Value			
Running Sum	RunningSum	A running sum or running total of the driving pen.	
Sum	Sum	The sum of two different driving pens.	Secondary Pen - The second driving pen.
Difference	Difference	The difference of two separate driving pens.	Secondary Pen - The second driving pen.
Linear Regression	LinearRegression	Will create a linear regression line for the driving pen.	

Configuring Calculated Pens

In this example, we will configure the following calculated pens on our Easy Chart: Constant, Moving Average, Upper Control Limit (UCL), and Lower Control Limit (LCL).

1. Drag an Easy Chart component onto your Designer window.
2. Drag a Tag onto the chart. In this example, we are using a **'Realistic1'** tag from Programmable Device Simulator.
3. Right click on the Easy Chart component and choose **Customizers >** Easy Chart Customizer.
4. The Pens tab will open and you'll notice a Calculated Pens Table at the bottom of the screen. Click the **Add** icon.

Note: Calculated pens are just like other pens, so you can specify the style, color, axis and subplot in the Edit Pen window.

5. This will open the Edit Pen window. Set the calculated pen options as follows:
   - **Name:** High SP
   - **Driving Pen:** Realistic1 (Since we only have one tag on our Easy Chart, the Realistic1 tag is the default).
   - **Function field:** Constant
   - **Constant Value:** 25
6. Click **OK**.

7. Click **OK** again to save your pen options. Now, view your **High SP** pen on your Easy Chart. The High SP value of 25 is represented on your Easy Chart by a blue horizontal line.

8. Next, let's add a second calculated pen. Double click on the Easy Chart to open the Style Customizer. On the **Calculated Pens Table** at the bottom of the screen. Click the **Add** icon.
9. On the Edit Pen window, set the calculated pen options as follows:
   - **Name**: MovingAvg
   - **Function field:** MovingAvg
   - **Driving Pen:** Realistic1
   - **Window Size:** .2

> **Note:** If you have more Tags that you dragged on to your Easy Chart from the Tag Browser, you'll have more pens to choose from in the Driving Pen dropdown list.

10. Click **OK**. The chart now displays a running average.

11. Lets add two more calculated pens: one for Upper Control Limit (**UCL**) and another for Lower Control Limit (**LCL**), and set the Driving Property to **'Realistic1.'** Select a pen color to change the default color.

12. Once you add all your calculated pens, you'll see all your pens in the Calculated Pens section at bottom of the window. Press **OK.**

13. Now, all your calculations are displayed on your Easy Chart. The blue pen is the Constant, the pink pen is the Moving Average, and UCL and LCL are brown and green respectively.

Hide Driving Pens

Once you have your Calculated Pens created, you'll notice that they disappear if you disable the pen driving them. If you want to remove the Driving Pen but leave the Calculated Pens, set the **Hidden** property of the Driving Pen to **'true'**.

Related Topics ...

- Easy Chart Customizer
- Using the Tag Browse Tree for Charting

Using the Tag Browse Tree for Charting

In the Designer, you can use a special built-in component called the Tag Browse Tree to create ad hoc charts where you can pick and choose which pens you want to put on an Easy Chart. This same functionality is also available for an operator in the Runtime.

1. In Designer, drag a **Tag Browse Tree** component and **Easy Chart** component from the component palette to your workspace.
2. Put the Designer in **Preview Mode** ▶.
3. Expand the Tag folders to see the Tags in your system. By default, the Tag Browser Tree component shows you all Tags even those Tags that are not logged in Historian.

4. Put the Designer back in **Design mode** ■.
5. Select the Tag Browse Tree component.
6. In the **Property Editor**, set the following properties:
   a. Set **Include Realtime Tags** to false.
   b. Set **Include Historical Tags** to true.
   c. Set the **Selection Mode** to Multiple - Discontiguous.

7. **Save** your project.
8. Launch a Vision Project or put the Designer in **Preview Mode** ▶.
9. Drag some tags over onto the chart. We chose Sine0, Sine2, Sine4, and Sine6. Note that because we previously set selection mode to Multiple - Discontiguous, we can choose several Tags using **Shift-Click**. The Tags don't have to be contiguous within the Tag browser.

10. You can click the **Delete** icon to the right of any of the pens to remove a Tag from the chart. You can also remove all pens and go back to an empty chart, and pick and choose which of the Tags you want to drag on to the chart.

Notes: Things to keep in mind when working with Ad Hoc Charting

- When working in the Designer, whatever pens you have on your Easy Chart when you saved your project, the same pens will also be displayed on the chart when the client is opened.
- You may have multiple axes set up for your Easy Chart, but when dragging Tags from the Tag Browse Tree component to an Easy Chart, there is no way for the user to set which axis to use. Because of this limitation, any Tag that is added in this way will attempt to match their Engineering Units property to an axis on the chart. If no match is found, the default axis will be used.

Related Topics ...

- Indirect Easy Chart

Indirect Easy Chart

Configuring an Indirect Easy Chart

> **This section assumes that Tags and Tag History have been configured**
>
> To learn more, go to the Tag and Configuring Tag History pages.
> The examples below use OPC Tags, but Memory Tags can be used instead.

On this page ...

- Configuring an Indirect Easy Chart

It is possible for the Easy Chart component to be indirect and point to a set of Historical Tags based on any parameter using the Cell Update Binding type. In this example, suppose you have a small tank farm consisting of Tank 101 through Tank 109. Every tank is identical and uses the same Tags. Each tank has a **Level** Tag and **Temperature** Tag which are set to log to the Historian.

Using an Indirect Easy Chart is a good way to see the history of these Tags for each tank in the tank farm. An convenient way to set this up is using a Dropdown List component where an operator can select various tanks to see a Tank's Tag history. This example will show you how to configure a cell update binding on an Easy Chart to be indirect and point to a set of Historical Tags.

1. In the Designer, drag a **Tag Browser Tree** and an **Easy Chart** component into your workspace.
2. Put the Designer in **Preview Mode** ▶.
3. Navigate to the Tank Tags in the Tag Browser Tree. Under Tank 101, drag the **'Level'** and **'Temperature'** Tags from the Tag Browse tree onto the Easy Chart.

4. Put the Designer back in **Design Mode** ■.

5. Drag a **Dropdown** component to your workspace. Using a Dropdown list will make it convenient for the operator to select a Tank and see the history of the **Level** and **Temperature** Tags for that tank.
6. With the Dropdown component selected, click on the **Dataset Viewer** icon to the right of the **Data** property in the Property Editor.
7. Click the **Add Row** icon for as many Tanks that you have in your Tank Farm. Then enter the **Value** and **Label** for each of the Tanks.
8. Click **OK**.

9. Put the Designer in **Preview Mode** ▶.
10. Click the **Dropdown** list to see the complete list of Tanks in your Tank Farm that you just entered.

11. Put the Designer back in **Design Mode** ■.
12. Right click on your Easy Chart and choose to **Customizers >** **Easy Chart Customizer**.

    You'll notice that the Tag Paths are pointing directly to Tank 101. The only difference between Tank 101 and all the Tanks in the Tank Farm is the Tank number (i.e., 101, 102, 103, etc.). You can manually point to a different Tag Path by replacing '101' in the Tag Path with a different Tank number such as '102,' but we'd rather have the Tag Paths change dynamically when the user selects a Tank. Press **OK**.

13. The Easy Chart component has a property called **Tag Pens** which stores all the configuration information that you have configured in the Easy Chart about your pens. With the Easy Chart selected, click on the **Dataset** icon in the Property Editor to view all the information for your **Tag Pens**.

14. The dataset is displayed. It contains one row for each Tag. The second column is the Tag Path that you will want to change dynamically. Press **OK** to close the Dataset Viewer.

15. Next we'll use a Cell Update Binding in order to change an individual cell of a dataset. With a Cell Update Binding, you can select one or more cells and dynamically bind them to a property or to a Tag that you have in your system. Click on the **Binding** icon next to **Tag Pens** property to open the Property Binding window.
16. Select the **Cell Update** binding type.
17. Select Tag Path cell for **Level**, and click the **Add** icon under the **Cell Bindings** table.

18. Select the first row you just added in the Cell Bindings Table, and click the **Insert Property Value** icon.
19. From the Property Window under the **Dropdown** folder, click the **Selected String Value**. This will grab the Tank number the operator selected from the dropdown list.

20. Click **OK**. Ignition fills in the Cell Binding value.
21. Repeat steps 17 through 20 for the Temperature Tag Path.
22. Next, expand the **Value** fields to make the Tag Path dynamic. Update the fields as follows.

Cell Binding to make Tag Path Dynamic
`[~]Tank/{Root Container.Dropdown.selectedStringValue}/Temperature` `[~]Tank/{Root Container.Dropdown.selectedStringValue}/Level`

23. Click **OK** to save the bindings.
24. Now you have an Indirect Easy Chart. To test it, put the Designer in **Preview Mode** ▶.
25. Select a Tank from the Dropdown List. The Level and Temperature values will change in the chart. Next select a different Tank from your Dropdown List to see how the history changes on the Easy Chart.

Related Topics ...

- Easy Chart Customizer

Charting - Right Click Menu

When viewed from the client (or the Designer's **Preview Mode** ▶) many chart components, such as the **Easy Chart** or **Bar Chart**, enable users to right-click and access a menu with additional features. Explanations of these features are listed below.

On this page ...

- Right Click Menu Functions

Right Click Menu Functions

Note that many of the behaviors listed below may be overridden by a property on the parent component. For example, the **Easy Chart** provides a way to define the plot background color with the Easy Chart Customizer, so the Background options on the right-click menu would not modify the behavior on the component when the user selects them.

Item Name	Description		
Mode	Specifies the functionality of a **left-click** on the chart. The following modes are available:		
	Zoom	**Zoom in**: Users can left-click and drag on the chart to zoom in or out. Dragging to the right creates a hollow rectangle. When the user releases the mouse button, the viewable space in the chart will zoom in to a rectangle. **Zoom out**: Left-click and dragging to the left will not create a rectangle, but releasing the mouse button will zoom back out.	
	Pan	Allows users to **Pan** the chart horizontally or vertically by left-click and dragging. The **Pan** mode may be used after using the Zoom mode: doing so allows panning within the zoomed in view-space.	
	Mark	While in **Mark** mode, a left click will target the closest point on the nearest pen, and return the value and timestamp of that point.	
	X-Trace	Similar to **Mark**, X-Trace will instead show the values of all pens at the selected point in time.	

Background	Allows the user to change the background color. Available options include **Black** and **White**.
Save as...	Provides a Save window that allows the user to save a PNG of the chart locally.
Print...	Provides a Print dialog that allows the user to print an image of the chart. This will send the chart to a physical printer. Note: a printer must be installed on the client computer.
Zoom In	Allows the user to Zoom in on the chart. There are three possible options: • **All axes**: Zooms in on both the Range and Domain. • **Domain axis**: Zooms in on just the Domain, or X- axis. Reduces the range of viewable values on the domain (typically a datetime), while leaving the Range axis untouched. • **Range axis**: Zooms in on just the Range, or Y-axis. Reduces the range of viewable values (typically numeric), while leaving the Domain axis unmodified. This is helpful when a trend shows very little change over a period of time, as this allows the user to view the trend with increased granularity.
Zoom Out	Allows the user to Zoom out on the chart. There are three possible options: • **All axes**: Zooms out on both the Range and Domain. • **Domain axis**: Zooms out on just the Domain, or X- axis. • **Range axis**: Zooms out on just the Range, or Y-axis
Auto Range	Automatically updates the range, or **viewable space**, on the chart to incorporate all data points. As mentioned above, many components have properties/customizations that restrict the capability of these (such as the date range on on the Easy Chart). • **All axes**: Adjusts the viewable space on both axes. • **Domain axis**: Adjusts the viewable space on just the Domain, or X-axis. • **Range axis**: Adjusts the viewable space on just the Range, or Y-axis.
Reset Axes	Adjusts the Axes back to their initial state.

Related Topics ...

- Easy Chart Customizer
- Easy Chart - Subplots
- Easy Chart

Easy Chart - Database Pens

Database Pens

Database Pens are driven by a SQL query, so they are ideal to use when trending Transaction Group data. However, they can query for data in any connected SQL database, so it is possible to show historical data recorded by other systems on the Easy Chart.

> **This section assumes that Tags and Tag History have been configured**
>
> To learn more, go to the Tag and Configuring Tag History pages.
> The examples below use OPC Tags from the Programmable Device Simulator driver, but Memory Tags can be used instead.

On this page ...

- Database Pens
- Configuring Database Pens

Configuring Database Pens

In this example, we'll configure a database pen driven by a SQL query to show historical data and display it on the Easy Chart.

1. Drag an Easy Chart component onto your Vision window.
2. Right click on the Easy Chart component and choose **Customizers > Easy Chart Customizer.**
3. In the Database Pens section, we'll create a database pen to trend data stored in our database. Click the **Add** icon to create the Database Pen.

4. The Edit Pen window will open. Here is where you enter your database pen data. In the very least, we'll need values for the following properties:

   We entered the following settings:

   a. **Name** - The name of the pen on the chart. Works the same as the name on any other pen.
   b. **Datasource** - The name of the database connection (as configured on the gateway) that contains the table we want to extract values from.
   c. **Table Name** - The name of the database table that contains the values that we want to represent on the chart.This dropdown will automatically populate with available table names once the Datasource property has a valid database selected.
   d. **Value Column** - The name of the column in the database table that contains the values we want to show, representing the value of the datapoint at a particular time. This column is ultimately responsible for the determining where each point lies on the chart's Y-axis. This dropdown will automatically populate with column names once a value for the Table Name property has been set.
   e. **Time Column** - The name of the column in the database table that contains a timestamp, representing the time of the datapoint. This column is ultimately responsible for determining where each point lies on the chart's X-axis. This dropdown will automatically populate with column names once a value for the Table Name property has been set.
   f. **Axis** - All pens require an axis. We can use the default here.

      Click **OK** to save your database pen settings.

5. In the Database Pens area, you will see your database pen that you created. Click **OK** to exit the customizer and view your Easy Chart.

6. The High Temp database pen is now trending data for the High Temp values we want to show at a selected time.

Related Topics ...

- Easy Chart Customizer

Using the Classic Chart

Chart

The Chart, sometimes referred to as the Classic Chart, is capable of rendering time series, XY, and Bar charts.

On this page ...

- Chart
  - Configuration
  - Datasets
  - Binding Techniques
  - Chart Type: XY vs Category
- Populating a Chart with Data
  - Example - Tag History
  - Customizing a Chart
  - Adding a Second Dataset to a Chart

Configuration

The basic idea behind configuring the classic chart is simple: add datasets, and fill them in with data in a format that the chart understands. You add datasets to the chart using the chart's customizer. You then use a bindings to populate each dataset. Commonly you'll use a SQL Query Binding. Since these datasets are just normal dynamic properties, you can also access them via scripting.

The Customizer also lets you add additional X and Y axes. There are various types of axes and each works slightly differently. Lastly, you can configure additional properties for each dataset, such as which axis it maps to, its visual style, subplot, etc.

Datasets

Each dataset should define one or more "series" (a.k.a "pens"). The expected anatomy of a dataset for the Classic Chart depends on several settings: namely the **Extract Order** property on the chart, as well as the type of renderer specified in the Chart Customizer's **Dataset Properties** tab.

Binding Techniques

The Classic Chart can be used to make almost any kind of chart, with some effort. Historical, realtime, dynamic pen selection, etc., is all possible. Your job is just to fill the datasets with the pertinent data, and the chart will display it. The most common idea is to make the chart dynamic by varying the date range that the dataset's SQL Query bindings run. This is easy to do by adding a Date Range component and using Indirect Bindings.

Chart Type: XY vs Category

The Classic Chart is typically in XY Plot mode. This means that the X-axis is either date or numeric, and the Y-axes are numeric. If your X-axis is categorical (names, not numbers), you can switch the Chart Type property to Category Chart in the Property Editor. Don't be surprised when you get a few errors - you'll need to go and switch your X-axis to be a Category Axis, and fill your dataset in with valid category data, that is, String-based X-values. This is most often used with the Bar Renderer (see the Chart Customizer).

Populating a Chart with Data

Populating the Classic Chart involves using Custom Properties to collect data. New Custom Properties are added to the component via the Chart Customizer. Multiple dataset properties may be configured on the component, and each additional dataset will show as a new subplot. Populating the Chart with historical data involves populating one of these Custom Properties.

The default configuration of the component expects a dataset where the first column is a timestamp (the first column **always** acts as the domain for the chart), and sequential columns are pens that should be drawn on the chart. The default dataset contains a **t_stamp** column with values for the domain, and two columns (**Process Temp** and **Output Temp**) with values that will be drawn against the range.

t_stamp	Process Temp	Output Temp
10/6/19, 3:15:00...	53	45
10/6/19, 4:15:00...	48	48
10/6/19, 5:15:00...	48	53
10/6/19, 6:15:00...	50	50
10/6/19, 7:15:00...	46	46
10/6/19, 8:15:00...	49	44
10/6/19, 9:15:00...	52	46
10/6/19, 10:15:0...	57	48
10/6/19, 11:15:0...	59	46
10/6/19, 12:15:0...	56	46
10/6/19, 1:15:00...	53	43
10/6/19, 2:15:00...	56	39
10/6/19, 3:15:00...	61	36

Example - Tag History

The Classic Chart is initially configured in a manner that easily displays Tag History with a Tag History Binding.

1. Drag a Chart component onto a window.
2. In the Vision Property Browser, scroll down to the **Data** property. It should be located at the bottom of the Property Editor when sorted by section. Click the **Binding** icon.
3. Select **Tag History** from the Tag section of the Binding window.
4. Browse the **Available Historical Tags**. For this example, we chose some ramp Tags. Drag the Tags over to the **Selected Historical Tags** ta ble on the right of the window.
5. Change the **Date Range** to **Realtime** with a **Most Recent** set to 10 minutes.

6. Click the **OK** button. The Chart will now show Tag History from the Tags you selected.

Note that the timestamp has automatically been included as the first column, so the domain has been automatically configured for you. If you wish to change the orientation of the axes, use the **Chart Orientation** property in the Property Editor to swap the position of the Domain and Range axes.

Customizing a Chart

Once trends are present on the chart, additional customization can be achieved through the Chart Customizer. When you open the customizer, you'll notice five tabs at the top: Datasets, X-Axes, Y-Axes, Dataset Properties, and Plot Properties. Each tab has its own set of properties and defaults.

An overview of each section in the Chart Customizer is listed below:

- **Datasets**: Allows additional datasets to be added to the component. Each dataset will appear as a separate custom property on the component. Data from multiple charts can be drawn on the same chart.
- **X-Axes** and **Y-Axes**: Allows for the creation of new X and Y Axes on the chart. There are six types of axes to choose from when configuring a chart, each having its own list of properties: Number Axis, Date Axis, Category Axis, Logarithmic Axis, Elapsed Axis, and Symbols Axis. Most of the X and Y axes properties are used in the customizer, and some properties are specific to the axis type and have their own unique properties. See the Chart Customizer page for more information on types of axis, associated properties, and examples.
- **Dataset Properties**: Specify which axes should a be used with each dataset. Also allows you to specify which subplot each dataset should be shown on.
- **Plot Properties**: Configure the look of each plot.

This example walks you through adding a new dataset to a chart, and modifying the existing chart properties using the Chart Customizer. When you first drag a chart to your window, you'll notice that it will display some data, that's because it's using the default dataset provided. For this example, you can add a new dataset by either copying and pasting the one below or adding your own.

1. Let's add a new dataset (i.e., SnowPackData) that measures the Snow Pack Level for the month of February. Double click on the chart to open the Chart Customizer.
2. Go to the Datasets tab and click the **Add** icon to add a new dataset.
3. Enter the Dataset name and then click **OK**.

4. In the Vision Property Editor, click on the **Dataset Viewer** icon.

5. Highlight and copy the dataset below, then click the **Paste Dataset from Clipboard** icon.

Classic Chart - SnowPack Dataset

```
"#NAMES"
"Date","Current Snow Pack","Normal Snow Pack"
"#TYPES"
"date","F","F"
"#ROWS","4"
"2018-02-02 10:41:22","2.25","3.75"
"2018-02-10 10:42:19","3.15","4.5"
"2018-02-15 10:43:27","2.15","5.25"
"2018-02-25 10:44:02","3.75","6.35"
```

6. The data will appear in the Dataset Viewer. Click **OK** to save the dataset.

7. The Chart component is now displaying both the default dataset and **SnowPackData** dataset. Let's disable the default database. Open the Chart Customizer and go to the Dataset Properties tab.
8. Select the default dataset (i.e., Data) and uncheck the **Enabled?** box.

9. Click **OK**. You'll notice the chart is now only displaying the values for the new dataset (i.e., SnowPackData).
10. This step uses the default Number Axis type, but if you want to add a new Axis Type, go the **X and Y Axes tabs** and select a new Axis type**.**
11. Next, let's change a few visual properties on our chart. Go to the **Y-Axes tab** and select Default Y Axis, and change the **Axis Label** (i.e, Snow Pack Level (Feet)).

12. Lastly, let's make the plot lines thicker on the chart. Go to the Dataset Properties tab, and change the **Line Size** renderer property from 1.0 to **3. 0**.

13. Here is the chart with the new dataset and updated properties. As you can see, you can easily configure additional properties for a dataset, as well as choose from a host of visual style properties to design your charts using the Chart Customizer.

Adding a Second Dataset to a Chart

In this example, we'll add an additional dataset to the chart.

1. Double click on the Chart component to open the **Chart Customizer (**or right-click and select **Customizers > Chart Customizer)**.
2. Once the Chart Customizer is open, make sure the **Datasets** tab is selected, and click the **Add** icon to add a new dataset.
3. A new row will appear. Give the new dataset a name by typing into the cell under the **Name** column. We'll call it **NewData**.

4. Click the **OK** button.
5. Check the bottom of the Property Editor. The newly created dataset will appear and may now be populated with data.

6. Now that we have a new dataset, you can add data however you like and it will show up in a second subplot.

Other Vision Trending Charts

Along with the Easy Chart and Classic Chart, there are several other types of charts contained in the Vision Module that can also visualize a trend and track a change in condition, output, or process through a single data point or multiple data points over time. This page describes the Sparkline and Status Charts and how they can be used.

On this page ...

- Sparkline Chart
  - Usage
- Status Chart
  - Series Data
  - Color Mapping

Sparkline Chart

The Sparkline Chart is a minimalistic chart that displays a line-chart history for a single data point that fits well with High Performance HMI screens. It's a great way to show a lot of contextual information in a very small amount of space. It provides a fast way to display the recent history of a single data point so the viewer can quickly discern the most recent trend.

Note, the Sparkline Chart can not show multiple trends, however, multiple Sparkline Charts can be stacked. This chart is often laid over an image or level display to show a simple trend with the current value.

INDUCTIVE UNIVERSITY

Sparkline Chart

Watch the Video

Usage

The trend on the Sparkline Chart is determined by the **Data** property located in the Property Editor. Bind the Data property to either a Tag Historian realtime query, or to a database query. The dataset should contain two columns: the first being a **date**, and the second a **number/value**.

Any additional columns in the dataset are ignored. Thus, the Sparkline Chart will only show a single pen per instance of the component. The data should be sorted by date in ascending order.

The red dot on the Sparkline Chart represents the most recent value. The Sparkline Chart can display a band of color across the back of the chart which indicates the desired operating range of the datapoint. Setpoints can be displayed on the chart by setting a value for the **Desired High** and **Desired Low** properties. This allows an easy way to view when the value fell out of bounds. Note that both properties must have a value for the **Desired Range Color** to appear.

Status Chart

The Status Chart allows you to visualize the status of one or more discrete data points over a time range. The X-axis is always a time series axis, and the Y-axis is a category axis, with one entry per data series. The Status Chart is populated with the **Series Data** property in the **Property Editor**. This chart is good for showing machine states or HOA values over time. It is recommended to include some sort of Legend when using this chart.

Series Data

The first column of the **Series Data** property must contain datetime values. Each additional column should be numeric (the default columns are doubles). The order of the columns (left-to-right) determines the order of the entries on the chart (bottom-to-top). Because of this, re-ordering the entries would involve changing the order of the columns as they appear in the Series dataset by modifying the mechanism that is populating the dataset (i.e., changing the order of columns in a query).

In **Wide** format, all of the columns but the first must be numeric. These "series" columns' headers will be used as the names on the y-axis. In **Tall** format, there should be exactly three columns. The first is the timestamp, the second is the series name, and the third is the value. For example:

Wide Format

t_stamp	Valve1	Valve2
2010-01-13 8:00:00	0	2
2010-01-13 8:02:00	0	2
2010-01-13 8:04:00	1	2
2010-01-13 8:06:00	1	1
2010-01-13 8:08:00	0	1

Tall Format

t_stamp	Name	Value
2010-01-13 8:00:00	Valve1	0
2010-01-13 8:00:00	Valve2	2
2010-01-13 8:02:00	Valve1	0
2010-01-13 8:02:00	Valve2	2
2010-01-13 8:04:00	Valve1	1
2010-01-13 8:04:00	Valve2	2
2010-01-13 8:06:00	Valve1	1
2010-01-13 8:06:00	Valve2	1
2010-01-13 8:08:00	Valve1	0
2010-01-13 8:08:00	Valve2	1

Color Mapping

Apart from getting the data into the series chart, the only other commonly configured option is the mapping of discrete values to colors. This is done in the Status Chart Customizer. Each named series can have its own mapping of colors, if desired. These mappings are stored in the expert-level dataset property **Series Properties Data** so they can be altered at runtime.

Alarming in Vision

The Vision visualization system has a few components that allow you to view alarming information within the client. The Alarm Status Table allows you to view the current alarms in the system, providing an easy way to inspect the alarm details, shelve the alarm, and acknowledge it. The Alarm Journal Table allows you to view a history of alarms by reading from an Alarm Journal's database tables. Note, that

Note: A valid database connection and an Alarm Journal Profile are required to use the Alarm Journal Table.

On this page ...

- Alarm Status Table
- Configuring the Alarm Status Table
  - Filtering Alarms
  - Viewing Alarm Information
- Alarm Journal Table Component

Alarm Status Table

The **Alarm Status Table** is a built-in component in Ignition that displays the current status of all alarms in the system into one view. Each alarm is color coded so you can quickly identify the state of the alarm. The Alarm Status Table is highly customizable and can be configured to show active, unacknowledged, cleared, and acknowledged alarms. Once alarms have been configured, most users will want to view the status of alarms from within a client. Alarm acknowledged and shelving are built directly into the Alarm Status Table making it easy for operators to respond quickly to alarms.

Alarms can be visualized in several ways. By far the most popular way to visualize alarm status is with the Alarm Status Table component. This component immediately displays the status of all alarms in the system from all Tag providers. In addition, the Alarm Status Table has an extensive list of filtering options where you can filter down to specific alarm criteria. For example, the component can filter by source path which means a single Tag provider, folder in a Tag path, or even UDT instances can be focused on by the component.

INDUCTIVE UNIVERSIT

Alarm Status Component

Watch the Video

Configuring the Alarm Status Table

The Alarm Status Table component can be configured to view alarms, and monitor and respond to alarms, but there are a host of properties for this component to become familiar with. The Alarm Status Table properties are configured in the Designer. When configuring the Alarm Status Table, you will have to toggle between both the Designer and Preview Modes in order to configure the Alarm Status Table properties and organize the alarm data in the table.

Note: Alarms must be set up on Tags for them to show up in the Alarm Status Table.

When you drag the Alarm Status Table component to your Designer, there are some default properties already configured. Some of the defaults may work for your client users, but you always have the option to change them. For example, the table shows all the "Active and Unacknowledged", "Active and Acknowledged", and "Cleared and Unacknowledged" alarms. It also displays the time the alarm went Active as well as the current State and Priority of the alarm, but you may want to narrow the scope of your alarm state entries and only display Active and Unacknowledged alarms with a priority of Medium and higher. You can update these property settings in the Property Editor of the Designer under the Filters group.

Filtering Alarms

The Property Editor has a dedicated Filtering group where you can configure filtering properties for displaying alarm data in your Alarm Status Table: alarm state, priority, source path, display path, and Tag provider. Once you configure your properties, the alarm table will refresh with new alarm data based on your filter property settings. When displaying alarms on the Alarm Status Table component, it is common to filter on either the **'Display Path'** or the **'Source Path.'**

The **Display Path** can be customized on each alarm. The default value for an alarm's Display Path is a Tag path that leads to the name of the alarm. The Integer Tag example below has an alarm named 'Alarm', and is located on a Tag path of 'Alarming Example/Integer Tag', thus the Display path will resolve to '**Alarming Example/Integer Tag/Alarm**'. However, the Display Path can be customized when configuring the alarm. This is generally utilized to display readable messages as to what the issue is: i.e., "Tank 105 High Temp Alarm".

The **Source Path** is also a path to the alarm, but also shows the Tag provider the alarm is located in. Again using the Tag below, if the name of the Tag provider is 'default', then the source path would resolve to 'prov:**default:**/Tag:**Alarming example/Integer Tag**:/alm:**Alarm**'. Unlike the Display Path, the Source Path on an alarm can never be overridden.

Understanding the functionality of Display Paths and Source Paths allows for much flexibility when filtering alarms by Tag path.

Viewing Alarm Information

In Preview Mode and the Vision Client, you can customize how you want to see the alarm data. You can right click in the header to hide or show columns, and move columns by clicking them and dragging them around to organzied the alarm data. You can also sort by column by simply left clicking, or sort by multiple columns by holding the **Ctrl** key while clicking the column headers. You will see small numbers next to the column header to indicate the sort order and direction (ascending vs descending). Client users can customize the alarm data in the Alarm Status Table to however they wish to view the data.

Alarm Journal Table Component

There is a pre-built component in Ignition called the **Alarm Journal Table** that allows the database to see alarm history. It provides a built-in view to explore alarm history that has been stored in your database. Before viewing alarm history, you must first tell Ignition to log alarms in your database by creating your Alarm Journal Profile. By default, the Alarm Journal Table does not filter out any of the alarms, it simply shows you all the alarms stored in the table for the time range set. When you add an Alarm Journal component to a window and have no start and end dates selected, it defaults to show the previous 8 hours of alarms and will not refresh itself until the start or end date properties change. You can use the Start Date and End Date properties to filter on alarms, or you can use the Date Range component to filter alarms during a specific time period.

Alarm Journal Component

Refreshing Alarm Journal Table Data

Alarm data in the journal does not refresh automatically. Toggle one of the filters in the Property Editor, or on the Calendar component you are using. You can also choose to make the journal table data refresh automatically by setting an expression binding on the journal's **End Date** property to `now()`.

Watch the Video

Event Time	Display Path	Event State	Event Value	Priority	Current State
6/10/19, 2:46 PM	Motors/Motor 3/Amps/Low Amps	Active	48	Critical	Active, Unacknowledged
6/10/19, 2:46 PM	Motors/Motor 3/Amps/Low Amps	Ack		Critical	Cleared, Acknowledged
6/10/19, 2:46 PM	Motors/Motor 3/Amps/Low Amps	Clear	53	Critical	Cleared, Unacknowledged
6/10/19, 2:46 PM	Sine/Sine3/High Setpoint	Clear	39.9582	Low	Cleared, Unacknowledged
6/10/19, 2:46 PM	Motors/Motor 3/Amps/Low Amps	Active	48	Critical	Active, Unacknowledged
6/10/19, 2:46 PM	Motors/Motor 3/Amps/Low Amps	Ack		Critical	Cleared, Acknowledged
6/10/19, 2:46 PM	Ramp/Ramp8/OPC Alarm	Clear	157.52	Critical	Cleared, Unacknowledged

28,300 events

The alarms are color coded so you know what each of the states represent. In **Preview Mode**, you can right click in the header to hide or show columns, and move columns by clicking them and dragging them around. You can sort by column by simply left clicking, or sort by multiple columns by holding the **Ctrl** key while clicking the column headers. You will see small numbers next to the column header to indicate the sort order and direction (ascending vs descending).

To view an alarm's **Details** and **Notes**, select an individual alarm, then click the **Search** icon in the lower right corner of the table to see all the information that is associated with the individual alarm.

To close the Details and Notes tabs, click the **Search** icon again or click the **Expand** icon.

Related Topics ...

- Configuring Alarms
- Alarm Journal

In This Section ...

Vision Alarm Status Table - Common Tasks

The Alarm Status Table has a lot of configuration options that can be used to do things like filter the list of alarms being displayed, to changing how the component displays them. Each of the pages in this section goes over setting up various aspects of the Alarm Status Table.

On this page ...

- Vision Alarm Status - General Filtering
- Vision Alarm Status - Filter on Associated Data
- Vision Alarm Status - Row Styles
- Vision Alarm Status - Marquee Mode
- Vision Alarm Status - Acknowledgment
- Vision Alarm Status - Shelving
- Vision Alarm Status - Tag History and Alarm History
- Using Alarm Status Tags in Vision

Vision Alarm Status - General Filtering

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.

Vision Alarm Status - Filter on Associated Data

Sometimes, the built in filters of the Alarm Status Table are not enough. The Alarm Status Table component allows you to filter alarms through scripting and filter on any of the alarms properties, including user created associated data properties.

Vision 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 a blinking style, allowing you to really draw attention to critical alarms!

Vision Alarm Status - Marquee Mode

Sometimes, the alarms displayed in the table can build up and it can be a pain to scroll through the list. The Alarm Status Table can be put into a Marquee Mode, where it will slowly scroll through the list of alarms at a pace that you specify. Learn how to set that up and how to use it.

Vision Alarm Status - Acknowledgment

Typically, the first step in fixing an alarm is acknowledging that the alarm is happening. Acknowledgement can play an important part in any alarm system which is why the ability to acknowledge alarms is built right into the Alarm Status Table component. Learn how to acknowledge alarms and what the different options are.

Vision Alarm Status - Shelving

Shelving alarms allows you to hit the snooze button on an alarm event, postponing the alarm until later. This can be useful when doing maintenance if Tags are constantly going in and out of alarm.

Vision Alarm Status - Tag History and Alarm History

The Alarm Status Table allows you to view Tag History for Tags that are currently in alarm. As long as history is being stored for that Tag, you can view the history right in the component to get an idea of what the value was prior to going into alarm. Was it a slow rise to the setpoint, or was it a large spike?

Using Alarm Status Tags in Vision

While not part of the Alarm Status Table, these system Tags provide the number of Tags in specific states of alarm. Thes can be used to easily create a small notice of how many alarms are in the system without having to show the entire Alarm Status Table.

In This Section ...

Vision Alarm Status - General Filtering

Filtering in the Designer

There are several different filtering properties on the Alarm Status Table component. Once you drag the Alarm Status Table component on to a window, by default it shows you all the alarms that are currently 'Active and Unacknowledged,' 'Active and Acknowledged,' and 'Cleared and Unacknowledged,' with a priority ranging from Diagnostic to Critical. Some users want to do some more targeted filtering like only seeing the 'Active and Unacknowledged' alarms for a specific area or piece of equipment. This is easily accomplished by configuring a few alarm properties in the Property Editor.

The Alarm Status Table below is similar to what you'll see when you first drag in an Alarm Status Table to your Designer window. You can filter the alarm list in the table to be a shorter list using the Filter properties in the Property Editor instead of scrolling through every single alarm in your system. In the **Property Editor**, there is a group called **Filters** that you can configure to focus on only those alarms you want to see. You can filter on Priority, State, Source Path, Display Path**,** and Tag Provider.

You'll find, as you continue reading this page, that it is really easy to do some basic filtering using the alarm filter properties on the Alarm Status Table component in both the Designer and Vision Client.

On this page ...

- Filtering in the Designer
  - Filtering on Priority
  - Filtering on Alarm State
  - Filtering on Source Path and Display Path
- Filtering in the Vision Client

Alarm Status - General Filtering

Watch the Video

Filtering on Priority

You can filter on any Priority by changing the **Min Priority** field in the Property Editor. You will only then see alarms displayed in the Alarm Status Table for the priority you set in the Property Editor and higher alarms. This example sets the **Min Priority** to **'High'** priority, and the table displays only High, and Critical alarms. The number of alarms operators have to view can be narrowed down by using the **Min Priority** filter.

Filtering on Alarm State

There are four alarm states that an alarm can be in; 'Active and Unacknowledged', 'Active and Acknowledged', 'Clear and Unacknowledged', and 'Clear and Acknowledged'. Each of the four alarm states has a different color so you can quickly identify and respond to the most critical alarms. To enable and disable filtering for any of the alarm states, go to the **Filters** group in the Property Editor.

This example shows only alarms for **Active and Unacked**, and **Clear and Acked**. A good business practice is to show only the Active alarms that are either Unacknowledged or Acknowledged. It also reduces the number of alarms an operator has to scroll through.

Filtering on Source Path and Display Path

The Alarm Status Table has the capability to filter alarms based on the Source Path or 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 Motor alarms in the Motors folder. You can enter ***Tank*** (using ***** as a wildcard) to look for all alarms with the Motor 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.

You can see in this example, all the alarms have **'Tank'** in the **Source Path**.

Source Path Examples

Example Filter	Result
	Retrieve alarm information from the alarm at precisely the specified path:

`prov:tagProvider:/tag:Inputs/PS_1:/alm:MyAlarm`	`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, **'High Speed'** is the **Name** that was 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 **'High Speed'**.

Filtering in the Vision Client

In the Vision Client, the Alarm Status Table provides a host of built-in filtering options that are immediately available and easy to modify to help you get started. You can choose to filter on any of the filter properties by right clicking in the Alarm Status header and selecting any of the available properties.

You can even arrange the columns in any order you want by simply right clicking on the individual column header and dragging it to the right or left within the table. You can also sort in ascending or descending order by clicking in the column header.

Related Topics ...

- Vision - Alarm Status Table
- Configuring Alarms

Vision Alarm Status - Filter on Associated Data

Filtering on Associated Data

Another way to filter alarms in the **Alarm Status Table** is using associated data that was added to an alarm. You can easily and quickly search, filter, and display on specific alarms based on associated data configured in an alarm.

Using a Script to Filter on Associated Data

It's a common design practice to associate alarm groupings on associated data of an alarm. Scripting gives you the ability to filter on associated data, but also gives you the freedom to filter on anything that you want. In the following example, we used a script to filter on associated data, but first we need to add associated data and setup an alarm group.

Here we have a Memory Tag called **'Speed'** with a configured alarm called **'High Speed.'** To add associated data, click the **Add** icon at the top of the Tag Editor, scroll down the list of alarm properties to Associated Data, rename **'New Data'** to **'Group'** and add a static value called **'Production**.' Click **Commit** and save your Tag.

Next, create a script to filter for all alarms in the Alarm Status Table that have the associated data called **'Production.'** Scripting allows you to use the **'filterAlarm'** extension function specifically for filtering on associated data.

1. Right click on your Alarm Status Table component, and scroll down and select **Scripting**.
2. In the Component Scripting window, under **Extension Functions**, select **'filterAlarm.'**
3. Click the **Enabled** checkbox.
4. Enter the code below into the **'filterAlarm'** script. You can filter on anything you want here, but in this example, we are going to filter on **'Production.'**

Extension Function - filterAlarm for 'Production'

```
group = alarmEvent.get("Group")
if group == "Production":
        return True
return False
```

Your script will look like the image below. This script will only display alarms matching the associated data for **'Production.'** For every alarm matching **'Production,'** it will return **'True'** and show alarm results in the Alarm Status Table. If the associated data does not match **'Production,'** it will return **'False,'** and the Alarm Status Table will be empty. Note, when you're finished filtering on associated data, don't forget to disable your script.

On this page ...

- Filtering on Associated Data
  - Using a Script to Filter on Associated Data

INDUCTIVE UNIVERSITY

Alarm Status - Filter on Associated Data

Watch the Video

5. Click **OK**.
6. Now the Alarm Status Table below shows all the alarms in the **'Production'** group. Select an alarm and click the **Search** icon to see all the **Details** about the alarm.

Note:

- If you see alarms that do not match your associated data, check your filter settings in the Property Editor of the Alarm Status Table. You may need to uncheck the 'Show Clear and Unacked' and 'Show Clear and Acked' settings depending on what you want your operators to see.
- If you have an error in this filtering script, it will return 'true' for every alarm instance (and show all alarms) instead of displaying many errors to your users. You can find more information about the error from the Output Console under the **Tools Menu** in the Designer, and from the Vision Client under **Help > Diagnostics** and selecting the **Console** tab.

Related Topics ...

- Alarm Associated Data

Vision Alarm Status - Row Styles

The Alarm Status Table comes with a particular set of colors associated with each of the alarm states. You can change these colors for each of the states by going to the Alarms Row Styles Customizer that the Alarm Status Table component provides.

By default, the Active and Unacknowledged alarms are at the very top of the table and are red and blinking, and Cleared and Unacknowledged alarms are teal and at the bottom of the table, as shown in the image below.

Active Time	Display Path	Label	Event Value	Priority	Current State
6/13/19, 7:19 AM	Speed/High Speed	High Speed	130	Critical	Active, Unackno...
6/13/19, 7:19 AM	Humidity/High Humidity	High Humidity	86	Critical	Active, Unackno...
6/13/19, 7:19 AM	Tank Level 2/Low SP2	Low SP2	0	Critical	Active, Unackno...
6/14/19, 11:58 AM	Motors/Motor 4/Amps/Low Amps	Low Amps	46	Critical	Active, Unackno...
6/14/19, 11:22 AM	Motors/Motor 6/Amps/Low Amps	Low Amps	53	Critical	Cleared, Unackn...
6/14/19, 11:22 AM	Motors/Motor 6/Amps/Low Amps	Low Amps	51	Critical	Cleared, Unackn...
6/14/19, 11:22 AM	Motors/Motor 6/Amps/Low Amps	Low Amps	55	Critical	Cleared, Unackn...
6/14/19, 11:27 AM	Motors/Motor 6/Amps/Low Amps	Low Amps	51	Critical	Cleared, Unackn...
6/14/19, 11:17 AM	Motors/Motor 4/Amps/Low Amps	Low Amps	52	Critical	Cleared, Unackn...
6/14/19, 11:48 AM	Motors/Motor 3/Amps/Low Amps	Low Amps	51	Critical	Cleared, Unackn...

Acknowledge Shelve

On this page ...

- Customizing Alarm Row Styles
  - Creating a Row Style

INDUCTIVE UNIVERSITY

Alarm Status - Row Styles

Watch the Video

Customizing Alarm Row Styles

In the Alarm Row Styles Customizer is where you can modify an existing row style, add more styles, or delete a style. Each style has an expression, a color and the option if you want it to blink. The expression allows you to do any evaluation you want using any filter properties of the alarm: Priority, State, Display Path, Active Time, Clear Time, and many more.

In the Designer, right click on the **Alarm Status Table** component, go to **Customizers > Alarm Row Styles** to see the default row styles. Alarm Row Styles is an ordered list and each style has an expression. How it works is, the first style that returns **'True'** for a given alarm is the one that is going to be used. So you want to make sure the order is the correct order that you want. If you want to change the order, select a row and click on the up or

down arrow icons. Click on each of the row styles to view the expression and row style.

Creating a Row Style

Let's create another state alarm with a new row style. Call this new alarm state **'ActiveUnacked'** and make it **'Critical.'**

1. With the Alarm Row Styles customizer open, click on the **Add** icon, and call this new alarm **'ActiveUnacked'** and make it **'Critical'.**
2. Copy and paste the expression from the code block below into the Expression block of the Alarm Row Styles window. Notice that we are making this alarm state a priority 4.

```
{state}='ActiveUnacked' && {priority}>=4
```

> **Alarm State Priority Numbers**
>
> Priorities are numbered behind the scenes, but named when you are creating them. 0 = Diagnostic, 1 = Low, 2 = Medium, 3 = High, and 4 = Critical. This allows you to use operations like '>=' (greater than or equal) when filtering on priority.

3. The new alarm state by default is added to the bottom of the list. Style order is very important. If you keep this style at the bottom of the list, the alarm will never be visible in the Alarm Status Table. Move **ActiveUnacked** with a **Priority >= 4** to the top of the row styles list so it is evaluated before the other styles.
4. Since this is a critical alarm, you want it to catch the operator's attention, so it's a good idea to make the style standout. Make the Foreground color **'Black'** and the background color **'Red.'** Check the **Blink Box** and make the foreground color **'Black'** and background color **'Yellow.'**
5. Press **OK**.

6. The critical alarms will now blink red and yellow and will definitely catch the operator's attention.

Active Time	Display Path	Label	Event Value	Priority	Current State
6/13/19, 7:19 AM	Speed/High Speed	High Speed	130	Critical	Active, Unacknow...
6/13/19, 7:19 AM	Humidity/High Humidity	High Humidity	86	Critical	Active, Unacknow...
6/13/19, 7:19 AM	Tank Level 2/Low SP2	Low SP2	0	Critical	Active, Unacknow...
6/14/19, 2:31 PM	Motors/Motor 2/Amps/Low Amps	Low Amps	50	Critical	Active, Unacknow...
6/14/19, 2:43 PM	Motors/Motor 4/Amps/Low Amps	Low Amps	46	Critical	Active, Unacknow...
6/14/19, 2:44 PM	Motors/Motor 3/Amps/Low Amps	Low Amps	45	Critical	Active, Unacknow...
6/14/19, 2:45 PM	Motors/Motor 1/Amps/Low Amps	Low Amps	46	Critical	Active, Unacknow...
6/14/19, 2:45 PM	Ramp/Ramp8/OPC Alarm	OPC Alarm	343.7333	High	Active, Unackno...
6/14/19, 2:30 PM	Motors/Motor 2/Amps/Low Amps	Low Amps	52	Critical	Cleared, Unackn...
6/14/19, 2:30 PM	Motors/Motor 2/Amps/Low Amps	Low Amps	51	Critical	Cleared, Unackn...

Acknowledge Shelve

To learn more about states and priorities including all the alarm configuration properties, refer to the sections on Alarm Event Properties Reference.

Vision Alarm Status - Marquee Mode

The Alarm Status Table can take up a lot of space on your Ignition client window as shown below. In many production environments, a small Alarm Status Table is often displayed at the very bottom of the Client window taking up very little space. The problem is when an operator has quite a few Active alarms, they have to manually scroll through the list of alarms to see them all. This can be a little tedious for operators. If you want to minimize the size of your Alarm Status Table, and not miss any alarms, you can turn on the **Marquee Mode** option.

On this page ...

- Setting Up Marquee Mode

INDUCTIVE UNIVERSITY

Alarm Status-Marquee Mode

Watch the Video

Setting Up Marquee Mode

The nice feature with Marquee Mode, is it automatically scrolls through all the alarms when there is not enough vertical space on the client window to show them all at once.

1. Select the Alarm Status Table. In the **Property Editor** of the Alarm Status Table, set the **Marquee Mode** to **'true.'**

2. This client screen shows you a typical Alarm Status Table using the Marquee Mode located at the bottom of the window. The Alarm Status Table automatically scrolls through all alarms that are currently visible. Marquee Mode makes it very easy for operators to see what's going on without having to manually scroll through the entire list of alarms to identify all the alarms that are Active or that need attention.

Set filters and inform operators that Marquee Mode is a scrolling list of alarms

When using Marquee Mode, it is even more important to pay attention to the filters configured on the Alarm Status Table since there is limited room. Make sure your operators know that even though the Alarm Status Table may be small, when Marquee Mode is enabled it becomes a scrolling list.

Vision Alarm Status - Acknowledgement

Using Alarm Acknowledgement

One of the most important things an operator is going to do in the client 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 or multiple alarms. You can also use the Shift+Click multi select feature to select a range of alarms, or the Select All checkbox in the header bar. Check all the alarms you want to acknowledge, and press the **Acknowledged** button. In this example, two alarms were checked, and two alarms were Acknowledged at the same time. The Alarm Status Table will record the time the alarm was acknowledged and the user that acknowledged the alarm.

If you don't have the Ack'ed By or Ack Time column in the header of the Alarm Status Table, you can simply add them in the Designer in **Preview Mode** as well as in the **Client**. Right click on the header bar, and from the dropdown list check **Ack'ed By** and **Act Time**, and any other information you want to display or remove from the Alarm Status Table.

If any of the selected alarms require **Acknowledgement Notes**, a small window will appear in which the operator will be required to add notes, otherwise, the alarm cannot be acknowledged. If the operator wants to cancel the Acknowledgement Notes and close the window, click on the **Search** icon or **Expand** icon to close the Acknowledgement Notes window.

Acknowledgement Notes are setup on the alarm's configuration settings. To setup Acknowledgement Notes, go to your alarm configuration settings and set **'Ack Notes Required'** to **'true.'** For more information refer to Configuring Alarms.

On this page ...

- Using Alarm Acknowledgement
- Security for Alarm Acknowledgement

Alarm Status - Acknowledgement

Watch the Video

Security for Alarm Acknowledgement

If you want to restrict who can use the Acknowledge button, there is the **Show Ack Button** in the Property Editor that can be set to **'False.'** By setting the **Show Ack Button** to **'false,'** this hides the Acknowledge button on the Alarm Status Table.

In order for operators to acknowledge alarms, the correct permission must be assigned. This example shows how to set permissions to acknowledge alarms for users in the **Operator** role. You can setup permissions for any role, user and user source in your system.

1. Select the Alarm Status Table component, and click the **Show Ack Button** 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 '**hasRole.'** This enters the function name.
4. Edit the expression to read: **hasRole("Operator")**
5. Click **OK**.

If you currently have the "Operator" role you will notice that the **Show Ack Button** property is now **'true,'** otherwise, it will be **'false.'**

Vision Alarm Status - Shelving

Another important feature of the Alarm Status Table is the capability to shelve alarms. 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 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.

> **Active Alarms**
>
> Active alarms must be present in the Alarm Status Table before you can shelve an alarm.

On this page ...

- How to Shelve an Alarm
- Custom Shelving Duration

INDUCTIVE UNIVERSITY

Alarm Status - Shelving

Watch the Video

How to Shelve an Alarm

To shelve an alarm, select one alarm or multiple alarms, then press the **Shelve** button. The **Shelving** tab will automatically open so you can set a duration to silence an alarm You can set a duration from 5 minutes to 4 hours in which to shelve selected alarms. Choose the duration and click **Apply**.

To view all the alarms that are currently shelved, go to the bottom right of the Alarm Status Table and click on the **Shelved Alarms** icon. A Shelved Alarm tab will open and show all of the shelved alarms, as well as when the shelved time period expires. To close the Shelved Alarm tab, click either the **Shelved Alarm** icon or the **Expand** button 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' in the Alarm Status list instead of 'Active.'

To unshelve an alarm, select one or more alarms under the **Shelved Alarms** tab, click the **Unshelved** link**,** and the alarm will return back to the Alarm Status list.

Go to **Preview Mode** and test out shelving some of your alarms.

Custom Shelving Duration

The shelving duration values on the Alarm Status Table component can be customized. By default, the component has several shelving preset 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 be temporary, not long term.

In the Property Editor, there is a property called **Shelving Times.** This is where you can add, remove, or change the shelving times for alarms. In the following example, **90 minutes** was added to the list of shelving times.

1. Select the Alarm Status Table component, and click on the **DatasetViewer** icon next to the **Shelving Times** property to open the Dataset Viewer.
2. Select a row and click the green **Add Row** icon to insert a new row. Enter the amount of time and unit of time (i.e., 90 minutes).
3. Click **OK**.

4. To verify your new time was entered correctly, select the alarms you want to shelve and click the **Shelve** button.
5. Choose **90 minute** shelve time from the dropdown list. Click **Apply** to temporarily silence your alarms. Note, you may need to close and reopen your window to see the new shelve time duration.

Vision Alarm Status - Tag History and Alarm History

The **Alarm Status Table** component has a unique feature built into it that lets you view the actual trend of a Tag's value and an alarm's history for any alarm that is currently active. It shows you exactly where the alarms occurred, when it was cleared, and when it was acknowledged.

For the trend feature to work, you must have your Tag history setup so it's logging to the Tag Historian as well as an Alarm Journal Profile configured so that the log history gets logged to a database. As long as both the Tag Historian and Alarm Journal are configured, you will see the trend for any given alarm that is currently active or in the Alarm Status Table list.

The Tag Browser will display the **Tag History** icon when alarm history is enabled.

Alarm Journal Property
If you have more than one alarm journal configured, the **Journal Name** property field is blank. Make sure you enter the name of the Alarm Journal you intend to use, otherwise your alarm log history will not get logged to the database.

In the Alarm Status Table, select any **alarm** and click on the **Trend** icon located on the bottom right of the table. The Trend Chart shows you the trend of the selected alarm's value over time. You can look at the last 5 minutes, 30 minutes, 1 hour, 8 hours or up to 24 hours to view the exact history of that Tag's value, as well as all the alarms that occurred over time.

The yellow triangles denote when the alarm was active, the green circles identify when the alarm was cleared, and a vertical black bar shows when the alarm was acknowledged. It's all built into the Alarm Status Table component and you don't have to go to a separate window to view an alarm trend or alarm history.

Related Topics ...

- Configuring Alarms
- Alarm Journal
- Configuring Tag History

INDUCTIVE UNIVERSITY

Alarm Status - Tag History and Alarm History

Watch the Video

Using Alarm Status Tags in Vision

Ignition displays four Gateway System Tags that give you the number of alarms that are Active and Acknowledged, Active and Unacknowledged, Cleared and Acknowledged, and Cleared and Unacknowledged. A quick way to see if any alarms are active, and to get an active alarm count is to add a Label component on a Vision navigation window.

Add a Label with Alarm Status

Let's setup an Active Alarm on a Navigation window using a Label component for the **Active and UnAcked** Tag showing how many alarms are currently active and unacknowledged.

1. In the **Project Browser** open your **Navigation window**, and drag in a **Label** component and name it **"Active Alarms."** This example uses a Tree View as the navigational component.
2. Next, we need to link the Label to the alarm Tag, but first we'll need a custom property. In the **Property Editor,** right click on the **Label** component, and select on **Customizers > Custom Properties**. The Custom Properties window will open.
3. To add a custom property, click the **Add** icon, and enter the name **'activeAlarms'** with an **'integer'** data type. Click **OK.**

4. You'll find your new custom property in the Property Editor under the **Custom Properties** category. Now, let's bind our new custom property, **'activeAlarms'** to the **'Active and Unacked'** Tag. Select the **binding** icon next to the **'activeAlarms'** property. The Property Binding Window will open.
5. Select **Tag** type and choose the **System > Gateway > Alarming > Active and Unacked** Tag.
6. Click **OK** to save.

7. Now, let's display the number of alarms that are active on the Label. Bind the **Text** property of the Label to the following expression. Click on the **binding** icon for the **Text** property and select the **Expression** binding type.
8. Copy and paste the following expression in the Expression area. You'll notice the the expression contains a literal string value **' Active Alarms.'** Click **OK** twice.

```
{Root Container.Active Alarms.activeAlarms} + " Active Alarms"
```

You can also create the expression by navigating to the custom property (i.e., activeAlarms). Ignition will build the expression for you. Click on the **binding** icon for the **Text** property and click on **Expression**. Click the **Property** icon and select the **activeAlarms** property, then click **OK** to save the property. Click **OK** again to save your binding.

9. The last thing to do is setup a style to make the Active Alarms label stand out on the Navigation window. Let's change the color to something bright and make it blink. Right click on the **Label** and select **Customizers > Style Customizer.**
10. For the **Driving Property**, select **'activeAlarms.'** For the **Available Properties**, select (one at a time): **Background Color**, **Foreground Color**, **Border** and **Visible**, and click on the green arrow pointing to the right and add them to the **Used Properties** list.

11. Under the **Styles** section, click on the Add icon twice times to add two **Values**. Click the **Expand** icon to open and customize the style for the Active Alarm Label. Set the following values for **Styles**:

Value 0
Background Color: **Red**
Border: **Etched (Raised)**
Foreground Color: **White**
Visible: **check the box**

Value 1
Click the **Animate** box. Use the green plus icon to add 2 **animation steps**. Change the values to

Step 1 Duration: **1000**
Background Color: **Red**
Border: **Etched (Raised)**
Foreground Color: **White**
Visible: **check the box**

Step 2 Duration: **1000**
Background Color: **Yellow**
Border: **Etched (Raised)**
Foreground Color: **Black**
Visible: **check the box**

Click **OK.**

Now, you have a label on the Navigation window that provides an active alarm count and blinks when there is an active alarm.

Vision Alarm Journal Table - Common Tasks

The Alarm Journal Table has a lot of configuration options that can be used to do things like filter the list of alarms being displayed, to changing how the component displays them. Each of the pages in this section goes over setting up various aspects of the Alarm Journal Table. Note that an 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 ...

- Vision Alarm Journal - General Filtering
- Vision Alarm Journal - Filter on Associated Data
- Vision Alarm Journal - Filter on Date Range
- Vision Alarm Journal - Focusing on Alarms
- Vision Alarm Journal - Row Styles
- Vision Alarm Journal - Searching

Vision Alarm Journal - General Filtering

The Alarm Journal Table component has many built in properties that allow you to filter on various parts of an alarm. Learn about the different built in filtering options that the Alarm Journal Table provides.

Vision Alarm Journal - Filter on Associated Data

The Alarm Journal Table provides the ability to write scripts to filter the alarm history being displayed. While the script can filter on any properties of the alarm, this page goes over how to filter on the user created associated data properties.

Vision Alarm Journal - Filter on Date Range

The Alarm Journal has start and end date properties that determine the time range from which to grab the alarm history. This can be bound to a Date Range component that allows you to filter for specific alarm events by choosing a start date and end date.

Vision Alarm Journal - Focusing on Alarms

Sometimes, it may be useful to see only specific events in the history. Without changing the filters, the Alarm Journal Table allows you to focus on a specific alarm, and see all instances of it by filtering out other events.

Vision Alarm Journal - Row Styles

The Alarm Journal Table colors each row a specific way depending on what type of event it is. These Row Styles can be completely configured, allowing you to not only change the colors and fonts for different events, but also to setup rows styles on based on other properties as well.

Vision Alarm Journal - Searching

The Alarm Journal Table has a built in search tool that allows you to search for specific events based on a set of keywords or priorities.

In This Section ...

Vision Alarm Journal - General Filtering

Note: To view alarm history, an Alarm Journal Profile must be created first before logging alarms to the database.

On this page ...

- Alarm Journal Table Filtering
  - Filtering in the Designer
  - Filtering in the Client

Alarm Journal Table Filtering

Ignition provides a host of filters to view alarm history data in the Designer and Vision Client. Not only can you filter the Alarm Journal Table on a date range to see alarm history data within a specific time period, but there are also a number of other filters you can use as well.

In the Designer you can filter alarms using the filter properties in the Property Editor. In the Vision Client, you can filter alarms by clicking on the **Filter** icon at the bottom of the Alarm Journal Table.

Filtering in the Designer

In the Designer, drag an Alarm Journal Table component into your window. By default, Ignition filters on four different type of alarm events: **System Events, Active Events, Cleared Events and Acked Events**. It also filters on all alarm priorities ranging from **Diagnostic** to **Critical**. In the Property Editor you can see each type of alarm event. You can show only those alarm events you want to see by checking the specified alarm event and leaving the other events unchecked.

Note:

When an Alarm Journal Table component is initially placed on a Vision window, the table displays the last 8 hours of alarm events stored in the Alarm Journal Profile. From here, you can filter alarms using the host of filtering properties in the Property Editor to expand or narrow the list of alarms.

INDUCTIVE UNIVERSITY

Alarm Journal - General Filtering

Watch the Video

You can also filter on **Source Path**, **Priority**, **Display Path** properties, and even create your own **Search String**. In the following example, we searched the **Display Path** for **'Sine'** and **'Speed'** and applied a basic filter to show only **Active** and **Acked** events as you can see by the different alarm state colors. We also set the **Priority** to see 'Medium' to 'Critical' alarms. You can see that all the **'Sine'** and **'Speed'** results are shown in the **'Di splay Path'** column of the journal table.

In Preview Mode and in the Vision Client, you can right click in the column header to show or hide columns, reorganize the columns, and resize the columns. You can also sort by column by simply left clicking, or sort by multiple columns by holding the **Ctrl** key while clicking the column headers. You will see small numbers next to the column header to indicate the sort order and direction (ascending vs descending).

You can also filter alarms based on a **Tag Path** and **Tag Provider**. If you configure alarms in particular folders, you can use the Source Filter to find alarms by Tag Path. In this example, ***Sine1*** was entered using wildcards in the Source Path field to find all the **'Sine1'** alarms that are **Active** and **Ac ked** in all the **'Sine'** folders with a **Priority** of 'Medium' to 'Critical.'

You can also filter on alarms by **Name** using the **Display Path** filter, which is the alarm name that was setup when you configured the alarm. In this example, **'High Speed'** was entered in the **Display Path** field to find all the alarms in the system with a **'High Speed'** name, and a **Priority** of **'Medium'** to **'Critical.'**

You can also create your own **Search String** to find specific alarms, events, and conditions. In this example, **'High Speed'** and **'High Level'** were entered in the **Search String** field to find all alarms that matched the string criteria.

Filtering in the Client

You can filter alarms in the Vision Client using the **Filter** tab by clicking on the **Filter** icon at the bottom of the Alarm Journal Table. Let's filter on **Priority** using the Filter tab. When you first open the Alarm Journal Table, it displays every single alarm on all the priorities from Diagnostic to Critical. If all you want to display is High to Critical alarms, select **'High'** from the first **Priority** dropdown on the left and **'Critical'** from the second dropdown box. You'll notice a **Search** field and **System**, **Active**, **Cleared** and **Acked** alarm events are also available to filter alarms.

In this example, we set the **Search** filter to find all **'Sine0' Active and Acked** alarm events with a **Priority** of **'Critical.'** To close the Filter tab, click on the **Filter** icon or the **Expand** icon.

All of the Alarm Journal Filter properties are bindable so you can design your Visions windows and let the operator choose and filter the alarm data based on how they want it displayed.

Related Topics ...

- Configuring Alarms
- Alarm Journal

Vision Alarm Journal - Filter on Associated Data

Alarm Journal Profile and Alarms Configuration
This section assumes that an Alarm Journal Profile, alarms, and associated data are already configured before filtering on associated data in the Alarm Journal Table. To learn more, go to the Configuring Alarms, Alarm Associated Data, and Alarm Journal page

On this page ...

- Set Up the Alarm Journal to Filter on Associated Data

Just like the Alarm Status Table you can filter the Alarm Journal Table based on associated data that you added to your alarms.

Set Up the Alarm Journal to Filter on Associated Data

It's a common design practice to associate alarm groupings on associated data of an alarm. If you create alarm groupings on associated data of an alarm, you'll most definitely want to filter on your associated data in the Alarm Journal Table. Associated data is outside the Source Path and Display Path, and scripti ng gives you the ability to filter on associated data using the **'filterAlarm'** extension function. It also gives you the freedom to filter on anything that you want, such as alarms in another area of the plant or for a certain set of alarms. The script is only going to show you the alarms matching the associated data. You can also add other filters as well.

In the following example, we'll use the **'filterAlarm'** extension function to filter on associated data, but first we need to add associated data on an existing alarm.

INDUCTIVE UNIVERSIT

Alarm Journal - Filter on Associated Data

Watch the Video

1. This example uses a Memory Tag called **'Speed'** that has an alarm configured called **'High Speed'**. An asssociated data property was added to the alarm called **'Group'** with a static value of **'Production.'** You can filter on what is shown in the alarm list of the Alarm Journal Table based on the associated data. In this example, we are going to filter on a **'Group'** called **'Production.'**

2. Once your associated data is configured in your alarm, setup your script to filter for all the alarms in your history that have the associated data named **'Group'** with a value of **'Production.'** Right click on your Alarm Journal Table component and select **Scripting**. The Component Scripting window will open.
3. On the left side of the window under the **Extension Functions** folder, click **'filterAlarm.'**
4. Check **Enabled** to enable the script, and enter the code you want to filter on for your associated data. The script used in this example is shown below. If you want to use this example, copy the code from the code block and paste it at the end of the script and click **OK** to save your script.

```
group = alarmEvent.get("Group")
if group == "Production":
        return True
return False
```

5. The filterAlarm script will find every alarm matching **'Production'** and return 'true' and display the alarms in the journal table. If the associated data does not match **'Production,'** it will return 'false' and no alarms will be displayed in the journal table.

Refreshing the Data in the Alarm Journal Table
By default, the Alarm Journal Table does not refresh the data in the table automatically if you are working in the Designer. Toggle one of the filters on the table to refresh the data and you'll see the table refresh with your associated alarm data. If you are opening a Client window, the data will automatically refresh.

6. The alarm list below shows all the alarms with associated data filtering for the **'Group'** called **'Production'** within the dates displayed on the Date Range component.

Event Time	Display Path	Name	Event Value	Event State	Priority
6/25/19, 1:07 PM	Speed/High Speed	High Speed	122	Active	High
6/25/19, 1:06 PM	Speed/High Speed	High Speed	97	Clear	High
6/25/19, 1:06 PM	Speed/High Speed	High Speed		Ack	High
6/25/19, 1:06 PM	Writeable/WriteableInteger1/Low Tank Level	Low Tank Level	15	Active	Critical
6/25/19, 1:06 PM	Writeable/WriteableInteger1/Low Tank Level	Low Tank Level	66	Clear	Critical
6/25/19, 1:05 PM	Writeable/WriteableInteger1/Low Tank Level	Low Tank Level		Ack	Critical
6/25/19, 8:44 AM	Speed/High Speed	High Speed	132	Active	High
6/25/19, 8:43 AM	Speed/High Speed	High Speed	99	Clear	High
6/25/19, 8:43 AM	Speed/High Speed	High Speed		Ack	High
6/25/19, 8:43 AM	Writeable/WriteableInteger1/Low Tank Level	Low Tank Level	18	Active	Critical

7. As long as the **'filterAlarm'** script is enabled, the Alarm Journal Table will display all alarms matching the criteria for the associated data. To disable the script, open the **'filterAlarm'** and uncheck **Enabled**.

Related Topics ...

- Configuring Alarms
- Alarm Associated Data
- Vision Alarm Journal - Filter on Date Range

Vision Alarm Journal - Filter on Date Range

Alarm Journal Profile and Alarms Configuration
This section assumes that an Alarm Journal Profile and alarms are already configured before proceeding with filtering alarms using a date range. To learn more about alarms, go to Configuring Alarms and Alarm Journal pages.

INDUCTIVE UNIVERSIT

Alarm Journal - Filter on Date Range

Watch the Video

Alarm Journal Table

The Alarm Journal Table displays the complete history of all alarms within a specific time period. Ignition offers several types of date pickers under the Calendar tab of the Component Palette, but it's far easier and more common to use the Date Range component for filtering alarms over time.

When the Alarm Journal Table component is initially placed on a window, the component shows the last 8 hours of alarm events stored in the Alarm Journal Profile. Typically, operators want to filter alarm events within a specific time period rather than scroll through a long list of alarm events. Selecting Start and End Dates is very easy using a Date Range component.

Setting Up Alarm Journal Table to Filter on Date Range

The Start Date and End Date are built into the Date Range component so you can select the Start Date and End Date all in one place.

1. In the Designer, drag the Date Range component from the Component Palette to your window and place it above your Alarm Journal Table.

2. Link the Alarm Journal Table component to the Date Range component beginning with the **Start Date**.

   a. Select the **Alarm Journal Table component**, click the **Start Date binding** icon. The Property Binding window will open.
   b. Click the **Property** binding type**.**
   c. Select the **Date Range** component's **Start Date.**
   d. Click **OK**.

3. Now, let's link the Alarm Journal Table to the Date Range component's **End Date.**

   a. Select the **Alarm Journal Table component**, click the **End Date binding** icon. The Property Binding window will open.
   b. Click the **Property** binding type**.**
   c. Select the **End Date** for the **Date Range** component.
   d. Click **OK**.

4. The Alarm Journal Table will refresh with all alarm events within that time period. Click on **Preview Mode** and drag the slider on a specified date range on the timeline. You can drag or stretch the selection box around the timeline. The selected date range is always a whole number of units, where the unit is determined by the current zoom level. You can also zoom in or out on specific time periods to see the alarm events that exist in the alarm history table.

Event Time	Event Id	Display Path	Event State	Priority
8/5/20, 8:58 AM	320304ec-eab8-4042-9cf8-17c63a944af2	Sine/Sine0/High level	Active	Critical
8/5/20, 8:58 AM	43556908-85b4-4228-89bf-6937fc8cc3ed	Sine/Sine0/High level	Ack	Critical
8/5/20, 8:58 AM	206e29f0-3ece-4080-9ea8-ee5c23146226	Sine/Sine0/High level	Clear	Critical
8/5/20, 8:57 AM	8682d4d5-b5d2-4f51-9ae8-77fea347b05a	Sine/Sine1/High Level	Active	High
8/5/20, 8:57 AM	867dfe3a-df3d-4242-91f2-e9e02cfedf24	Sine/Sine1/High Level	Ack	High
8/5/20, 8:56 AM	206e29f0-3ece-4080-9ea8-ee5c23146226	Sine/Sine0/High level	Active	Critical
8/5/20, 8:56 AM	1cf37a29-9f4f-4ff2-a5b0-23a99fd1c2bf	Sine/Sine0/High level	Ack	Critical
8/5/20, 8:56 AM	6fc7d382-6ca8-4d64-97a9-c012023fe42e	Sine/Sine2/Low Level	Active	Critical
8/5/20, 8:56 AM	73b0cc70-f763-4dff-9222-3624e8584704	Sine/Sine2/Low Level	Ack	Critical

Note: Alarm Journal Table fetches data only when the window is first opened and when the Start or End Dates change. To see new alarm events come into the table, use the slider to change the dates.

Related Topics ...

- Vision - Date Range Component

Vision Alarm Journal - Focusing on Alarms

The Alarm Journal Table can focus on specific events in the alarm list. The Focus feature filters out alarms while reducing the number of alarm events displayed in the Alarm Journal Table, thus, allowing you to view specific instances of an alarm source. You can focus on one **Target Alarm Source** or the **Target Event Id** to look at when alarms occur. It's a nice way of filtering through the Alarm Journal Table without having to scroll through the entire list of alarms to match up the Alarm Events, Times and States since the Active, the Cleared and the Acknowledged states are not going to be in sequential order depending on when the event times actually happened.

The Alarm Journal Table shows you a complete history of all alarm events within a specific time period which can be quite large. The following examples will show you how to use the focus feature to pare down the list of alarm events and to focus strictly on those alarms of interest to you.

On this page ...

- Using the Focus Feature
  - Target Alarm Source
  - Target Event Id

Using the Focus Feature

The Alarm Journal Table can become quite large with alarm events over a period of time. The **Focus** feature allows you to focus in on one specific alarm and view the history for that alarm using **Target Alarm Source** and **Target Event Id**. Notice, the examples on this page are using the Date Range component to focus on the Target Alarm Source and Target Event Id within a given time period.

Target Alarm Source

If you want to see alarms from just a single Source Path, select any alarm event in the journal table, click on the **Focus** icon, and choose **Target Alarm Source**. This returns the complete alarm history of that Source Path within that given time range. Note, make sure to have your **Source Path** column set in the journal table. Refer to Alarm Journal Table Component for more information on configuring table columns.

In this example, we focused on the High Speed alarm with a source path: `prov:default:/tag:Speed:/alm:High Speed`.

Once the Target Alarm Source option was clicked, all alarm events from the selected Source Path were displayed in the journal table within the specified date range, including any changes to the state, as well as acknowledgments.

Event Time	Source Path	Display Path	Event V...	Event State	Priority
6/26/19, 1:53 PM	prov:default:/tag:Speed:/alm:High Speed	Speed/High Speed	101	Active	High
6/26/19, 1:53 PM	prov:default:/tag:Speed:/alm:High Speed	Speed/High Speed	99	Clear	High
6/26/19, 1:51 PM	prov:default:/tag:Speed:/alm:High Speed	Speed/High Speed		Ack	High
6/26/19, 1:48 PM	prov:default:/tag:Speed:/alm:High Speed	Speed/High Speed	103	Active	High
6/26/19, 1:48 PM	prov:default:/tag:Speed:/alm:High Speed	Speed/High Speed	96	Clear	High
6/26/19, 1:48 PM	prov:default:/tag:Speed:/alm:High Speed	Speed/High Speed		Ack	High
6/26/19, 1:27 PM	prov:default:/tag:Speed:/alm:High Speed	Speed/High Speed	107	Active	High
6/26/19, 1:27 PM	prov:default:/tag:Speed:/alm:High Speed	Speed/High Speed	90	Clear	High
6/26/19, 1:26 PM	prov:default:/tag:Speed:/alm:High Speed	Speed/High Speed		Ack	High

To remove the Focus, click on the **Focus** icon again, and you will see all alarms events in the journal table refresh within the given date range.

Target Event Id

In this example, `Motors/Motor 1/Amps/Low Amps dated 6/26/19 at 1:59 PM` with an Active event state was selected and the **Target Event Id** option was chosen. Note, make sure you have the **Event Id** column set in the journal table. Refer to Alarm Journal Table Component for more information on configuring table columns.

Event Time	Event Id	Display Path	Event Value	Event State	Priority
6/26/19, 1:59 PM	5d5a562a-10a5-424b-81e5-2c61f2621eeb	Motors/Motor 1/Amps/Low Amps	46	Active	Critical
6/26/19, 1:59 PM	b3af1bca-2a9d-4d22-b769-91a407f39646	Motors/Motor 1/Amps/Low Amps		Ack	Critical
6/26/19, 1:59 PM	f818582c-5297-4f1a-8193-a0320578407a	Motors/Motor 1/Amps/Low Amps	51	Clear	Critical
6/26/19, 1:59 PM	ca433c63-acf0-4ebc-b9fe-1d7284800e47	Motors/Motor 3/Amps/Low Amps	46	Active	Critical
6/26/19, 1:59 PM	f3f1cddb-91bc-4e83-9a87-7903e3b3547e	Motors/Motor 3/Amps/Low Amps		Ack	Critical
6/26/19, 1:59 PM	f818582c-5297-4f1a-8193-a0320578407a	Motors/Motor 1/Amps/Low Amps	48	Active	Critical
6/26/19, 1:59 PM	89d89f8b-2ff3-4aa5-9c21-4cd4aa617f6a	Motors/Motor 1/Amps/Low Amps		Ack	Critical
6/26/19, 1:58 PM	ac1a8c0f-ec26-45f0-a1bb-ec10d318ea9a	Motors/Motor 4/Amps/Low Amps	51	Clear	Critical
6/26/19, 1:58 PM	2d4a7902-ba33-472b-8d39-1025a8901a13	Motors/Motor 1/Amps/Low Amps	54	Clear	Critical

The Target Event Id returned all the alarms events with that specific Event Id during within the given date range. You can see when the alarm went Active, Cleared, and Acknowledged.

Click the **Focus** icon to remove the focus**,** and you will see all alarms events in the journal table refresh within the given date range.

Vision Alarm Journal - Row Styles

The Alarm Journal Table allows you customize row styles for different states of alarm history. Just like the Alarm Status Table, the Alarm Journal Table comes with a particular set of colors associated with each of the alarm states as shown in the image below. You can change these colors for each of the states by going to the Alarms Styles Customizer that the Alarm Journal Table component provides.

On this page ...

- Customizing Alarm Row Styles
  - Creating Row Styles for Different Alarm States

Alarm Journal - Row Styles

Watch the Video

Customizing Alarm Row Styles

Alarm Row Styles is where you can modify an existing row style, add more styles, or delete a style. The Alarm Row Styles Customizer gives you a head start for building a new expression. The expression allows you to do any evaluation you want using any filter properties of the alarm: Priority, State, Display Path, Active Time, Clear Time, and many more.

In the Designer, right click on the Alarm Journal Table component, go to **Customizers > Alarm Row Styles** to see the default row styles. Alarm Row Styles is an ordered list and each style has an expression. How it works is, the first style that returns **'True'** for a given alarm is the one that is going to be used. So you want to make sure the order is the correct order that you want. If you want to change the order, select a row and click on the up or

down arrow icons. Click on each of the row styles to view the expression associated with the row style.

Creating Row Styles for Different Alarm States

The main reason to create a new row style is for overlapping conditions for styling. Let's create another state alarm with a new row style and state.

1. With the Alarm Row Styles customizer open, click on the **Add** icon.
2. By default, the new alarm state is added to the bottom of the list. The following expression creates a new state for alarms **'Ack by the user admin.'** Copy and paste the expression into the Expression area of the Row Styles window.

```
{ackUser}='usr-prov.default:/usr:admin'
```

3. To make this alarm state catch the attention of the operator, let's make the row style standout by making it blink. Make the foreground color **'B lack'** and the background color **'Yellow.'** Check the **Blink** box and make the foreground color **'Black'** and background color **'Red.'**
4. Move the new state for **'Ack by the user admin'** above the **'Ack'** state so it gets evaluated first, otherwise if **'Ack'** state is evaluated first, it will become **'True'** first and the new state **'Ack by the user admin'** will never be evaluated.
5. Press **OK** to save your updates.

6. Now, when the user **'admin'** acknowledges an alarm, it will blink yellow and red in the Alarm Journal Table.

Refer to the Alarm Event Properties Reference section to learn more about alarm properties and how they are used in the Alarm Row Style Customizer.

Vision Alarm Journal - Searching

Using the Search Feature

The Alarm Journal Table component can search through alarm history and filter for particular alarms with any criteria. The Search feature is built into the Alarm Journal Table component and found on the bottom right side of the table. Click on the **Filter** icon to start your search.

The Filter tab will open at the bottom of the Journal Table. You can type in any search criteria you want as well as filter on **Priority** and **Event** types. When you hit the carriage return in the Search field, or change the Priority or Event fields, the alarm list will automatically refresh.

This example shows how to find all **'Tank 100'** event alarms within the given time period on the Date Range component with a priority of **'Medium'** to **'Critical.'** You can increase or decrease the size of the Filter window by dragging the top edge up or down. Click either the **Expand** or **Filter** icons to close the Filter tab window.

On this page ...

- Using the Search Feature

This example searches the alarm list for all **'Motors'** alarms with **'Active'** events and a priority of **'High'** to **'Critical'** within the given time period on the Date Range component.

As you can see, the operator has the ability to search and filter on anything they want by entering the alarm criteria in the Alarm Journal Table filter area.

Reporting in Vision

Included with the Reporting module are many unique Vision components, all of which are added into a new **Reporting** section of the Component Palette. The commonly used Report Viewer allows you to view your reports within a Vision Client. The others, such as the PDF Viewer and Row Selector, do not directly interact with a Report, but instead enhance the capabilities of your Vision Clients.

On this page ...

- Report Viewer
  - Report Viewer Properties
  - Using the Report Viewer
  - Save and Print a Report
- Row Selector
- Column Selector
- PDF Viewer
- File Explorer

INDUCTIVE UNIVERSITY

Reports in Vision Clients

Watch the Video

Report Viewer

The Report Viewer component provides an easy way to view reports in Vision windows. Once you drag a Report Viewer component onto your window, you choose the report you want to view from the **Report Path** property dropdown list. Parameters added during report creation are provided as properties in the Report Viewer, and will override any default values set in the Report Resource. Scroll down to the bottom of the Property Editor and you'll see your report parameters available as properties. The Report Viewer allows you to bind your report parameters to other components on your window.

Report Viewer Properties

While you're in the Property Editor, take a moment to scroll through all the Report Viewer properties. You'll notice that you can customize the visual style of your report by adding page number, change the zoom factor, and the background colors of the report. More information on the Report Viewer's properties can be found in the the Appendix.

Using the Report Viewer

Once you have your reports designed, you're ready to view them in a Vision window using the Report Viewer component. (You need to have a report created before you can view it using the Report Viewer. For information on how to design a report, refer to Report Design.

1. In the **Designer**, drag a **Report Viewer** component to a window.
2. With the Report Viewer selected, go to the **Property Editor** and set the **Report Path** property to display any of your existing reports from the dropdown list.

3. Scroll down to the bottom of the Property Editor, and you'll see **Report Parameters**. This sample report is using a parameter we called "maximum." Any events that are longer than the duration set in the maximum parameter are highlighted.

4. Put the Designer in **Preview Mode** ▶ to see the report as it would appear in the Client.

Save and Print a Report

In **Preview Mode** and from the **Client**, you can right click on the report and print it. You can even save the report to PDF, HTML, or PNG format.

Row Selector

The Row Selector component allows users to filter a dataset based on unique values of one or more columns. Each level in the sorting tree is based on these properties. The user will see a dynamically generated expandable tree that groups their data by any number of choices. As a user clicks down the tree, objects bound to the dataset will indicate the filtered data.

INDUCTIVE UNIVERSITY

Row Selector

Watch the Video

A common way to filter the data is by time. For example, you can feed the Row Selector an input dataset that represents a large time range, and have it break it up by Year, Month, and Day. Then you can power a report with the output dataset that lets the user dynamically create reports for any time range. When configuring the Row Selector for the first time, you'll notice some default Date filters in the Row Selector Tree Customizer to help you quickly configure and filter raw data by time. If you don't want to filter your data by time, then simply delete the default filters and create your own.

See the Appendix for more information on the Row Selector.

A Row Selector has two important properties: **Data In** and **Data Out**. The Row Selector component filters the data in the **Data In** property and pushes the filtered result to the **Data Out** property. Let's configure a Row Selector to filter on some raw data. (This example uses a Power Table component which gives you the option of using some sample test data, or you can create your own data).

1. Drag a Power Table component on to your window and change the **Name** of the Power Table to "**Data In Table**".
2. Scroll down the Property Editor and set the **TestData** property to **'true'**. This will populate the **Data** property with some test data. Alternatively, you could manually populate the **Data** property using either the Dataset Viewer or by creating a binding on the property.
3. Drag a Row Selector component on to your window. With the Row Selector selected, click the **Binding** icon on the Row Selector's **Data In** property.
4. Select the **Property** Binding Type, and bind it to the **Data** property in the **Data In Table**, and click **OK**.

5. Drag another Power Table component on to your window, change the Name to "**Data Out Table"**.
6. With the **Data Table Out** still selected, bind the **Data** property to the Row Selector's **Data Out** property, and click **OK**.

Note:

Don't worry about the 'Unknown' nodes in the Row Selector component. The Row Selector has three default Date filters which haven't been configured for your data yet. You will need to reconfigure them, or delete them and configure your own unique set of filters. Once configured, the 'Unknown' nodes will disappear.

7. Now let's configure your filters. Right click the Row Selector component and scroll down to **Customizers > Row Selector Tree Customizer.**

The Row Selector Tree Customizer provides three default Date filters. Here you can customize the parameters of each Date filter or choose another filter type that is more appropriate for your dataset. Each column on the Data In Table will be listed in the Available Filters tree, and the types of filters available to each column depends on the datatype of the column.

8. To start customizing, select the first filter, then change the **Column Name** to Data Column. (This example uses the **Date** column to filter on the Month, Day, and Time combination.)
9. Change the **Format String** to MMMM yyyy.
10. Click **OK**.

11. Put your Designer into **Preview Mode** ▶.

12. Select Month, Day, or Time to filter on. The filtered results are displayed in the **Data Out** Table. In this example, we filtered on June 20th, so the Data Out Table only contains records that match that date.

Note: When designing your report window, it's not necessary to display the Data In Table only the Data Out Table. You also don't need a component to house the data: the Data In property on the Row Selector could simply retrieve the raw data with a binding.

Here are a few more Row Selector examples:

- A Line Graph bound to a Row Selector - Set up grouping to be first by month and year, then day, then hour, like the example above. Clicking on a month and year will dynamically update the graph for that time period. Further clicking to a specific day or hour will re-filter the graph for that period.
- A Report Viewer bound to a Row Selector - Grouping by department (String) would allow selection by department, automatically regenerating the Report on selection.
- An "Alarm History" Table bound to a Row Selector - This could first be broken down severity level (Integer), then broken into "Alarm Acknowledged" / "Not Acknowledged" (Boolean based). Clicking "Severity 3" would filter the table to all Severity 3 alarms. Selecting "Unacknowledged" would then filter the table to Unacknowledged alarms of Severity 3.

Column Selector

The Column Selector is similar to the Row Selector except that instead of filtering rows, it filters out entire columns from the output dataset. Each column from the input dataset is shown as a checkbox and allows users to show or hide variables in the datasets via the checkboxes, then output the resulting dataset.

The Column Selector allows users to choose which columns in a dataset they wish to use. If an object is bound to the Column Selector it will update itself whenever a user checks or unchecks a column. This allows users to dynamically show or hide Table columns, "pens" on a graph, data in a Report Viewer, or any other component set up to use a dataset.

1. Drag a Power Table component on to your window and change the **Name** of the Power Table to "**Data In Table**".
2. Scroll down the Property Editor and set the **TestData** property to True. This will populate the **Data** property with some test data.
3. Drag a Column Selector component on to your window. With the Column Selector selected, click the **Binding** icon on the Column Selector's **Data In** property.
4. Select the **Property** Binding Type and bind it to the **Data** property in the **Data In Table**, and click **OK**.

5. Drag another Power Table component on to your window, change the Name to "**Data Out Table**".
6. Bind the **Data** of the **Data Out Table** property to the Column Selector **Data Out** property, and click **OK**.

7. Put your Designer into **Preview Mode** ▶. In Preview Mode, you can can hide specific columns by unchecking the boxes in the Column Selector.
8. Use the check boxes to select columns you want displayed or hidden. In this example, we hid the **Boolean** and **Int** columns as shown in the Data Out Table.

9. If you want to customize the column display for all users, put the Designer back into Design mode.
10. Right click on the Column Selector and choose **Customizers > Column Selector Customizer**.
11. Click the **Excluded from Selection** box next to any column that you don't want displayed. The "Excluded from Selection" option determines if the user is allowed to hide the column from the client via the Column Selector.
12. Click **OK**.

13. Now, the columns are excluded from the selection in the Customizer (Int and Boolean columns) and are not displayed in the Column Selector of the Client and prevented from being hidden by the user.

See the Appendix for more information on the Column Selector.

PDF Viewer

The PDF Viewer allows you to take PDF files and embed them into your Ignition Client. This can allows you to view PDF documents without having to leave the Ignition Client, perfect when running a client at full screen, or to view any reports that were saved in PDF format.

File Explorer

The File Explorer component displays a file system tree structure that allows users to navigate around various folders. It can be rooted to any folder including shared network folders, and can filter file types by their file extension like 'pdf.' The File Explorer is typically used in conjunction with the PDF Viewer component in order to create a PDF viewing window. This is very useful for viewing documents from within a project.

INDUCTIVE UNIVERSITY

File Explorer and PDF Viewer

Watch the Video

Let's setup the File Explorer and the PDF Viewer to create a window to view a PDF documents.

1. In Designer, drag a **File Explorer** component and **PDF Viewer** component on to a window and place them side by side.
2. Select the **PDF Viewer** component and click the binding icon next to the **File Path** property.
3. Select the **Property** binding type, and drill down to the **File Explorer Selected Path** property. Click **OK**.

4. Next, select the **File Explorer** component. In the Vision Property Editor, set the **Selected Path** property to a folder path. You can type in a path or bind the root directory by clicking on the binding icon for the **Selected Path** as shown in the image below. The Root Directory restricts which directories are accessible on the component. This is typically used to restrict access to a particular folder so that the user doesn't have access to the entire file system. All folders nested within the Root Directory can be accessed on the component.

> **Note:** If you set the File Explorer's Root Directory to a network folder, all clients will be able to access documents within all folders in that network folder.

5. To filter for only PDF file types, enter **'pdf'** (without quotes) in the **File Extension Filter** property, otherwise, all file types will be displayed inside the Root Directory.

6. In **Preview Mode**, click on one of the reports in your Root Directory. The **PDF Viewer** works by passing the file path of the file you selected to the PDF Viewer. You'll also notice several PDF Viewer properties that drive the appearance of the PDF Viewer: Page Fit Mode, Hide or Show Toolbar, Utility Bar, Highlight, Select, Save, Print, and more.

> **Note:** Clicking on the Save icon in the runtime saves a copy of the report to the **client** computer, not the Ignition Gateway.

File Edit View Project Component Alignment Shape Tools Help
Project Browser
Filter
File Explorer
Label
Label 1
Label 2
Tag Browser
Tag Value Data Type Traits
Tags
System
Vision Client Tags
All Providers
Vision Property Editor
Common
Name File Explorer
Enabled true
Visible true
Border Other Border
Mouseover Text
Cursor Default
Behavior
Data
Selected Path User Manual 8.0 .pdf
Appearance
Font Dialog, Plain, 12
Ignition Documents
Ignition Modules-overview.pdf
Reporting.pdf
Welcome - Ignition User Manual 8.0 .pdf
1 of 3
100%
Welcome
Ignition
by Inductive automation
Welcome to Ignition by Inductive Automation
Page 1 / 3
Welcome
Main Window 9b
Main Window 10
Main Window 11

Vision Client Tags

Client Tags, as the name implies, are only available for use in Vision Clients. Their values are isolated to a Client runtime. All clients will have the same list of client Tags, however, the actual values are unique and independent for each running Client. In other words, even though client Tags are created in the Designer, each client will create their own instances. This makes them very useful as in-project variables for passing information between screens, and between other parts of the clients, such as scripting.

Client Tags support most of the data types that standard Tags do (including datasets), but excluding the array types. Additionally, Client Tags do not have a Tag Group property, so the value will only update when the polling property executes, or a reference in the Client Tag's expression updates.

Often, users will create parameterized windows that take in a line ID or machine name. A Client Tag can store this value for indirection to be used across multiple windows without users on different clients fighting over the current value as they would with a standard Tag.

Note: Client Tags are only available to Vision resources, and are unavailable to Perspective and Gateway scoped resources. If you are using Perspective see the Session Properties page for a similar system.

On this page ...

- Expression Type
- Create a Client Tag
- Using Vision Client Tags
- Overriding Vision Client Tags
- System Client Tags

INDUCTIVE UNIVERSIT

Client Tags

Watch the Video

Expression Type

Client Tags can be configured in one of several ways, based on the **Expression Type** property. The following options are available.

- **None**: Causes the Tag to behave like a Memory Tag.
- **Expression**: Allows the Tag to utilize Ignition's Expression Language, much like an Expression Tag.
- **Query**: Executes a SQL Query, similar to a Query Tag.
- **Named Queries**: Client Tags may call Named Queries

Create a Client Tag

This example shows how to create a client Tag.

1. In the Tag Browser, select Vision Client Tags in the Tag Provider Selector.

2. Click the **Add** icon and select **Client Tag**. The Tag Editor will open.
3. In the **Tag Editor** enter the Tag name and the following general properties, then click **OK.**

   Name: **Area**
   Value: **Processing**
   Data Type: **String**

4. A new Tag called **Area** is created in the **Client** folder.

Using Vision Client Tags

Once a Client Tag is created, it can be used just like any other Tag. You can drag-and-drop, bind to it, use it in scripting, or add it to a Transaction Group.

Overriding Vision Client Tags

When you open a Client, your Vision Client Tags default to the value that was saved in the Designer. If you want to open a Client with different values in your Client Tags, you can override them in the Vision Client Launcher (either for all applications, or an individual application). A Client Tag can be overridden in the **Client Tag Overrides** section of the Vision Client Launcher.

- Use the **Settings** button to override a client Tag in all applications.
- Select **Manage** on the individual application to override a tag on that specific application.

Client Tags can be overridden within a folder. You can also pass client Tag overrides with spaces by using a '+' icon as an escape character. This happens automatically if you are configuring overrides in the Client Launcher.

The following example shows how to override client Tags on a single application.

1. In the **Designer**, create a client Tag inside a folder and bind it to something that can be displayed on a window. This example has two Tags: one Tag is directly under Vision Client Tags folder, and another one under a nested folder. Each **TagA** tag is bound to a different Label components.

2. Open the **Vision Client Launcher**. On your application, click the **More Options** ⋮ icon to open the **Manage** screen.

3. Select the Client Tag Overrides tab.
4. Click the **Add** icon on the right side of the table to add a Tag override. In this example, let's configure two client Tag overrides. The first TagA is for a client Tag directly under Vision Client Tags. Enter the **Tag Name** and **Tag Value.**
5. The second override is for the nested TagA Tag. In the Tag Name, the folder name must precede the Tag name followed by a forward slash. Notice how the value has a **'+'** that denotes a space. The launcher will automatically enter the space for you in the client launcher when you save your updates. Enter the **Tag Name** (including the folder name) and **Tag Value.**

6. Click the **OK** button. This will take you back to your **My Applications** screen.

7. Select your project and click the Open button. You will see the new values that you overrode in the Client Launcher.

System Client Tags

System Tags provide status about the Ignition system, such as memory usage, performance metrics, and so on. Every individual Vision client is going to have its own values like IP address, hosting name, username, and more. You cannot modify System Client Tags. For more information, see System Tags.

Related Topics ...

- Vision Client Launcher

Vision Project Properties

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

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

Alternatively, you can click on the **Project Properties** icon at the top of the Project Browser.

On this page ...

- Vision Design Properties
- Vision General Properties
- Vision Launching Properties
- Vision Login Properties
- Vision Permissions Properties
- Vision Timing Properties
- Vision User Interface Properties

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

Vision Design Properties

This section of properties apply to the Vision Client in general.

Window & Template Committing	
Property	**Description**
Commit on Close	**Prompt** - Prompt whether the user wants to commit changes when closing a window or template. **Always** - Always commit changes when closing a window or template.

Template Auto Commit	**On** - Always commit changes when switching to a different window or template. **Off** - Do not automatically commit changes when switching to a different window or template.
Component Manipulation	
Constrain to Parent Container Bounds	Disabling the constraint on parent bounds allows you to position components outside of their parents bounds, which can be helpful in advanced layouts.
Nudge Distance	The number in this box is the distance (in pixels) that a nudge moves (when using the arrow keys) or resizes a component.
Alt-Nudge Distance	The number in this box is the distance (in pixels) that an alt-nudge moves (when using the arrow keys plus the Alt key) or resizes a component.
Default Component Layout	
Layout Mode	**Relative** - All newly created components will be configured with a Relative layout, further configured by the **Relative Layout Options** property. **Anchored** - All newly created components will be configured with an Anchored layout, further configured by the **Anchored Layout Options** property.
Relative Layout Options	When the **Layout Mode** property is set to Relative, these options determine the layout options of new components. For more information, refer to Component Layout.
Anchored Layout Options	When the **Layout Mode** property is set to **Anchored**, these options determine the layout options of new components. For more information, refer to Component Layout.
Default Color Mapping	The initial color mapping when configuring a new **number-to-color** binding.

Vision General Properties

General	
Property	**Description**

Timezone Behavior	The Vision Client can emulate any timezone. By default, it will appear to be in the same timezone as the **Gateway Timezone**. This has the effect of all Clients behaving the same, regardless of the timezone setting on the Client's host operating system. Depending on your project's requirements, this may not be optimal. You can have the Client use the host's timezone by choosing the **Client Time zone** option, or you may specify any explicit timezone for all Clients to emulate.
Notify / Push	Updates sent to the Client when a designer Saves the projects can either Notify the user by showing an update banner at the top of the Client, or Push the the changes without warning the users. See Client Update Modes.
Touch Screen	
Touch Screen Mode Enabled	All Clients can operate in Touch Screen mode. When enabled, clicking on editable numeric and text entry fields (i.e., Text Fields, Numeric Text Fields, etc.) will pop up on-screen keyboards that can be used for data entry. You can optionally set the width of any scrollbars (number of pixels wide/tall).
Touch Screen Mode Active on Startup	Configures the Clients to start up with the Touch Screen mode active. More details can be found on the Using Touch Screen Mode page.
Data	
Disable Tag History Data Cache	The Clients normally maintain a cache of data retrieved from Tags History, improving repeat operations on graphs and tables. When this option is disabled, no data is cached, and the full queries execute against the Gateway each time data is required.

Vision Launching Properties

These properties affect the Vision Client's launching process.

Gateway Launch Page	
Property	**Description**
Default Launch Mode	Determines the mode for a Client launched from the **Launch** button that appears next to the project in the Client Launcher. Available modes are Windowed or Full Screen.
Windowed Button/Full Screen Button	Each launch mode can also be enabled individually, which allows that mode to appear in the Dropdown list next to the **Launch** button on the Gateway **Home** page.
Hide From Launcher	This option hides the project from the Client Launcher and prevents the project from being selected in the Vision Client Launcher.
Launch Icon	
Launch Icon	The image specified here is used to represent the project on the launch page and desktop shortcut. This needs to be a path to an image that has been uploaded to the Gateway. Use the browse button to choose or upload a new image.
Windowed Properties	
Vendor	This property is displayed as the project is launching through Java Web Start, as well as in the JWS application manager.
Homepage	A URL that is displayed in the JWS application manager.

Width	The width of the Client window when launched in Windowed mode.
Height	The height of the Client window when launched in Windowed mode.
Screen Index	The number here is the index of the screen to use for Full Screen mode, it starts at 0. This property is not supported on all operating systems.
Start Maximized	If the box is selected, when the Client is launched in Windowed mode, it starts maximized. Note that this is not the same thing as Full Screen mode, which is only available when the Client is launched in Full Screen mode. In Full Screen mode, the width, height, and start maximized properties have no effect. When launched in Full Screen mode, the user is given an **Exit** button on the login screen by default. For terminals where the application should not be exited, this button can be removed by checking the **Hide Exit Button** box .
Start Centered	If the box is selected, when the Client is launched in Windowed mode, it starts centered.This property is ignored if **Start Maximized** is enabled.
Hide Exit Button	If the box is selected, when the Client is launched in Full Screen mode, the exit button is hidden to prevent the application from closing.
Client Memory	
Initial	Governs how the Client use RAM resources on its host machine. The initial memory setting is how much memory the Client will require on startup. While this is typically left alone, boosting it a bit can improve performance somewhat.
Maximum	Governs how the Client use RAM resources on its host machine. The maximum memory setting sets a cap on how much memory the Java VM is allowed to use. When you launch a Client on a machine with plenty of RAM, you'll also need to boost this setting to allow the Client to use more RAM.

Vision Login Properties

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

Login Screen	
Property	**Description**
Authentication Strategy	This feature is new in Ignition version **8.1.0** Click here to check out the other new features Authentication strategy setting. Options are Classic or Identity Provider. **Option** / **Description** Classic: User Source system that works as it did before release 8.1. Identity Provider: Identity Provider mode uses the system Identity Provider (IdP) to log into the Designer.

Welcome Message	The message that appears in the upper-left corner of the Login screen. If left blank, no message is displayed. (HTML formatting is allowed).
Welcome Image	The image that appears in the upper-left corner of the Login screen. If left blank, uses a default image. Images are resized/forced to fit into a square format. If you use a more rectangular image, the scaling on the image will automatically be adjusted.
Username Text	The text that appears next to the username field. **Note:** Any value in this field is ignored if the Authentication Strategy is set to Identity Provider.
Password Text	The text that appears next to the password field. **Note:** Any value in this field is ignored if the Authentication Strategy is set to Identity Provider.
Login Button Text	The text that appears on the Login button.
Show Locale Selector	Determines if the Locale Selector should appear on the Login screen. This property interacts with Ignition's Localization system. • **Automatic** - Show the selector if more than a single Languages exists in the project. • **Show** - Always show the selector, regardless of how many Languages exist. • **Hide** - Never show the selector.
Auto Login	
Enable Auto-Login	By enabling auto-login, you can have the launched Client skip the login process. The Client will log in behind the scenes using the cre dentials supplied here. If they fail, the login screen will be presented.

Property	Description
Username	User name of the user to automatically log in when Client is launched.
Password	Password of the user to automatically log in when Client is launched.
Password (repeat)	Password of the user to automatically log in when Client is launched.

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

Vision Permissions Properties

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

Permissions	
Property	**Description**
Enable?	Determines if the Client has access to this category. If unselected, the category will be disabled in the Client for all users. If selected, the **Required Client Roles** text field will also be used to determine if the Client has access.
Required Client Roles	A **comma separated list** of Managing Users and Roles and/or Security Zones that have access to the category while the **Enabled** ch eckbox is selected. While these fields have focus, the **down arrow key** on the keyboard may be pressed to make a list of available roles appear. There are several ways to use roles and zones in this field, and they can be used together in the comma separated list. • **Blank Field** - anyone can use this, regardless of their roles. • **Role name** - users with this role can use this regardless of their zone. • **roleName@zoneName** - users must have this role AND be logged in from this zone. **Caution:** When creating a new project, all of these settings will be disabled by default.
Category Descriptions	The following is a list of the initial categories. Note that the categories you see in your Designer are dependent on which modules are installed on the Gateway. Additionally, third-party modules can add to this list.

Category	Description
Alarm Management	Allows the Client to cancel, shelve, and acknowledge alarms. Applies to both local and remote Alarms.
Datasource Management	Allows the Client to modify Gateway datasource connections.
Device Management	Allows the Client to modify device connections.
DNP3 Management	Allows the Client to freeze DNP3 operations via scripting.
Legacy Database Access	Allows Clients to run queries directly against the database. This doesn't effect named queries.
OPC Server Management	Allows the Client to modify OPC server connections.
SFC Management	Allows the Client to start or stop Sequential Function Charts.

Tag Editing	Allows the Client to add, edit, or delete Tags through scripting.
Translation Management	Allows the Client to modify translations in the localization system.
User Management	Allows the Client to modify schedules, holidays, and users through scripting or components.

Vision Timing Properties

These properties affect the Vision Client polling rate and timeout settings.

Setting Project Polling Base Rate

Watch the Video

Timing	
Property	**Description**
Polling Base Rate	The base rate, in milliseconds, for all polling bindings.
Connect Timeout	The maximum amount of time to wait for connections to the Gateway to be established. Specified in milliseconds.
Read Timeout	The maximum amount of time for socket connection to the Gateway to remain open. Specified in milliseconds.
OPC Browse Timeout	Maximum amount of time, in milliseconds, to wait for the response to a request. (default: 120,000)
Connection Concurrency	By default, Clients are not limited by the number of concurrent connections to the Gateway. These connections are used to send Tag writes, return database results, as well as any other action that requires information to be passed between the Gateway and the Client . Depending on what is running in the Client, your network's bandwidth could be hindered. Enabling this property will limit the amount of concurrent connections the Client can maintain. Note that this may negatively impact Client performance, but is usually preferable on busy networks.

Vision User Interface Properties

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

Setting Client Minimum Size

Watch the Video

User Interface	
Property	**Description**
MInimum Size	Typically, a Vision Client is designed to run on multiple different resolution and sizes of monitors. The various component layout features help design elastic screens, but sometimes you need to set a lower bound as to how small you'll allow the Client's usable area to shrink. This is what the Minimum Size settings are for. You can see these settings visually represented in the Designer as lines on the Vision workspace when the Root Container is smaller than the configured Minimum Size.. Whenever the usable space shrinks smaller than these bounds, scrollbars will appear, capping the **Width** and **Height** to these minimums. This defaults to 800 x 600. In the image below, the project was set to a minimum size of 400 x 300. Since the window is smaller, the outline is visible.
Client Background Color	This option allows you to specify the color of the Vision workspace which will be visible when not obscured by windows.
Client Menu	These options change the appearance of the Client Menu:

Client Menu	
Property	**Description**

	Menu Font	Changes the font type, font style, and font size of the Menu Bar.
	Hide Menu Bar	Hides the entire Menu Bar in the Client. Usually enabled in situations where users should not be able to close the client. Changes to this setting are applied on client startup, meaning clients will need to be relaunched after changing this setting.
	Hide Windows Menu	Hides the automatically-generated "Windows" menu that lets users switch between open windows. Enabled when users should not be able to close windows in the Client. See also Menubar Scripts.
Touch Screen Keyboard Width	Determines how wide the Touch Screen Keyboard should appear in the Client. Percentage of the client window the touch screen keyboard displays.	
Docking	Docked windows are locked to the edges of the Client and fill all the space on that edge. Docking options are:	

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

Related Topics ...

- Client Update Modes
- Using Touch Screen Mode
- Working with Components
- Docked Windows - Axis Precedence

In This Section ...

Client Update Modes

Notify Versus Push

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

- **Notify**
  This is the default mode to automatically **notify** operators when project updates are available. In this mode, every time you update a project in the Designer and save the changes, Clients will display an orange information bar at the top of their display. This orange bar notifies the operator that an update is available. By clicking on the notification banner, the new project modifications are downloaded and applied.

- **Push**
  When you save your changes in the Designer, this mode automatically pushes all project changes and updates to all running clients with no operator interaction, that is, the new version is downloaded and applied automatically. This is often desirable when a Client is running in a situation where keyword and mouse access is inconvenient, such as in a large overhead display.

On this page ...

- Notify Versus Push
  - Setting Client Update Modes

About Client Updates

Watch the Video

Note: This property (and many other project properties) are only read when a client is launched. When you change from Notify to Push Mode, client updates will not get pushed when you save, clients need to be re-launched to start using the Push Update Mode.

Setting Client Update Modes

This example shows how to change the Client Update Mode.

1. In the Project Browser of the Designer, select **Project > Properties** from the top menubar. The **Project Properties** window will open.
2. Under the **Vision** folder, select **General**, set either **Notify** or **Push** as your Client Update Mode. The default is set to Notify.
3. Click **OK.**

Project Properties
Project
General
Permissions
Designer
Vision
Design
General
Launching
Login
Permissions
Timing
User Interface
Vision / General
Timezone Behavior
Gateway Timezone
Notify
Push
Touch Screen
Touch Screen Mode Enabled
15
Scrollbar Width (pixels)
Touch Screen Mode Active on Startup
Data
Disable Tag History Data Cache
OK
Apply
Cancel

Now, in the Client (with Update Mode set to **Notify),** the operator will have to click the banner to update the Client.

Then the operator clicks **Update** on the confirmation message.

Now the client version is updated with the most recent version of the application.

Related Topics ...

- Using Touch Screen Mode
- Vision Project Properties

Setting Up Auto Login

Client Auto Login

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

Set the Clients to Automatically Login

In this example, we will set the project properties to allow Ignition clients to automatically login.

1. In Designer, go to **Project > Properties**.
   The **Project Properties** window is displayed.
2. Scroll down to **Vision > Login**.
3. Click the **Enable Auto-Login** checkbox.
4. Enter a Username and Password.
5. Click **OK.**

6. Select the **File > Save All** to save your changes to the Gateway and your project.

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

On this page ...

- Client Auto Login
  - Set the Clients to Automatically Login

INDUCTIVE UNIVERSIT

Setting up Client Auto Login

Watch the Video

Related Topics ...

- Gateway Backup and Restore

Using Touch Screen Mode

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

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

Simply click in the Username field and the Touch Screen keyboard will appear. Enter your Username and click enter, do the same for your Password, and then click Login.

On this page ...

- Enabling Touch Screen Mode
- Invoke the Touch Screen Keyboard with Scripting
- Change the Size of the Touch Screen Keyboard

INDUCTIVE UNIVERSITY

Using Touchscreen Mode

Watch the Video

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

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

Enabling Touch Screen Mode

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

1. From the Project Browser, click on the Project Properties icon, or from the menubar go to **Project > Project Properties**. The **Project Properties** window is displayed.

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

3. Change the settings as desired then click **OK**.

> **Note:** Make sure Comm Read/Write is enabled in Project Properties of the Designer.

These settings are helpful for mixed-use projects, that is, those that are launched on both touch screen devices and traditional computers and laptops. Once Touch Screen mode is enabled through Project Properties, the Touch Screen icon will appear on top right corner of the project login screen. Click in any of the login fields to bring up the Touch Screen keyboard.

Invoke the Touch Screen Keyboard with Scripting

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

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

See also system.gui.setTouchscreenModeEnabled.

Change the Size of the Touch Screen Keyboard

You can control the size of the Touch Screen keyboard that is displayed in a Vision client.

1. In the Designer, go to **Project Properties > Vision > User Interface**.
2. Change the percentage for the Touch Screen Keyboard Width property to anything you want.

3. When you launch a Vision client, the Touch Screen keyboard will be displayed in the default percentage of 75%. After you're logged into the client, the Touch Screen keyboard will be set at the value you entered into the Touch Screen Keyboard Width property field.

Alphanumeric Keyboard

The Touch Screen Keyboard Width setting controls both the alphanumeric keyboard and the numeric keypad.

Numeric Keypad

Related Topics ...

- Setting Up Auto Login

Common Tasks in Vision

This section contains examples for items we've identified as "common" tasks: undertakings that many users are looking to utilize when first starting out with a specific module or feature in Ignition . Additionally, this section aims to demystify 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 Ig nition. While they 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 ...

- Component Animation
- Custom Input Template
- Client Tags for Indirection
- High Performance HMI Techniques
- Open Dynamic Windows on Startup
- Tank Cutaways

Component Animation

Creating animation within a Vision project. The Component Animation page walks through the different ways to animate graphics on a window. The different methods vary in complexity and offer different solutions for different needs.

Custom Input Template

Creating a simple template that can be used many times to create simple user input fields. The Custom Input Template is relatively simple to put together, but can be a powerful tool that can quickly build out screens that are heavy on user input.

Client Tags for Indirection

Vision Client Tags can be used as a variable across all windows to indirectly point to a set of Tags, such as an area of the plant. For example, setting up Client Tags for Indirection can enable the user to choose an area of the plant from a Dropdown List component, and have Ignition display the correct windows for that area.

High Performance HMI Techniques

Using techniques that relay information more quickly than standard a P&ID display. High Performance HMI Techniques help create simpler screens that have much less noise and useless "fluff" that takes up screen space. The suggestions and examples help guide better practices to increase efficiency and performance across any industry.

Open Dynamic Windows on Startup

It can sometimes be necessary to Open Dynamic Windows on Startup, so that certain users see one set of windows while other users with potentially different privileges sees a different set. While it is easy to set a window to open on startup, that same window will always open for anybody that logs in to the project. By using scripting with the Client Event Startup Script, you can customize which windows get opened based on any criteria.

Tank Cutaways

The Tank Cutaways in Symbol Factory may seem like just a random shape, but they can be used to remove parts of other SVG diagrams to create a way to "look inside" the tank or other machine and view information like how much material is available.

In This Section ...

Component Animation

Animation can be a useful tool to help visualize what is happening at any given time. Animations make it easy to tell if a machine is running, or a conveyor is moving with a quick glance at the screen, and it can help highlight which components aren't currently in use. There are two main ways of creating animation: actually moving all or part of a component, or cycling through a few different static images of a component that give the illusion of moving. Each method has its advantages. For example, making a dump truck move forward and backwards may be better accomplished with actually moving the component. It would be fairly simple to make the whole component move forward on its x coordinate, and then back the same amount. But, if the truck was instead stationary, and just its bed were moving up and down to simulate the truck dumping its cargo, it may be difficult to seamlessly move the component, since it would not just be moving on one axis, but would require careful rotation and movement combined. In this instance, it would be far easier to create a few static images of the truck with its bed in various states, and then cycle through those images fairly quickly so it looks like it is moving. Let's go over both methods below.

On this page ...

- Animation in Scripting
- Components that Actually Move
- Giving the Illusion of Moving

Animation in Scripting

The best way of moving components around on the screen is using the scripting function system.gui.transform. This function can move and resize components from a Python script. It provides many options for moving and resizing components, all in one simple scripting function. This can be called on a property change when a user enters a new value, or based on other conditions that happen on the window.

Component Animation

Watch the Video

Components that Actually Move

To actually move components around during runtime usually involves binding all or part of an image or SVG to the value property of a timer or signal generator. The Symbol Factory Enhanced components all contain an Angle property. Binding the Angle property of a spinning part of a motor can help us visualize when the motor is running or not.

Note:

When searching Symbol Factory, make sure to select the Enhanced radio button in the search window. The enhanced symbols have groupings that enable you to more easily animate them

1. Pick out a component that you would like to animate. For this example, we used the **Single Stage Compressor** from Symbol Factory. You will also want to drag a **Timer** component onto your window.

1. In the Project Browser, click the **Expand** icon to expand the Single Stage Compressor component. You will notice that the whole component is made up of many smaller pieces. We want to select the piece called Group_Impeller.

2. In the Property Editor, bind the **Group_Impeller's Angle** property to the **Value** property of the **Timer**.

3. We now need to modify the properties of the Timer to ensure a good rotation. Select the **Timer** component, and set
   Delay (ms): **200**
   Step By: **10**
   Bound: **360**
   Running?: **True**

4. Put the Designer in **Preview Mode** and your Compressor symbol component will now animate. You can adjust the Delay (ms) to be lower or higher to adjust the speed at which the component rotates.

Giving the Illusion of Moving

With this method, you can make components look like they are moving without actually moving anything. What we do here is duplicate the component multiple times and modify each component to be a little different than the others, and then show and hide them in the correct order to make it seem like they move.

The following example uses a Signal Generator component to drive the animation, but any incrementing value can be used, such as the Value property on a Timer component, an accumulating value in a PLC, or the current time in seconds.

1. Pick out a component to animate. For this example, we chose the **Horizontal Conveyor with Perspective** from the Symbol Factory. We also placed a signal generator symbol on the window.

2. Duplicate this component 10 times for a total of 11 conveyors on the window.

3. Select the first instance of the component then choose **Component > Ungroup**.
4. In the Project Browser, expand the Group_Conveyor_Belt. Select **Path 5**.

5. Move **Path 5** to the left and then move **Path 4** the same distance to the left. In this example, we moved them both an equal distance left so that the leftmost component is on the left edge of the conveyor.

6. Repeat this with the rest of the components, except move them slightly to the right of the previous instance. Notice how the gray bars slowly move from left to right over all the components when viewed in sequence.

7. Select each Conveyor image individually select **Component > Group**.
8. Next, select the first Conveyor from the Project Browser (the one that has the bars on the far left), and place an expression binding on its **Visible** property that looks like this:

```
if({Root Container.Signal Generator.value} = 0, 1, 0)
```

Duplicated this across all of the conveyors, but increment the first number by **1** each time. The last conveyor (the one that has the bars on the far right) should have the following expression:

```
if({Root Container.Signal Generator.value} = 10, 1, 0)
```

9. Repeat **Step 8** across all of the conveyors, but increment the first number by **1** each time. The last conveyor (the one that has the bars on the far right) should have the following expression:

```
if({Root Container.Signal Generator.value} = 10, 1, 0)
```

10. Next, stack all of the conveyer images them all on top of each other exactly. This is easily done by selecting all of the conveyors, going into the Alignment menu, then selecting **Align Centers Horizontal** icon and **Align Centers Vertical** icon.
11. Select the **Signal Generator** component. Set the **Signal Type** to **Ramp**, the Period to **1000**, the Values/Period to **11**, the **Upper Bound** to **11** , and the **Lower Bound** to **0**.
12. Set the **Running?** property of the Signal Generator to **True** and it will now cycle through showing all of the conveyors, which will make it look like it is moving.
13. **Save** the project.

Custom Input Template

Sometimes you may need to have text fields in your project for the user to input data. Rather than copying and pasting these text fields into each window, you can create a template that includes a single label and text field. Parameters can then be passed in so that the label and text can be used for different types of input. The template can then contain an expression to validate that there is data in the text field. This template can be reused many times on multiple windows to allow users to input data.

On this page ...

- Custom Input Template Example

Custom Input Template Example

In this example, we'll create a template containing a Label and Text Field components. We'll add two parameters to be passed into the components. We will also copy the expression in the code block below to let the user know that there is data in the text field. Once you get the template created, you can copy the template to a window and test it out.

1. In the Project Browser, right-click **Templates** and select **New Template**.
2. Right-click on ***New Template*** and click **Rename** to change its name it to something meaningful such as **Text Entry**.
3. To add a parameter, right-click the checkered area and select **Customizers > Custom Properties**.
4. In the **Custom Properties** window, add two **Template Parameters** by clicking the Add icon twice and entering the following:

   1st Parameter
   Name: **display**
   Type: **String**
   2nd Parameter
   Name: **text**
   Type: **String**

5. Click **OK.**
6. From the Component Palette, drag a **Text Field** and a **Label** to the checkered area of the template, resizing so that the label and the text box occupy the majority of the space.

7. Select the **Label** component, go to the **Property Editor**, and click on the **Binding** icon of the **Text** property. The Property Binding window is displayed.
8. In the **Property Binding** window, click on the **Property** binding type, choose the **display** propert y. Make sure the **Bidirectional** box is not selected, and click **OK.** Don't worry if the label on the template disappears. It is simply displaying the value of the display custom property, which is currently blank.

9. Now select the **Text Field** component, go to **Property Editor**, and click on the **Binding** icon of the **Text** property.
10. In the **Property Binding** window, click on the **Property** binding type, choose the **text** property. Make sure the **Bidirectional** box is selected, and click **OK**.

11. Next, make the background color of the Text Field change depending on whether the user entered a text value or not. While the **Text Field** component is still selected, go to **Property Editor**, and click on the binding icon of the **Background** property.
12. In the **Property Binding** window, click on the **Expression** binding type, and copy and paste in the following expression.

Expression - Input Validation

```
// Inside the if statement is the len and the trim functions that
are available when doing expression bindings.
// The trim function will trim the blank spaces from the text
therefore validating that there is actual text rather than spaces.
// The len function will count the length of the recently trimmed
text.
// The if statements asks the question: Is the length of the trimmed
text greater than zero?


if(len(trim({textBox.Text Field.text}))>0,color(255,255,255),color
(255,0,0))
```

13. Click **OK**.
14. Close the template, click **Save** on the top menubar, and click **Save** again.
15. To test it out, drag a few of the **Text Entry** templates from **Project Browser** to your window. Notice that the text field is red because it is expecting some data.

16. In the **Property Editor**, go to the display custom property and enter something like **Name**. Do something similar for the other templates. Notice how the label gets filled in with the data we entered.

17. Put the Designer into **Preview Mode**, and enter in some values into the text fields. Notice that the background color changed to white, and the information that was entered is now in the **text** p roperty that can be bidirectionally bound so that it writes to a Tag or used in a script.

Client Tags for Indirection

Vision Client Tags can be used as a variable across all windows to indirectly point to a set of Tags, such as an area of the plant. Suppose that a window has a Dropdown List component that allows a user to select different areas of the plant. If the drop-down is bidirectionally bound to a Client Tag's selected string value property, the user can change the drop-down's value, therefore resulting in the indirect binding throughout the project reflecting the change in the Client Tag.

The Client Tag can be bound to a custom property on a window's Root Container. This Custom property can be bound to the Client Tag. The components inside the window can have indirect bindings on their properties that leverage the Custom property on the window. Therefore, the components on the window will be dependent on the Client Tag. Client Tags are managed inside the Client, therefore each Client will be independent of other Clients.

On this page ...

- Client Tag Indirection Example
  - Testing Your Work

INDUCTIVE UNIVERSITY

Using Client Tags for Indirection

Watch the Video

Client Tag Indirection Example

In this example, we will setup Client Tags for indirection which can be shared between different windows in our project.

1. We need two windows and a way to navigate between them. If you don't already have a project that meets this criteria, create a new project using the **Vision Tab Nav** project template.

2. Add a Client Tag. In the Tag Browser, right click on **Vision Client Tags**, then choose **New Tag > Client Tag**.

3. Name the Tag **Machine Number** and set the **Value** to **1**.

4. Add a window to show data based on the Client Tag. Create a new window named **Machine Overview**.
5. Next, we'll add a Custom Property to the Root Container of the window.
   a. In the Project Browser, select the **Root Container** for the Machine Overview window.
   b. Right click on the desktop and choose **Customizers > Custom Properties.**
   c. Click the **Add** ✚ icon.
   d. Enter **MachineNumberRef** as the name, and make sure the Type is **Integer**. Click **OK** to save the new custom property.

6. Next we'll bind the custom property to the Client Tag. In the Project Browser, select the Root Container for the Machine Overview window.
    a. In the Vision Property Editor, click on the **Binding** icon next to the **MachineNumberRef** custom property.
    b. Choose the **Tag** binding type.
    c. Select the **Machine Number** Vision Client Tag.
    d. Click **OK** to save the binding.

7. Drag a **Label** component onto the window. Next we'll add an Expression binding to show the page title.
    a. In the Vision Property Editor, click on the **Binding** icon next to the **Text** property.
    b. Choose the **Expression** binding type. Enter the following expression:

```
"Machine "+{Root Container.MachineNumberRef}+" Overview"
```

c. Click **OK** to save the binding.

8. Next we'll set up an LED Display component for the value.
   a. Drag an **LED Display** component onto the window.
   b. Click on the **Binding** icon next to the **Value** property.
   c. Select the **Indirect Tag** binding type.
   d. Click on the **Tag** icon. Navigate to a Tag you want to use. We used the **Sine1** tag from the simulator.
   e. Click **OK** to save the Tag path.

f. Next, click the **Insert Property Value** icon.
g. Choose the **MachineNumberRef** property from the root container. Click **OK**.

h. In the Indirect Tag Path field, delete the "1" before the {1}.

i. Click **OK** to save the binding.
j. **Save** your project.

9. Next we'll make a details screen so we can switch between the two. In the Project Browser, right click on the **Machine Overview** window and select Duplicate.

10. **Rename** the copy as "Machine Details."
11. On the Machine Details window, select the **Label** component.
12. Click on the **Binding** icon next to the **Text** property. Change the word **"Overview"** to **"Details**." Edit the Expression binding as follows:

```
"Machine "+{Root Container.MachineNumberRef}+" Details"
```

13. On the LED Display, update the Indirect Tag binding on the **Value** property to point to the **Ramp** Tag instead of the Sine Tags.
    a. Select a **Ramp** tag from the Tag selector and click **OK**.

    b. Click the Insert **Property Value** icon. Choose the **MachineNumberRef** property from the root container and click **OK**.

c. In the Indirect Tag Path field, delete the "1" before the {1}. Click **OK** to save the binding.

14. Next we'll add the two new windows to the Navigation. Open the Navigation window.
    a. Right click on the **Tab Strip** component and select Tab Strip Customizer.

b. In the Tab Strip Customizer, click on the **Empty** tab. Change the Window name and the display name to **"Machine Overview**."
c. Click the **Add Tab** button. Change the Window name and the display name to "**Machine Details**."
d. Click **OK** to save the changes to the Tab Strip.

15. Add a way to change the Client Tag value in the Client. There are two ways to do this:
    a. Drag a **Numeric Text Field** component onto the Navigation window, and then drag the **Machine Number** Client Tag onto the **Numeric Text Field** component.
    b. If your Header doesn't have a background component, you can just simply drag the **Machine Number Tag** directly onto the window and select **Control > Numeric Text Field**.

c.

16. **Save** the project.

Testing Your Work

Remember that the point of a Client Tag is to have different values across multiple Clients, so there is no way to test it in the Designer. To test the functionality of this example, do the following:

1. Open two Vision Clients.
2. In the first Client, change the Client Tag Value from "1" to "2" in the header.
3. Click between the Machine Overview and Machine Details screen. Notice that they are both showing the Tags for Machine 2.
4. Switch to the second Client. Notice that the Machine Overview and Machine Details screens still show the Tags for Machine 1.

High Performance HMI Techniques

About High Performance HMI Techniques

High performance HMI techniques and practices call for designs and displays which help the viewer make the best decision in the shortest amount of time after interacting with the HMI.

High performance HMIs often look basic and simplistic. They typically use gray-scale colors rather than the traditional graphics and bright colors for their displays. Conceptually, the High Performance HMI operates under the idea of visually contrasting critical and non-critical states. The power of this design philosophy is when something does go wrong, a high performance HMI will quickly guide the user to the source of the problem.

Here is a comparison of a traditional HMI next to a high performance HMI.

On this page ...

- About High Performance HMI Techniques
  - Traditional HMI
  - High Performance HMI
- Use of Color
  - Colors and Alarm Indicators
  - Technical Considerations with Colors
  - Accommodating Color Blind Viewers
- Alarm Indicator
  - Reducing Ambiguity

Traditional HMI

High Performance HMI

In Ignition, you create the high performance HMIs by using components such as moving analog indicators, sparkline charts, and radar charts.

Moving Analog Indicator

Sparkline Chart

Radar Chart

Use of Color

Colors are an important consideration when designing a high performance HMI. Gray-scale colors are used instead of the traditional bright red, green, and blue colors.

The person that designs the HMI must understand the diverse audience that may view the HMI. For example, an operator may look at a motor on a traditional HMI. If it the HMI has colored the motor green, the operator may conclude that the motor is running. However, if a maintenance technician looks at the same motor, he may conclude that the motor is not faulted. These are logical conclusions. They reflect the respective interest of the person viewing the HMI where the operator wants the motor to run and the maintenance technician wants it to be working.

In reality, the motor is simply "scheduled" to run, meaning when it runs depends on its control mechanism. For example, the motor may run when there is product on the line or boxes on the conveyor. In other words, the motor may be periodically starting and stopping automatically.

A high performance HMI can eliminate this confusion by introducing a color that signifies a state of "scheduled." A common high performance HMI practice is to use a dark gray to signify a "scheduled" state for equipment. This color should never compete with more alerting colors that "pop" from the HMI resulting in the viewer's eye's being drawn to the portion of the HMI where the problem may be occurring.

INDUCTIVE UNIVERSITY

Use of Color

Watch the Video

Colors and Alarm Indicators

Color can play an important role in how an operator responds when problems do occur. High performance HMI design refrains from coloring equipment when the equipment is in a state of fault. For example, some equipment may still run when faulted. Instead, an optimized solution is to place an alarm indicator near the equipment in such a way that when the equipment is undergoing some fault, the alarm indicator renders with the appropriate color and shape. The object and the consequential color should signify the most important alarm state occurring for the equipment at that current time.

Technical Considerations with Colors

Some HMIs in industrial settings may temporarily lose their ability to render color because of various environmental factors. High performance HMI design incorporates this possibility by encouraging the use of descriptive text with color. For example, motors of two different colors may look the same on a color deficient HMI resulting in confusion for the viewer. Even worse, the viewer may misinterpret the motor state and assume everything is fine. However, if each motor has a descriptive text such as "Motor 1 is Faulted" and "Motor 2 is Running", the problem associated with a faulty HMI failing to display color is largely reduced by the HMI's high performance design.

Accommodating Color Blind Viewers

Common color combinations such as red and green and blue and purple cannot adequately be distinguished by those with color blindness. High performance HMI design accommodates color blind uses by combining colors with descriptive text as well as incorporating alarm indicators in unique shapes.

Alarm Indicator

The high performance HMI design techniques make use of an object called the **Alarm Indicator** which displays a colored shape when there is a problem. This works well with the high performance HMIs as color is only used when there is a problem. The Alarm Indicator can contain descriptive text in addition to the shape and color, and is usually placed near the component that is causing the problem. You can import the Alarm Indicator, shown in the example below, from Ignition Exchange.

The Alarm Indicator represents different levels of alarm with different shapes, color, and descriptive text. For example, a motor that exists in an industrial setting is monitored by a high performance HMI. There are two alarms on the motor. The first is a critical alarm associated with the motor becoming seized, will display as a red rectangle with the letter 1. The second is a high priority alarm associated with the motor when overheating, will display as a yellow triangle with the number 2. An Alarm Indicator is placed near the motor, positioned in such a way to clearly show motor that the indicator is referring to.

Given this scenario, a high performance HMI will show the Alarm Indicator as in the following examples:

Example 1

The motor is critically faulted. This is the highest priority.

Example 2

Motor is overheating resulting in a high alarm.

Reducing Ambiguity

A high performance HMI design technique to reduce ambiguity incorporates a line from the Alarm Indicator to the object experiencing the alarm. The following example shows how a simple dotted line can ensure that the viewer associates the correct Alarm Indicator with the correct motor.

Open Dynamic Windows on Startup

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

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

Code Snippet - Client Startup Role Check

```
# Checks the users role and opens a main window depending on the role

# Grabs a list of the users roles
roles = system.security.getRoles()

# Checks if they have Administrator role
if 'Administrator' in roles:
        system.nav.openWindow('Administrator Screen')

# Checks if they have Operator role
elif 'Operator' in roles:
        system.nav.openWindow('Operator Screen')

# If they have neither Administrator or Operator
else:
        system.nav.openWindow('Welcome Screen')
```

INDUCTIVE UNIVERSIT

Open Dynamic Window(s) on Startup

Watch the Video

Client Event Scripts
Client Event Scripts
Startup
Shutdown
Shutdown-Intercept
Keystroke
Timer
Tag Change
Menubar
Message
Client Startup Script
Project startup script that runs in each Client
# Checks the users role and opens a main window depending on the role
# Grabs a list of the users roles
roles = system.security.getRoles()
# Checks if they have Administrator role
if 'Administrator' in roles:
system.nav.openWindow('Administrator Screen')
# Checks if they have Operator role
elif 'Operator' in roles:
system.nav.openWindow('Operator Screen')
# If they have neither Administrator or Operator
else:
system.nav.openWindow('Welcome Screen')
OK
Apply
Cancel

Tank Cutaway

The Symbol Factory images in the Basic > Tank Cutaways category work well when combined with the other symbols, especially tanks from the Tank Category. Use the following technique to make a dynamic cutaway tank display:

1. From your Symbol Factory SVG symbols, drag a **Tank** and a **Cutaway** symbol onto the window. (We used *tank 3* and *jagged cut-away 2*.)
2. Align the cutaway symbol on the tank where you'd like the cutaway to be placed.

3. Select the tank symbol **first**, and then select the cutaway while holding **CTRL** to select both symbols.
4. Click the **Difference** icon to use the cutaway symbol to make it appear that the area of the tank is cut away.

5. Place a **Level Indicator** component (drag from the Component Palette) on the area removed by the cutaway.

6. With the Level Indicator selected, in the Property Editor, enter a value for the **Value** property, or use a binding to put a value on the **Level Indicator**.

7. Choose **Alignment > Move Back** to put the Level Indicator behind the tank.

8. This is an optional step, but you can select the tank, including all the graphics, right click and Group them so now they can move around as one unit. You can even make a template out of it, so you can use it multiple times.

Related Topics ...

- Symbol Factory

Dropdown List Example

Dropdown lists are used when you want to select a single item from a list of options. The Dropdown component is under the Input section of the component palette. Simply drag it on to your window. The most important property of a Dropdown component is the Data property. It is a Dataset that contains one or more rows of data. Each of the rows are different options that you see on the component. Select the Dropdown component and click on the **Dataset Viewer** icon for the Data property to manually add some options.

On this page ...

- Data Property's Dataset Modes
  - Number/Label Pair
  - Single Label Column
  - Code/Label Pair
- Setting Dropdown Options
- Displaying Multiple Columns in a Downdown List

INDUCTIVE UNIVERSIT

Dropdown

Watch the Video

Data Property's Dataset Modes

There are three modes you can use the **Data** property's dataset: a number/label pair, a single label column, and a code/label pair. Which mode is used depends on what columns are in the **Data** property's dataset, and will determine the values of the **Selected Value**, **Selected String Value** and **S elected Label** properties. Any additional columns that are added to the dataset will not affect these properties.

Number/Label Pair

In the Number/Label Pair mode, the first column of the Dataset is an integer (often an id) and the second column is a string (often a label). The first column is the **Value** column which is invisible to the user and is usually used in binding. The second column is the **Label** column and is visible to the user in the dropdown list.

In this example, the dataset has two rows. Under the **Value** column, you see the integers **'1'** and **'2'** respectively. Under the **Label** column, you see the **'Realtime'** and **'Historical'** options.

In **Preview Mode**, select from the list of dropdown options. Notice, you can only see the options in the **Label** column. Select the **'Realtime'** option. Y ou can see under the Data property section, the **Selected Value** is **'1'** which is the first (integer) column in the dataset and is not visible to users. The **Selected String Value** and the **Selected Label** are the same and show the **'Realtime'** option which is the second (string) column in the dataset.

Single Label Column

The Single Label Column mode you can use is a string/string combination which are two columns that are both strings.

1. In the Dataset Editor, remove the **Value** column from our example by selecting a cell in that column and clicking on the Vertical Delete ic on.
2. Add another column by clicking on the Vertical Add icon.
3. Call the new column **'Code,'** set the column position to **'0'** and make it a **'String.'** Click **OK**. Both columns are now strings.

4. Under the **'Code'** column enter **'v1'** and **'v2.'** Click **OK.**

Note: Because we added the 'Code' column in position '0,' the users will see the same two options in the dropdown list: 'Realtime' and 'Historical.' If you see 'v1' and 'v2' instead, that means your 'Code' column is not the first column.

5. In Preview Mode, click on **'Historical'** and you can see the Selected String Value is **'v2'** because the value is in the first column and the Selected Label is**'Historical.'**

Code/Label Pair

The Code/Label Pair mode simply uses a one string column.

1. In the Dataset Editor, remove the **Code** column from our example by selecting a cell in that column and clicking on the **Vertical Delete** ic on.
2. You are going to see the same two options in the Dataset Viewer: **'Realtime' and 'Historical.'**

> **Note:** This only applies if there is exactly one column in this dataset.

3. In **Preview Mode**, select the **'Realtime'** option. You will see the same value in the **Selected Label** and **Selected String Value** properties since there is only the one column.

Setting Dropdown Options

Now, you can set these dropdown options manually or bind the Data property. In this example, you can take a Dropdown List on the window and bind the Data property using a SQL query.

1. Select the Dropdown component, and click on the **binding** icon for the Data property.
2. Select the **SQL Query Binding Type** and enter a query that brings back an ID and Name from one of your tables in the database.

3. Click **OK** to see all the options that came back from the database.
4. In Preview Mode, select any one of these options and you can see the **Selected Value**, **Selected String Value**, and **Selected Label** as shown in the following example.

Displaying Multiple Columns in a Downdown List

Another feature of the Dropdown List component is you can show more than one column to a user.

1. Drag another Dropdown component to your window.
2. Go to the **Dataset Viewer** and add some options manually. Instead of having only one or two columns you can add as many as you want. The first column needs to be either an **integer** or a **string**. Any additional columns will show up in the dropdown.
    a. Open the Dataset View, under **Value**, enter **'1'** and **'2.'**
    b. Under **Label**, enter '**Realtime**' and '**Historical**.'
    c. Add another column and call it '**Col3**' in '**Position 2'** , and make it a **string** and click **Add Column.**
    d. Under Col3, enter **'This is real'** and **'This is historical'**.
    e. Click **OK**.

3. In the Property Editor, set the **Dropdown Display Mode** to **'Table'** to see both columns.

4. Now you can see **Label** and **Col3** and select between those different rows. It's a nice way to show more information in the Dropdown list. The **Selected Value**, **Selected String Value**, and **Selected Label** properties will behave the same as previous examples ignoring any columns beyond the second.

Project Browser
Filter
Project Properties
Dropdown 2
Navigation
Root Container
Templates
Named Queries
Reports
Vision Property Editor
Appearance
Font
Dialog, Plain, 12
Foreground Color
46,46,46
Background Color
250,250,251
Selection Background
71,169,230
Dropdown Display Mode
Table
Max Row Count
8
Realtime
Label
Col3
Realtime
This is real time
Historical
This is historical

Multi-Monitor Clients

Multiple Desktops

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

Client Tags and Property Values

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

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

Handles

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

On this page ...

- Multiple Desktops
  - Client Tags and Property Values
  - Handles
  - Spotting the Primary Desktop
  - Project Updates
- Opening Another Desktop
  - Navigating Windows in Desktops
- Opening a Desktop on Each Monitor

INDUCTIVE UNIVERSITY

Multi-Monitor Clients

Watch the Video

Spotting the Primary Desktop

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

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

Project Updates

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

Opening Another Desktop

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

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

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

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

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

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

Navigating Windows in Desktops

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

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

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

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

Opening a Desktop on Each Monitor

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

```
# Get the screen information for all of your monitors.
screensDataset = system.gui.getScreens()

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

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

        # Open an empty frame on the next monitor.
        # Assign a handle and apply the width/height for the monitor you are opening on
        handleName = "Monitor %d" %monitorNum
```

```
system.gui.openDesktop(screen=screenIndex, handle=handleName, width=screenWidth, height=screenHeight)

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

Related Topics ...

- system.gui.openDesktop

Local Client Fallback

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

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

Note: In order to use local client fallback, **port 6501** must be open on the local machine.

On this page ...

- Enable Fallback
- Test Local Fallback
- Automatically Transfer Back

Enable Fallback

To enable local Vision client fallback, do the following steps:

1. Go to Config > Gateway Settings in the local Gateway.
2. Scroll down to the Local Vision Client Fallback section and select **Enable Local Fallback**.
3. Enter the name of the Fallback Project.

   Note: The selected project must be published in the local Gateway, and it must have at least one main window.

4. Optionally, you can change the Seconds to Failover setting to a value other than 60 seconds.

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

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

Test Local Fallback

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

Automatically Transfer Back

Local Vision Client Fallback will not automatically transfer back to the main Gateway when it is running again, as simplicity was key in this system. You can, however, provide your own solution to automatically switch back. One example is to add a retarget script to your Timer Client Event Scripts to silently try to reconnect.

```
# add this to a Client Timer Script running every 30 seconds
# change the ipaddress and project names to match your system
```

```
# allow the main copy of Ignition 10 seconds to give a response
status = system.util.getGatewayStatus("ipaddress:8088/main", 10000)
# if it's running again, retarget
if status == "RUNNING":
    system.util.retarget("my_project", "ipaddress:8088")
```

Client Launchers Commands

Command Line/Terminal

Clients can be launched from the Client Launcher via command/terminal. When called in this way, many of the application properties may be overridden for the one call. The overrides use the same property names as noted in the Application Property Reference Table, under the "JSON name" column.

```
Windows:
"C:\ClientLauncher\visionclientlauncher.exe" application=myproject window.
mode=window

Linux:
./visionclientlauncher.sh application=myterminal window.mode=fullscreen
screen=0
```

There are a few important notes when using the Command Line/Terminal to launch a project from the Vision Client Launcher.

- The Vision Client Launcher must be installed and have an application added for the Command Line/Terminal commands to work.
- The **application** argument requires the application name in the Launcher, not the project name. You can open the launcher to determine what the application name is. Adding new applications in the launcher uses the Project title by default.
- Applications may contain spaces in their name. However, when launching from command /terminal, spaces should be escaped with **%20**. For example, if our application was named **my project**, then we could all it with the following:

```
"C:\ClientLauncher\visionclientlauncher.exe" application=my%20project
```

Command Line Arguments

Argument	Description
`application`	The name of the application to launch.
`window.mode`	Controls the client mode. Available options are: • `window` : Launches the client in Windowed Mode • `fullscreen` : Launches the client in Fullscreen Mode
`screen`	The screen index indicates which monitor to use.
`fallback.application`	The name of the application to use if the number of retries has been exceeded. The fallback is only utilized if the **Retries** setting is greater than 0.
`timeout`	The maximum number of seconds to allow for any gateway communication. Any communication that exceeds this amount will cause the Vision client launcher to abort the communication and try again if configured.
`retries`	How many times to attempt to contact a gateway again if an error occurred during communication. Available values are: • -1 : Retry indefinitely, or until the launcher is manually closed. • 0 : Zero retries, or abort after the first failure. • 1 (or more): Determines the number of retries: i.e., a value of "5" means five retries. If the number of retries is exceeded, then the launcher will attempt to launch the Fallback Application. If the number of retries is exceeded, then the Launcher will attempt to launch the Fallback Application
`init.heap`	The initial heap size (memory) for the client.
`max.heap`	

On this page ...

- Command Line/Terminal
  - Command Line Arguments

	The maximum heap size (memory) for the client

`-Djavaws.launchparams`	Defines client tags that can be overwritten upon launch. The use of this argument alone only defines the client tags that will be overwritten. Setting a value on the tags can be done by an additional argument that utilizes the tag names delimited by a semicolon:
	
	`// Establishes the tag names`
	` -Djavaws.launchparams="Tag1;Tag2"`
	
	`// Sets values on the tags`
	`-Djavaws.launchparams.Tag1=10`
	`-Djavaws.launchparams.Tag2=20`
	
	`//An actual call would look like:`
	`"C:\ClientLauncher\visionclientlauncher.exe" application=myproject -Djavaws.launchparams="Tag1;Tag2" -Djavaws.launchparam.Tag1=10 -Djavaws.launchparam.Tag2=20`

`config.json`	Allows you to point the launcher to a launcher configuration file command line. Doing so will start running an instance of the launcher using the configurations in the file as temporary overrides. The argument expects a path to a JSON export file, specifically the same file that created by the that's created by the **Export Launcher Config** button under the launcher's **Settings** menu. `"C:\ClientLauncher\visionclientlauncher.exe" config.json="C:\Users\MyUser\Desktop\vision-client-launcher.json"`

Related Topics ...

- Vision Client Tags

Deploying Vision Client Launchers

Deploying Launchers

In some cases, it may be desirable to deploy a pre-configured Launcher to remote clients. Doing so bypasses the installation process on the remote machine. The process to do so is outlined below. The steps in this example are operating system dependent, as the launchers were designed to run on a single operating system.

Three Key Pieces to a Configured Launcher

- The launcher itself.
- The JRE for the launcher. The "jre" directory should be in the same directory as the launcher.
- The configuration JSON file. The launcher always expects this file to be located at a certain directory, otherwise the launcher will create a new, blank, file.

The steps below demonstrate how to configure a launcher, and then migrate each piece to a new file system into an appropriate location.

On this page ...

- Deploying Launchers
  - Three Key Pieces to a Configured Launcher
  - Pre-Configure a Launcher
  - Create a ZIP
  - Deploying the Launcher
- Redundancy
- Command Line/Terminal
  - Command Line Arguments
- Launchers and SSL
  - Certificates Signed by a CA
  - Self-Signed Certificates
- Troubleshooting

Pre-Configure a Launcher

1. Install a Vision Client Launcher somewhere on network. Take note of the Installation directory you use, as a later step will involve finding this directory.
2. Configure each application. Be mindful of the Gateway Address setting: the deployed launcher will need to be able find the Gateway based on the address you enter.

At this point, we should have a single launcher configured with one or more applications. Next we can zip up the the launcher and its configuration files.

Create a ZIP

In these steps, we will create a single ZIP, containing all of the required files.

1. Navigate to the Launcher's installation directory. It should look similar to the image below:

Name	Date modified	Type	Size
jre	4/26/2019 3:49 PM	File folder	
clientlauncher.log	4/26/2019 3:49 PM	LOG File	1 KB
unins000.dat	4/26/2019 3:49 PM	DAT File	20 KB
unins000.exe	4/26/2019 3:48 PM	Application	820 KB
visionclientlauncher.exe	4/22/2019 2:36 AM	Application	21,043 KB

2. Copy all of the files shown in Step 1, and paste them into the Client Launcher Data directory. This is the same directory that the configuration JSON file is located. The path differs by operating system, but should generally look like the following:

```
C:\Users\userName\.ignition\clientlauncher-data\
```

Linux

```
/home/user/.ignition/clientlauncher-data/
```

After pasting, the directory should look something like the following:

Name	Date modified	Type	Size
certificates	3/4/2019 12:28 PM	File folder	
jre	4/26/2019 2:39 PM	File folder	
clientlauncher.log	4/26/2019 3:49 PM	LOG File	1 KB
designer-launcher.json	4/26/2019 11:11 AM	JSON File	3 KB
designerlauncher.log	4/26/2019 11:11 AM	LOG File	86 KB
launchclient.jar	4/24/2019 10:17 AM	Executable Jar File	3,565 KB
launchclient.jar.chk	4/24/2019 10:17 AM	Recovered File Fra...	1 KB
unins000.dat	4/26/2019 3:49 PM	DAT File	20 KB
unins000.exe	4/26/2019 3:48 PM	Application	820 KB
visionclientlauncher.exe	4/22/2019 2:36 AM	Application	21,043 KB
vision-client-launcher.json	4/26/2019 3:53 PM	JSON File	4 KB
visionclientlauncher.log	4/26/2019 2:30 PM	LOG File	35 KB
VisionIcon.ico	4/25/2019 2:54 PM	Icon	306 KB

We now have a single folder containing the launcher, a JRE, and the application configurations.

3. Navigate up a directory to the ".ignition" folder. You will see the folder we were just in: "clientlauncher-data"

4. In this directory (the **.ignition** directory), create a new folder called **.ignition**. Note the leading period. In some operating systems you may have to attempt to rename with a trailing period as well.

```
.ignition.
```

5. Copy the **clientlauncher-data** folder into the new **.ignition** folder. Note that you want to copy, not move. Moving would make prior application configurations on the local system unavailable.
6. Zip up the newer **.ignition** directory.

We now have a ZIP we can deploy to a new system.

Deploying the Launcher

1. Take the ZIP, and move it to another computer.
2. Unzip the file at the user directory. The location of the directory is based on the local operating system.

Windows
`C:\Users\userName`

Linux
`/home/user/`

3. The files should be unzipped. Navigate to the launcher at:

```
{user folder}/.ignition/clientlauncher-data/visionclientlauncher
```

4. Run the Launcher. It should contain all of the configured applications.
5. At this point, consider creating a shortcut to the launcher, and placing the shortcut at a more accessible location. Alternatively, the launcher can be accessed via command line/terminal, so you could automate client launchers in a number of ways.

Redundancy

The client launcher can take advantage of a redundant Gateway setup. Whenever a connection is established with a master Gateway, the backup Gateway IP address is automatically stored in the client launcher configuration file. If the master Gateway cannot be contacted the next time the client launcher is run, an attempt is made to contact the backup Gateway. If the backup cannot be contacted, the client launcher switches between contacting the primary Gateway and the backup Gateway until one responds or the user closes the launcher.

Command Line/Terminal

Clients can be launched from the Client Launcher via command/terminal. When called in this way, many of the application properties may be overridden for the one call. The overrides use the same property names as noted in the Application Property Reference Table, under the "JSON name" column.

```
Windows:
"C:\ClientLauncher\visionclientlauncher.exe" application=myproject window.mode=window

Linux:
./visionclientlauncher.sh application=myterminal window.mode=fullscreen screen=0
```

There are a few important notes when using the Command Line/Terminal to launch a project from the Vision Client Launcher.

- The Vision Client Launcher must be installed and have an application added for the Command Line/Terminal commands to work.

- The **application** argument requires the application name in the Launcher, not the project name. You can open the launcher to determine what the application name is. Adding new applications in the launcher uses the Project title by default.
- Applications may contain spaces in their name. However, when launching from command/terminal, spaces should be escaped with **%20**. For example, if our application was named **my project**, then we could all it with the following:

```
"C:\ClientLauncher\visionclientlauncher.exe" application=my%20project
```

Command Line Arguments

Argument	Description
`application`	The name of the application to launch.
`window.mode`	Controls the client mode. Available options are: • `window` : Launches the client in Windowed Mode • `fullscreen` : Launches the client in Fullscreen Mode
`screen`	The screen index indicates which monitor to use.
`fallback.application`	The name of the application to use if the number of retries has been exceeded. The fallback is only utilized if the **Retries** setting is greater than 0.
`timeout`	The maximum number of seconds to allow for any gateway communication. Any communication that exceeds this amount will cause the Vision client launcher to abort the communication and try again if configured.
`retries`	How many times to attempt to contact a gateway again if an error occurred during communication. Available values are: • -1 : Retry indefinitely, or until the launcher is manually closed. • 0 : Zero retries, or abort after the first failure. • 1 (or more): Determines the number of retries: i.e., a value of "5" means five retries. If the number of retries is exceeded, then the launcher will attempt to launch the Fallback Application. If the number of retries is exceeded, then the Launcher will attempt to launch the Fallback Application
`init.heap`	The initial heap size (memory) for the client.
`max.heap`	The maximum heap size (memory) for the client

`-Djavaws.launchparams`	Defines client tags that can be overwritten upon launch. The use of this argument alone only defines the client tags that will be overwritten. Setting a value on the tags can be done by an additional argument that utilizes the tag names delimited by a semicolon:
	`// Establishes the tag names`
	` -Djavaws.launchparams="Tag1;Tag2"`
	
	`// Sets values on the tags`
	`-Djavaws.launchparams.Tag1=10`
	`-Djavaws.launchparams.Tag2=20`
	
	`//An actual call would look like:`
	`"C:\ClientLauncher\visionclientlauncher.exe" application=myproject -Djavaws.launchparams="Tag1;Tag2" -Djavaws.launchparam.Tag1=10 -Djavaws.launchparam.Tag2=20`

config.json	Allows you to point the launcher to a launcher configuration file command line. Doing so will start running an instance of the launcher using the configurations in the file as temporary overrides. The argument expects a path to a JSON export file, specifically the same file that created by the that's created by the **Export Launcher Config** button under the launcher's **Settings** menu. `"C:\ClientLauncher\visionclientlauncher.exe" config.json="C:\Users\MyUser\Desktop\vision-client-launcher.json"`

Launchers and SSL

When SSL is enabled on a Gateway, the Vision Client Launcher can take advantage of the enhanced security features associated with SSL.

Certificates Signed by a CA

When the Gateway's SSL certificates are signed by a recognized Certificate Authority, no additional configuration is required on the launcher.

Self-Signed Certificates

When using a self-signed SSL certificate, the certificates for the Gateway must be locally accessible to the client launcher in the following directory:

```
{user folder}\.ignition\clientlauncher-data\certificates
```

When a certificate is placed in the directory above, the launcher will attempt to automatically add the certificate(s) to the local keystore upon application launch.

Troubleshooting

If the Vision Client Launcher fails to launch a client for some reason, the log file can be found at:

```
{user folder}\.ignition\clientlauncher-data\visionclientlauncher.log
```

This log contains any errors that occurred.

Related Topics ...

- Vision Client Tags

Vision Client Launcher Settings

You have the option of customizing the Vision Client Launcher Settings. This can be accomplished with either the GUI or by modifying a file on the local file system. Regardless of method, both utilize the properties specified under the Vision Client Launcher Settings Reference Table.

On this page ...

- Launcher Settings
  - Launcher Configuration Using JSON
  - Vision Client Launcher Settings Reference Table
  - Trusted Certificates
  - Delete a Certificate
  - Export a Certificate
- Launcher Application Settings
  - Application Property Reference Table

Launcher Settings

You can find the **Settings** button in the upper right corner of the Vision Client Launcher.

You can choose to use the default settings or customize your client applications. There are three different **Action** buttons in **Settings** in the Vision Client Launcher.

- **Export Launcher Config** - Exports a configuration of the the Vision Client Launcher, as a JSON file (i.e., `vision-client-launcher.json`). Taking an export is useful in cases where you want multiple launcher instances to use the same settings. The content of the export file is based on the last saved configurations settings of the launcher.
- **Import Launcher Config** - Imports your launcher configurations from a Vision Client Launcher export file.
- **Import Application Config** - Creates a new client application configuration from an export, adding a new application to the launcher. The import file is the same generated by an application export.

When finished updating any of the client launcher settings, click the **Save Changes** button. Refer to the list of additional launcher settings in the reference table below.

Launcher Configuration Using JSON

In addition to using the GUI, these configurations changes can be made by modifying the `vision-client-launcher.json` file (created during installation of the launcher). The file is located at:

```
{user folder}\.ignition\clientlauncher-data\vision-client-launcher.json
```

The preferred method of configuration is with the GUI, but JSON modification is useful when attempting to load multiple application configurations simultaneously.

A reference to the various properties that can be modified via JSON are listed in the Vision Client Launcher Settings Reference Table. However, the JSON file has the following setting that isn't available from the GUI:

lock.configuration	When set to true, the application list and individual application configurations cannot be modified. Effectively restricting the ability to add applications and edit the launcher's configurations.

Vision Client Launcher Settings Reference Table

Launcher Settings - Configurations for the Launcher		
Property Setting	**Description**	**JSON Name**
Default Application	When set, the Launcher will attempt to automatically launch the specified Application when running. If left blank /null, an Application will not automatically start	`default. application`
Logging Level	Level of logging that will be used for the launcher: INFO, TRACE, DEBUG, WARN, and ERROR.	`logger. level`
Multicast Address	The address that will be used to listen for multicast broadcasts from Gateways.	`multicast. address`
Multicast Ports	The port that will be used to listen for multicast broadcasts from Gateways.	`multicast. receive. port`
Auto Exit on Launch	Automatically close the launcher window when a client is started.	`autoexit`
Application Defaults - Default Configurations for New Applications		
Property Setting	**Description**	**JSON Name**
Timeout	Maximum number of seconds allowed for any Gateway communication. Any communication that exceeds this amount will cause the client launcher to abort the communication and try again if configured.	`timeout`
Retries	The number of times a Client will to attempt to contact a gateway again if an error occurred during communication. Available values are: • -1 : Retry indefinitely, or until the launcher is manually closed. • 0 : Zero retries, or abort after the first failure. • 1 (or more): Determines the number of retries: i.e., a value of "5" means five retries.	`retries`
Initial Heap	Amount of memory to be allocated at startup.	`init.heap`
Max Heap	Maximum amount of memory allocation for heap size.	`max.heap`
JVM	Arguments to append to client startup. These should start with a '-D' and be separated by semi-colons.	`jvm. arguments`
Client Tag Overrides	Client Tags and their values that are set when the client launches.	`client.tag. overrrides`

Trusted Certificates

You can add security certificates to the launcher.

While attempting to launch an application, the launcher will check against the application's host Gateway, and attempt to automatically download any required certificates. In addition, certificates can be manually added using the example below.

1. Access Vision Client Launcher Settings by clicking **Settings** on the top right of the Vision Client Launcher window.
2. In the Trusted Certificates section, click the **Add** icon.

3. Navigate to the location of the certificate on your system and choose **Open**.
4. Information about the certificate is displayed. To view complete details, click the **Expand** ▶ icon next to Details. If everything looks OK, click **Trust Certificate**.

5. Once a certificate is imported, you can view its details by clicking the **Information** icon.

Delete a Certificate

1. To delete an imported certificate, access Vision Client Launcher Settings by clicking **Settings** on the top right of any Vision Client Launcher window.
2. In the Trusted Certificates section, select the certificate you want to delete.
3. Click the **Delete** icon.

4. You'll get a confirmation window. If the certificate is the correct one you want to delete, click the **Delete** button.

Export a Certificate

1. To export an imported certificate, access Vision Client Launcher Settings by clicking **Settings** on the top right of any Vision Client Launcher window.
2. In the Trusted Certificates section, select the certificate you want to export.
3. Click the **Export** icon.
4. Navigate to the location you want to save the certificate and click **Save**.

Launcher Application Settings

For each project application, you can update the default settings by clicking the **More** icon (three vertical dots) next to the project name, and select **Manage** to open the Manage Project Settings window. The **Export** command exports the configuration settings to a `.json` file. The **Delete** command deletes the project application from the launcher.

You can choose to use the default settings or customize your project client applications. There are three different **Action** buttons under **Manage Project Settings** in the Vision Client Launcher.

- **Launch Application** - Launches the your application in the Vision Client Launcher.
- **Export Config** - Exports your updates to the Vision Client Launcher file (i.e., `vision-client-launcher.json`). The resulting export file contains the last **saved** configuration of the application.
- **Create Desktop Shortcut** - Creates a shortcut to your application on your desktop, using the last **saved** application configuration.

When finished updating any of the launcher settings, click the **Save Changes** button.

Application Property Reference Table

Setting Name	Description	JSON Name
Application Name	The descriptive name of the application. This is independent of the project name that corresponds to a project on the gateway. Instead, this is the name of the application as it is defined in the Launcher.	`application`
Project	The Vision project that exists on the gateway that this application will load.	`project`
Description	An optional brief note about the application. The description will appear in the Launcher. If left blank, a description will not appear.	`description`

Window Mode	Controls the client mode. Available options are: • `window` : Launches the client in Windowed Mode • `fullscreen` : Launches the client in Fullscreen Mode	`window.mode`
Screen Index	The screen index indicates which monitor to use.	`screen`
Image Path	This will allow the icon of the application to be set within the designer and it will be downloaded and displayed as the icon for the application as well as the shortcuts. If no path is set the default icon is used. If this is set to a different icon path, that icon will be used.	`image.path`
Fallback Application	The name of the application to use if the number of retries has been exceeded. The fallback is only utilized if the **Retries** setting is greater than 0.	`fallback.application`
Timeout	The maximum number of seconds to allow for any gateway communication. Any communication that exceeds this amount will cause the Vision client launcher to abort the communication and try again if configured.	`timeout`
Retries	How many times to attempt to contact a gateway again if an error occurred during communication. Available values are: • -1 : Retry indefinitely, or until the launcher is manually closed. • 0 : Zero retries, or abort after the first failure. • 1 (or more): Determines the number of retries: i.e., a value of "5" means five retries. If the number of retries is exceeded, then the launcher will attempt to launch the Fallback Application. If the number of retries is exceeded, then the Launcher will attempt to launch the Fallback Application.	`retries`
Init Heap	The initial heap size (memory) for the client.	`init.heap`
Max Heap	The maximum heap size (memory) for the client	`max.heap`
Client Tag Overrides	Allows client tags to be overridden on the Vision Client. These are set using the table on the application configuration screen by adding rows with the names of the tags and the corresponding values. When applying the overrides from command line, the Tags must first be defined as launch parameters, then a value can be set on the parameters. Vision Client Tag overrides within folders and passing client tag overrides are supported by using a '+' icon as an escape character. This happens automatically if you are configuring overrides in the Client Launcher. To learn more about how to configure Client Tag Overrides, refer to Overriding Vision Client Tags.	`client.tag.overrides`

Related Topics ...

- Vision Client Tags

Perspective

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

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

The following are some key features of the Perspective module:

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

On this page ...

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

INDUCTIVE UNIVERSIT

Perspective Project Elements

Watch the Video

Views and Containers

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

INDUCTIVE UNIVERSIT

Anatomy of a View

Watch the Video

Components

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

The Perspective Workstation is a single application that acts as both a launcher, and desktop “wrapper” for Perspective Sessions. Perspective Workstation allows you to run your Perspective applications as first class desktop applications designed for HMI's, panel PCs and dedicated workstations.

Perspective Workstation has built-in features to run in Windowed mode and full screen Kiosk mode eliminating any distractions from the underlying operating system. There are also features to manage applications running on dedicated workstations with multiple monitors configured to run on specific monitors.

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

Perspective Workstation

The Perspective Workstation is a single application that acts as both a launcher, and a desktop "wrapper" for Perspective Sessions. Perspective Workstation allows you to run Perspective applications as first class desktop applications designed for HMI's, panel PCs and dedicated workstations. Workstation supports launching across multiple monitors. A single application configured in Perspective Workstation can open multiple application windows.

Perspective Workstation has built-in features to run in Windowed mode and full screen Kiosk mode eliminating any distractions from the underlying operating system. There are also features to manage applications running on dedicated workstations with multiple monitors configured to run on specific monitors. When launching into an application that has multi-monitor configured, and authentication is required, the “primary” screen (determined when configured) shows a login screen, while the other application windows show an intermediary screen until the user logs in.

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

Perspective Symbols

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

Bindings

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

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

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

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 s ecurity (TLS).

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

For more information, see Security in Perspective.

PingID

okta

DUO

ADFS

Related Topics ...

- Working with Perspective Components
- Containers in Perspective
- Pages in Perspective
- Perspective Workstation

In This Section ...

Perspective Sessions

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

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

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

On this page ...

- 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
- Session Disconnections

INDUCTIVE UNIVERSITY

Perspective Project Elements

Watch the Video

Session Anatomy

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

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

Component

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

Container

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

View

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

Page

A Page represents a collection of views that are displayed in a single space such as a browser window or tab, and is the main navigation element in a session. Each page consists of a primary view, but multiple views can be defined from within a single page. Some views can be docked to specific

edges of the page, or “popups” floating on top of the other views. Multiple pages can be open as part of the same session, but can show different information. A URL is defined for a page, which means the Forward and Back buttons in a web browser can be used to navigate to pages that have already been visited.

Properties

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

A Simple Example

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

Launching a Perspective Session

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

Watch the Video

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 **Launch 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.

INDUCTIVE UNIVERSITY

Session App Bar

Watch the Video

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

You can change the location of the Maximize/Minimize icon on the app bar in the Property Editor. Scroll down the list of properties 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.

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.

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

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

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

Session Disconnections

If a Perspective Session loses its connection to the Ignition Gateway, a message like the one shown below will be immediately displayed on the upper right-hand corner of the Session:

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

Related Topics ...

- Pages in Perspective

In This Section ...

Ignition Perspective App

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

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.

On this page ...

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

INDUCTIVE UNIVERSITY

Perspective App

Watch the Video

Download the App

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

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

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

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

Added in version .96, the Ignition Perspective App can now be configured via iOS and Android enterprise services (as defined by the AppConfig Community), 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. However, 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.
hide_demo	If true, the built-in demo project will become hidden.
initial_applications	Allows you to add projects to the main app page. Each application has the following keys: **Key** — **Description**

<table>
<tr><td></td><td>
<table>
<tr><td>project_url</td><td>A URL leading directly to the application, for example:
<code>http://www.someurl.com:8088/data/perspective/client/myproj</code>

<code>OR</code>

<code>perspective://www.someurl.com:8088/myproj</code></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:
<code>http://www.someurl.com:8088</code></td></tr>
</table>
</td></tr>
</table>

Session Properties

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

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

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

On this page ...

- Restrictions from System Properties
- Session Properties Table

INDUCTIVE UNIVERSITY

Session Props

Watch the Video

Restrictions from System Properties

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

Session Properties Table

Name	Description
id	Unique Session identifier.

host	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 Session.
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	Property Type
authenticated	True if the user is authenticated. False if the user is unauthenticated. Null if the user's authentication status is unknown.	value: boolean
user	Contains information about the user, if they are authenticated.	object

Name	Description	Property Type
id	The IdP's unique identifier for this user. Null if the user is not authenticated.	value: string
userName	The user's username. Null if the user is not authenticated.	value: string
firstName	The user's first name. Null if the user is not authenticated, if the IdP did not provide this attribute, or if no mapping was configured for this attribute.	value: string
lastName	The user's last name. Null if the user is not authenticated, if the IdP did not provide this attribute, or if no mapping was configured for this attribute.	value: string
email	The user's email address. Null if the user is not authenticated, if the IdP did not provide this attribute, or if no mapping was configured for this attribute.	value: string
roles	The roles that the IdP assigned this user. Null if the user is not authenticated, if the IdP did not provide this attribute, or if no mapping was configured for this attribute.	value: string
timestamp	A timestamp representing the last time the current user authenticated against the Identity Provider.	value: timestamp

Name	Description	Property Type
securityLevels	The deepest security levels in the tree granted to the current user, starting with the children of the Public security level. The Public security level is never shown since all Sessions include Public.	array

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

Name	Description	Property Type
idpId	The identity provider's ID. Replaced by the `idp` property	value: string
idp	The name of the Identity Provider configuration set on the project.	value: string
idpAttributes	Represents the JSON object returned by the identity provider after logging in. The structure of this object will match that of the JSON provided by the Test Login Identity Provider page. **Note:** The Designer does not authenticate against identity providers in Ignition 8.0, so this object will always appear empty in the designer. Use the Test Login page to determine the shape of this property, or use a simple binding to something visual (i.e. a label) display and parse the results while developing your project.	object

gateway	

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

Name	Description	Property Type
address	Remote host address of the connected Gateway.	value: string
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	object
connected	True when connected to a websocket. All tabs in the Session must have a connected websocket to become true. If a Session disconnects but is still open in the browser, the property will not change, as the property write from the Gateway can't write to the disconnected Session.	value: boolean

device

Properties for the device that is running the Session.

Name	Description	Property Type
type	Type of device that created this Session. Read only. Options are ios, android, designer, browser, and workstation. Empty string if device is unknown during loading.	value: string
identifier	Unique ID representing this device. This is a convenience property not intended/suited for security purposes. May change via device/application re-installs or browser cache clears.	value: string
timezone	Document providing time zone information. **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	object
userAgent	User agent string of the connected device.	value: string
settings	Array of settings for the device. **Name** / **Description** / **Property Type** preventSleep — Prevents the device from sleeping while viewing project in the mobile Perspective App. Default is false. — value: boolean	object
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	object

bluetooth

Options and data provided by device Bluetooth services.

Name	Description	Property Type

enabled	If true, enables bluetooth capability.	value: boolean
options	Bluetooth options.	object

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

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

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

Name	Description	Property Type
eddystone	Eddystone open beacon format.	object

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

Name	Description	Property Type
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	value: string

									must be specified in order to receive iBeacon data.			

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

geolocation

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

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

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: numeric

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

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.

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

symbols

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

Settings relevant to the components on the Perspective Symbols Palette.

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

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 View. Every view and container in Perspective has an associated layout, simply put, it is a way of defining and describing the way that elements inside the container interact.

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

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

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

Related Topics ...

- 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 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* configuratio n 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:

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.

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** obje ct 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 ...

- Containers in Perspective
- 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)

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.

INDUCTIVE UNIVERSITY

Creating Pages

Watch the Video

Page Configuration

All page configuration happens on the Perspective Page Settings tab. Any number of pages can be created for a project each with their own Primary View and Docked Views. When configuring a docked view in Page Configuration, you can choose to dock the view on a specific Page URL with a set primary view, or you can choose to share the view across all pages under Shared Settings. Each project can configure a **Shared Settings** that will be “inherited” by all pages in that project, 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 **Perspective** 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: **Option** — **Description** visible — The docked view is always expanded/displayed. onDemand — The docked view is collapsed, but allows the user to display the view by clicking on the docked view's handle. auto — Automatically shows or hides the docked view depending on how much space is available in the session: showing the view if the page is wider than the width specified in the auto-breakpoint setting. (Works in conjunction with the Auto Breakpoint property).
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. **Option** — **Description** push — Opening or closing the docked view causes the content in the center to resize: the center view will be 'pushed' out of the way. cover — When opening the dock, it slides in front of the center view, obscuring part of the center view: the dock will 'cover' part of the center view. auto — Acts like the cover option when the viewport is smaller than the Auto Breakpoint value. Acts like the push option when the viewport is larger than the Auto Breakpoint value.
Anchor	Allows you to make a view always visible while scrolling. Only available on North docked view configurations.

INDUCTIVE UNIVERSITY

Docked View Properties

Watch the Video

	<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.

INDUCTIVE UNIVERSITY

Docked View Parameters

Watch the Video

4. You will see a popup on which you can 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.

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. 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 **Perspective** 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).

This feature is new in Ignition version **8.1.1**
Click here to check out the other new features

If a page configuration is using a view that does not exist, a **Warning** icon is displayed next to the page configuration and next to the Primary View setting.

Configuring Docked Views

You have the option of making a docked view shared across all pages of our 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.

Note: 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 **Edit** icon 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.

Note: 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. 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 TopNave **Edit** icon 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.

Note: 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: MyProject.

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

Related Topics ...

- Views in Perspective
- Containers in Perspective
- Navigation Strategies in Perspective

Views in Perspective

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

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

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

On this page ...

- View Properties
- Input/Output Parameters
- Configuring Views

INDUCTIVE UNIVERSITY

Anatomy of a View

Watch the Video

View Properties

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

Each view contains exactly one "root" level container, which may be any of the available container types: 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.

Configuring Views

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

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

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

d. Click **Create View**.

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.

> **Note:** 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.

Related Topics ...

- Containers in Perspective
- Pages in Perspective

In This Section ...

Embedded Views

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

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

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.

INDUCTIVE UNIVERSITY

Embedded View

Watch the Video

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

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

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

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

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

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

Related Topics ...

- Containers in Perspective
- Pages in Perspective

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

INDUCTIVE UNIVERSITY

Popup Views

Watch the Video

Title Bars on Popups

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

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

Configuring a Popup View

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

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.

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

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

4. 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 ...

- Containers in Perspective
- Pages in Perspective

Containers in Perspective

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.

On this page ...

- Container Types
- Breakpoints
- Nesting Layouts Using Containers
- Wrapping Existing Components into a New Container

Container Types

The following table describes each of the container types in Perspective.

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. 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 use 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.

Wrapping Existing Components into a New Container

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

Perspective features a one-step wrapping option. The Wrap In Container action, will "wrap" a component or group of components into a new container within the existing container. This feature provides design flexibility in that you can introduce additional containers to an existing app design in Perspective.

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

The following is the process going on "behind the scenes" in the Designer.

1. Create a new container of the type you picked
2. Copy the component you clicked on's "position" props into this new container
3. Cut the component you clicked on
4. Replace the component you clicked on with the new container
5. Paste your component into the new container

In the following example we have a view with two Vessel components and a Time Series chart. We've decided that the Time chart needs to be in a separate container.

1. Right click on the Time Series Chart.
2. Select Wrap in
3. Select the type of container you want, in this case we chose a Coordinate container.

4. The Designer creates the Coordinate Container and moves the Time Series Chart Component inside. The component is deep selected.

For more information on options on the right-click menu, see Working with Perspective Components, Right-Click Menu.

Related Topics ...

- Pages in Perspective
- Views in Perspective

In This Section ...

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

INDUCTIVE UNIVERSITY

Coordinate Container

Watch the Video

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.

city	country	population
Folsom	United States	77,271
Helsinki	Finland	635,591
Jakarta	Indonesia	10,187,595
Madrid	Spain	3,233,527
Prague	Czech Republic	1,241,664
San Diego	United States	1,406,630
San Francisco	United States	884,363
Shanghai	China	24,153,000
Tokyo	Japan	13,617,000
Washington, DC	United States	658,893
Wellington	New Zealand	405,000

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

A Column 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 component does not scale with the height of the component 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

INDUCTIVE UNIVERSIT

Column Container

Watch the Video

Easy Screen Flexibility

A Column container can be used a simple way to set up responsive design. Responsive design is 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 container 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: numeric</td></tr><tr><td>order</td><td>Where component is places among its siblings within its row. Ordering is independent per row.</td><td>value: numeric</td></tr></table>	array

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.

> **Note:** 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 **menuType** 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 Navigation
- Nested Tabs

INDUCTIVE UNIVERSIT

Tab Container

Watch the Video

Tab Navigation

The Tab container 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, payment, and more. 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

Breakpoint Container

Watch the Video

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 **Perspective** 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 value 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

INDUCTIVE UNIVERSIT

Flex Container

Watch the Video

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 pixels. 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 pixels.

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, depending c 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, depending 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 width v

	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 will ac

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
alignConte nt	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, Project Browser, and Tag Browser. The dockable panels can be rearranged as you wish and will snap into place as you move them around the screen. Each workspace remembers its layout, which is the docking arrangement of the panels around it.

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

INDUCTIVE UNIVERSITY

The Designer User Interface

Watch the Video

Perspective Menubar

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

File Menu

See General Designer Interface.

Edit Menu

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

Function	Description

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

View Menu

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

Function	Description
Reset Panels	Resets the Panels in the Designer so that they are in the default configuration.
Panels	A list of all the available Panels for the Designer in Perspective. Select the checkbox next to the Panel name to display that panel.
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.
	Zoom into the currently open window.
	Zoom out of the currently open window.
	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 remainder of the icons on the Perspective toolbar are for aligning components. See aligning for more information.

Resource Tabs

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

Right-Click Menu

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

Option	Description
Close & Revert	Closes the selected view, and reverts any changes that have been made, but not yet saved. This option should only be used when you wish to lose any changes made to the view.
Close	Closes the selected view, and commits changes (meaning changes will be retained in the designer session).
Close Others	Closes all views except for the view selected. Changes made to the views are committed.
Close All	Closes all views. Changes made to the views are committed.

Project Browser

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

Badge Icons

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

Badge Icon	Name	Description

	Binding	Indicates the component has a binding 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 Tag Browser.

Perspective Component Palette

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

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

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

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

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

Recently Modified Views

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

Keyboard Shortcuts

There are a lot of ways to speed up development once you are familiar with how Ignition works. There are many keyboard shortcuts in Perspective Des igner 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 Perspective Designer toolbar also has options for aligning components.

Related Topics ...

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

Working with Perspective Components

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

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

On this page ...

- Component Categories
- Component Properties
  - Variants
  - List View vs Tile View
- Adding Components
- Selecting Components
  - Mouse Selection
  - Tree Selection
  - Deep Selection
- Right-Click Menu
- Manipulating Components
  - Moving
  - Resizing
  - Aligning
  - Alignment Guides
  - Alignment Guides Example
  - Rotating
- Image Source
  - Web Address
  - Image Management

Component Categories

Components are separated into the following categories in Perspective:

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

Component Properties

Properties on Perspective components are separated into categories.

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

INDUCTIVE UNIVERSIT

Component Overview

Watch the Video

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

Variants

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

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

List View vs Tile View

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

Adding Components

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

Selecting Components

Mouse Selection

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

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

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

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

Wrap in Container

This next option on the menu is Wrap In Container. This option takes the selected component(s) and puts them into a container that you specify. This option is covered in detail here on the Containers in Perspective page.

Configure Events

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

Configure Scripts

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

Manipulating Components

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

Moving

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

Resizing

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

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

Aligning

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

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

Icon	Function	Description
	Align Left	Align the left edges of a group of components.
	Align Right	Align the right edges of a group of components.

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

Align and Normalize Example

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

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

2. Enter a padding distance in pixels. We use 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. The Rotate property is located on Position Properties section of the Perspective Property Editor. For a definition of all the position properties, see 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.

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

Rotation Examples

Example 1

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

Example 2

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

Image Source

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

Web Address

Simply enter the 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 ...

- Property Categories
- User Created Properties
- Property Data Types
- Restricting Property Access
  - Restricting Access to Component Properties
  - Writing to Private or Protected Session Property
- 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
- Right-Click Menu
  - Actions
  - Structure
  - Binding
  - Options

INDUCTIVE UNIVERSITY

Component Properties

Watch the Video

Property Categories

Each Perspective component has a list of available properties. Each property is placed into one of several categories, and each property category grou ps 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.

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.

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" 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. Dataset properties have an dataset viewer and editor in Perspective's Property Editor. Once a property is bound to a dataset, you can access the viewer by clicking on the **Dataset Browser** icon. 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

Note:

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

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

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

Restricting Property Access

Perspective components have the ability to restrict access to properties from the app/browser. Property access settings do not restrict or inhibit built-in component interactions with bindings and python scripts. Instead they protect against malicious code execution in the browser. Normally a user can

potentially execute arbitrary JavaScript code via developer tools (which generally are included with all web browsers) to interact with components and properties in the session. However setting property access level to **Private** or **Protected** will prevent such approaches, as browser-side script execution will be unable to access property values on the server side.

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

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

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. The existence of a binding on a property will result in the property being re-created in the session.

Bindings and Persistence

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

Imagine a table component, with a binding on the table's data property. In most cases, you'll likely want the data in the table to be generated in the session from the binding, retrieving the most 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

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 property to an embedded view using a parameter with property bindings, Tag bindings, or even scripts. For more information on embedded views, refer to Pass a Property into an Embedded View Using a View Parameter and View Properties.

Page

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

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

Search Filter

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

Bindings

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

Styles

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

Right-Click Menu

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

Actions

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

Structure

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

Binding

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

The Binding section of items allows you to configure or manipulate existing bindings on a property.

- **Configure Binding...**: Opens the binding configuration window, allowing you to add a binding to the clicked property.
- **Copy Binding**: Copies the binding configuration from the clicked property. This item is disabled if the property does not have a binding configured.
- **Paste Binding**: If a binding configuration was previously copied via **Copy Binding**, this option will paste the binding configuration on the clicked property. This item is disabled if a binding configuration was not copied.
- **Disable Binding**: Disables the binding on the clicked property. This item is hidden if the clicked property does not have a binding configured.
- **Remove Binding**: Removes the binding configuration from the clicked property. This item is hidden if the clicked property does not have a binding configured.

Options

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

Related Topics ...

- Property Bindings in Perspective
- Containers in Perspective
- Pages in Perspective

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 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.

Note: 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

INDUCTIVE UNIVERSITY

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

3. You have the option of saving the image as a link or embedding the image in the view. The Link Image screen is displayed. Choose the **Save and Link** radio button and then click **Upload**. 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.

INDUCTIVE UNIVERSITY

SVGs

Watch the Video

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

> **Note:** 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.

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 **Uplo ad**. 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 Style s and Style Classes.

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. If the image is larger than 100KB, you will 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. 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 from Ignition's Translation system. Once terms have been defined, translations can be enabled by the use of the "locale" session property. This page covers using localization in Perspective. For the Vision module, see Localization in Vision.

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

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

On this page ...

- Selecting a Language
- 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 locale tags: https://www.oracle.com/java/technologies/javase/jdk8-jre8-suported-locales.html#util-text

Switching Languages within a Session

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

Now we'll 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
  - 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
  - Workstation Mode Action

INDUCTIVE UNIVERSIT

Events and Actions

Watch the Video

Configure Events and Actions

To configure an event and action on a component:

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

3. Choose an event from the left hand side.

Note: You can configure actions for as many events as you'd like, but you'll need to configure the actions separately for each one.

4. Next choose one or more actions to associate with the event, by clicking the **Add** icon. Descriptions of the options for each component action are described in the sections on this page.
5. Configure the actions as you want 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.

Each action type is described below. Click on the expand panel to see the description and details for that action.

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*.

Action Setting	Description
Continuous	Begins recording accelerometer data, which repeatedly updates the **accelerometer** object in the current session's Session Properties. • **Sample Rate** indicates how often the accelerometer object should be updated. • **Context** provides the opportunity to pass a custom object through to the Accelerometer Data Received Session Event. A **session** object is already provided to that event script.
Batch	Records accelerometer data at a specified interval for a specified rate, then sends the accumulated data to the Gateway, to be handled by the Accelerometer Data Received Session Event. • **Sample Rate** indicates how often a reading should be made. • **Duration** specifies for how long data should be logged. • **Context** provides the opportunity to pass a custom object through to the Accelerometer Data Received Session Event. **Note:** A **session** object is already provided to that event script.

Off	Turns off accelerometer data recording.
Enabled	Specifies whether the action should be used or ignored.
Prevent Default	Prevents the browser's default behavior from occurring. Useful when you want to prevent the browser's built-in right-click menu from showing.
Stop Propagation	Prevents events from higher up in the component hierarchy from triggering when the selected event triggers.

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

Alter Logging Action

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

Action Setting	Description
Remote Logging Enabled	Indicates whether the browser logs should also be sent to the Gateway to log in the Gateway logs.

	Note: For all messages to be visible, the **Perpective.Client** logger must be set to the same logging level as the level indicated in this action.
Set Logging Level	Dictates how verbose the session's logging should be. A logging level of **all** (or **trace**) shows all possible records, while a level of **off** shows none. Options are: • all • trace • debug • info • warn • error • fatal • off
Enabled	Specifies whether the action should be used or ignored.
Prevent Default	Prevents the browser's default behavior from occurring. Useful when you want to prevent the browser's built-in right-click menu from showing.
Stop Propagation	Prevents events from higher up in the component hierarchy from triggering when the selected event triggers.

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.
Enabled	Specifies whether the action should be used or ignored.
Prevent Default	Prevents the browser's default behavior from occurring. Useful when you want to prevent the browser's built-in right-click menu from showing.
Stop Propagation	Prevents events from higher up in the component hierarchy from triggering when the selected event triggers.

Fullscreen Action

Enters full screen mode. Requesting to enter full screen mode only works with events that originate from user interactions. Some browsers may not support full screen requests.

Action Setting	Description
Enter	Enters full screen mode.
Exit	Exits full screen mode.
Toggle	Changes the Session's browser to whichever mode it is not currently in.
View	Enters full screen mode on a targeted view.

Page	Enters full screen mode on a targeted page.
Enabled	Specifies whether the action should be used or ignored.
Prevent Default	Prevents the browser's default behavior from occurring. Useful when you want to prevent the browser's built-in right-click menu from showing.
Stop Propagation	Prevents events from higher up in the component hierarchy from triggering when the selected event triggers.

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.

The table below shows available settings on the Login Action. The Logout Action does not have any settings.

Action Setting	Description
Ask the IdP to re-authenticate users	Determines how re-authentication requests sent to the Identity Provider will be handled. Identity Providers can choose to ignore re-authentication requests, defaulting to their own behavior. Options are as follows: • Project - Use the re-authentication setting located in the General category of Project Properties. • Enable - Prompt the user to provide their credentials, even if they're already logged into the session. • Disable - When selected, the user will not have to provide their credentials if they're already logged in.
Enabled	Specifies whether the action should be used or ignored.
Prevent Default	Prevents the browser's default behavior from occurring. Useful when you want to prevent the browser's built-in right-click menu from showing.

Stop Propagation	Prevents events from higher up in the component hierarchy from triggering when the selected event triggers.

Navigation Action

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

Type	Description
Page	Navigates to a separate page.

Type	Description
Set Page	A string denoting the page URL of the target page.
Enabled	Specifies whether the action should be used or ignored.
Open in new tab	Specifies whether the newly opened page should replace the current page, or open in a new browser tab.
Prevent Default	Prevents the browser's default behavior from occurring. Useful when you want to prevent the browser's built-in right-click menu from showing.
Stop Propagation	Prevents events from higher up in the component hierarchy from triggering when the selected event triggers.

View	Replaces the current main View with a new main View.

Type	Description
Select View	The path to the view that should be used.
Parameters	A set oparameters 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
Enabled	Specifies whether the action should be used or ignored.
Prevent Default	Prevents the browser's default behavior from occurring. Useful when you want to prevent the browser's built-in right-click menu from showing.
Stop	Prevents events from higher up in the component hierarchy from triggering when the selected event triggers.

	Propagation
Url	Navigates to an external web address.

Type	Description
Enter Url	The URL that the action should navigate to. Example: `https://inductiveautomation.com/`
Enabled	Specifies whether the action should be used or ignored.
Open in new tab:	Specifies whether the newly opened page should replace the current page, or open in a new browser tab.
Prevent Default	Prevents the browser's default behavior from occurring. Useful when you want to prevent the browser's built-in right-click menu from showing.
Stop Propagation	Prevents events from higher up in the component hierarchy from triggering when the selected event triggers.

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 dismissable	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.
Enabled	Specifies whether the action should be used or ignored.
Prevent Default	Prevents the browser's default behavior from occurring. Useful when you want to prevent the browser's built-in right-click menu from showing.
Stop Propagation	Prevents events from higher up in the component hierarchy from triggering when the selected event triggers.

Perspective Popup Actions can be set to use Exact Positioning, or Relative Positioning. Relative allows the popup to be positioned based off of the mouse cursor position when used with a Mouse Event.

Action Setting	Description
Position Exact	Exact Positioning controls where in the session the popup should appear, and how large it should be. If no specifications are given, the popup will open centered at its default size. • **Top**, **Left**, **Bottom**, and **Right** control how far the popup should be offset from each margin of the session. • **Width** and **Height** specify how large the popup should appear.
Position Relative	Relative Positioning enables the popup to be positioned based off of the mouse cursor position when used with a Mouse Event. The Circle icon in the center represents the mouse location. The arrows icons represent the location the popup will appear in relation to the user's mouse cursor. Click on an arrow to position the popup window.

Viewport Bound	This feature is new in Ignition version **8.1.3** Click here to check out the other new features If selected then Popups will be "shifted" on open to always open within the bounds of the viewport. If the Popup would have been larger than the viewport then Popup will be resized to fit within the bounds. This setting also prevents a Popup from being dragged even partially off-screen.

Refresh Action

Reloads the current browser tab.

Enabled	Specifies whether the action should be used or ignored.
Prevent Default	Prevents the browser's default behavior from occurring. Useful when you want to prevent the browser's built-in right-click menu from showing.
Stop Propagation	Prevents events from higher up in the component hierarchy from triggering when the selected event triggers.

Scan Barcode Action

Allows the user to scan a single barcode on their mobile device, which is then sent to the Gateway and handled by the Barcode Scanned Session Event.

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

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

Action Setting	Description
Barcode Type	Indicates the format of the barcode to be scanned. **Any** can be used to catch all barcodes.
Barcode Background Color	This feature is new in Ignition version **8.1.3** Click here to check out the other new features This option enables the reading of barcodes with light or dark backgrounds, or previously white barcodes with black backgrounds couldn't be read. Options are Light, Dark, or Auto.
Context	Provides the opportunity to pass a custom object through to the Barcode Scanned Session Event. **Note:** A **session** object is already provided to that event script.
Enabled	Specifies whether the action should be used or ignored.
Prevent Default	Prevents the browser's default behavior from occurring. Useful when you want to prevent the browser's built-in right-click menu from showing.
Stop Propagation	Prevents events from higher up in the component hierarchy from triggering when the selected event triggers.

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.

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.

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.
Enabled	Specifies whether the action should be used or ignored.
Prevent Default	Prevents the browser's default behavior from occurring. Useful when you want to prevent the browser's built-in right-click menu from showing.
Stop Propagation	Prevents events from higher up in the component hierarchy from triggering when the selected event triggers.

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.

Script Action

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

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

Enabled	Specifies whether the action should be used or ignored.
Prevent Default	Prevents the browser's default behavior from occurring. Useful when you want to prevent the browser's built-in right-click menu from showing.
Stop Propagation	Prevents events from higher up in the component hierarchy from triggering when the selected event triggers.

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.
Enabled	Specifies whether the action should be used or ignored.
Prevent Default	Prevents the browser's default behavior from occurring. Useful when you want to prevent the browser's built-in right-click menu from showing.
Stop Propagation	Prevents events from higher up in the component hierarchy from triggering when the selected event triggers.

Workstation Mode Action

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

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

When called from a session running in Perspective Workstation, changes the mode of workstation from Windowed Mode to Kiosk Mode.

Action Setting	Description
Windowed	Enters Windowed Mode.
Kiosk	Enters Kiosk Mode
Toggle	Toggles the current mode between Windowed and Kiosk.
Enabled	Specifies whether the action should be used or ignored.
Prevent Default	Determines if the default action of the user agent should occur.
Stop Propagation	Determines if the event should be propagated during the capturing and bubbling phase.

Event Configuration on Button
Events
System Events
onStartup
onShutdown
Component Events
onActionPerformed
Mouse Events
onClick*
onContextMenu
onDoubleClick
onMouseDown
onMouseEnter
onMouseLeave
Configure [onClick] Actions
Organize Actions
1. Workstation Mode
Configure Workstation Mode Action
Requesting to enter fullscreen mode only works with events originating from user interactions. Some browsers may not support fullscreen requests.
Mode
Windowed
Kiosk
Toggle
Options
Enabled
Prevent Default
Stop Propagation
OK
Cancel
Apply

Perspective Project Properties

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

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

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

On this page ...

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

Perspective General Properties

These general properties apply to the Perspective Sessions.

General	
Property	**Description**
Enable Update Notification	Enables notifications for sessions when the project is changed. When notification is enabled, this is the message that will be displayed. The token {timeLeft} will be replaced with the seconds remaining until update. If false, update will be immediate.
Project Locale	Select a project locale.
Include regional variations	Enables regional variations for the locale.
Hide from Launch Page and Native Apps	When selected, the project to be hidden from the Session Launcher and mobile app project listing.
Launch Icon	The image specified here is used to represent the project on the launch page and 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. **Note:** 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	Message to be displayed if system.perspective.closeSession is invoked.
Page Closed Message	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.

INDUCTIVE UNIVERSITY

Tag Drop

Watch the Video

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

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

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

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

Component Bindings Configuration	This section is to set up bindings when a Tag is dropped.
Component Type	Dropdown list of the available components in Perspective.
Tag Property	The name of a Tag's property to be used in a binding. "`value`" is most commonly used.
Prop Path	The property path on the component where the binding will be created. props.value or props.text are most commonly used.
Bidirectional	Check box to indicate if the binding should be bidirectional. The **Bidirectional** property is set by default when adding Input Components.

Perspective Inactivity Properties

Perspective has Inactivity Timeout settings in Project Properties. You can set an Inactivity timer that either closes the Perspective session or logs the user out if no activity is detected after a specified number of minutes. Activity is considered opening a new tab, clicking, typing. If Perspective is running on a mobile device, activity is considered a swipe or tap. The Gateway is the "time keeper" for inactivity timer.

This inactivity feature pertains to perspective sessions only. When the Inactive Session Action occurs (regardless of whether "Logout" or "Close Session" is selected), the user will still be logged into the Identity Provider.

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

The following are the properties for the Inactivity Timeout:

Inactivity	
Property	**Description**
Enabled	Whether the automatic inactivity detection option is enabled.
Inactivity Timer	Time (in minutes) in which a session can be inactive before the user is logged out or the session is closed. The maximum value is 2,147,483,647.
Grace Period	Grace period (in seconds) after the Inactivity Timer time has passed but before the user will be logged out or the session closed.
Grace Period Message	Message to be displayed before a session becomes inactive. Use the placeholder {seconds} to indicate remaining time. (Optional.)
Inactive Session Action	Action that occurs when the session becomes inactive. Options are: • Logout - Log the user out of the session. Note that the user will still be logged in with the • Close Session - Close the session.

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 through the user interface, they are described in the Style Reference.

On this page ...

- Style Editor
- Style Example

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

As of 8.1.4, CSS properties are also available through a properties dropdown menu. This menu provides access to all available properties in the schema.

INDUCTIVE UNIVERSITY

Styles

Watch the Video

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

Style Editor

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

The following menus are available in the style editor:

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

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

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

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

You can minimize the Applied Styles panel by clicking on the **Arrow** icon to the left of the name. Click the arrow again to reopen the panel.

Expanding a category enables you to set the properties associated with it. Here is an example of the Style Editor with the Text menu expanded and a few options selected:

Note:

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

Some components can have multiple individual Style elements, with each one focusing on a different part of the component. When these styles have a conflict, the more specific style wins out, and sets the style for that particular property.

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:

In This Section ...

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.

Initial Theme

Ignition installations come with the following themes:

- light - white background
- dark - dark background
- light-warm
- light-cool
- dark-warm
- dark-cool

Theme Examples

Light:

On this page ...

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

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.

Theme Colors

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

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

Caution:

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

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

Using Theme Colors

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

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

A Style Class is a group of Style settings that are saved together. Style Classes can be applied to multiple components to provide consistency in design. A Style Class enables you to define style elements in one place, and then quickly apply that style to different components. Style Classes are stored in the Styles folder in the Project Browser.

Within the Styles folder, Style Classes can be placed into folders in order to keep them organized. In the following example, we created separate folders for banner styles and logo styles in the Designer. As projects grow and increase in complexity, use folders to more easily manage your Style Classes.

Creating a Style Class

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

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

On this page ...

- 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

INDUCTIVE UNIVERSITY

Style Classes

Watch the Video

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 the 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	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 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.

INDUCTIVE UNIVERSITY

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.

Name	Description
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 applied 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** checkbox in the **Options** section of the **Property Binding** window. Typically this would be done on one of the PROPS properties of an Input component like a multi-state button or a numeric input.

Binding Interface

A property can have many different types of bindings, for example it can have a Tag or an Expression binding. Instead of setting a label statically, the text might change based on a PLC value or on-screen selection. There many ways to bind your components to show values from PLCs, databases, other compo nents, 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`

./		`./LabelA.position.x`

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

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 property 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.

INDUCTIVE UNIVERSITY

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 property 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 property 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 property 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 Structure Binding

Watch the Video

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.

Query Binding

Watch the Video

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.

INDUCTIVE UNIVERSITY

Tag History Binding

Watch the Video

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.

> **Note:** 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**

INDUCTIVE UNIVERSITY

HTTP Binding

Watch the Video

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)).

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 Ignition prompts you for what type of component you want to create, and by dragging and dropping a dropping a Tag directly onto a component that already exists on a window. In both cases, Ignition automatically creates the Tag bindin gs 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.

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 r epresented 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 Tag path 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.

INDUCTIVE UNIVERSIT

Indirect Tag Binding

Watch the Video

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 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.

In This Section ...

Drop Configuration

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

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

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

The param and action Properties

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

param

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

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

On this page ...

- The param and action Properties
- action

UDT Drop Config
- Example

Data Type Drop Configurations

action

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

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

UDT Drop Config

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

INDUCTIVE UNIVERSITY

Tag Drop

Watch the Video

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

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

Example

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

```
{
  "name": "",
  "tagType": "Provider",
  "tags": [
    {
      "name": "UDT_Drop_Target",
      "tagType": "Folder",
      "tags": [
        {
          "name": "New Instance",
          "typeId": "UDT_Drop_Target/My_Type",
```

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

It should look like the image below. Notice for this example, I've placed both the instance and definition into their own separate folders.

2. Create two views. The first will be a big view that we'll be dragging Tags onto, and a small view which we'll associated to the UDT above. For the purposes of this example, we'll use Coordinate Containers as the root container for both views.

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

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

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

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

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

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

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

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

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

Data Type Drop Configurations

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

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

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

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.

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

INDUCTIVE UNIVERSITY

Property Binding

Watch the Video

Property to Property Binding

A property binding simply binds one component's property to another. When that property changes, the new value is pushed into the property that the binding is 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., BigView) from the dropdown list.
   b. In the **Property Editor**, click the **Add Object Member** icon under props.params.
   c. In the Parameters list, select the sliderValue parameter.

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

Using a Tag Binding

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

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

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.

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

INDUCTIVE UNIVERSITY

Expression Binding

Watch the Video

Using Expression Bindings

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

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

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

Example 1

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

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

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

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

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

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

Example 2

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

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

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

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

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

Expression Structure Bindings in Perspective

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

INDUCTIVE UNIVERSITY

Expression Structure Binding

Watch the Video

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 **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

Query Binding

Watch the Video

Binding Properties

Property Name	Description
Path	Here you can enter in the path to the Named Query. Click on the **Search** icon to get a list of all available Named Queries.
Return Format	The Return format specifies how the query results are returned. Options are auto, json, dataset, or scalar. • **auto**: Query results are returned in the format native to the database (typically data set). • **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.
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.

t_stamp	Sine0	Sine2
1,566,328,579,173	87.67	47.31
1,566,328,579,773	86.16	46.20
1,566,328,580,373	81.03	39.78
1,566,328,580,973	80.37	38.96
1,566,328,581,573	73.70	29.30
1,566,328,582,173	68.59	22.19
1,566,328,582,773	66.22	18.89
1,566,328,583,373	58.98	10.66
1,566,328,583,973	58	9.55
1,566,328,584,573	49.16	2.88
1,566,328,585,173	42.79	0.96

1 2 3 4

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.

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.

Note: 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.

Headers	Used to pass key/value pairs in the header of our HTTP requests.

Field	Definition

	Key — Allows dropdown selection from common header keys, or the ability to specify a custom one. **Value** — The Value field is an expression.
Body	An expression indicating what to send in the body of our HTTP requests.
Authentication Type	Indicates what HTTP authentication type to use. Equivalent to specifying the **Authorization** key in the header. Options are None, Basic, Bearer, or Digest.
Authentication Value	The **Authentication Value** field takes an expression that should indicate what authorization string or token should be used in combination with the associated authentication type. For example, if the header should contain the string: `Authorization: Basic aWduaXRpb246cGFzc3dvcmQ=` Then the Authentication Type should be **Basic** and the Authentication Value should be **aWduaXRpb246cGFzc3dvcmQ=**.
Connect Timeout	Indicates how long the Ignition Gateway should wait for a response to our connect request.
Socket Timeout	Indicates how long the Ignition Gateway should wait for a response to a given HTTP request.
Enabled	Indicates whether the binding should be active.
Overlay Opt-Out	Indicates whether the component should reflect a bad quality binding via a Tag overlay.
Allow Cookies	Indicates whether to allow the remote web server to store cookies.
Polling	Controls how frequently HTTP requests should be issued, and therefore how often the binding should be updated.

Transforms

Transforms offer a chance to alter the value returned from a binding. For example, you can bind a property to an integer value and use a transform to map the numerical value to a particular color, all from the same interface.

Transforms can be used as a way of splitting up complex expressions. You can make several simple expressions and chain them together as several transforms to manipulate your original value.

When multiple transforms are applied to a single binding, they are executed in order from top to bottom.

On this page ...

- 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.

The mapping table defaults to a value for both the input and output, but they can both be changed to one of several types. Select the **Input Type** and **Output type** for the mapping using the pull down arrows next to the Input/Output Type headers to make your selections.

Input Type

Input Type is The value coming into the transform (from the binding or previous transform)

- **Value:** Individual numeric values designed to exactly match the input value.
- **Numeric Range:** A range of values that the incoming value will fall between. Use brackets [x,y] to indicate values that are to be inclusive, or parenthesis for values that are exclusive (x,y), and you can mix them as needed. You can also omit start or end values to indicate no end to the range. See the examples below for clarification.
- **Expression:** Used in a similar manner to a Value Input Type. Use the Expression language to create a Value.

Output Type

Output Type is the outgoing value from the transform.

- **Value:** An alphanumeric value.
- **Color:** A color.
- **Expression:** A value result calculated by an expression.
- **Document:** A manually created jSON document (You can also copy an existing or custom made property and paste it in).
- **Style:** A formatting style (best for simple edits of formatting style). To map to a named Style, use the "Value" output type to call the name.
- **Style Class:** A style class to use. Once selected, a dropdown list shows the available style classes.

Use the **Add** icon to add rows to the Map table. The **Delete** icon will delete the selected row. The up/down arrows will sort the order execution for the mapping. The first mapping (from top to bottom) that matches will be the mapping that is assigned.

You can then double-click to set values for each cell except the Fallback cell. The Fallback allows you to create a value to use in case non of your values, expressions, or ranges match to the input.

Examples

Numeric Range to Integer Value

Input Type: **Numeric Range**

Output Type: **Value**

You can see in the Binding Preview at the bottom that the original value of the property binding is zero. Since the Map Transform input value is 0, we get an output value of 99, the Fallback value. This is because the first mapping uses an exclusive 0 (with a parenthesis) so it is testing for strictly greater than 0, not "greater than or equal to."

Numeric Range to Number Mapping	
(0,25]	0
(25,40]	1
[40,50]	2
Fallback	99

On this page ...

- Input Type
- Output Type

Examples
- Numeric Range to Integer Value
- Integer to String
- Expression to Color
- Numeric Range to Expression

INDUCTIVE UNIVERSIT

Map Transform

Watch the Video

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

INDUCTIVE UNIVERSITY

Format Transform

Watch the Video

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

INDUCTIVE UNIVERSITY

Script Transform

Watch the Video

Using Complex Properties in Your Transform

Any time you reference a complex component property (like an array or object), it will contain a Qualified Value(s). This means they have a quality code and timestamp attached to each piece of data. Using your script to access it will result in a Qualified Value but you can access the actual value manually using the **.value()** function. This means that fetching a complex property like an array must be looped through and converted manually if you want a basic array. For ease, the output of any binding or transform will automatically be stripped of quality and timestamp so you can use the value argument directly.

Example 1) Using the results of an array Property Binding in your script transform: You can reference the property value directly by calling the value parameter that is passed in.

Example 2) Fetching an additional array Property inside your Script Transform: You must manually convert the array property to a simple array.

Example Scripts

A lot of the components in Perspective expect arrays or JSON structured data in order to display data, but several binding types return single values or datasets. In order to transform one into the other, you can add a Transform to the binding and fill in a script to change the data format. Here are a few examples of code that can be added to a **Script Transform**.

Note: Don't forget to tab in correctly if you are copying scripts from this page. The Script Transform is a function (starts with "def") and every line of the code below should be tabbed in one from the edge.

These examples are not necessary with Tag History or Query binding types. For both of these, there is a dropdown setting at the bottom of the binding page that allows you to select the return type of DOCUMENT.

Dataset to Array of Objects

This is a script to take a dataset and transform it into a json array. All header names will be included in the resulting structure. This type of array is expected on many Perspective components like the Table component.

Dataset to Array of Objects

```
# convert the incoming value data
pyData = system.dataset.toPyDataSet(value)
# get the header names
header = pyData.getColumnNames()
# create a blank list so we can append later
newList = []

# step through the rows
for row in pyData:
        # create a new blank dictionary for each row of the data
        newDict = {}
        # use an index to step through each column of the data
        for i in range(len(row)):
                # set name/value pairs
                newDict[ header[i] ] = row[i]

        # append the dictionary to list
        newList.append(newDict)

# return the results
return newList
```

Dataset to Array with Renamed Columns

This is another script to take a dataset and transform it into a json array. In this example, you create the header names to be included in the resulting structure. This type of array is expected on many perspective components like the XY chart, Dropdown, etc.

Dataset to Array of Named Objects

```
# set the header names. For the XY chart, these must match the values in the series property.
header = ["Column 1", "Column 2"]

# convert the incoming value data
pyData = system.dataset.toPyDataSet(value)
# create a blank list so we can append later
newList = []

# step through the rows
for row in pyData:
        # create a new blank dictionary for each row of the data
        newDict = {}
        # use an index to step through each column of the data
        for i in range(len(row)):
                # set name/value pairs
                newDict[ header[i] ] = row[i]

        # append the dictionary to list
        newList.append(newDict)

# return the results
return newList
```

Sparkline Chart

The data expected in a Sparkline chart is a bit different than other charts. Instead of having values paired with a timestamp, it just takes an array of values to draw in order. This is a script to take a Tag History binding, apply the Script Transform, and output only the values in an array. Make sure the Value Format dropdown is set to Dataset.

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

INDUCTIVE UNIVERSIT

Expression Transform

Watch the Video

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
  - Scripting
  - Perspective Data Types
- Object Traversal
- Scripting Transforms
- Extension Functions

Perspective Scripting Fundamentals

Though Perspective uses the same basic platform (Jython) as other scripting environments in Ignition, interfacing with some unique features in Perspective might make it feel like a new scripting experience. Here are some key details that might be important to Perspective script writers.

Scopes

Perspective does not have a "Client" scope, because unlike Vision, Perspective does not have clients. All Perspective scripting is run on the Gateway, although session-specific functions (like navigation) will only affect a single session. Critically, this means that:

- Client-scoped scripting functions (`system.file`, `system.gui`, and `system.nav` functions) will *not* work in Perspective.
- Other scripting functions, like `system.util.getLogger()`, will run in a Gateway context.

Perspective Data Types

Component properties in Perspective are structured as JSON. However, interacting with them does not require any kind of specialized knowledge. Critically, every property in Perspective is one of three types:

Type	Description	Example
Value	A value is a simple piece of data, usually a number or string. Assigning a value to a value property is just like assigning a value to an ordinary Python variable.	`self.props.text = "My Text"` `self.props.startAngle = 5`
Object	An object is structured like a Python Dictionary, holding any number of elements. Each element can be a value, object, or an array.	`motorObject = {"motorNum": 1,"motorState":"Running"}` `self.custom.myObject = {"operationNum":15," motorObject":motorObject}`

Array	An array is structured like a Python List. Unlike an object where each element in the data type has an associated key, elements in an array only have a position, or offset. Elements in an array can be values, objects, or other arrays.	`rowObject1 = {"city":" Folsom","country":"United States","population":77271}`
		`rowObject2 = {"city":" Helsinki","country":" Finland","population": 625591}`
		`self.props.data = [rowObject1,rowObject2]`

Object Traversal

In scripting, we can use component properties and methods to access related components, and view and session info. See Perspective Component Methods for details.

Scripting Transforms

Any property binding can make use of a Script Transform to apply any python script to the output value of the binding. For more information, see Transforms and Script Transforms.

Extension Functions

Extension Functions can be found in the Script Configuration window in certain Perspective Components. They allow users to add extended functionality to the component via scripting. These functions are generally more advanced and require better understanding of Python. Extension Functions differ from Event Handlers in that they are not event driven and are called by their component for a specific purpose when appropriate. Extension Functions are often called when the component first loads on a view or when it receives new input. For example, the Perspective Alarm Status Table has an Extension Function named **filterAlarm**. This function is called by the Alarm Status Table for each event before it is displayed in the table and it allows users to control what alarm events are displayed on the table or not.

From an object-oriented point of view, Extension Functions create a custom "subclass" of the base component type. Your subclass can then override and implement parts of the functionality of the component itself, in Python. Following Python object-oriented methodology, each extension function's first argument is called **`self`**. That is because these are methods that belong to the component's class itself, instance methods. The value of **`self`** will always be the component itself. Notice that this is different than Event Handler scripts where you are also given an **`event`** object in your scope. When you write an Extension Function, there is no **`event`** object so the component is given to you as the **`self`** object instead.

Each component Extension Function comes with its own documentation built-into the function's default implementation using a standard Python "docstring". The built-in documentation will contain descriptions for the arguments the Extension Function takes in as well as its return value, if any. You will find that you are unable to edit the function's signature or docstring. Changing the method's signature (arguments or function name) would prevent the component from calling it correctly. Changing the docstring could be misleading or confusing as you would lose the documentation for how your implementation of the function should work.

In This Section ...

Perspective Component Methods

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**
.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()`
.view	Calling this from anywhere within a view will return the parent view of the object.	`self.view`
.getView()	A method that will return the view, similar to ".view" above.	`self.getView()`

.page	Returns a page object associated with the page the current component is on.
	.close() can be called on the page object, and can accept a message string as a parameter.
		`page = self.page`
		`pageID = page.props.pageId`
		`pagePath = page.props.path`

.getPage()	A method that will return the page, similar to ".page" above.	`self.getPage()`
.session	Returns the current Perspective Session you are in. From this session you can get any of the existing attributes of the Session. Similar to the view.session object.	`session = self.session` `sesName = session.props.`

	• .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. • .getInfo() returns a list of session objects. This feature is new in Ignition version **8.1.4** Click here to check out the other new features • .getProjectInfo() returns a dictionary of project meta data, including name, title, description, lastModified, lastModifiedBy, views, and pageConfigs.	`gateway.address` `sesProp = session.custom.propertyName`
.getSession()	A method that will return the session, similar to ".session" above.	`self.getSession()`
.getChild (string)	Method that looks for a change component of a given name. Returns None if not found. String can either be the name of a child object, or a path to an object delimited by a forward slash, allowing you to move through multiple items in the hierarchy in a single call.	`self.getChild('Label_0')` `self.getChild('Container/Label_0')`
.getSibling (string)	Returns a reference to an object in the same container that the source component is located in. Similar to calling self.parent.getChild('component').	`self.getSibling('Label')`
View Objects		
.rootContainer	Returns the root container of the View.	`view.rootContainer`
.id	Returns a string that uniquely identifies the view instance. For example: Perspective/Views/path/to/view yields path/to/view@C	`view.id`

.session	Returns the current Perspective Session you are in. From this Session you can get any of the existing attributes of the Session.
	
	This is similar to the self.session object.
	
	.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.
	
	.getInfo() returns a list of session objects.
		`session = view.session`
		`sesName = session.gateway.address`
		`sesProp = session.custom.propertyName`

Object Traversal Examples

If you want to get other component properties in a view while scripting, you can use the above methods and properties to move around the View.

These examples assume you have the following structure/components in a View:

- View
  - Button 1
  - Text Field 1
  - Container 2
    - Text Field 2

- Container 3
    - Text Field 3
    - Container 4
        - Button 4
        - Text Field 4

Scripting Example: Get component properties from a Button script

```
# this example code exists on the 'Button 1' in the above hierarchy.

# get to the view that the button is in
view = self.view

# get the text from 'Text Field 1'
text1 = self.getSibling('Text Field 1').props.text

# get the text from 'Text Field 2'
text2 = self.getSibling('Container 2').getChild('Text Field 2').props.text

# get the text from 'Text Field 3'
text3 = self.getSibling('Container 3').getChild('Text Field 3').props.text

# get the text from 'Text Field 4'. Either of these will work the same.
text4 = self.getSibling('Container 3').getChild('Container 4/Text Field 4').props.text
text4 = self.getSibling('Container 3').getChild('Container 4').getChild('Text Field 4').props.text
```

Scripting Example: Get component properties from a Button script

```
# this example code exists on the 'Button 4' in the above hierarchy.

# get to the view that the button is in
view = self.view

# get the text from 'Text Field 1'
text1 = self.parent.parent.getSibling('Text Field 1').props.text

# get the text from 'Text Field 2'
text2 = self.parent.parent.getSibling('Container 2').getChild('Text Field 2').props.text

# get the text from 'Text Field 3'
text3 = self.parent.getSibling('Text Field 3').props.text

# get the text from 'Text Field 4'
text4 = self.getSibling('Text Field 4').props.text
```

Built-In Methods

Perspective components contain several shared methods. This section details such methods. Some methods are only available to certain types of components. In these cases, the description for the method will state any limitations.

Refreshing Bindings

The `refreshBinding` function can be used to manually fire a binding, and is designed to be used on bindings that can poll (like query and Tag history bindings). In these instances, using refreshBinding in lieu of polling can save Gateway resources. The refreshBinding() function takes a string as a parameter, corresponding to the property that should be refreshed:

```
self.refreshBinding("props.data")
```

It is often useful to use refreshBinding() from a component message handler, since we can then refresh several applicable bindings via a single system.perspective.sendMessage call.

Requesting Focus

The `focus` method can be called by a component to request focus in a view. This is useful if you wish to control where keyboard input is directed after a particular action.

Due to the nature of focus, calling the `focus` method is only effective on components that can have focus. Input components such as the Text Field and Numeric Entry Field components can gain focus, but Display components like Labels and Images can not gain focus.

```
self.focus()
```

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.

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

6. Click **OK** to commit your method.
7. To call this method, use:

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

For example:

```
self.myMethod("Hi!","This is a test")
```

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 Handlers

Watch the Video

Component Paths

Watch the Video

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)
```

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.

Note: In this simple example, the major issue you may run into is the Message Type. Recall from #4 in Step 1 of the example, that Message Type is case-sensitive, so make sure the script on the Button is correctly referencing the message type, and try again.

Passing Parameters Example

Let's make the previous example more complicated and pass some values with the message. The example above can be modified to determine the timestamp. When passing parameters in a script, the most direct approach is to include any parameters along with the message. Message Handlers have a built-in argument called a **payload**, which is used to transfer values to the handler. The payload is simply a Python Dictionary, so please see the Dictionaries page for more information.

Alternatively, we could create a session property to hold the value, and have the label reference the session property. However this would require that we either create a new property to hold the value or overwrite the value of another property. Thus, our next example will demonstrate how to utilize the payload.

Avoid Storing Values in Tags
Since Tag values are shared by all Perspective sessions, you may not want to write the parameters in Tags. Doing so would result in each session instance potentially trying to overwrite the same value.

Step 1 - Update the Button Script Action

1. Right-click on the Button component and select **Configure events...**
2. Replace the code with the following:

Python - Check the Time, Send a Message

```
messageType = 'my-handler'

# Look up the current time.
currentTime = system.date.format(system.date.now(), 'HH:mm:ss')

# Create a payload to be passed with the message
payload = {'time':currentTime}

# Send the message, pass the payload, limit the scope to the view
system.perspective.sendMessage(messageType, payload, scope = 'view' )
```

3. Make sure the code is properly indented. It should look like the following:

Note: On line 8, we're creating a dictionary, creating a key called "time", and storing the current time with the "time" key. When our handler receives the payload, it can retrieve the value we passed by referencing the "time" key.