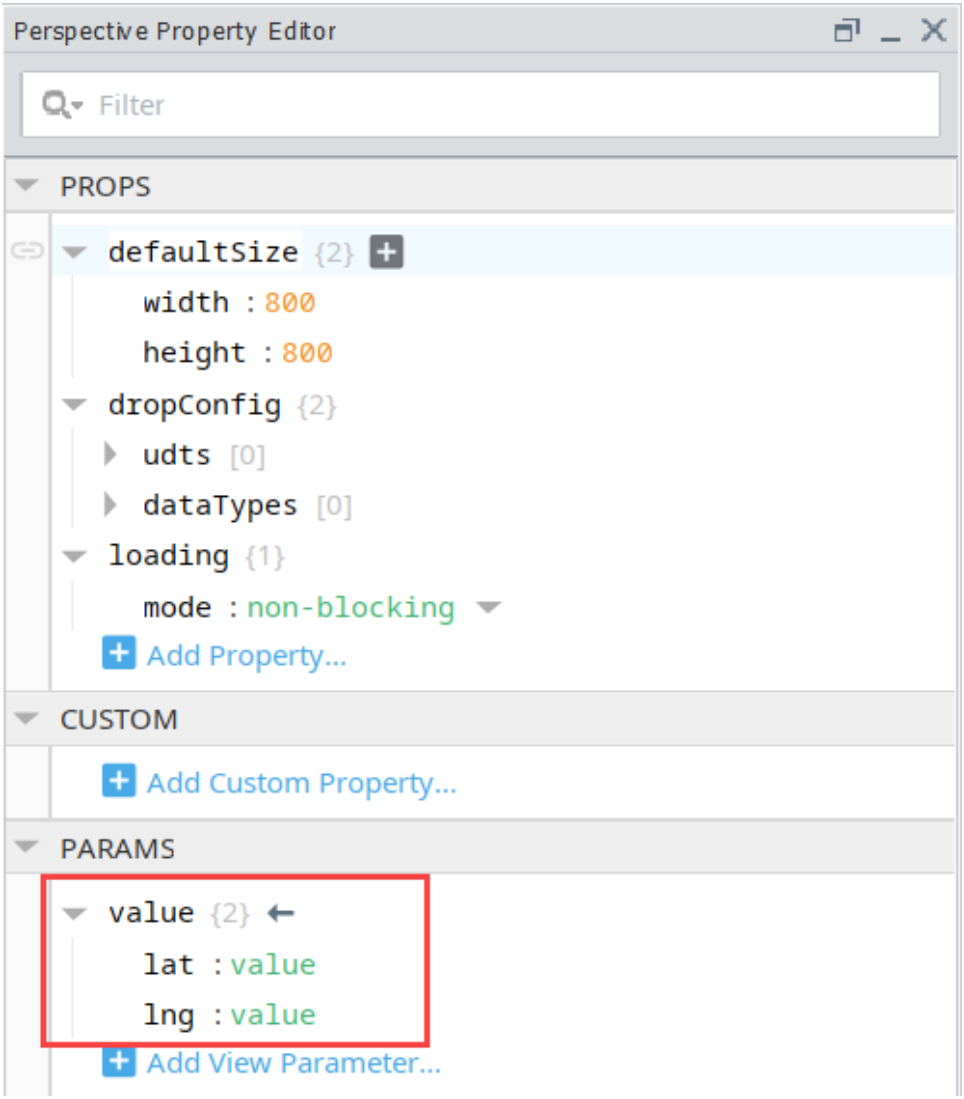
1. In the Project Browser, right click on Views to create a view. Name the new view **CityMaps**. Set it as a **Flex** layout, so the map easily takes up all available space. Lastly, do not check the Page URL option, as we don't need a corresponding page.



2. Drag a Map component onto the **CityMaps** View.
3. Set the Map's **Grow** property to **"1"** so it resizes to take up the entire view.
4. Click on the **CityMaps** view in the Project Browser. In the Property Editor, click the **Add View Parameter** link under **Params** and choose **Object**.
5. Double click on key, and enter **value** as the object name.
6. Next, we'll add two parameters to that value object.
    a. Click **Add Object Member** link under Params and choose **Value.**



b. Double click on key, and enter **"lat"**. This matches the lat (latitude) column from the Table on the CityStats view. This name must *exactly* match the column name in the table.
c. Click the **Add Object Member** icon next to value and choose **Value**.
d. Double click on key, and enter "**lng"**. This matches the lng (longitude) column from the Table on the CityStats view. This name must *exactly* match the column name in the table.

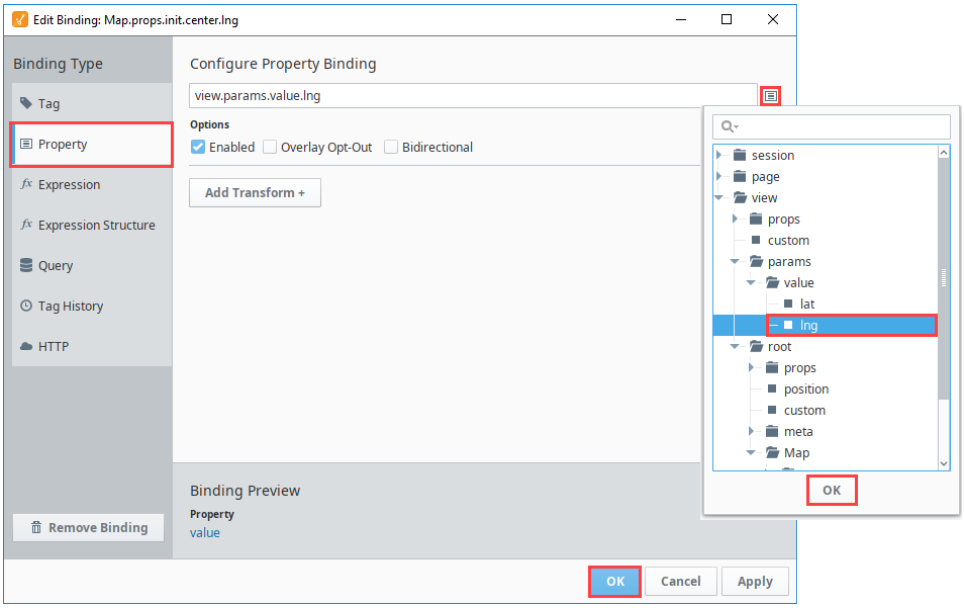


7. Next select the Map component. We need to set the map's initial geographic center to the view parameters. In the Property Editor, expand the **init.center** property.
    a. Click on the **Binding** icon next to the **lat** property.
    b. On the Edit Binding screen, select **Property** as the binding type.
    c. Click the **Browse Properties** icon. Navigate to view, the params, the value, and then the **lat** property.
    d. Click **OK**, then click **OK** again to save the binding.

> At this point, the **init.center.lat** property is bound to **view.params.value.lat** where **view.params.value.lat**'s value is "value" instead of a valid latitude number. This will cause a Component Error which is expected.

    e. Click on the **Binding** icon next to the l**ng** property.
    f. On the Edit Binding screen, select **Property** as the binding type.
    g. Click the **Browse Properties** icon. Navigate to view, the params, the value, and then the **lng** property.
    h. Click **OK**, then click **OK** again to save the binding.

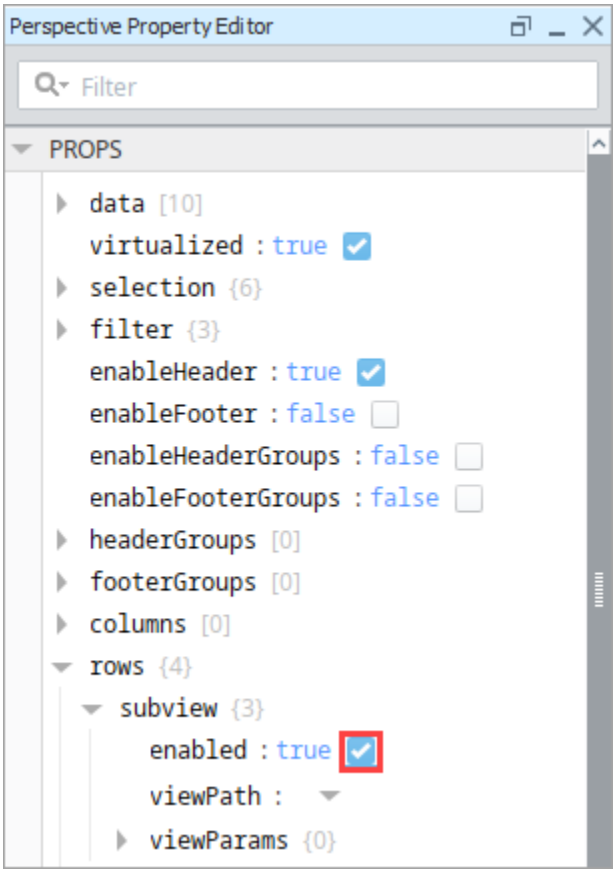
> At this point, the **init.center.lng** property is bound to **view.params.value.lng** where **view.params.value.lng**'s value is "value" instead of a valid longitude number. This will cause a Component Error which is expected.



Use the Maps View as a Subview for the Table

Lastly, we need to tell the CityStats View to use CityMaps as its subview.

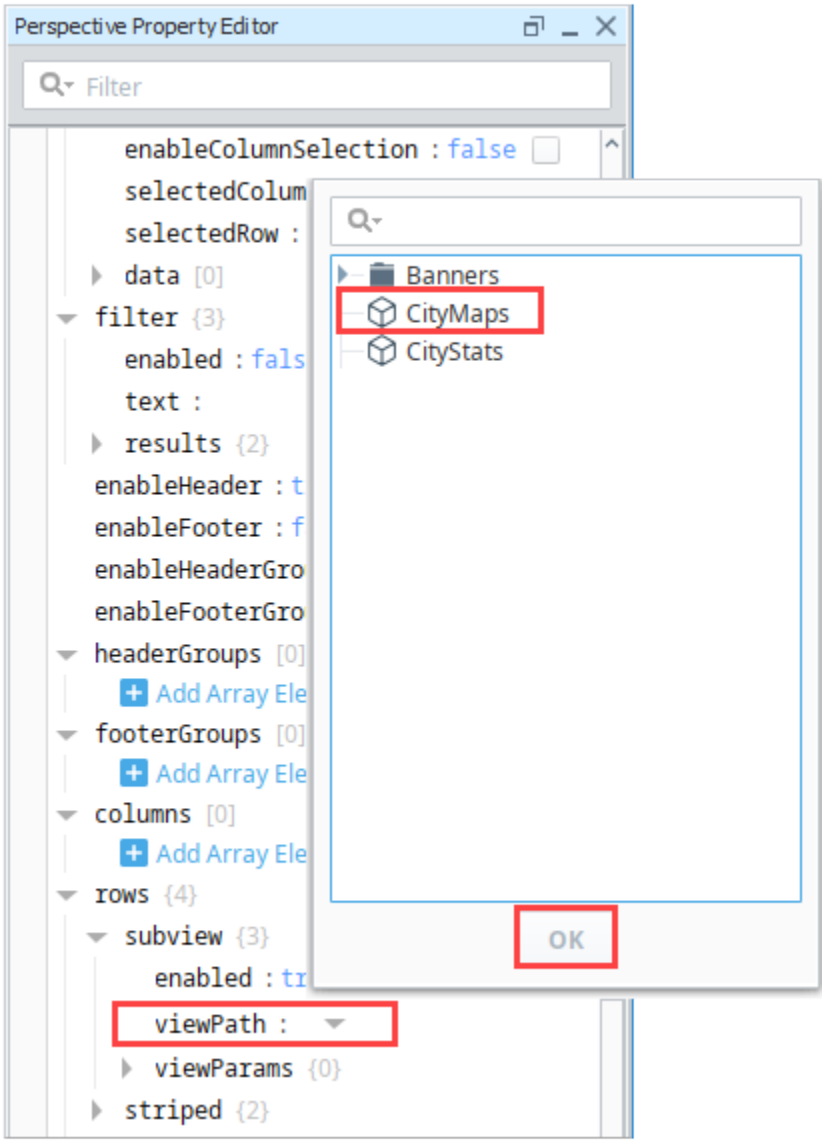
1. On the CityStats View, select the **Table** component.
2. In the Property Editor, scroll down to the **rows.subview.enabled** property.
3. Next, enable the **"enabled"** property.



As a result, you'll notice the table now has **Expand** ▶ icons for each row.

	city	country	population	lat	lng
▶	Folsom	United States	77,271	38.68	-121.18
▶	Jakarta	Indonesia	10,187,595	-6.21	106.85
▶	Madrid	Spain	3,233,527	40.41	-3.70
▶	Prague	Czech Republic	1,241,664	50.07	14.45
▶	San Diego	United States	1,406,630	37.71	-117.14

4. Next, find **rows.subview.viewPath**, and click the dropdown to see the a list of possible views. Choose **CityMaps** from the list and click **OK**.



5. **Save** your project.
6. Put the Designer into **Preview** mode. Click on the **Expand** ▶ icon next to one of the cities. You'll see a map of the city appear underneath the table row for that city. To close the map, click the **Collapse** ▲ icon.

	city	country	population	lat	lng
▶	Folsom	United States	77,271	38.68	-121.18
▶	Jakarta	Indonesia	10,187,595	-6.21	106.85
▼	Madrid	Spain	3,233,527	40.41	-3.70

▶	Prague	Czech Republic	1,241,664	50.07	14.45
▶	San Diego	United States	1,406,630	37.71	-117.14
▶	San Francisco	United States	884,363	37.78	-122.42
▶	Shanghai	China	24,153,000	31.23	121.50
▶	Tokyo	Japan	13,617,000	35.69	139.75
▼	Washington, DC	United States	658,893	38.91	-77.05

1

Self-Hiding Navigation Drawer

Navigation Drawer

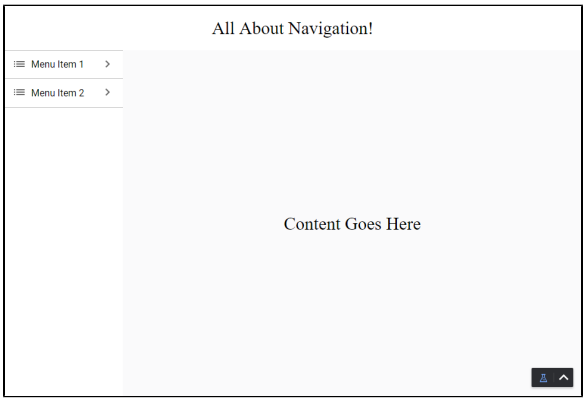
On this page

...

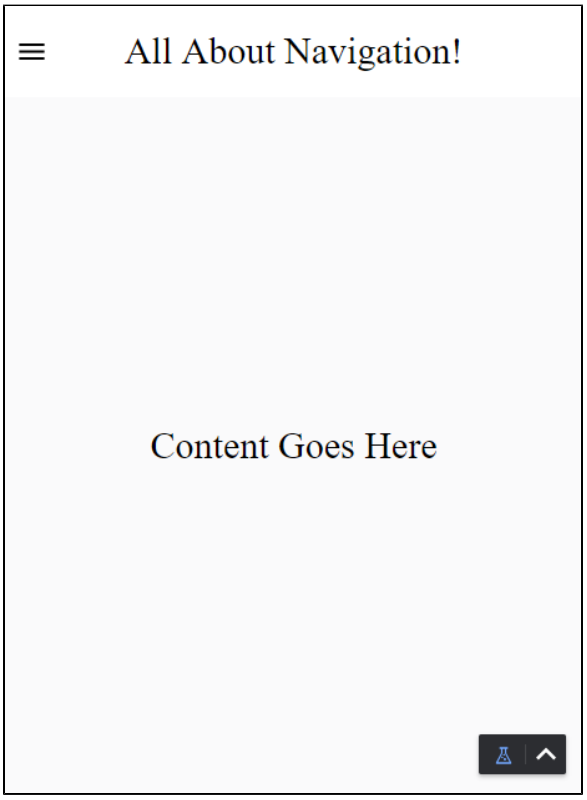
- Navigation Drawer
- Configuring a Navigation Drawer
  - Navigation View
  - Header Views
  - Page Setup
  - Configure Menu Icon

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

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



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



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

⚠ 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.

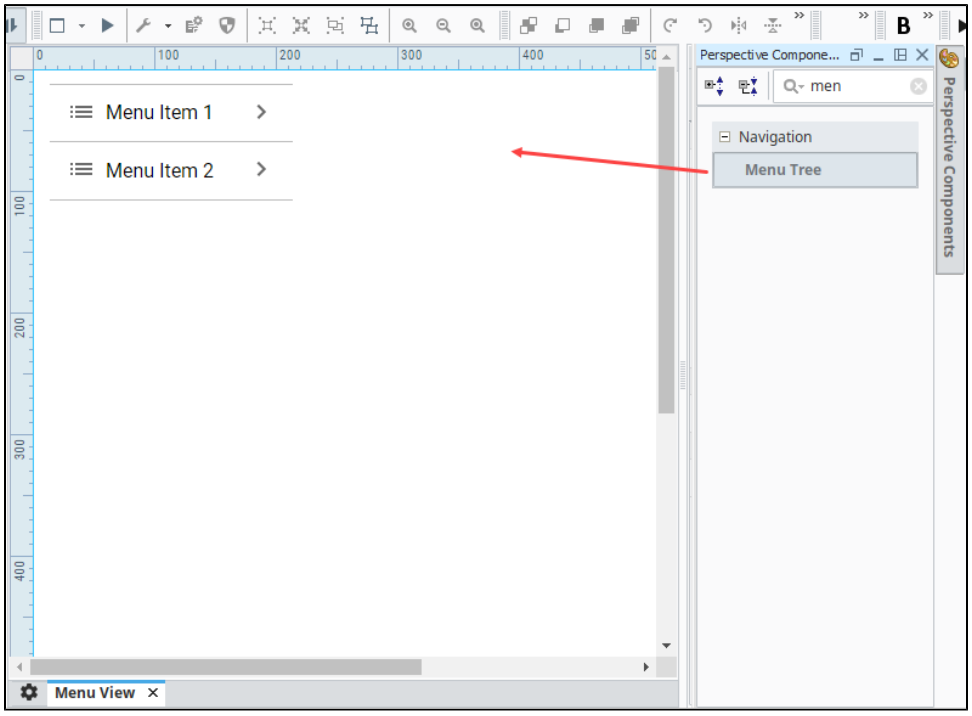
Configuring a Navigation Drawer

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 layout to **Coordinate**. This is where you'll put the navigation menu. Click **Create** View**.**
3. Set the width of the view to 200 pixels.
4. Drag a **MenuTree** component onto the Menu View. Configure the component as you would if you were using a standard docked view.

   Tip: 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.

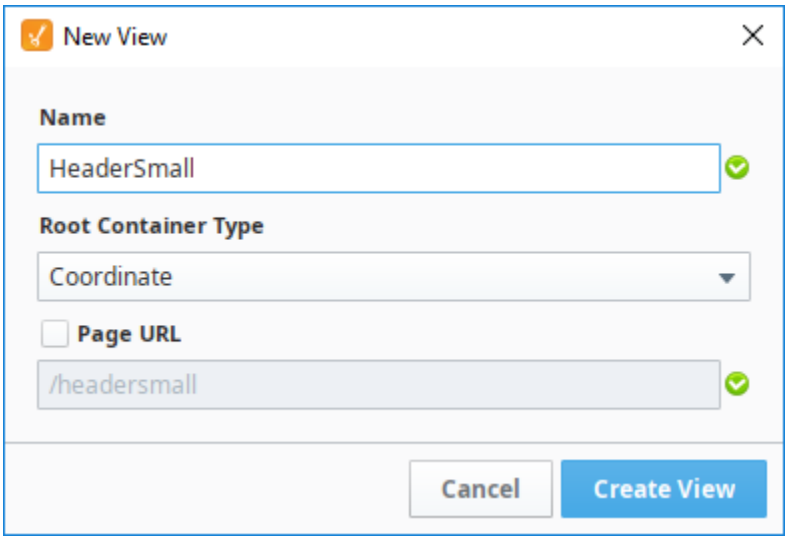


Header Views

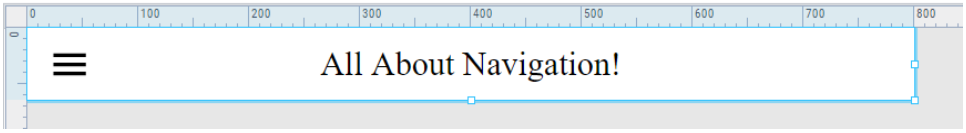
Next, we will create our header views. The two views will be setup in a **breakpoint** container, which will swap between a small header with an icon, and a big header without one.

1. Create a small header view for our mobile UI.
   a. In the Project Browser, right click on **Views** and click **New View**.

b. Name the new view **HeaderSmall**, with a layout of Coordinate. Click **Create View.**

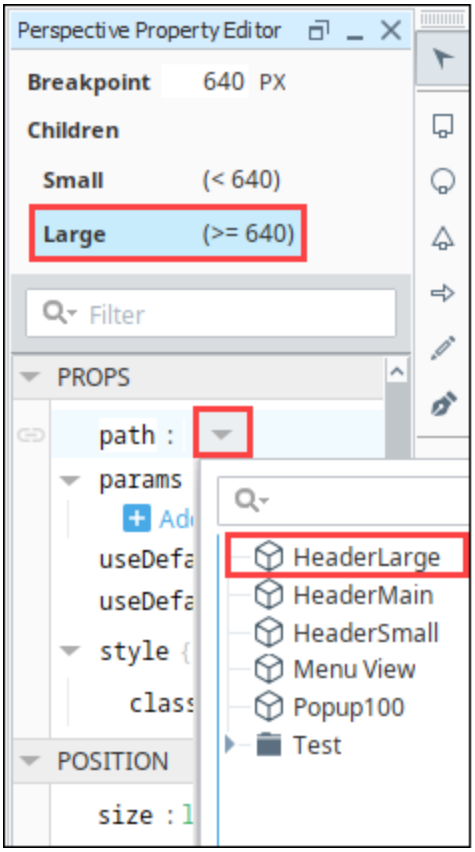


c. Set the height of the view to 75 pixels.
d. Drag an Icon component onto the upper left side of the view, then click on the component to select it.
e. In the Property Editor, its **path** property to **material/menu**. You can of course use whatever icon you'd like; a list of all the icons in the material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



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

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

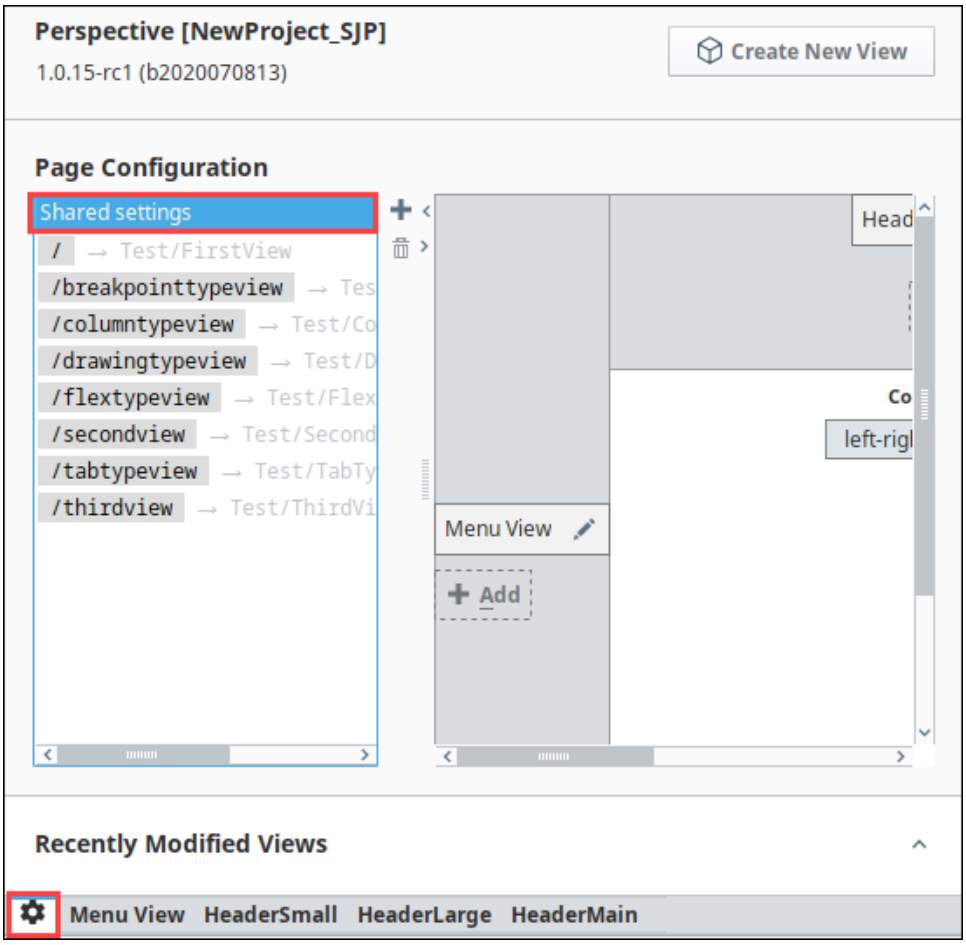


f. Now we'll do the same for the small child of this Breakpoint container. In the Property Editor, click on **Small** (under Children).
g. Drag an Embedded View component on the HeaderMain view.
h. In the Property Editor, click on the Expand ▼ icon next to the **path** property then select **HeaderSmall**.

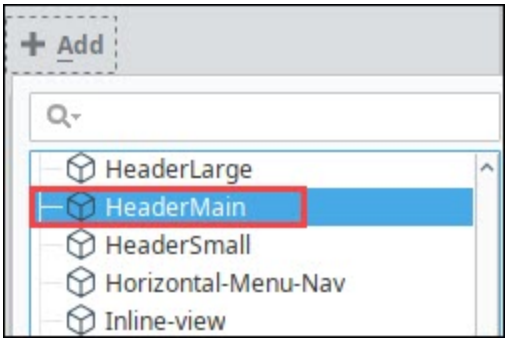
Page Setup

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

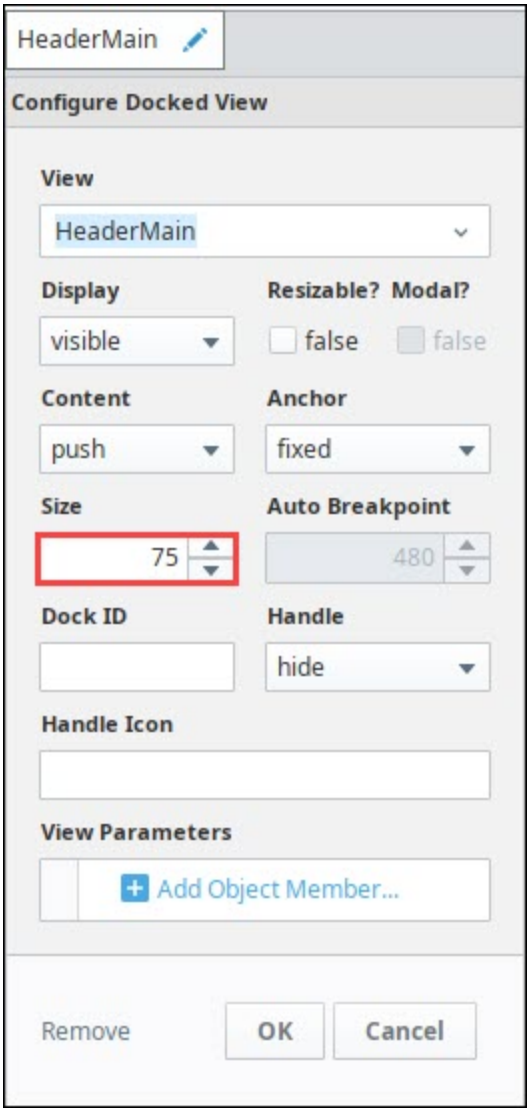
1. Click on the Settings ⚙ icon in the bottom left of the designer to access the Page Configuration menu. Select **Shared settings**. We're going to add our two docked views here, so they show up on every page.



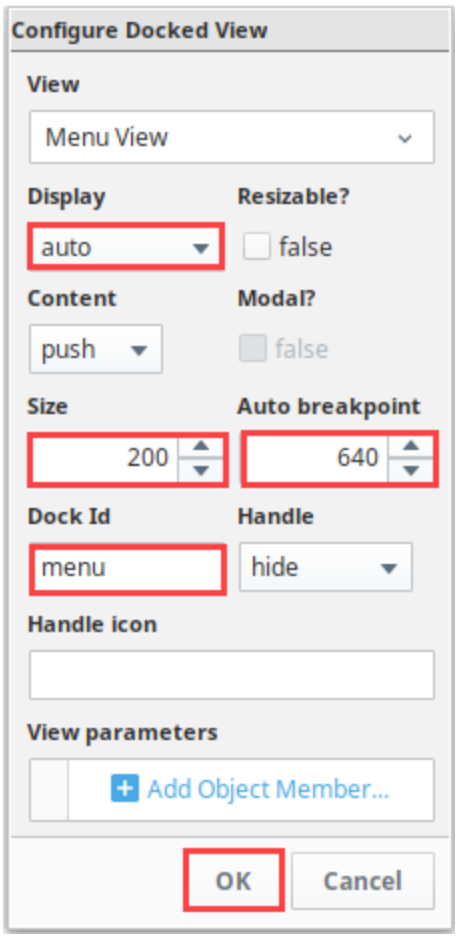
2. On the top dock, click the Add ✚ icon. Select the **HeaderMain** view and click **OK**.



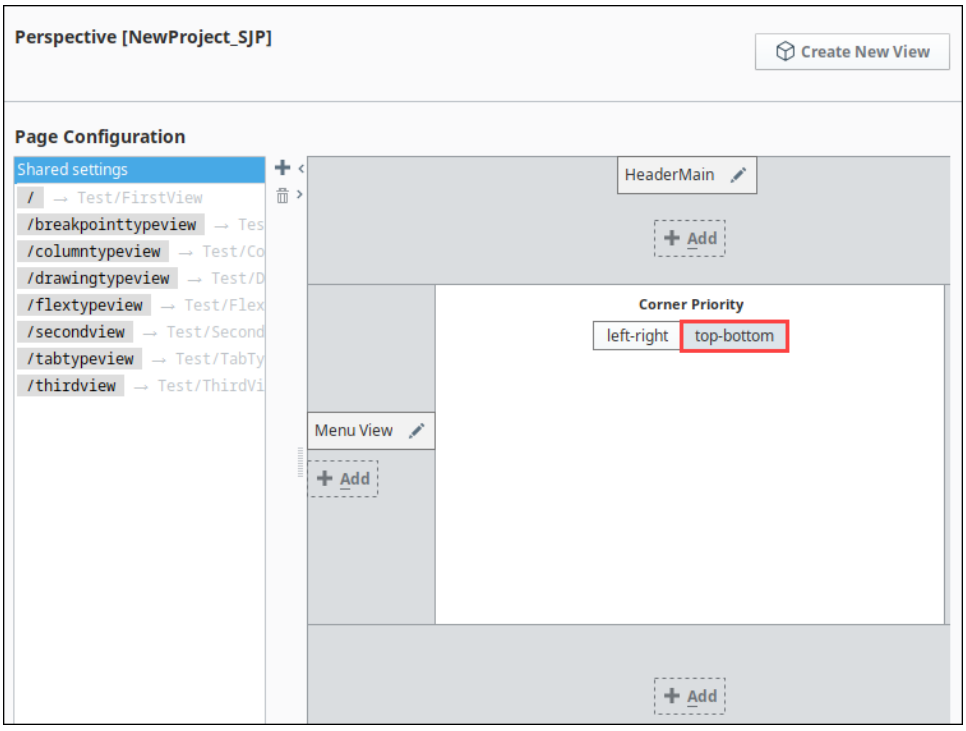
3. Click the Edit icon next to the HeaderMain docked view.
4. In the Configure Docked View menu, set the **size** to 75.



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.)



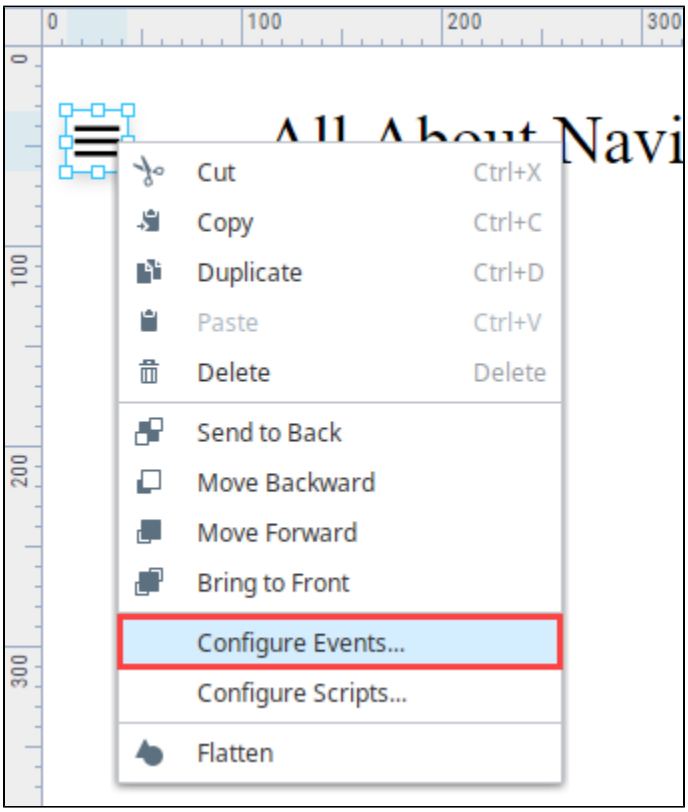
6. In the center of the Page Configuration menu, set the **Corner Priority** to **top-bottom**. The other option won't look quite right.



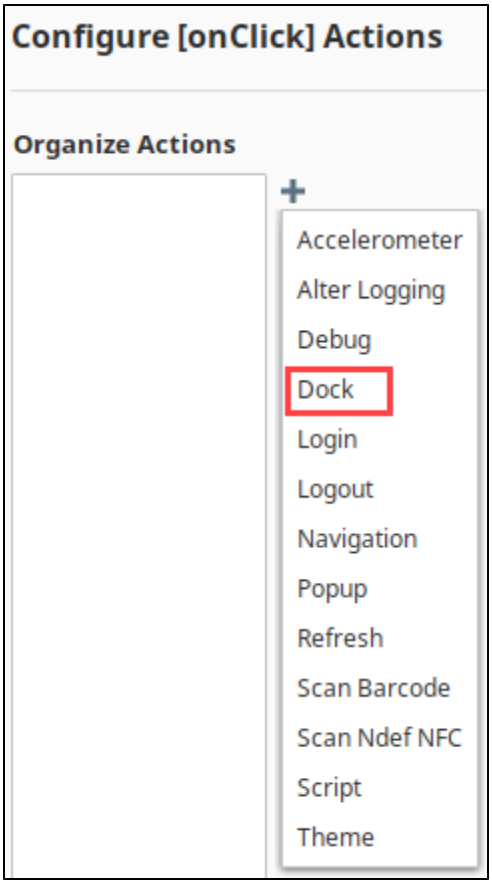
Configure Menu Icon

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

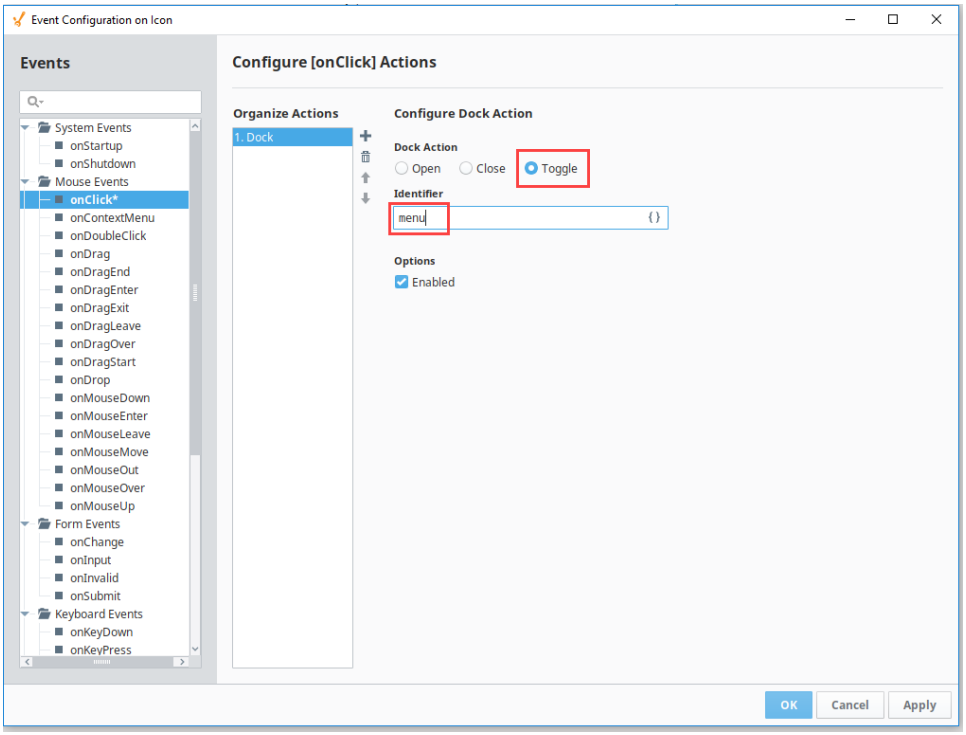
1. Open the HeaderSmall View.
2. Right-click on the Icon component, and select **Configure Events...**



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



5. This action needs the identifier we created in step 9. Set the Dock Action to **Toggle**, and the Identifier to **menu**. Click **OK**.



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

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

Configuring a Dashboard

The Dashboard exposes widgets to end users in a Perspective Session so they can customize their dashboard layout for their individual needs. Widgets are views that are pre-configured in the Designer and made available to Perspective Session users. End users have the flexibility to add, remove, resize, move around, and even configure widgets in the dashboard of their Perspective Session without having access to the Designer. Users can interact with widgets in a session on both desktop and mobile devices. There may be some minor variances in how a user can interact with their dashboard between desktop and mobile devices, but the principle is still the same.

On this page

...

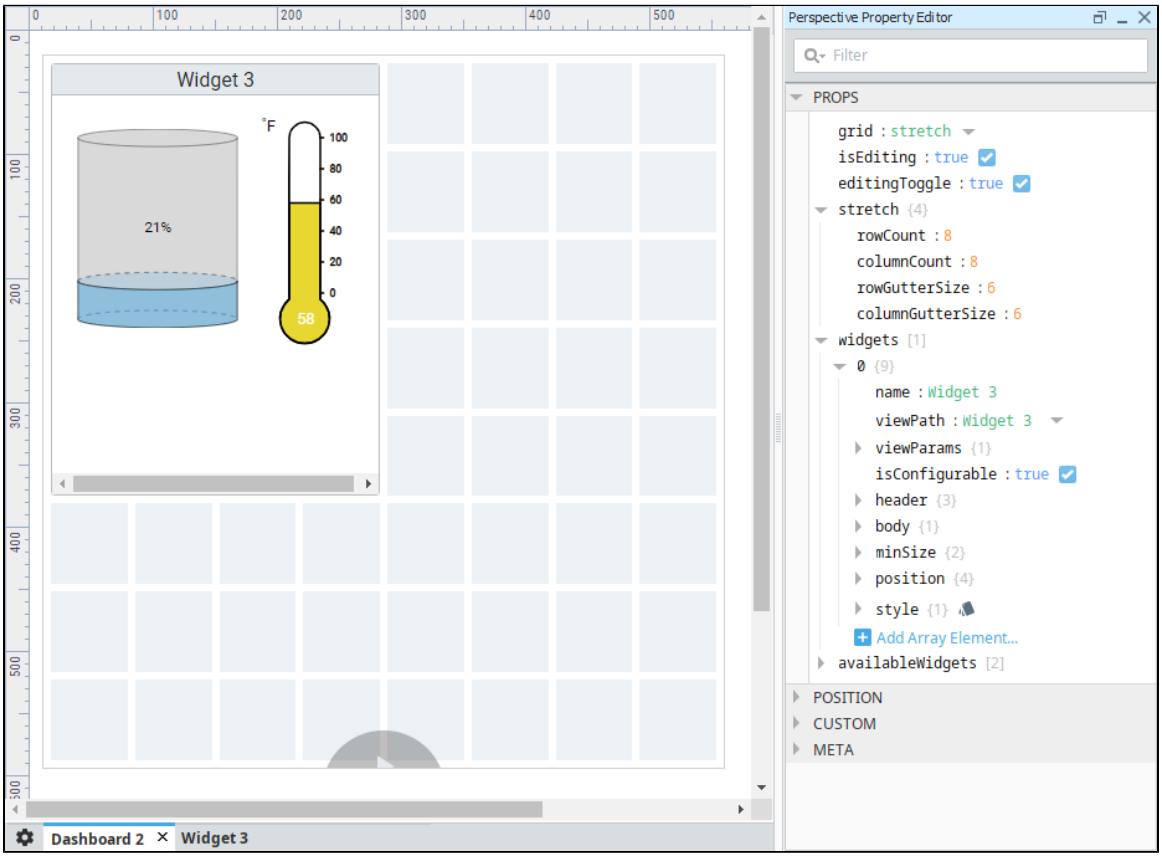
- Configuring a Dashboard Component
- Setting Up a User Dashboard
  - Adding a Widget
  - Removing a Widget
  - Moving a Widget
  - Resizing a Widget
- Configuring a Widget
  - Setting a Widget as Configurable in the Designer
  - Creating a Configurable View in the Designer
- Saving Perspective Session Edits and Populating Widgets

Configuring a Dashboard Component

Configuring a Dashboard starts with designing widgets and having a selection of pre-configured widgets for users to choose from to configure their individual dashboards. Designers create the widgets and make them available for end users to use in their individual dashboards. By making the widgets available using the 'availableWidgets' property, the widget overlay modal is populated with a searchable list of all the available widgets a user can add to their dashboard. The dashboard component contains a host of addtional properties that can be configured based on the end-user requirements.

The Dashboard component uses a grid system based off of CSS grid specifications to position and place your widgets. The Property Editor settings of the Dashboard component control the general layout of the grid. They specify the responsive mode: fixed or stretch, if the dashboard is editable, and if each widget is configurable and available in a Perspective session. The image below shows one widget on a dashboard in the Designer along with some of its properties.

To learn more about Dashboard properties, refer to the Dashboard component page.

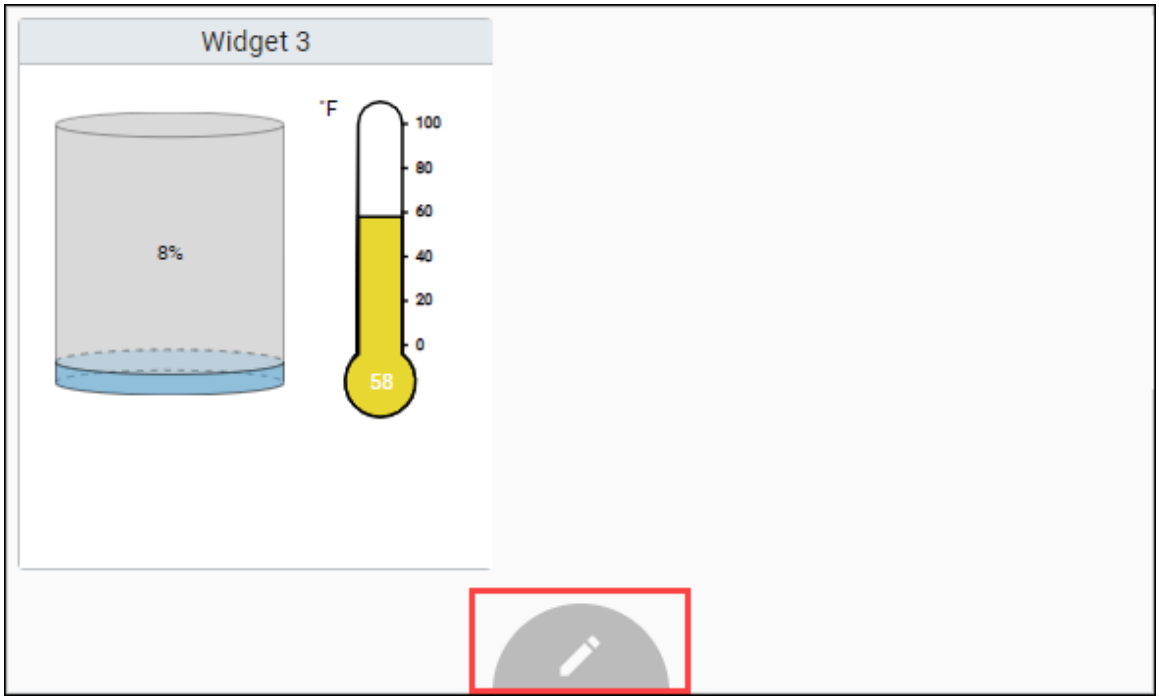


Setting Up a User Dashboard

Setting up a dashboard starts with users choosing from a list of pre-configured widgets to configure their dashboards in a Perspective Session based on their individual needs.

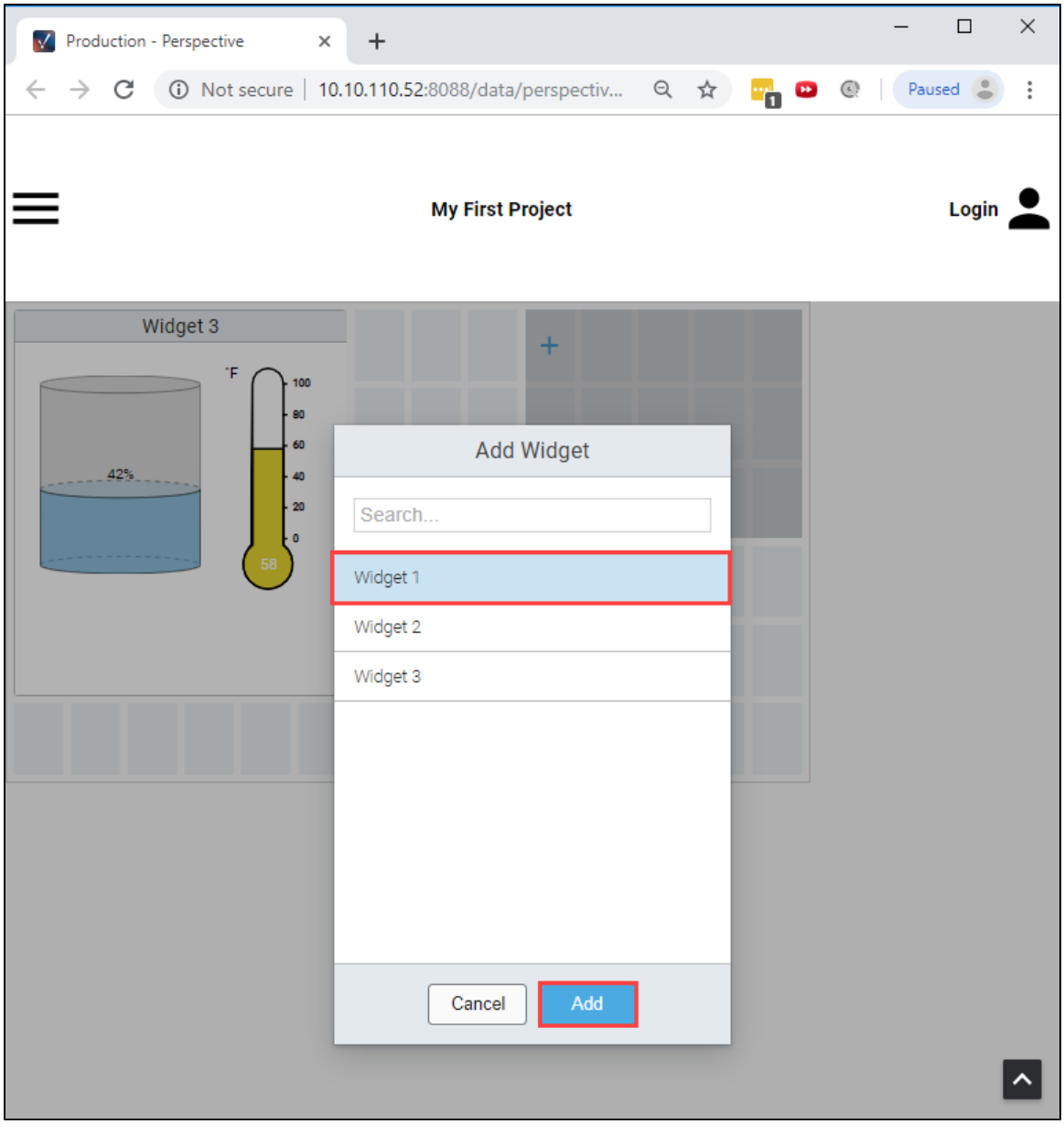
To edit the dashboard in a Perspective Session, the user can put the dashboard into Edit mode by clicking the Edit icon at the bottom of the dashboard and deciding what widgets they want, where they want them, and how they want them configured. They can add, remove, resize and configure widgets, including the ability to interact with widgets such entering text in a text field or displaying/hiding components in a widget. You can also remove this control entirely and implement your own by configuring the `'editingToggle'` property on the component. Refer to the Dashboard component properties for more details.

The following sections on this page describe how to setup your own dashboard.



Adding a Widget

There are two ways a user can add a widget in a Perspective Session: by clicking on a single grid cell, or by dragging a grid cell over multiple grid cells that opens an add widget overlay as shown in the image below. Both ways result in displaying the add widget modal which provides a searchable list of all of the available widgets a user may add to their dashboard.

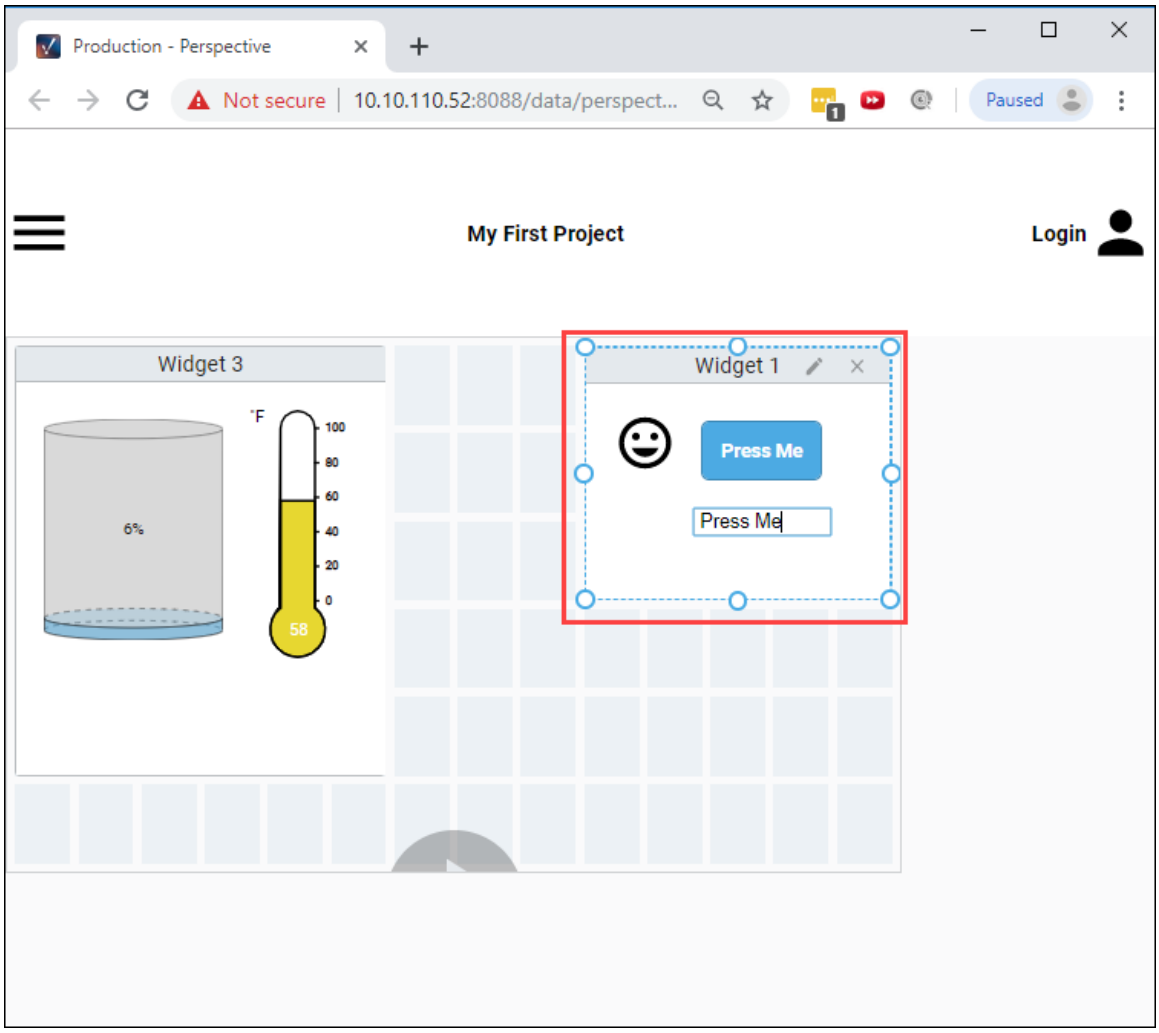


Dragging a grid cell creates an add widget overlay that specifies the desired dimensions of the widget to add. If the desired widget position overlaps other widgets, the overlapped widgets will be moved to any available space on the dashboard. Widgets do not overlap when being added, resized, and moved unless there happens to be no space for a widget so that it is placed within the grid.

Widget's Minimum Dimensions

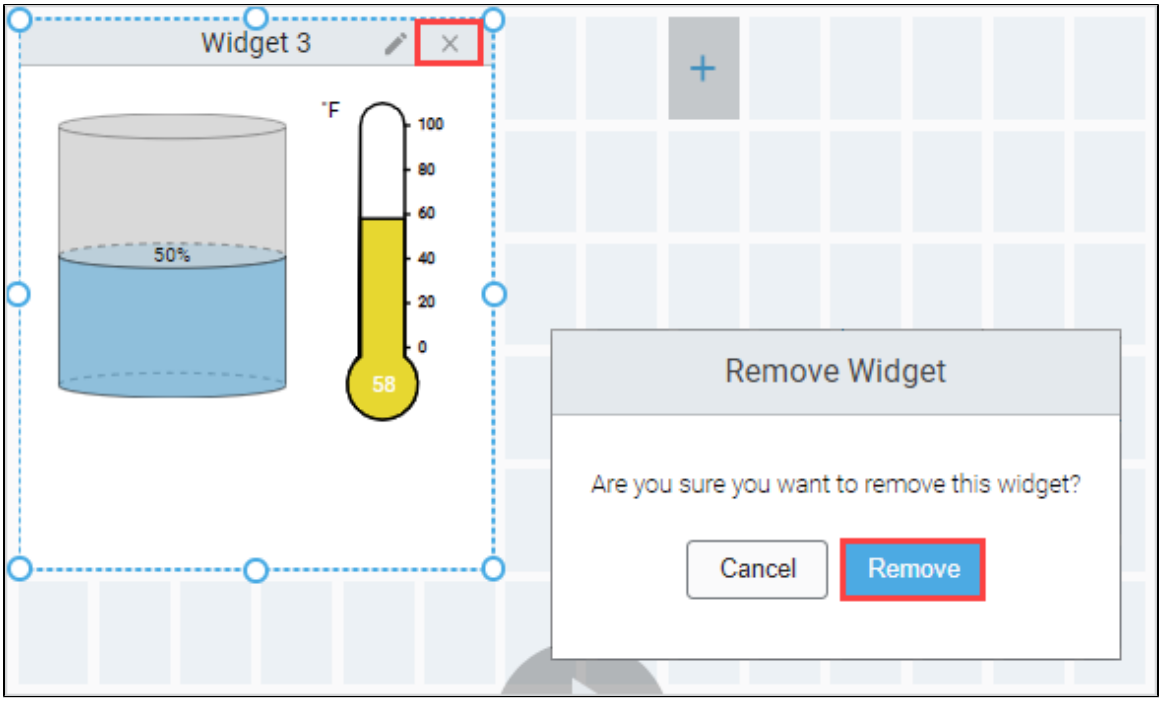
When adding a widget, if the desired dimensions are less than the configured minimum dimensions, the desired dimensions will get overridden by the minimum dimensions. If a single grid cell is clicked, the configured default dimensions will be applied, if and only if, the default meets the required minimum dimensions, otherwise the minimum dimensions are applied. By default, the minimum and default dimensions for a widget are 1x1.

On mobile devices, activating a grid cell requires a long-press of about a second. Once a grid cell is activated, you can then drag to create the add widget overlay. The image below shows Widget 1 dropped over the multiple selected grid cells in the dashboard. You'll notice the active widget has a dashed blue border.



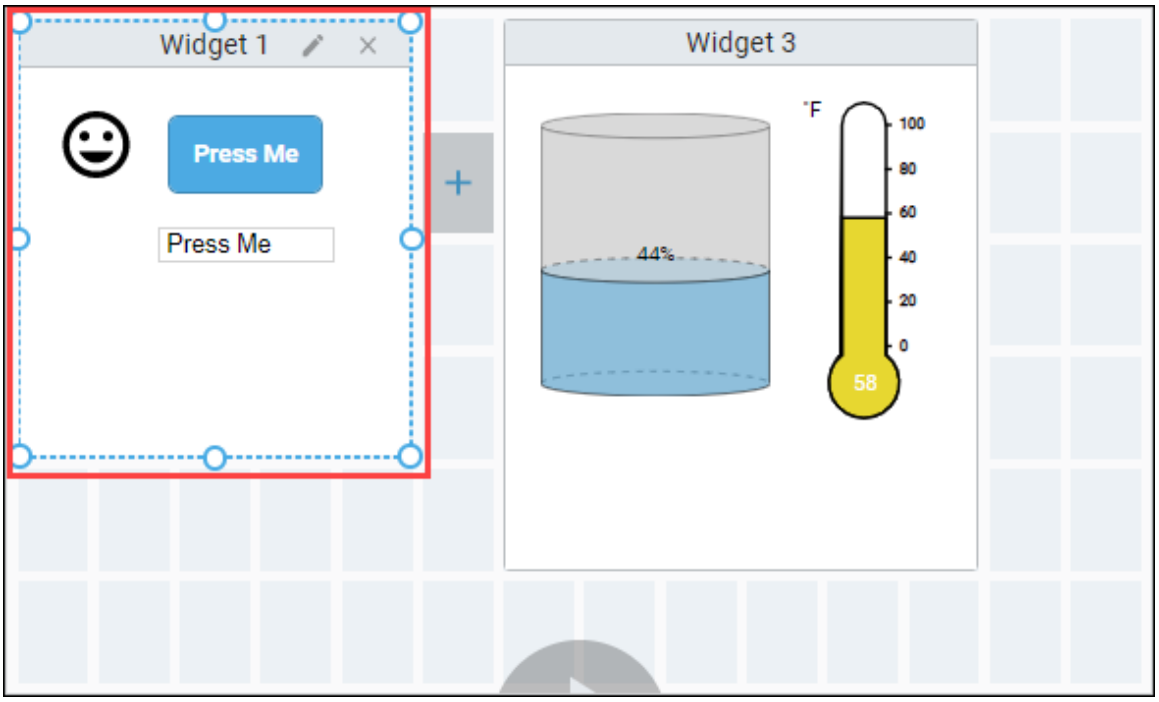
Removing a Widget

Click the **Edit** icon, select the widget, and you'll notice the widget has a dashed blue border indicating the widget is active, then click the **Delete** '**X**' icon in the top right corner to remove the widget from the dashboard. You will then be prompted with a confirmation modal to delete the widget. Click **Remove**.



Moving a Widget

Put the dashboard in Edit mode, select the widget so that it becomes active (dashed blue border). Drag the widget to the desired position. As you move the widget, any overlapped widgets will be repositioned into the first available space.



Resizing a Widget

To resize a widget, put the dashboard in Edit mode, then simply select the widget you'd like to resize and drag one of the resize handles. If, while resizing, the widget overlaps other widgets, the overlapped widgets will be repositioned into the first available space.

Configuring a Widget

The dashboard allows your users to configure a widget in a Perspective Session. To do this, you need make a few changes to your view and Dashboard component configuration.

Setting a Widget as Configurable in the Designer

To make a view allow configuration, you need to set the **isConfigurable** property for each widget that needs to be configured. This will set a param value on your view (in the runtime) that you can use to create a configuration display in your view. The purpose of this parameter is to avoid having to make a separate widgets for each possible variation of the same view.

1. Select the Dashboard component.
2. Expand the **availableWidgets** parameter, and expand the array object for the widget that you want to make configurable.
3. Set the **isConfigurable** property to 'true' for this widget.

Creating a Configurable View in the Designer

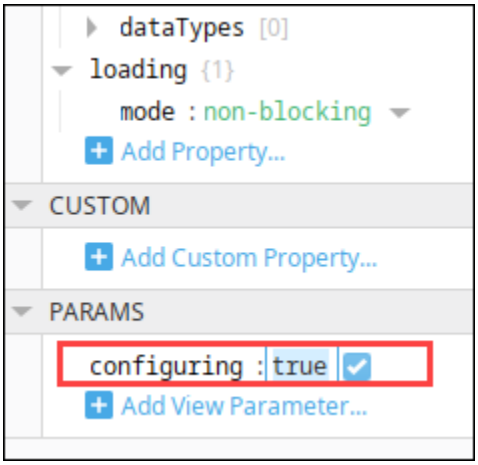
To make a view configurable, you need to do a bit of work to alter what is in the view. This is possible because the Dashboard component was created to use a parameter named **`configuring`** that is set to **'true'** when the widget is in put into configuration mode. The idea here is to have a second 'mode' or 'display' version of the view that has controls on it to effect the primary display. The best way to do this is to create two containers in your view; one for configuration, and one for display. You can then bind the visibility on each container so only one is shown at a time.

To learn more about using parameters to pass properties, refer to the Perspective Component Properties page. You will not need to pass any value into the param though, it is done automatically for you if you get the param name correct.

1. Create a new Coordinate view. For the example, we named our Configurable_View.

   > ⚠ If you use a Flex container, some of the settings will be different further down in the example.

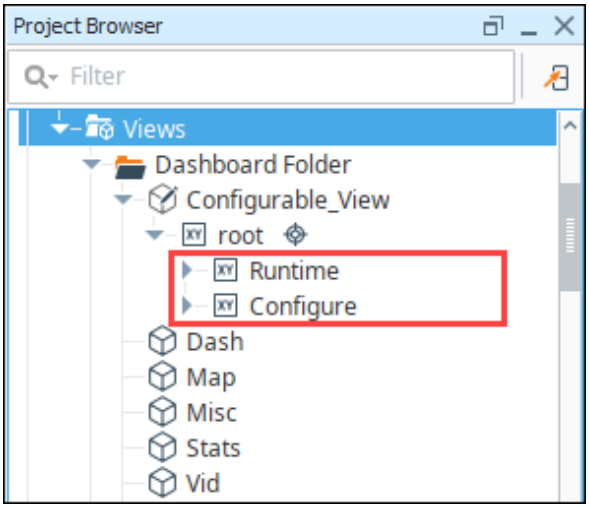
2. In the Property Editor, add a view param. Name the param **`'configuring'`** and set the value to 'true'. Note the spelling and (lack of) capitalization.

3. Drag a Coordinate Container component inside your view. Give the container a good name like 'Runtime.'
4. Add any display components you want.

   > ⚠ If you started from an existing view, move all existing components into the new container then make the container fill your entire view.

   a. Deep Select the Configure container.
   b. Drag a **Cylindrical Tank** component into it.
   c. Bind the value property to a Tag.
   d. Drag a **Temperature Gauge** component into it.
   e. Bind the value property to a Tag.
5. We need to create a second new container in the view for your configuration. **Duplicate** the Runtime container. Give the container a good name like 'Configure.' This container will be a sibling to the Runtime container, not inside the Runtime container.



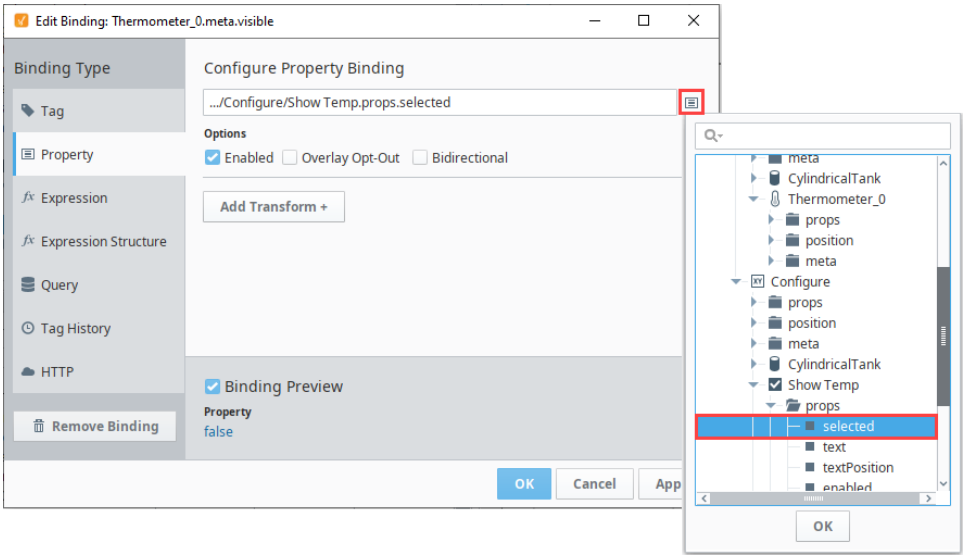
6. Add a Checkbox to toggle the temperature component visibility.

> ⚠ If you started with an existing view, this step is completely up to you. You will decide what should be configurable and create controls for that in your configuration container. For example, you could create a list of Tags for the user to select between and display only the selected Tags on a chart.

   a. Deep Select the Configure container.
   b. Drag a **Checkbox** component into it.
   c. Set the Name and Text to "Show Temp".

7. Now we need to alter the Runtime container components to listen to our new controls.

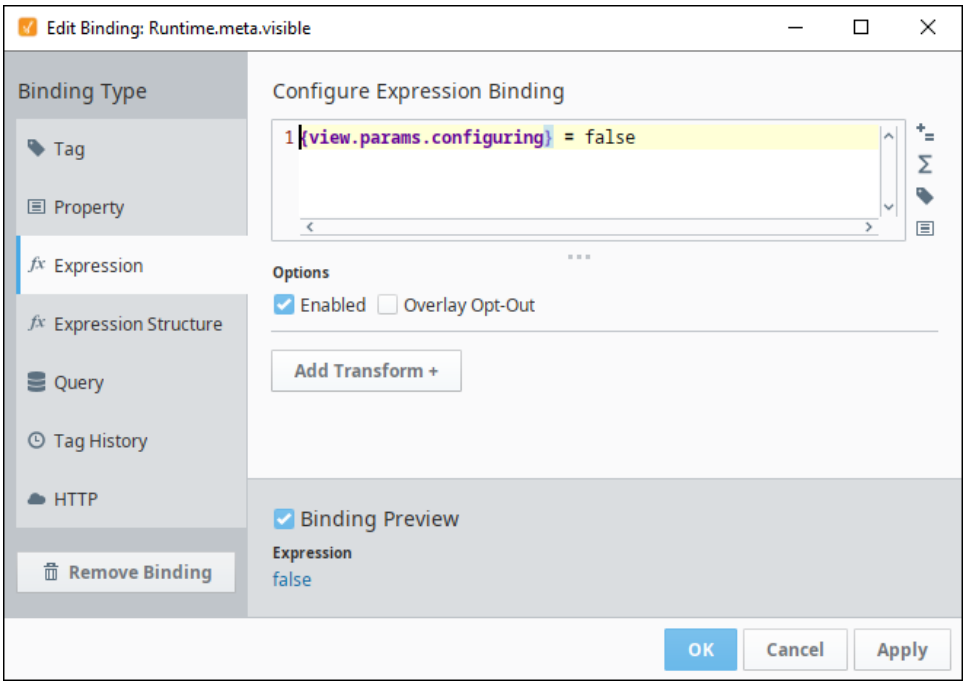
   a. Deep Select the Runtime Container.
   b. Select the Temperature Gauge component.
   c. In the Property Editor under META, bind the **Visible** property to the Selected value of the Show Temp Checkbox component.



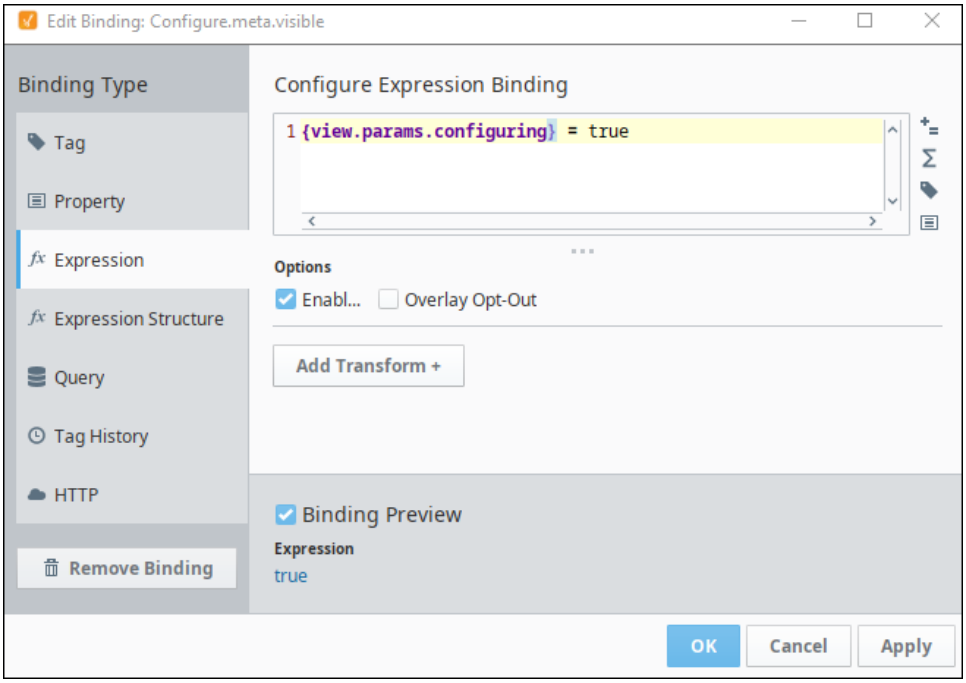
8. Now we just need to show one container at at time.

> ⚠ If you used a Flex container at the start of this example, then use the '`display`' property instead of the '`visible`' property in the following steps.

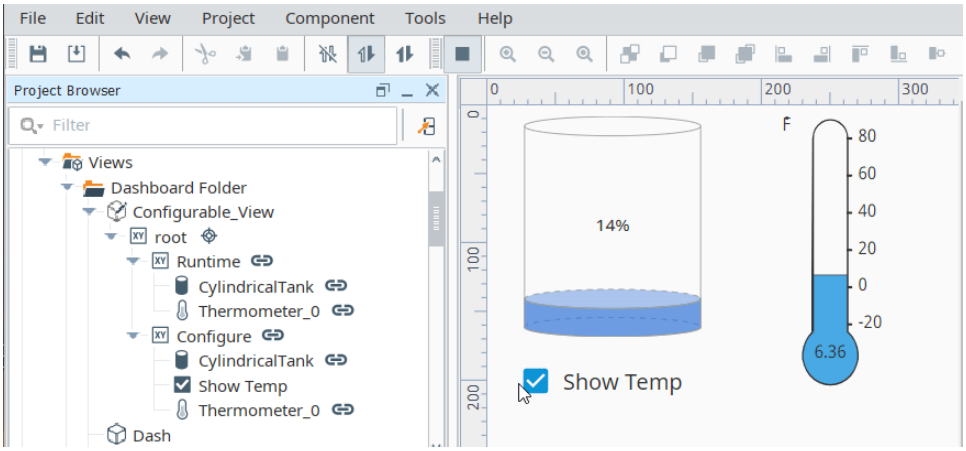
   a. Bind the '`visible`' property for the Runtime container using an expression. It should be true when the **`configuring`** param is false.



b. Bind the '`visible`' property for the Configure container using an expression. It should be true when the **`configuring`** par am is true.



9. Save your project and then put the Designer into Preview mode. When you click on the Temp Show button, you'll see the Temperature component appear or disappear.

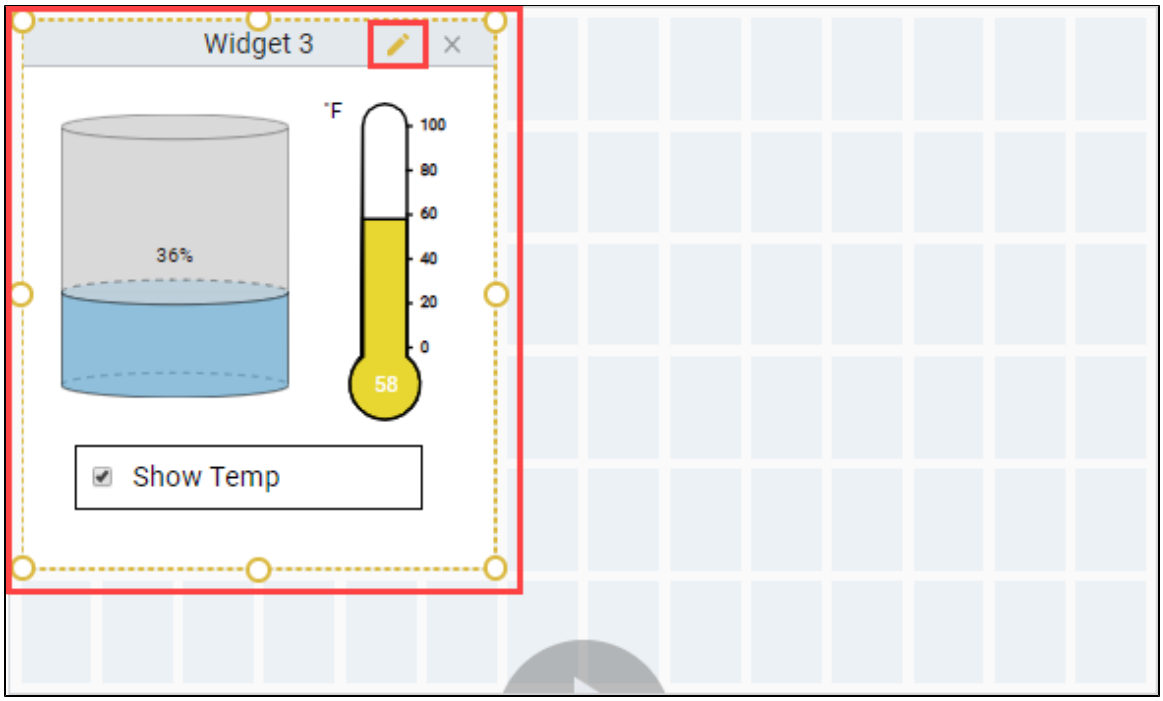


10. Now you can use this view in your dashboard.

Configuring in the Dashboard Component in the Runtime (Perspective Session)

This section is for the people using the Dashboard in a Perspective Session.

1. To use a configurable widget in the runtime, put the dashboard in Edit mode and select the widget you'd like to configure.
2. Click the edit icon (pencil) in the top right corner of the widget. The widget's border will change colors from blue to orange (shown in the image below).
3. The view changes to show the 'configuring' mode you set up previously for the view, allowing users to configure the widget.



Saving Perspective Session Edits and Populating Widgets

Edits that an end user makes in their dashboard in a Perspective Session are not automatically saved and do not persist when the end user's session restarts. A session can be refreshed within the same session. One possible solution for populating widgets for the next editing session is to add a property change script on the `'widgets'` prop to listen for changes and then write that value back to a database along with any user information derived from the active session. The value of the `'widgets'` prop will be an array of QualifiedValues, which you'll

need to handle accordingly.In similar fashion, consider adding an '`onStartup`' event action that will query the database and then populate the '`widgets`' prop with the users last saved configuration and optionally populate the '`availableWidgets`' prop (possibly for varying user roles).

Related Topics ...

- Perspective Dashboard Component
- Passing Parameters
- Views and Containers

Download and Upload Files

Downloading and uploading files from a Perspective session typically involves storing and retrieving files from a database. A table will store all of the available files, and each row of the table represents a new file. This allows for long term storage that is accessible from any project.

The examples on this page show suggested methods of uploading files from a session, as well as how to download them.

On this page

...

- Query Examples
- Uploading a File
- Downloading a File

Query Examples

Before following along with the examples on this page, you'll need to create a table in the database that will hold the files. This process can vary by database, along with the column datatypes.

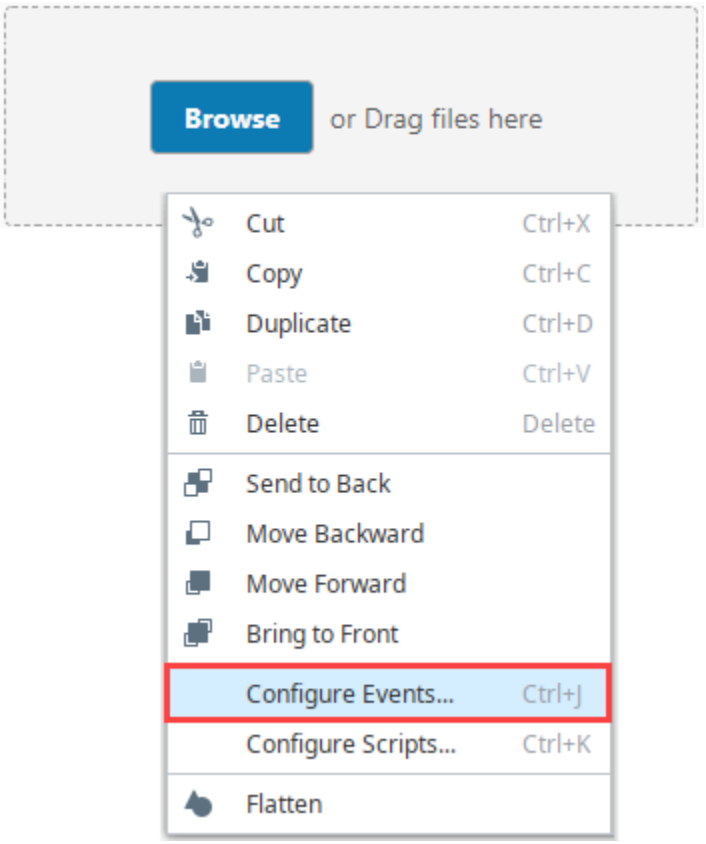
For the sake of brevity, the example assumes the files will be stored and retrieved from a SQL Server database. You may need to modify the query examples on this page if using a different database. The "files" database table used by these examples contains the following columns:

- **id** - integer, primary key, identity
- **filename** - varchar (255)
- **filedata** - varbinary (MAX)

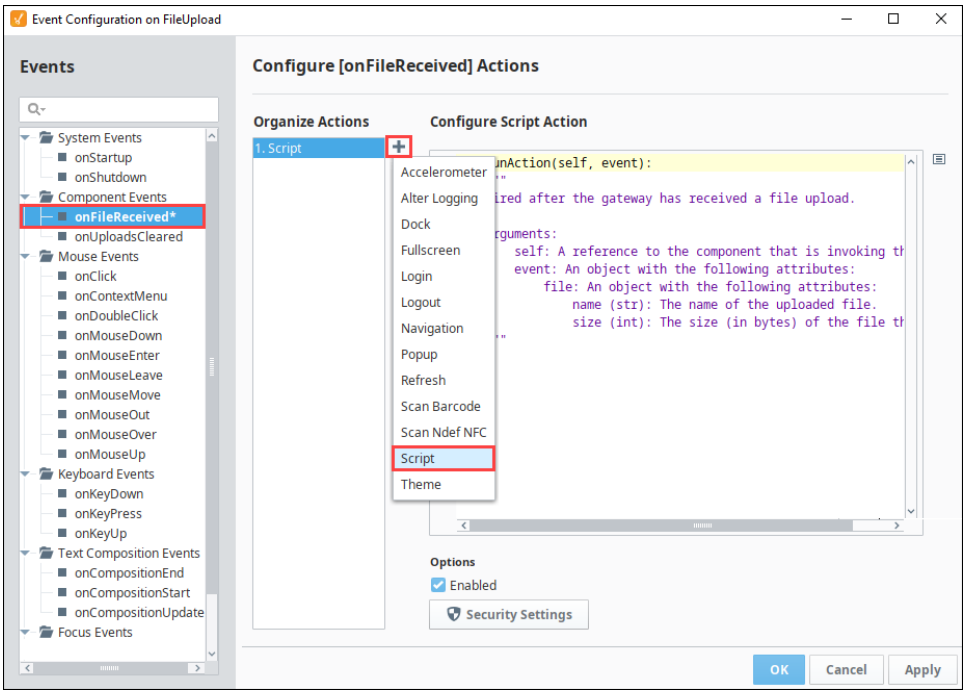
Uploading a File

To upload a file in Perspective, we will want to use the File Upload component. This allows us an easy way to manage the upload.

1. Add the File Upload component to a view. The File Upload component has everything we need to upload a file into the database.
2. Right click on the File Upload component and select **Configure Events**.



3. Select the onFileReceived event and click the **Add** + icon to add a script action to it.



4. Add the following script to the script action:

```
# Grab the file name and data
filename = event.file.name
filedata = event.file.getBytes()

# Use a query to insert the file
query = "INSERT INTO files (filename, filedata) VALUES(?, CONVERT(varbinary(MAX), ?))"
args = [filename, filedata]
system.db.runPrepUpdate(query, args)
```

   As mentioned above, the query will also vary based on the database used.

5. Click **OK**. You can test out the upload functionality by dragging a file onto the File Upload component, either from a session, or the designer while it's in preview mode.

Downloading a File

To download a file that is stored in the database in Perspective, we will want to use the system.perspective.download function. This will allow us to download the file data that we receive from the database.

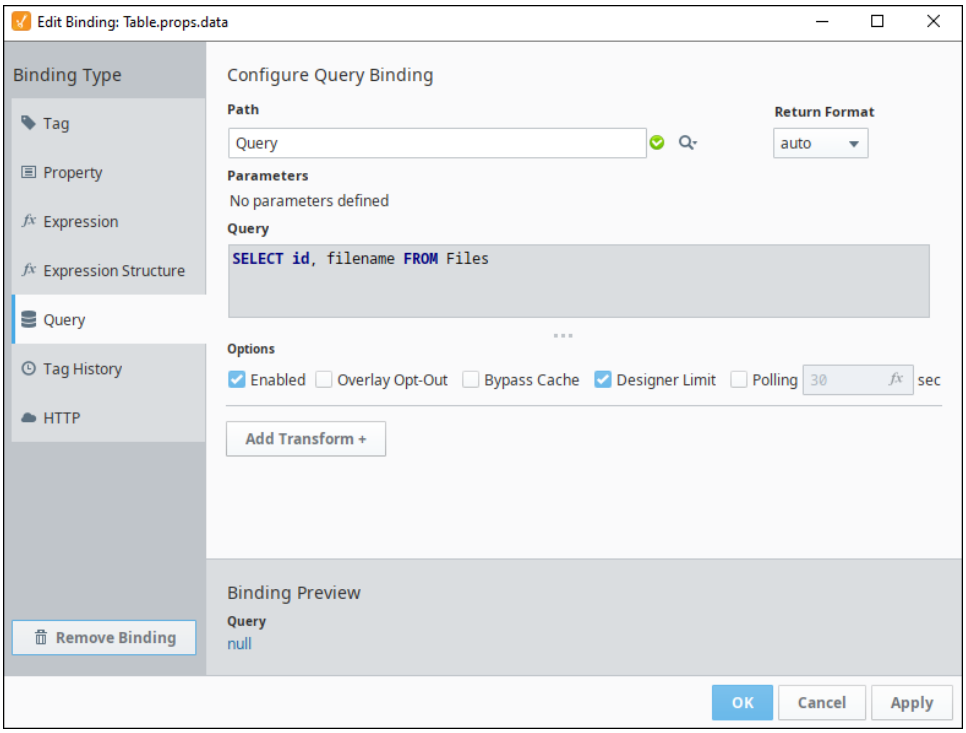
This example will show you how to do several things:

- Create a named query, that will return the contents of our file database table
- Create a table component, that shows a listing of potential files to download, using the named query above in conjunction with a Named Query Binding.
- Add a button component, that will allow users to download a file, assuming one of the rows in the table component are selected.

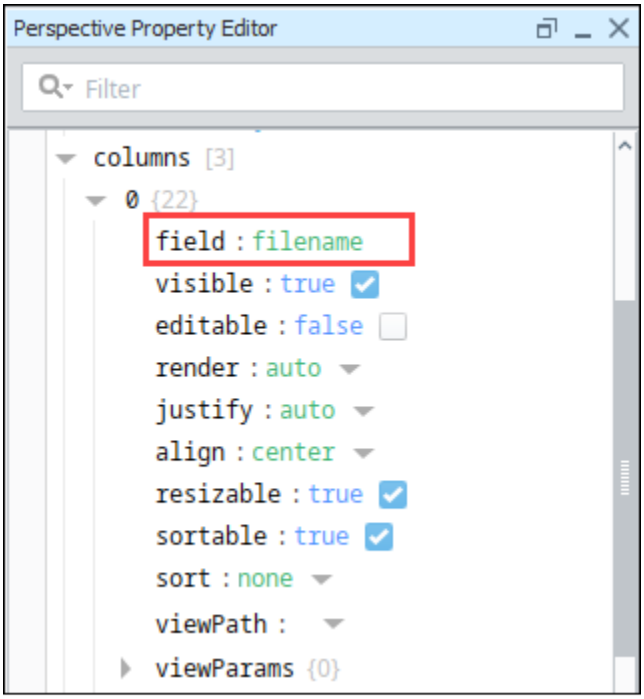
1. Create a Named Query that we will use to pull a list of files out of the database table. We're using a named query here since a named query binding is the easiest way to run a query when the view loads.
   The query should pull out the id of the row which we can use to later query the data, as well as the filename which the user can use to identify the file.

```
SELECT id, filename FROM Files
```

2. On a view, add a Table component. This will display a list of all files we currently have in the database table.
3. On the Table's **data** property, set up a binding. The binding should be a Query type, and it should use the query that we just made. We want to return the data in a JSON format, and you can enable polling so that it automatically updates if new files get uploaded.



4. On the Table's columns property, add an array element. Set **columns.0.field** to the name of the column that holds the filename. This will display only the filename column, as the id column does not need to be visible.

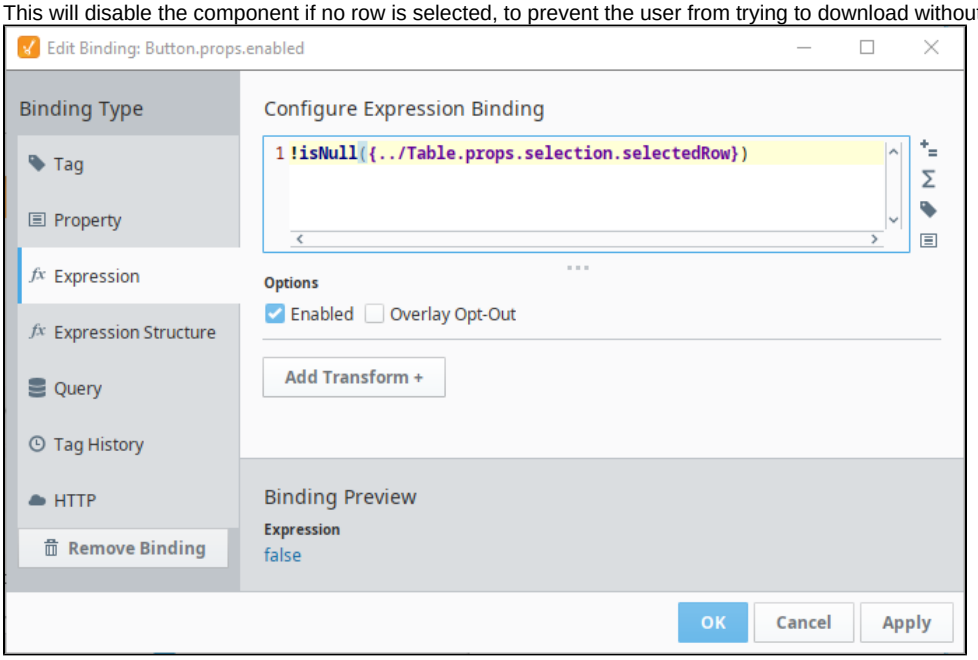


5. Add a Button to the view. This button will be used to download the file after the user has made a selection. However, we also want to make sure the user can't press the button unless a row in the table is selected.

6. On the Button's enabled property, configure a binding. The binding type should be an expression. The expression should check to see if the Table's selected row is null, and invert it.

```
!isNull({../Table.props.selection.selectedRow})
```

This will disable the component if no row is selected, to prevent the user from trying to download without making a selection.



7. Right click on the Button and go to **Configure Events**.
8. Select the onActionPerformed event, click the **Add** ✚ icon to add a script action to it.
9. Add the following script to the script action:

```
# Grab the selected row
selectedRow = self.getSibling("Table").props.selection.selectedRow

# Use the selected row to grab the id of the file at that row
id = self.getSibling("Table").props.data[selectedRow].id

# Use the id to grab the file data out of the database, along with its corresponding name.
query = "SELECT filename, filedata FROM Files WHERE id = ?"
args = [id]
data = system.db.runPrepQuery(query, args)

# Pull out the file name and data
filename = data[0][0]
filedata = data[0][1]

# Download the file
system.perspective.download(filename, filedata)
```

10. Test the script by selecting a row in the table and clicking on the button while in Preview mode.

Table Column Configurations

In a Perspective Table component, you have the ability to replace a cell with something other than just text or a number. Instead, you can have the column render other objects altogether such as progress bars or a view. This page contains several examples of changing how a column renders.

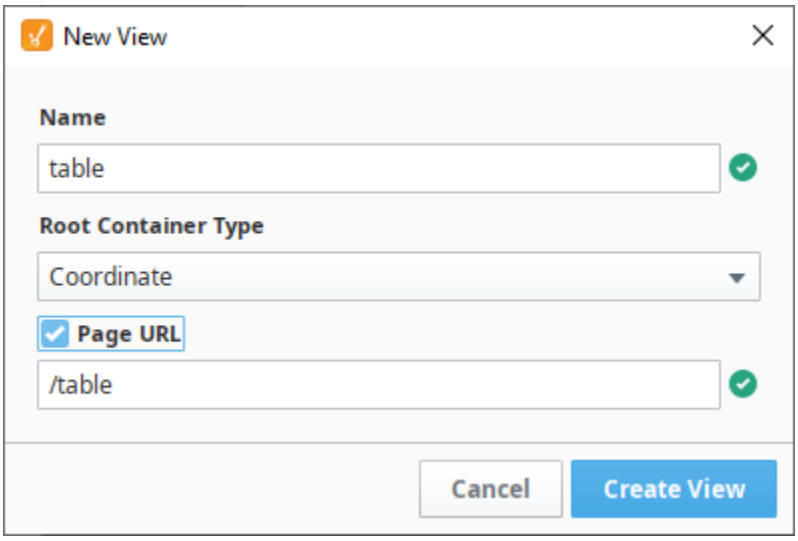
On this page

...

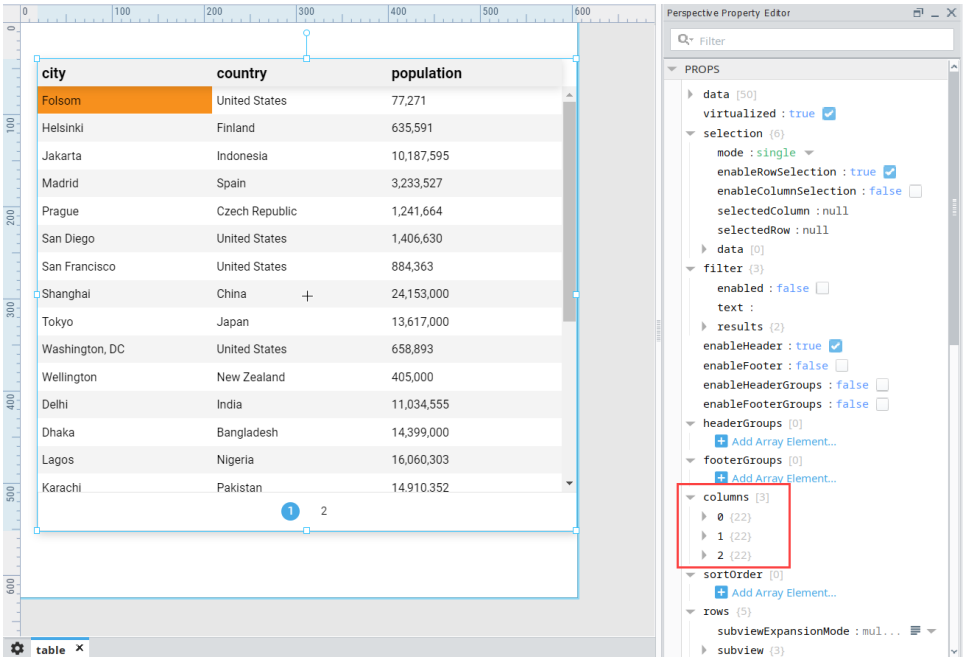
- Replacing a Value in a Cell with a Progress Bar
- Embedding a View in a Table Cell

Replacing a Value in a Cell with a Progress Bar

1. In the Project Browser, right click on Views to create a view. In this example, the view will be named **table**. Set it to have a **Coordinate** Root Container Type.



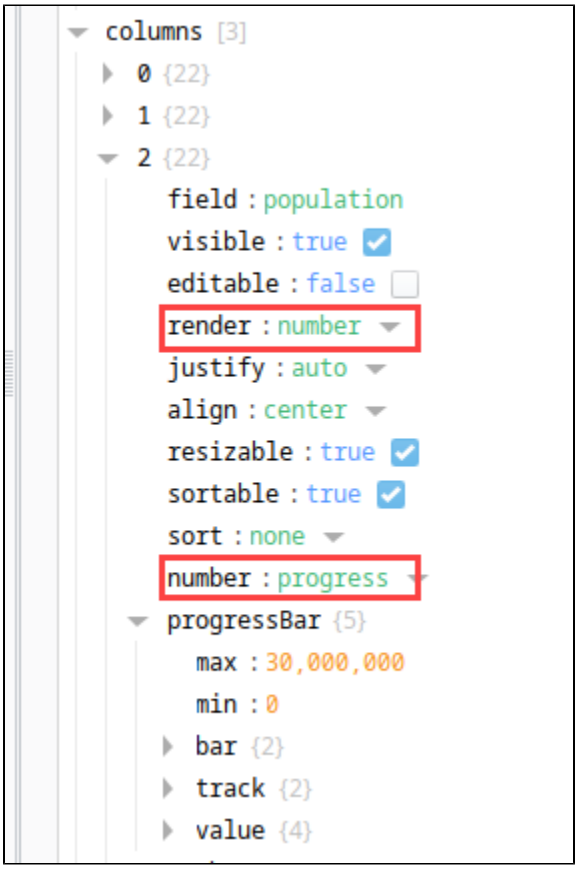
2. Drag a Perspective Table component onto your table view. We'll use the default population information that the component is initially configured with.
3. Add three array elements to the columns property of your table as shown in the Property Editor.



4. There is a **field** property inside each of the three columns array elements. Set the **"field"** property values to match each of the column names in your table. This is how we associate a column in the table (which is really just a key in the underlying data on the table) with one of these custom column configurations.

    a. Enter a **field** property value of **"city"** for the **"0"** column's element.
    b. Enter a **field** property value of **"country"** for the **"1"** column's element.
    c. Enter a **field** property value of **"population"** for the **"2"** column's element.

5. Let's display a progress bar on the table to show the population value. To do this, go to the columns array element for "population" and set its **render** property to **"number"** and set its **number** property to **"progress"** as shown in the image below.

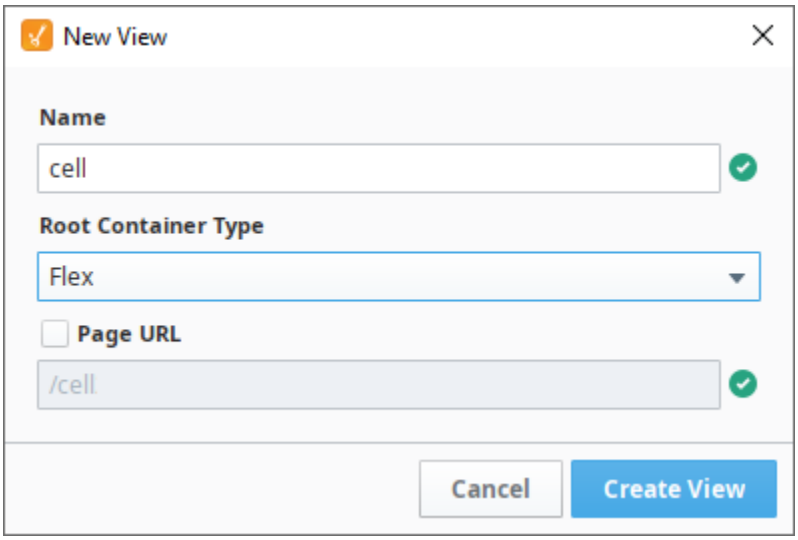


6. Set the **progressBar.max** value to 30,000,000 to account for cities with a large population.
7. The table will look like this after the above configurations are applied.

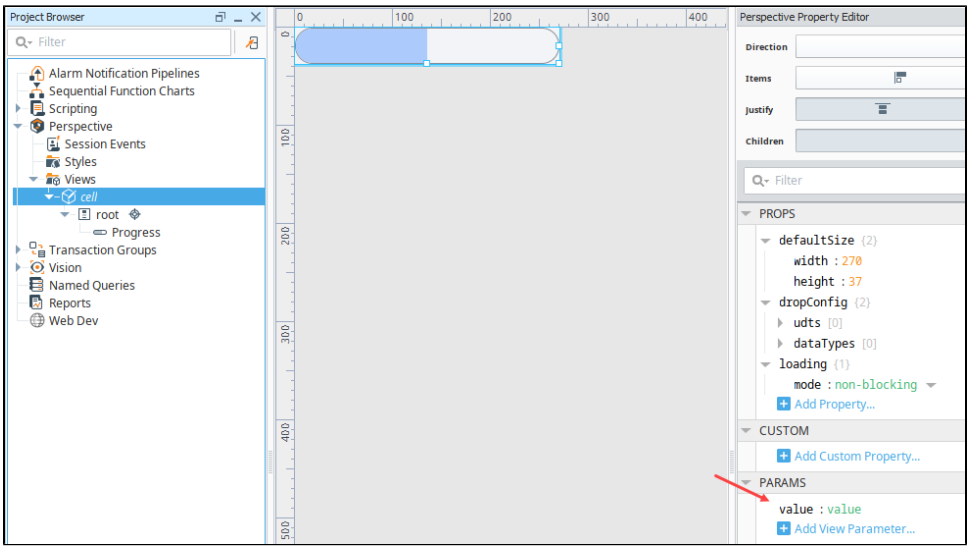
Embedding a View in a Table Cell

In a Perspective Table, it is possible to embed a view inside a table cell. In this example, instead of using the table's built-in progress bar, we'll embed a view that contains a custom progress bar, using the Progress component.

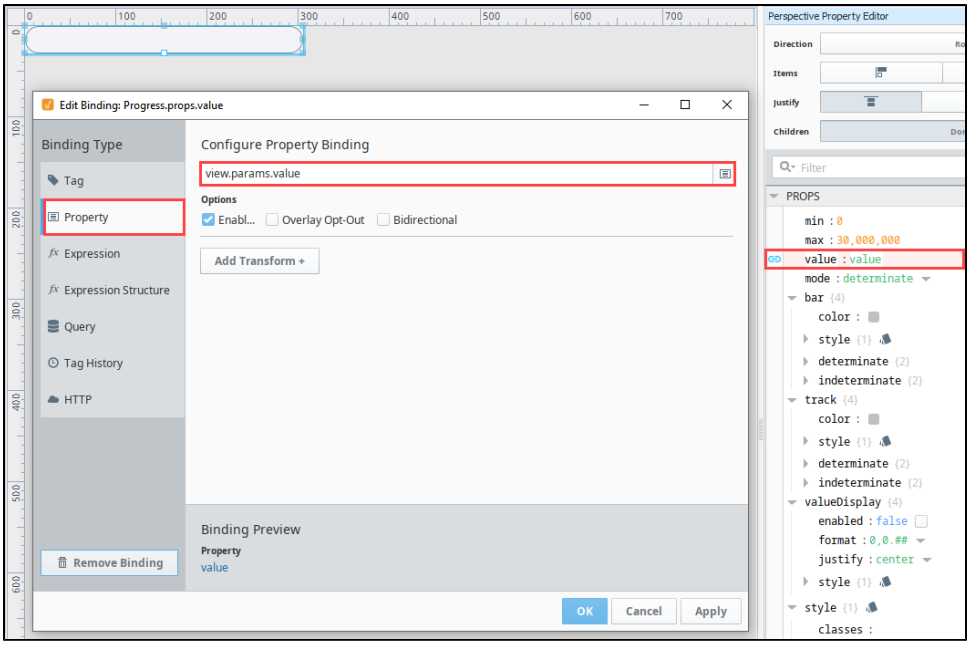
1. Let's create the cell view first. In the Project Browser, right click on Views to create a view. In this example, the view will be named **cell**. Set it as a **Flex** layout.



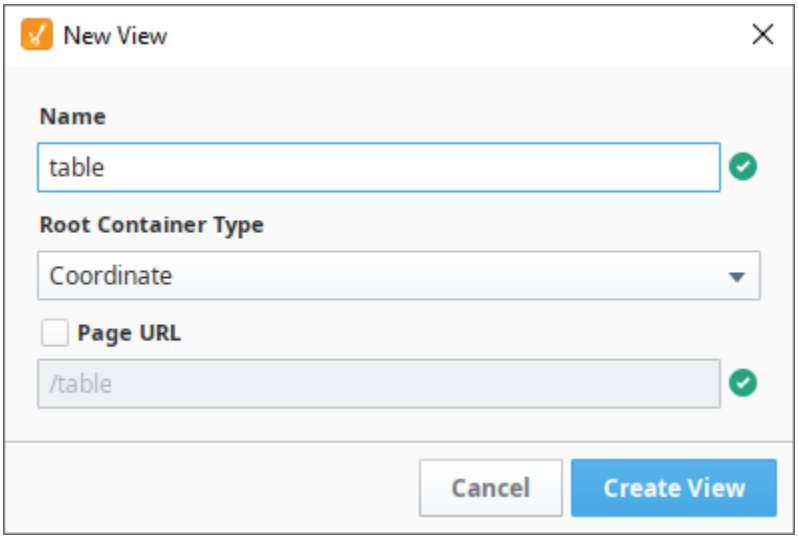
2. Drag and drop a Progress Bar component onto the cell view.
3. Select the Progress Bar component and set the **position.grow** property to **"1"** so that the bar takes up as much space in the container as possible.
4. In the Project Browser, select **"cell"** view, and resize the view so it is closer in size to the cell in the table on our **"table"** view.
5. Create a view parameter named **"value"** by clicking on the **"Add View Parameter"** option while your view is selected in the Project Browser as shown below.



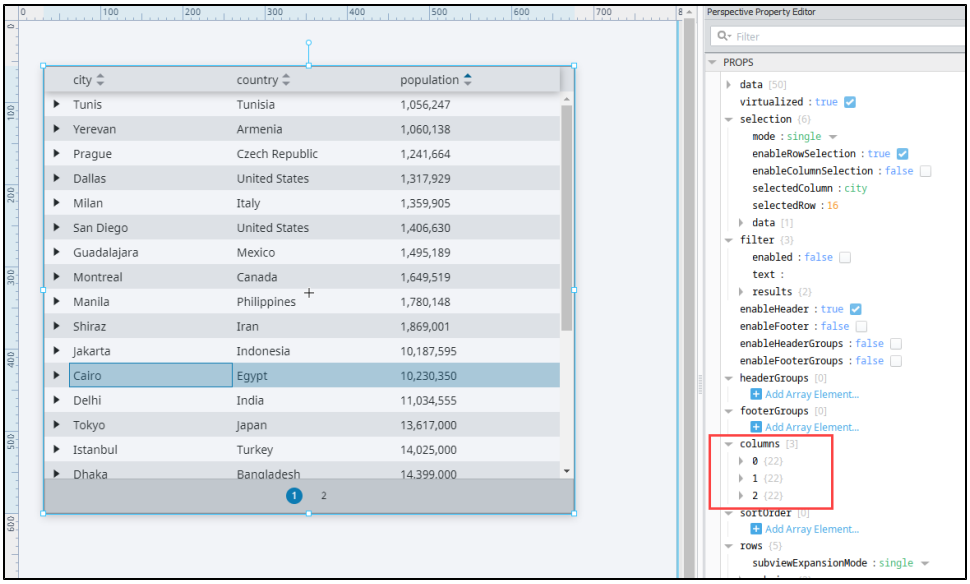
6. On the Progress Bar component, set its **max** property value to 30,000,000 to account for large values.
7. Bind the Progress Bar's **value** property to the view parameter created in Step 3 as shown below.



8. Now let's create our table view. Right click on Views to create a view. In this example, name the view **table**. Set it to have a **Coordin ate** Root Container Type.

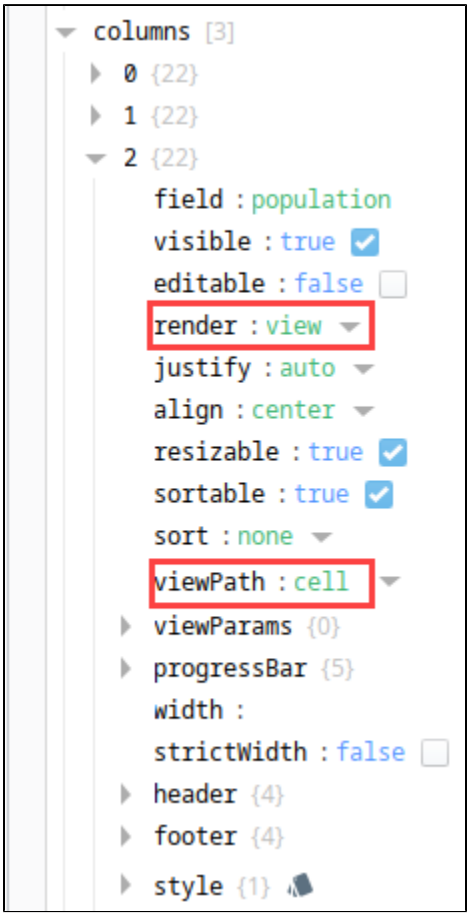


9. Drag a Perspective Table onto the table view. The default population information that comes on the factory configured table component will be used.
10. Add three array elements to the columns property of the table like as shown in Property Editor as shown in the image below.



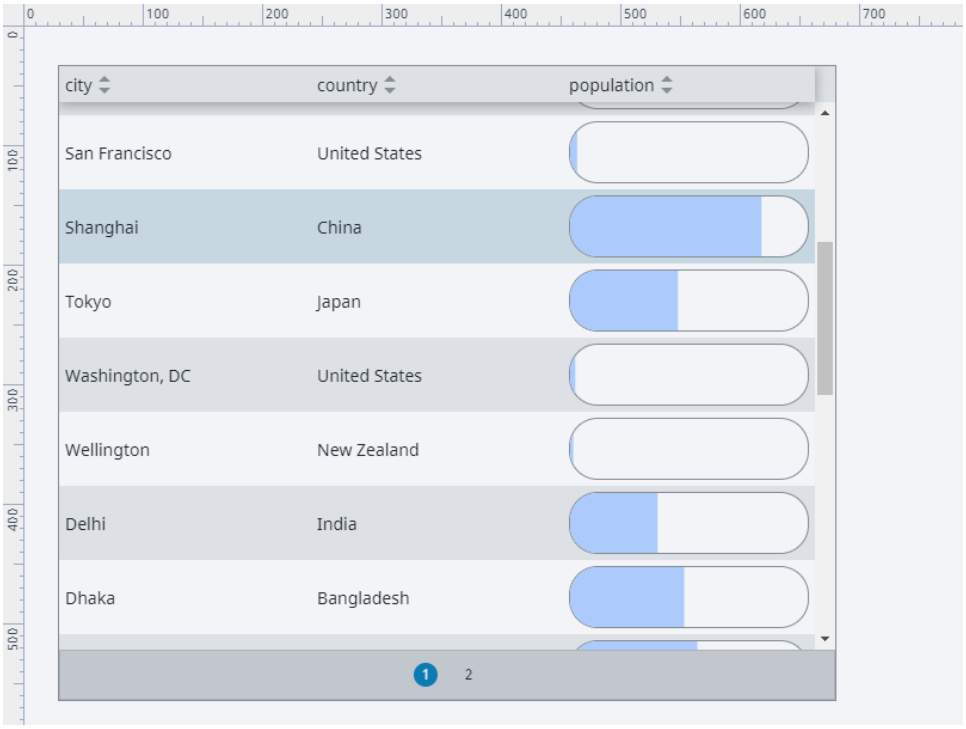
11. There is a **field** property inside each of the three column array elements. Set the field values to match each of the column names in your table.
    a. Enter a **field** property value of **"city"** for the **"0"** column's element.
    b. Enter a **field** property value of **"country"** for the **"1"** column's element.
    c. Enter a **field** property value of **"population"** for the **"2"** column's element.

12. The **cell view** is going to be embedded into the **Population cell** values. Go to the column array element with the **"field"** value of **"population"** and set its **"render"** value to **"view"** and its **"viewPath"** to **"cell"** as shown below.



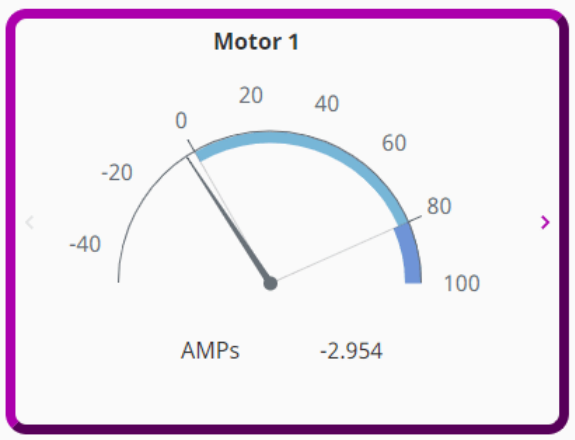
13. After the population column is pointed to the cell view, the population number from the table cell will be passed to the cell view. Since the cell view's Progress Bar has its value property bound to the cell view's input parameter, the population value will then be displayed on the table by the Progress Bar in the cell view. If you wanted to resize the progress bar, simply change the **"height"** and

"width" properties under the **defaultSize** property on the **"cell"** view.



Carousel Component Example

Perspective offers strategies for dragging (or swiping) left and right as a way of navigating between views. The **Carousel** component is specifically for this strategy, which is perfect when working with several instances of the same view (including a dynamic number of them). Maybe you're looking for a way of collapsing a lot of content onto a small screen, and need a way of scrolling through it.



On this page

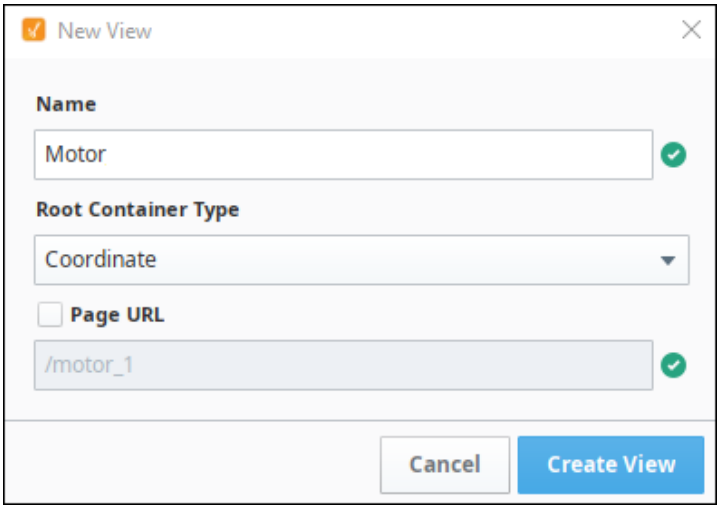
...

- Initial Project Setup
- Set Up the Carousel View

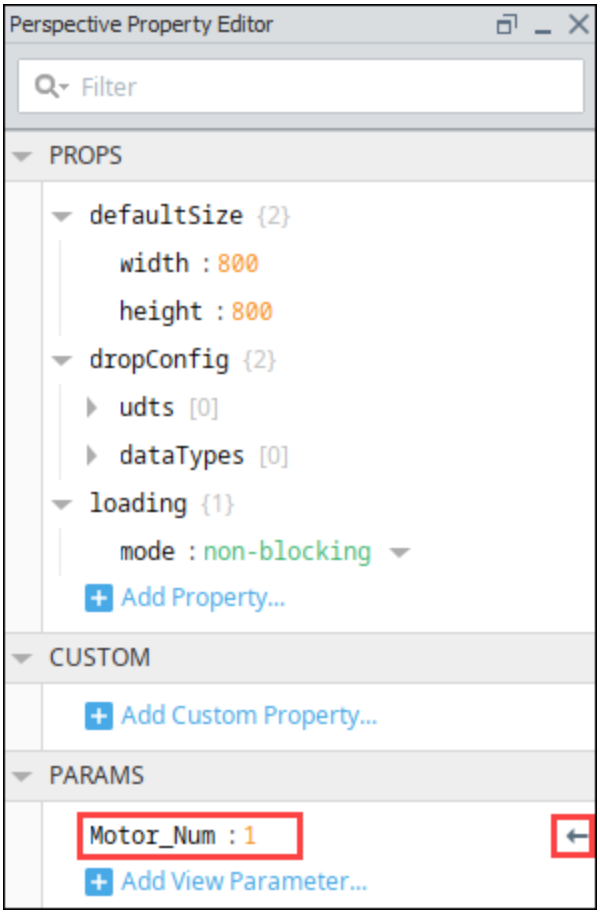
Initial Project Setup

Configuring side scrolling through a carousel is pretty straightforward. First we'll create three views for the carousel to scroll through.

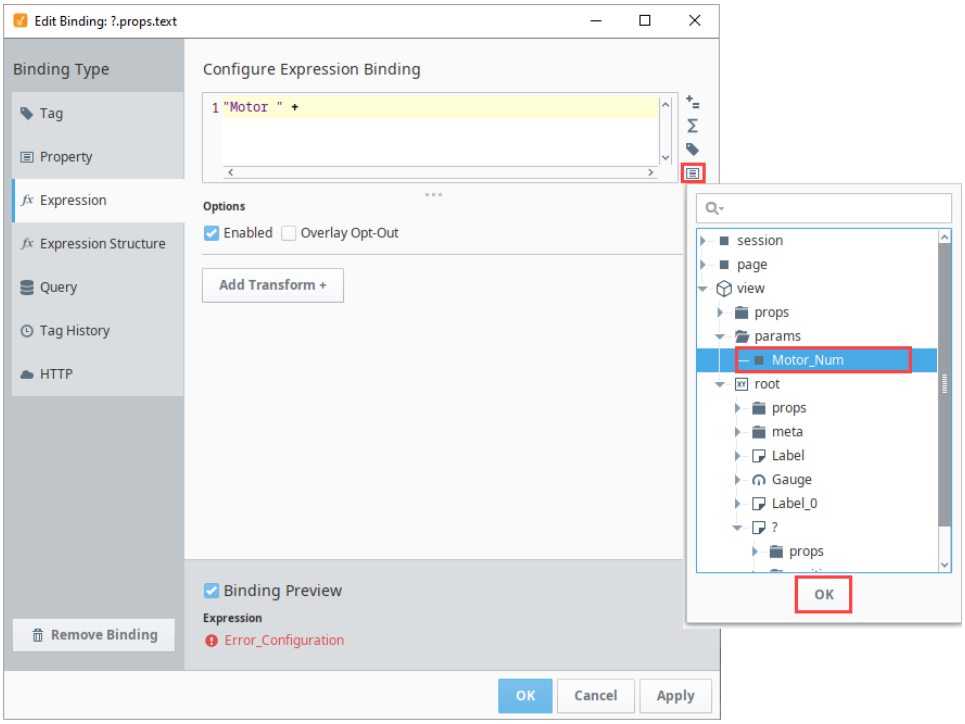
1. In the Project Browser, right click on Views and select the **NewFolder** option. We named our folder "Carousel Example".
2. Right click on the Carousel Example folder and select the **NewView** option.
   Name: **Motor**
   Layout: **Coordinate**
   Page URL: unchecked



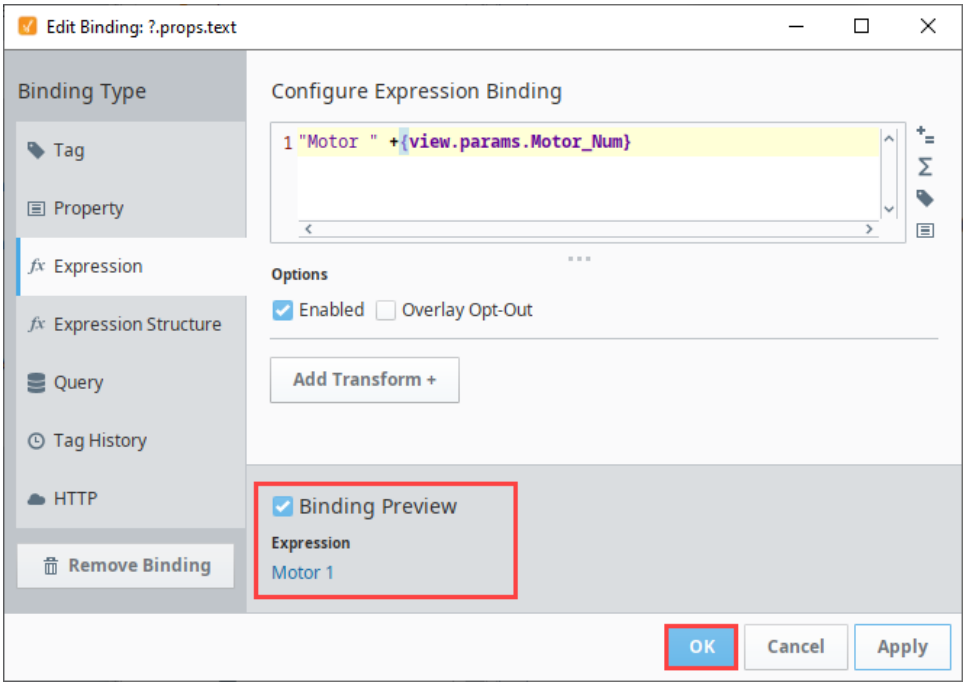
3. Click **Create View**.
4. Next we'll create a view parameter that we can use for the motor number. In the Property Editor under PARAMS, click on the **Add View Parameter** icon.
   a. Select **value**.
   b. Change "key" to "Motor_Num".
   c. Change "value" to "1".
   d. Make sure the arrow icon is facing to the left, as an input parameter.



5. Next we'll make a title that will change depending on the Motor that's being displayed. Drag a Label component onto the view.
6. Bind the text property to the view parameter as follows:
    a. Click the **binding** icon next to the text property.
    b. Select **Expression** as the binding type.
    c. Enter the following: "Motor " +
    d. Click the **Property** icon then scroll down and select the Motor_Num view parameter.

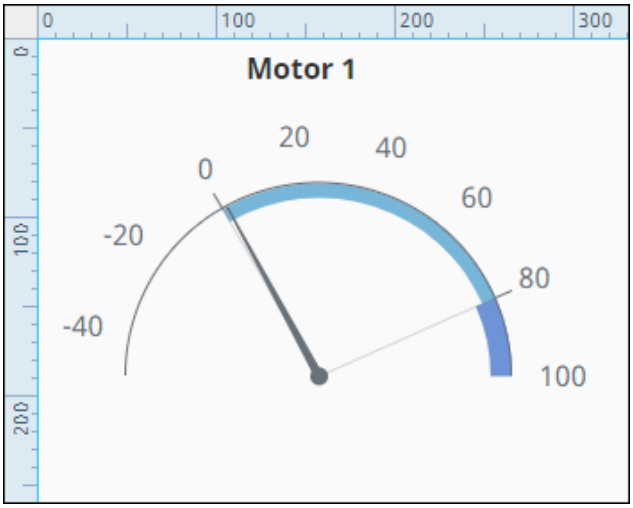


e. Click **OK**. You'll now see the binding preview shows "Motor 1."

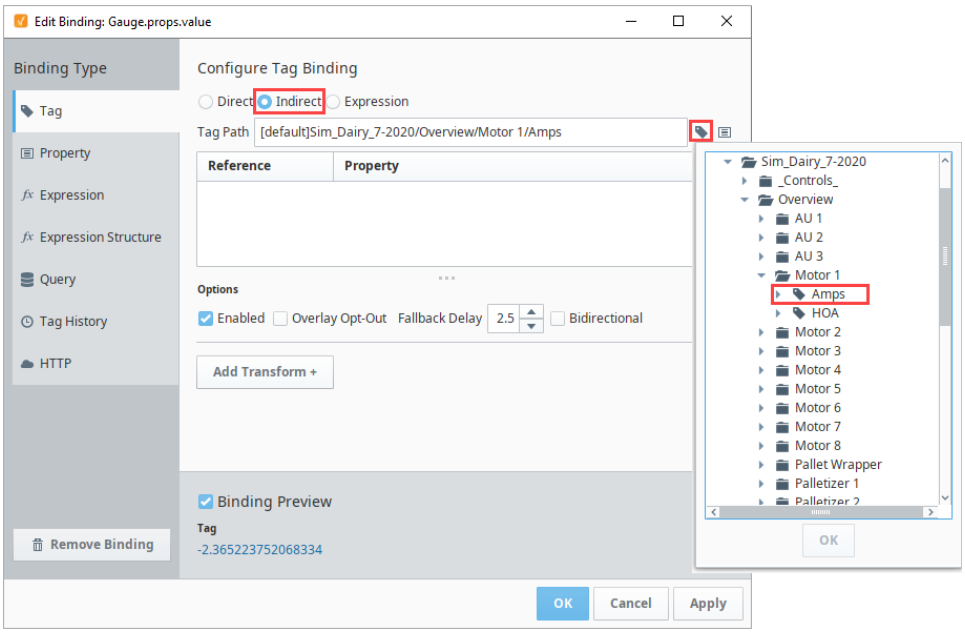


f. Click **OK** to save the binding.

7. Next we'll setup a Gauge component to display motor Amps. Drag a Gauge component onto the view.
8. Align the Motor label so that it is approximately centered abouve the Gauge component.
9. In the Property Editor, expand the outerAxis properties. Change the properties as follows:
   minValue: **-50**
   maxValue: **100**

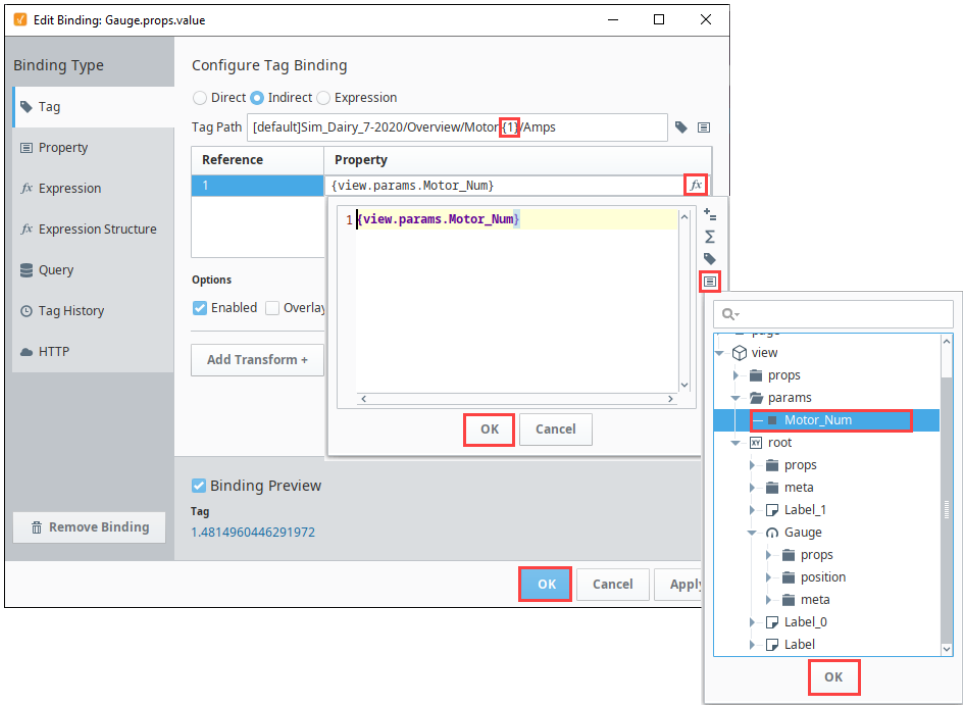


10. Now we'll set up an indirect Tag binding using one of the motor Tags in the Dairy simulator. (For more information, see Programmable Device Simulator.) Select the Gauge component.
    a. In the Property Editor, click the **binding** icon next to the value property. On the Configure Tag Binding page set the following:
    b. Choose **Tag** as the binding type.
    c. Select the **Indirect** option.
    d. Next to the Tag Path field, click on the **Tag** icon and navigate to the the Motor 1/Amps Tag in the Dairy simulator program.
    e. Click **OK**.

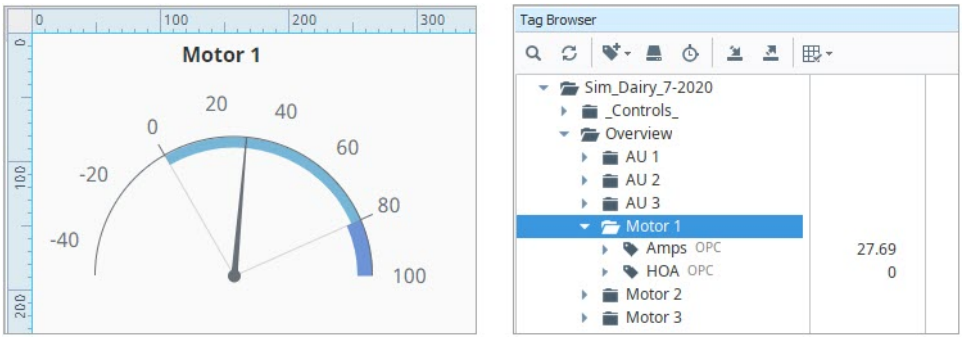


    f. In the Tag Path field, replace the 1 with {1}.
    g. In the References list, under Property, click on the **Functions** icon.
    h. Click the **Properties** icon.
    i. Scroll to the view parameters and select Motor_Num. Click **OK**.
    j. Click **OK** again. You'll see the binding in the preview area.

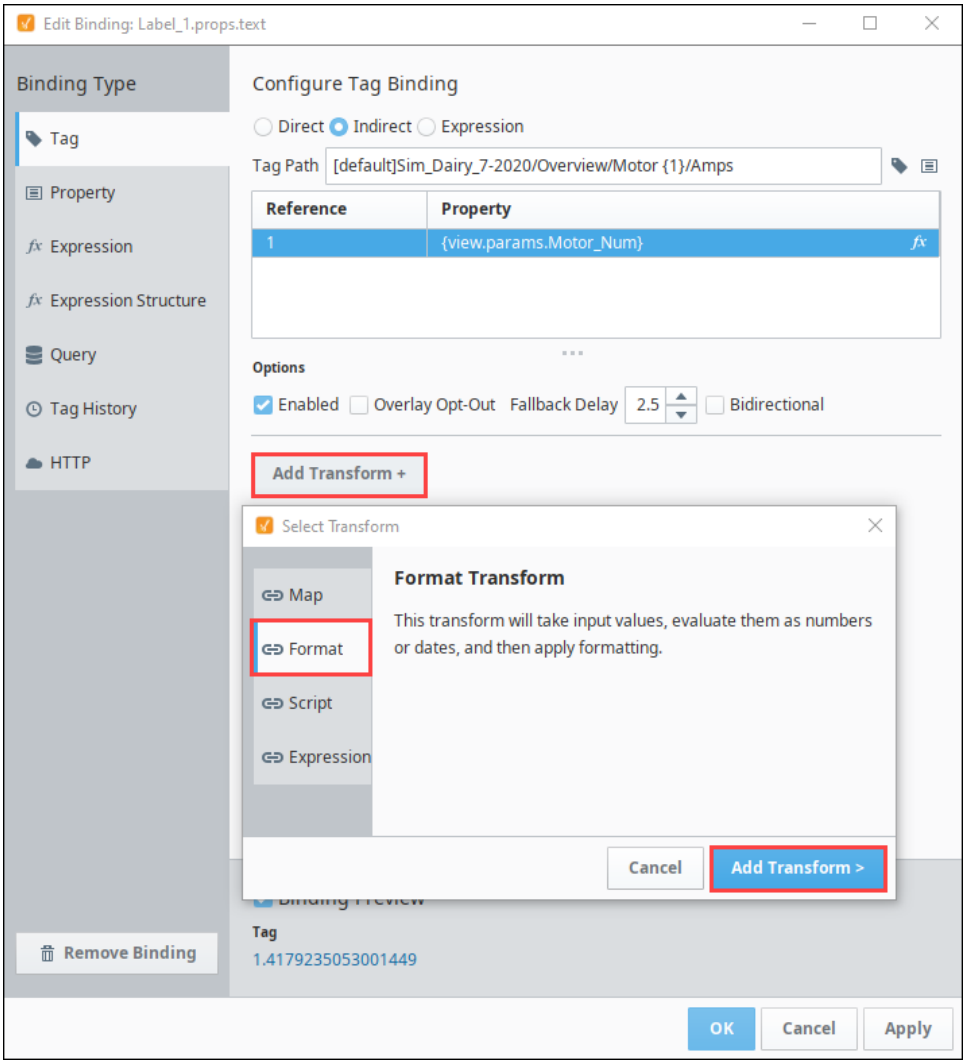
k. Click **OK** to save the binding.



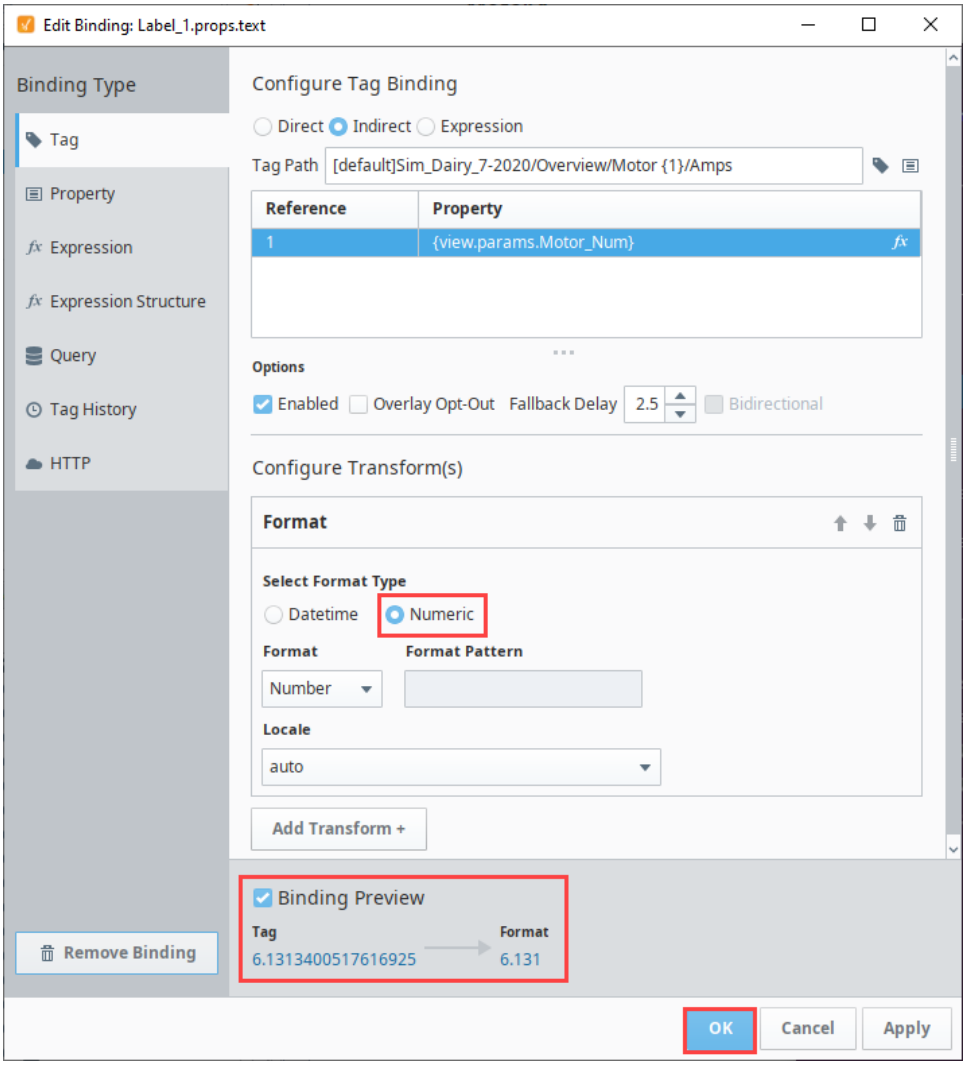
11. The Gauge now displays the value of the AMPS Tag.



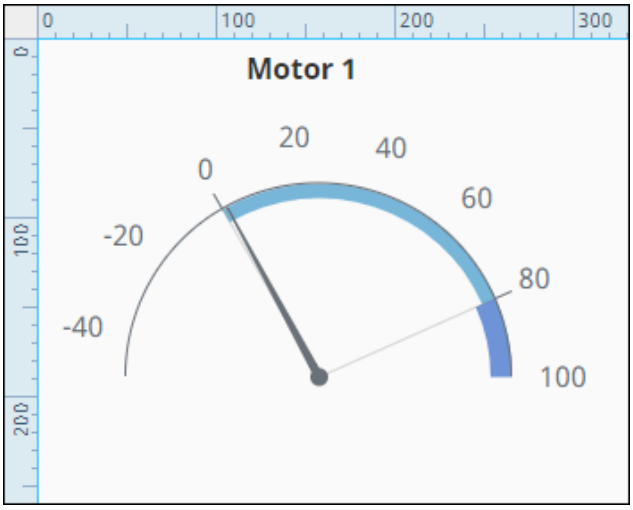
12. Drag a Label component onto your view. Place it under the Gauge component and change the text property to "**AMPS**".
13. Drag another Label component onto your view and place it next to the AMPS label. We'll set up a similar indirect Tag binding on this label.
    a. Select this Label and click the **binding** icon next to the value property.
    b. Select **Tag** as the binding type and click the **Indirect** radio button.
    c. Next to the Tag Path field, click on the **Tag** icon and navigate to the the Motor 1/Amps Tag in the Dairy simulator program.
    d. Click **OK**.
    e. In the Tag Path field, replace the 1 with {1}.
    f. In the References list, under Property, click on the **Functions** icon.
    g. Click the **Properties** icon.
    h. Scroll to the view parameters and select Motor_Num. Click **OK**.
    i. Click **Apply**. You'll see the binding in the preview area.
    j. For this Label component we don't want the full Tag value displayed. So we'll add a Transform to limit it to two decimal points. Click on **Add Transform.**
    k. Select **Format**, then click **Add Transform**.



l. Select **Numeric** as the Format type. The displayed value will now be shortened to just two decimal points. You can see the format in the Binding Preview.
m. Click **OK** to save the binding.



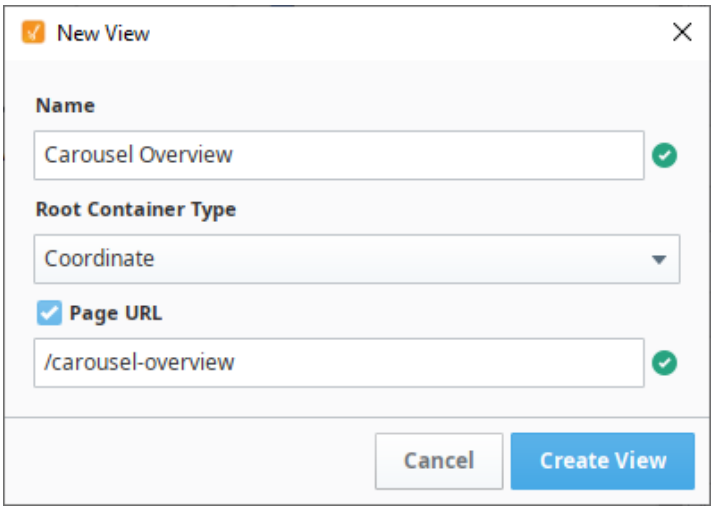
14. Your view should look something like this now:



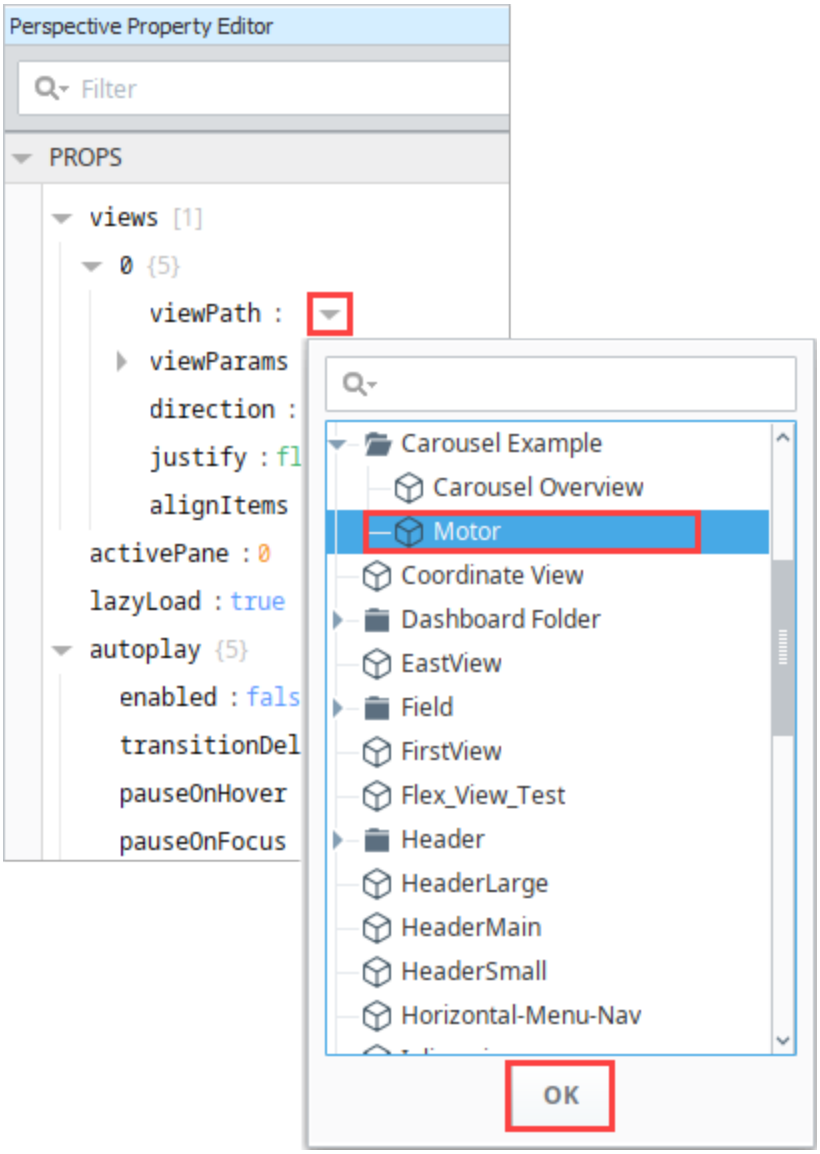
Set Up the Carousel View

Now we'll set up a view that holds the Carousel component.

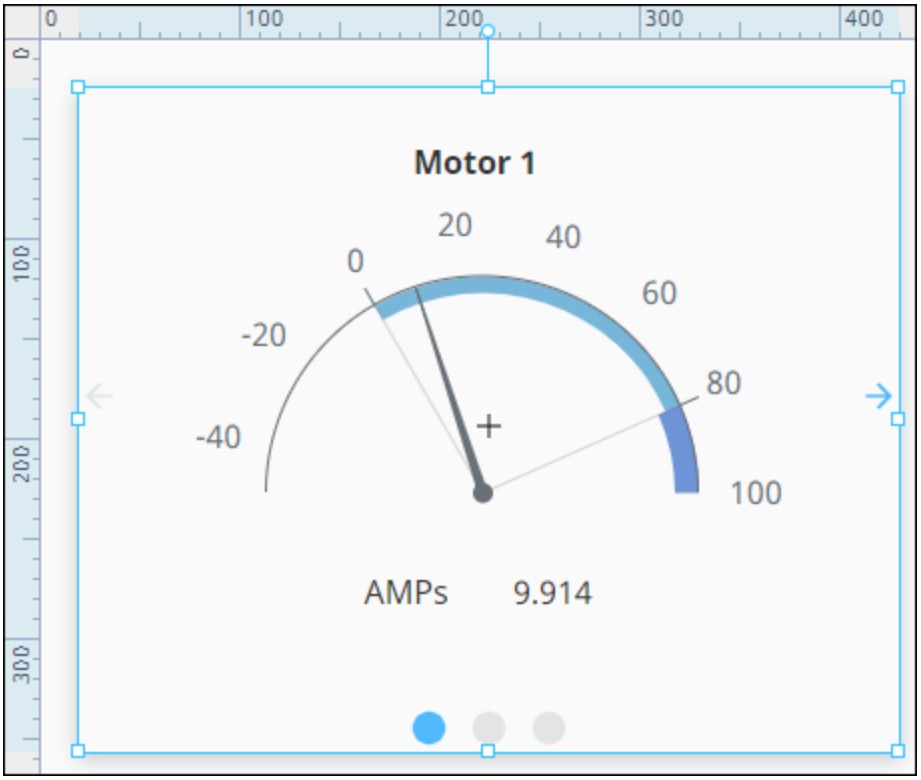
1. Right click on the Carousel Example folder and select the **NewView** option
2. Name the View Carousel Overview. Check the Page URL option.
3. Click **Create View**.



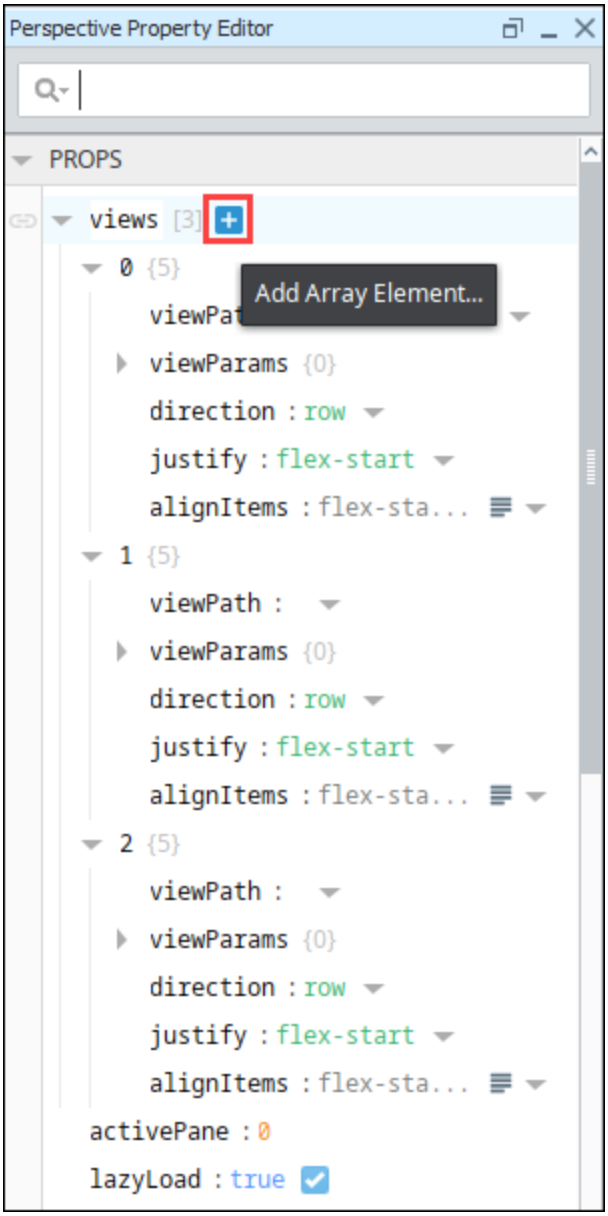
4. Drag a Carousel component onto the view.
5. In the Property Editor, expand the props.views.0.viewPath property.
6. Click the Expand icon in the viewPath property and select the **Motor** view.



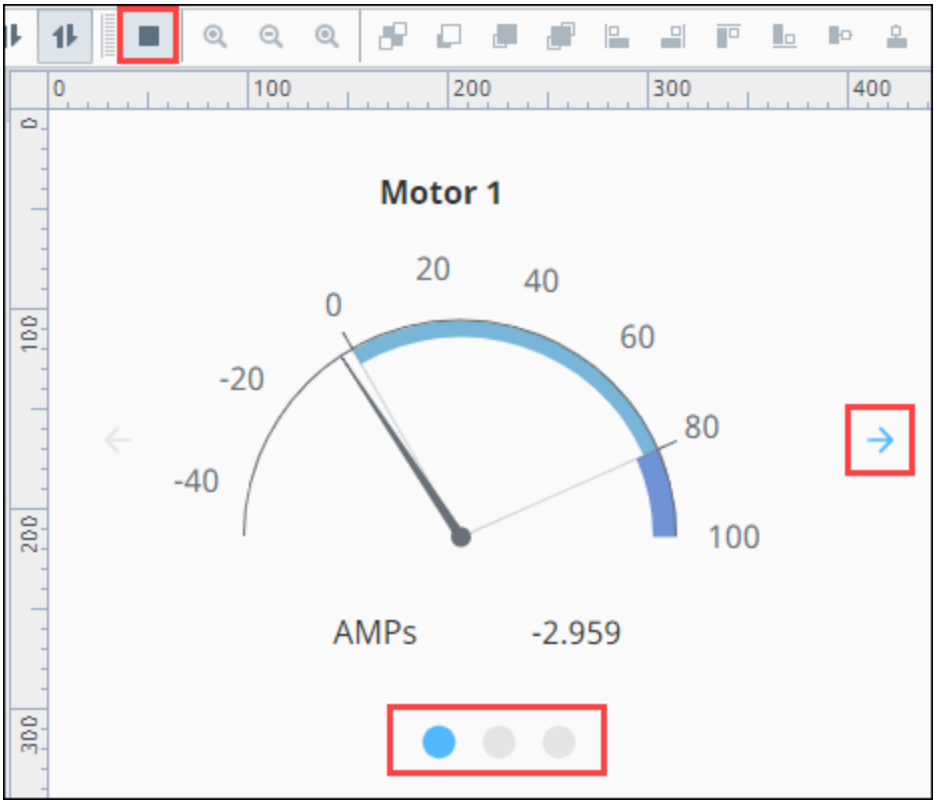
7. Under viewParams, click the **Add** icon then choose value.
8. Replace "key" with "Motor_Num" and replace "value" with "1".
9. Click **OK**. The Motor view now appears in the Carousel component. You may need to expand your Carousel component slightly to fit the Motor view in without scrollbars.



10. Next we'll add two more views. Click on the **Add** icon two times.



11. Under the props.views.1.viewPath property, choose the **Motor** view.
    a. Under viewParams, click the **Add** icon then choose value.
    b. Replace "key" with "Motor_Num" and replace "value" with "2". This will point this instance to the Motor 2/AMPS Tag.
    c. Click **OK**.
12. Under the props.views.2.viewPath property, choose the **Motor** view.
    a. Under viewParams, click the **Add** icon then choose value.
    b. Replace "key" with "Motor_Num" and replace "value" with "3". This will point this instance to the Motor 3/AMPS Tag.
    c. Click **OK**.
13. Save your project.
14. Put the Designer into Preview mode. Click the left right arrows or the dots to scroll between the three Motor views.



15. We decided to change a few properties on the Carousel to update the appearance. Here are the settings we used:

Property	Value
props.appearance.dots.enabled	false
props.appearance.arrows.next.iconPath	material/navigate_next
props.appearance.arrows.next.fillColor	#AC00AC
props.appearance.arrows.previous.iconPath	material/navigate_before
props.appearance.arrows.previous.fillColor	#AC00AC
props.style.borderStyle	outset
props.style.borderColor	#AC00AC
props.style.borderWidth	7
props.style.borderTopLeftRadius	15
props.style.borderTopRightRadius	15
props.style.borderBottomLeftRadius	15
props.style.borderBottomRightRadius	15

16. Put the Designer into Preview mode. Click the next or previous arrow to scroll between the three Motor views.