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

...

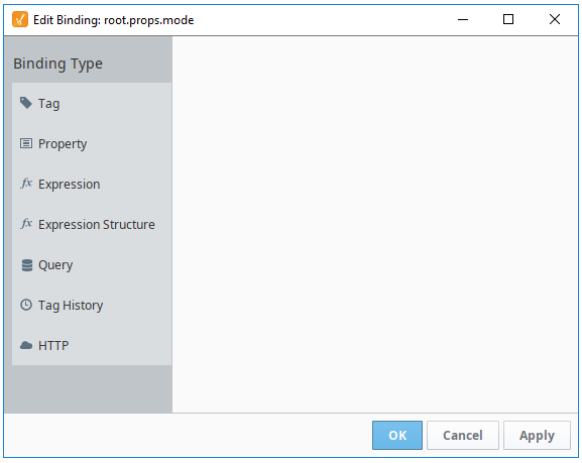
Bidirectional Bindings

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

Binding Interface

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

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



Property Paths

Many of the bindings can utilize a string property path, such as a Property Binding, or Indirect Tag Binding. Because of this, it can be helpful to understand how the paths work. This section details the various keywords and operators associated with these paths. Only properties on components in the same view are eligible to be used in this way. You may not have a binding refer to a property in another view instance, even view instances that may be embedded in your view. (Views may expose property values to their parent via output parameters.)

The format of the property path is like a file system path to get to the component combined with a dot-referenced object path to get to the property. The section referencing the property must begin with the property scope (e.g., "props" or "position" or "meta"). For the following examples, suppose we are designing a View with the following component hierarchy, and that each of these components has an "x", "y" property in "position", as well as a property called "complex" in "props" which is a map containing "foo", which is a number, and "bar" which is an array of numbers.

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

Operator /Keyword	Description	Example

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

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

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

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

Tag Binding

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

The following modes are available:

- **Direct**: Bind the property to a Tag path.
- **Indirect**: Allows properties to be placed in the Tag path, providing a way to make the binding dynamic.

- **Expression**: Utilizes the Expression language to build a Tag path. The Tag path in the Expression is expected to be a string. Unlike the Expression binding, this mode allows the bound property (Tag) to be bidirectional.

Watch the Video

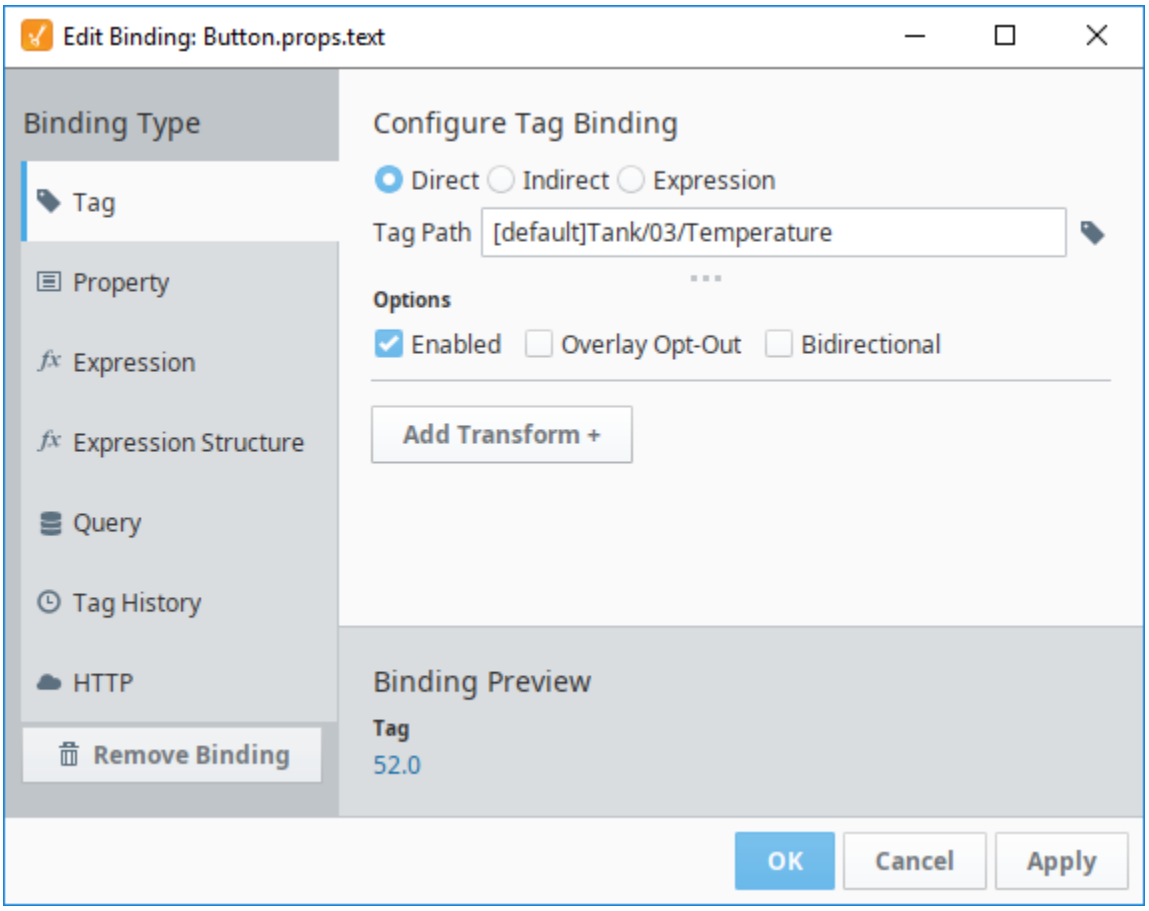
These options are available:

- **Enabled:** Allows the component to be active/in use /interactive on the screen.
- **Overlay Opt-Out:** Choosing the Overlay Opt-out option will ignore the quality of the chosen Tag (or expression), making it have no effect on the component's quality overlay.
- **Bidirectional:** Allows user input or parameter changes on the component to be passed back to the Tag or property that the binding refers to.

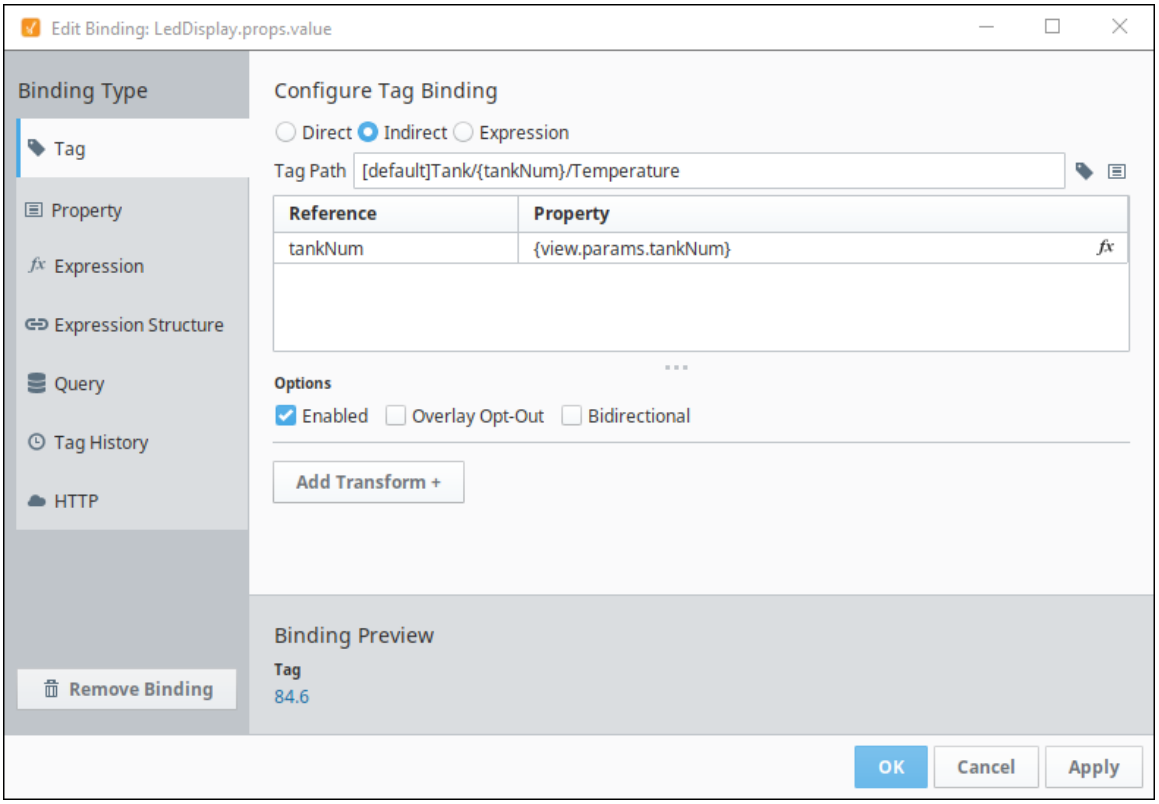
Transforms can be added:

Transforms offer a chance to alter the value returned from a binding. For example, you can bind a p roperty to an integer value and use a transform to map the numerical value to a particular color, all from the same interface. For more information, see Transforms in Perspective.

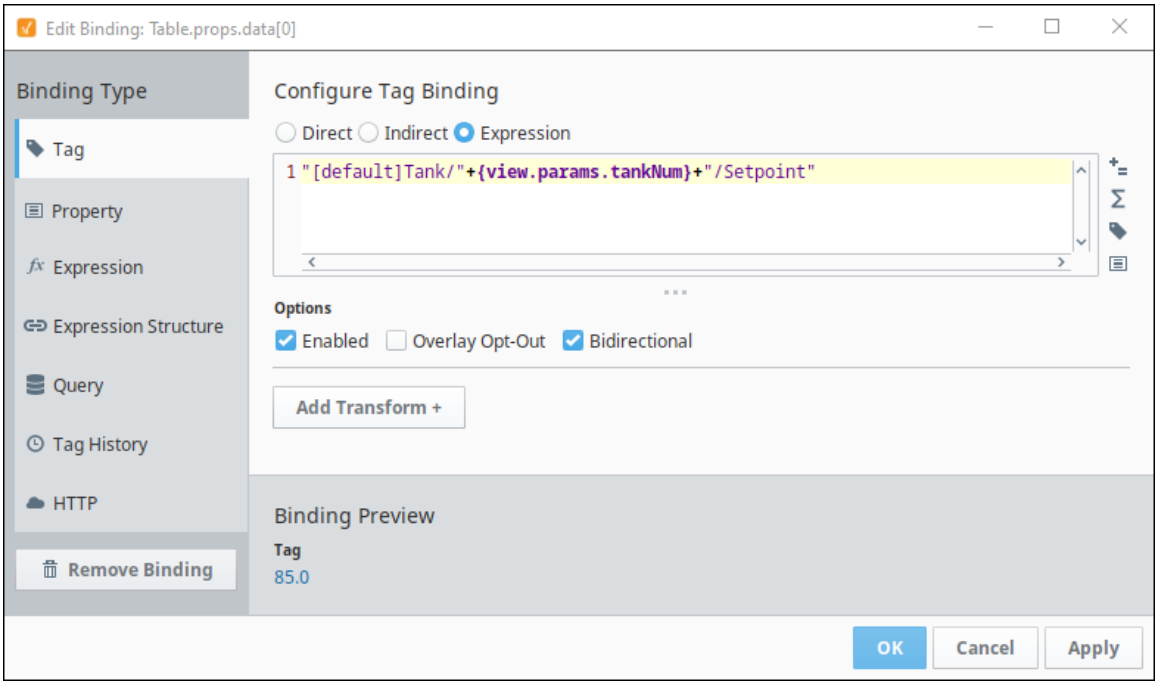
Direct Example



Indirect Example



Expression Example



Property Binding

A Property Binding binds the value of one property, to another. This binding is initially unidirectional.

These options are available:

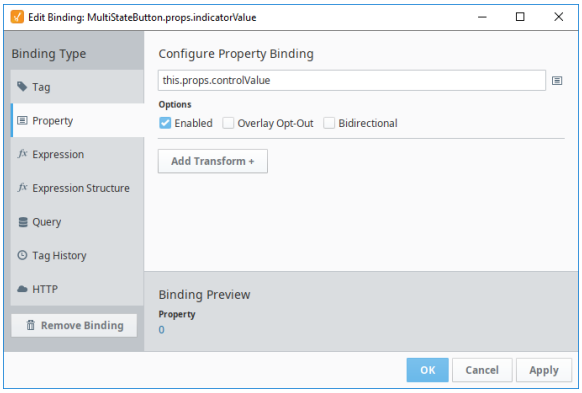
- **Enabled:** Allows the component to be active/in use /interactive on the screen.
- **Overlay Opt-Out:** Choosing the Overlay Opt-out option will ignore the quality of the chosen Tag (or expression), making it have no effect on the component's quality overlay.
- **Bidirectional:** Allows user input or parameter changes on the component to be passed back to the Tag or property that the binding is reading.

Transforms can be added:

Transforms offer a chance to alter the value returned from a binding. For example, you can bind a p roperty to an integer value and use a transform to map the numerical value to a particular color, all from the same interface. For more information, see Transforms in Perspective.

Property Binding

Watch the Video



Expression Binding

An Expression binding uses the expression language to generate a value. This value is passed onto the Property that is linked to the binding. The Expression Binding is unidirectional only.

These options are available:

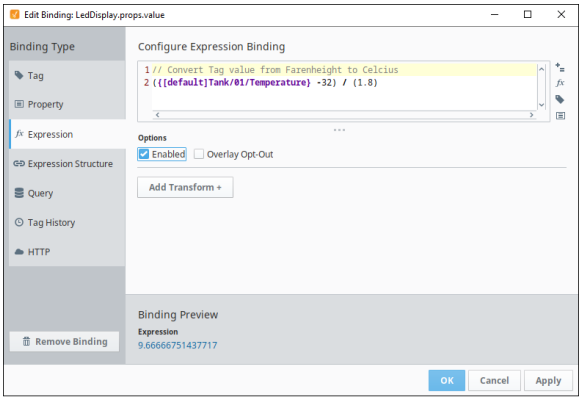
- **Enabled:** Allows the component to be active/in use /interactive on the screen.
- **OverLay Opt-Out:** Choosing the Overlay Opt-out option will ignore the quality of the chosen Tag (or expression), making it have no effect on the component's quality overlay.

Transforms can be added:

Transforms offer a chance to alter the value returned from a binding. For example, you can bind a p roperty to an integer value and use a transform to map the numerical value to a particular color, all from the same interface. For more information, see Transforms in Perspective.

Expression Binding

Watch the Video



Expression Binding Examples

Expression on a button that references its custom property

```
if({this.custom.sourceString}!="abc","Return String 1","Return String 2")
```

Expression on a label that references a button's custom property

```
if({../Button.custom.selected},"Return String 1","Return String 2")
```

Expression on a container that references a button inside it.

```
if({./Button.custom.intValue}>15,1,0)
```

Expression Structure

An Expression Structure binding allows you to build a json document and bind it to a property. Each value within the Expression Structure can be bound to an Expression builder. This allows for a dynamically changing values based on bindings.

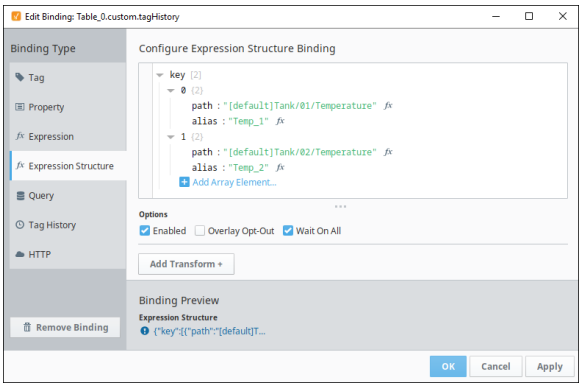
These options are available:

- **Enabled:** Allows the component to be active on the screen.
- **Overlay Opt-Out:** Choosing the Overlay Opt-out option will ignore the quality of the chosen Tag (or expression), making it have no effect on the component's quality overlay.
- **Wait On All:** Waits to evaluate the Expression Structure binding until all Bindings within the Expression Structure have been evaluated.

For this example we also linked a Tag History Expression binding to this expression structure. By using Binding Expressions to modify the "path" field This method can be used to create indirect Tag history bindings. (See Tag History (Expressions) for the other part of this example).

Transforms can be added:

Transforms offer a chance to alter the value returned from a binding. For example, you can bind a p roperty to an integer value and use a transform to map the numerical value to a particular color, all from the same interface. For more information, see Transforms in Perspective.



Query Binding

A Query Binding is a polling binding type that runs a structured Query against any of the database connections configured in the Gateway.

Return Format: the Return format specifies how the query results are returned.

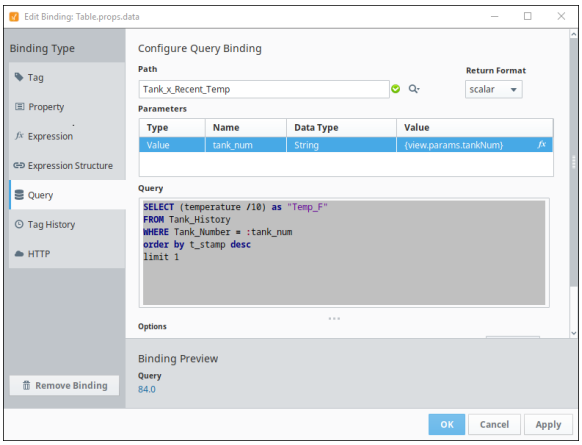
- **auto**: Query results are returned in the format native to the database (typically dataset).
- **json**: Query results are returned in json format. This format is recommended for XY Charts.
- **dataset**: Query results are returned in dataset format. This format is recommended for tables.

- **scalar**: Returns the first element from the query result. This format is best when a single value is expected.

Parameters: If the structured query you select requires parameters you can add them here. The value you enter for the parameters can be modified using an expression builder (*fx*).

Options:

- **Enabled:** Allows the component to be active/in use /interactive on the screen.
- **Overlay Opt-Out:** Choosing the Overlay Opt-out option will ignore the quality of the chosen Tag (or expression), making it have no effect on the component's quality overlay.
- **Bypass Cache:** This will cause the query to bypass/Ignore any cached values from the Named Query and run every time it is called.
- **Designer Limit:** This setting will force the results of the query to be limited to a few rows when run in the Designer.
- **Polling:** This setting will cause the query to run/poll the database using based on a poll time (set) in the Designer.



Tag History

A Tag History binding runs a query against the Tag Historian. There are several ways data can be polled and returned.

Return Format:

- **Wide**: Returns data value and time stamp value from the Tag Historian.
- **Tall**: Returns additional data such as quality code of data from the Tag Historian.
- **Calculations**: Performs calculations on the data (average, sum, count, etc).

Query Mode:

- **PointCount**: Data returned will be the amount of datapoints specified and spread evenly across the date range specified.
- **Periodic**: Data returned will be sampled from the Tag Historian at regular intervals. The regular interval adjustable in the Period filed of the binding configuration.
- **AsStored**: Data returned is exactly as is from the Tag Historian. No data interpolation of the time or points are done.

Time Range:

- **Realtime**: The data is sampled from the most recent time and back from a specified time range. The time range can be adjusted using an Expression builder (*fx*).
- **Historical**: The Start date and End Date can be defined using an expression builder (*fx*).

Select Tags:

- **Direct**: Tags are referenced using a builder that looks directly at the Tag Browser. You can rename the display name of the tag by entering a name in the Alias column.
    - **Alias:** A different name to display for the tag being shown.

    - **Aggregate:** Determines how the tag value will be interpreted. Several functions are available: (Average, MinMax, LastValue, SimpleAverage, Sum, Minimum, Maximum, DurationOn, DurationOff, CountOn, CountOff, Count, Range, Varience, StdDev, PctGood, PctBad).
- **Expression** : The Tag History binding expects a JSON document of objects to read in the desired Tag paths. The best way to do this is to create a custom property and make an array of objects. Each object should contain a value called "path". Optional values to include are "alias" and "aggregate". You can create a custom property structured as mentioned or you can create an "Expression Structure" (refer to Expression Structure Bindings in Perspective for further details). This is the preferred way to make a dynamic Tag path for fetching Tag History values.

Default Aggregation Mode: Any tags left with the Aggregation Mode set to "(default)" will use this setting.

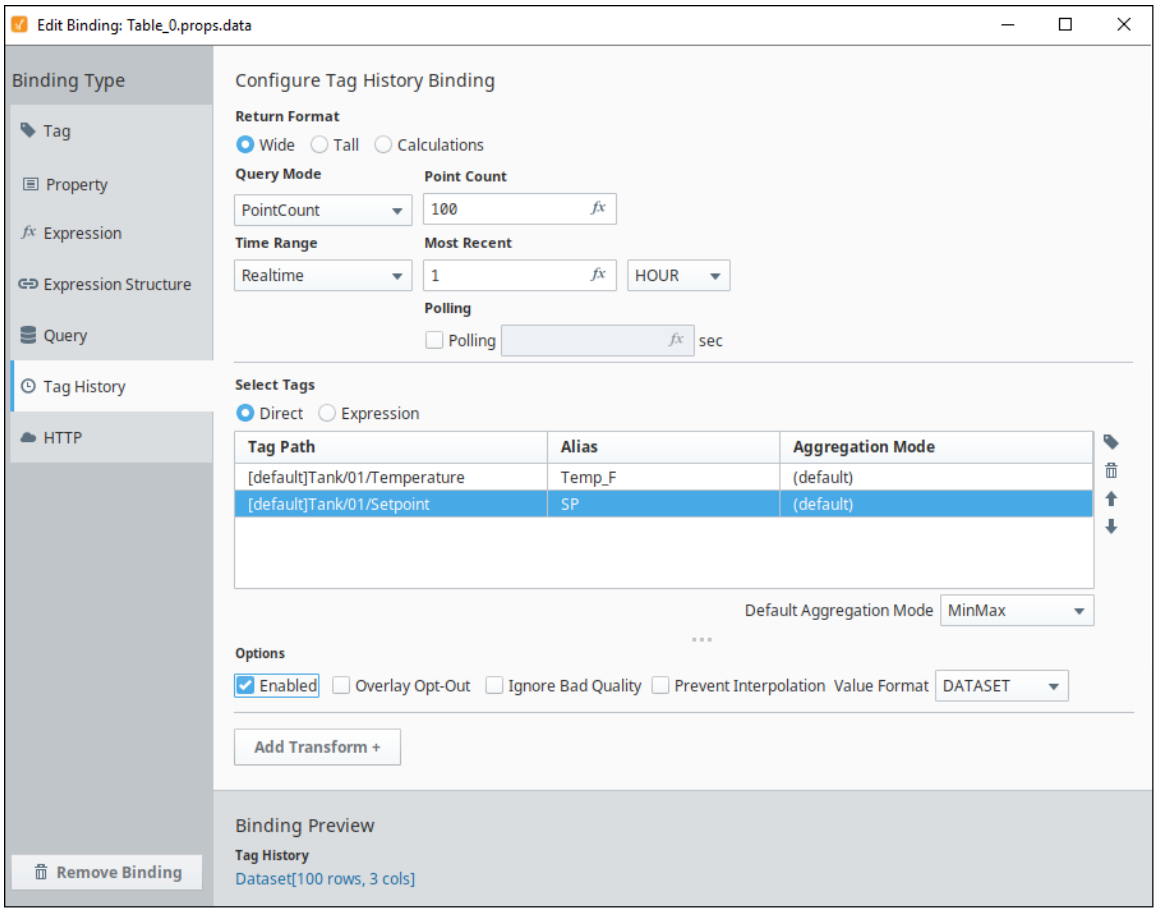
Options:

- **Enabled**: Allows the component to be active/in use /interactive on the screen.
- **Overlay Opt-Out**: Choosing the Overlay Opt-out option will ignore the quality of the chosen Tag (or expression), making it have no effect on the component's quality overlay.
- **Ignore Bad Quality**: This will force the system to ignore any data results that have a bad quality associated with it.
- **Prevent Interpolation**: This forces the data displayed to not use any interpolation. The data will not be truncated or averaged to to fit the display in a nice manor.

Value Format:

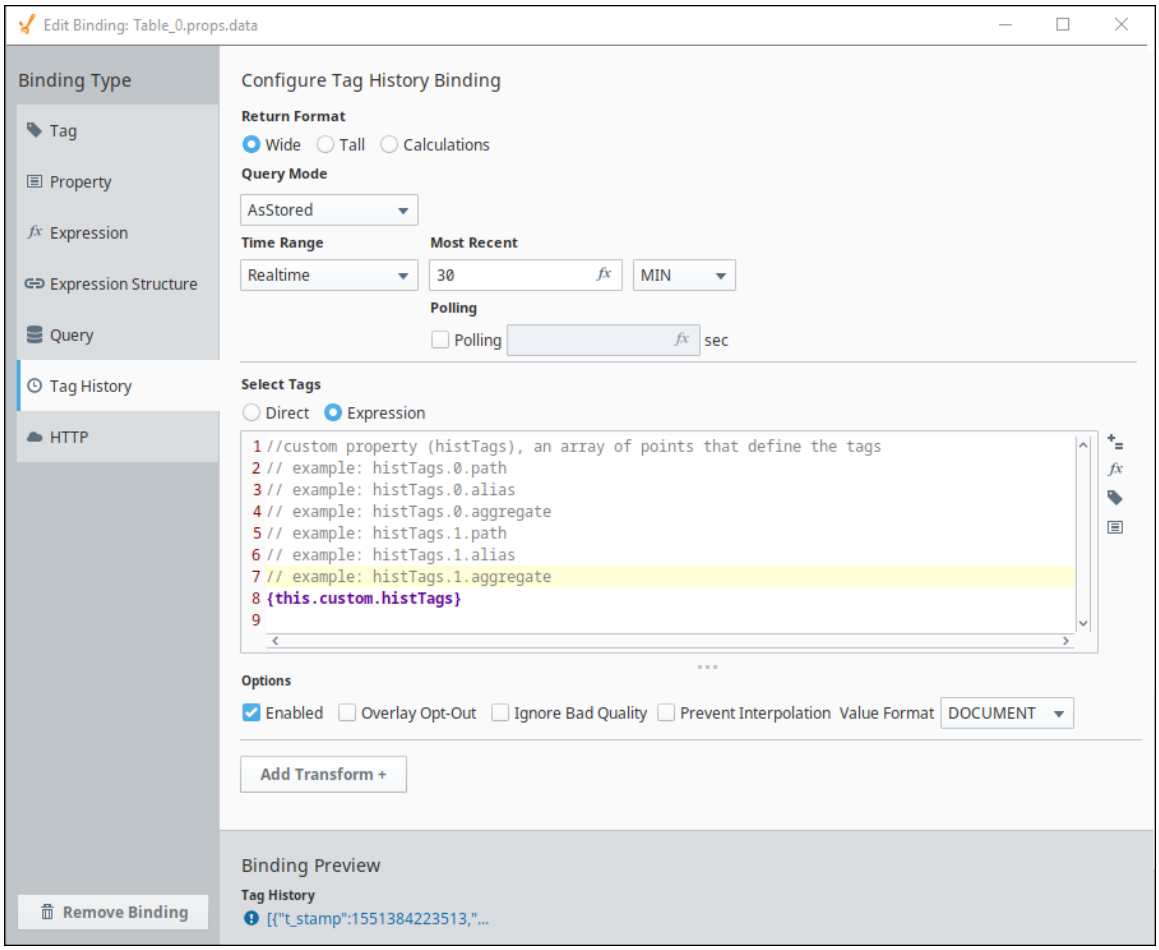
- **Dataset**: Returns the data in a dataset. This format is best suited for use in a table component.
- **Document**: Returns the data in a JSON document. This format is best suited for use in an XY Chart.

Direct Tag Example



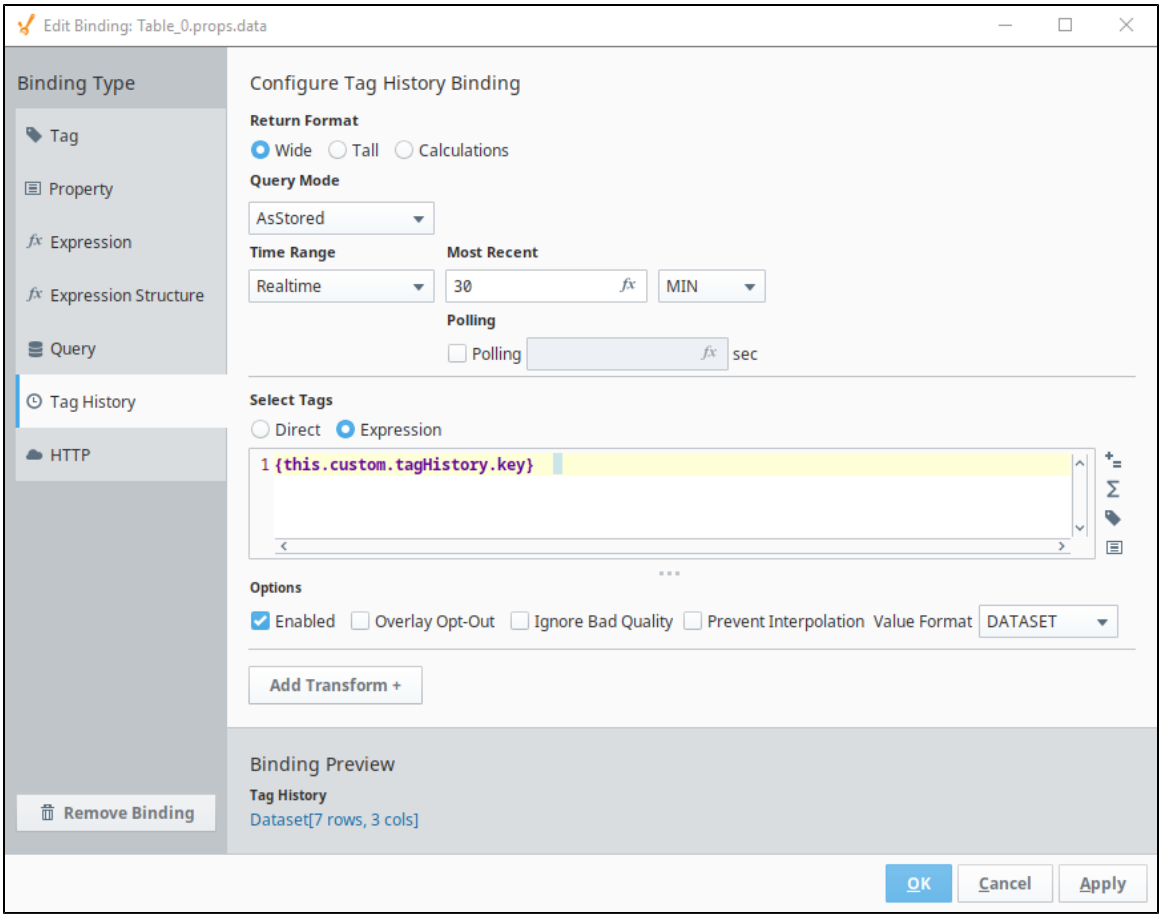
Expression Tag Example 1

For this example, we created a static custom property. The comments in the example show the structure used.



Expression Tag Example 2

For this example we created an Expression Tag structure.



HTTP Binding

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

URL: The web address you are communicating with.

> ⚠ The URL should be inside quotations as it needs to be a string for this binding to work.

Method: The method to communicate with the URL. Available methods are listed here [GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH].

Headers: If any headers are needed they are filled in here:

- **Key**
- **Column**

Body: The body of the HTTP message. This is most commonly used in POST method where it represents the information that is being posted.

Authentication Type: Authentication methods required by the URL. Options are:

- **None**
- **Basic**
- **Bearer**
- **Digest**

Authentication Value: Authentication credentials can be edited via an Expression binding.

Connect Timeout: Amount of time to wait before failing a connection attempt.

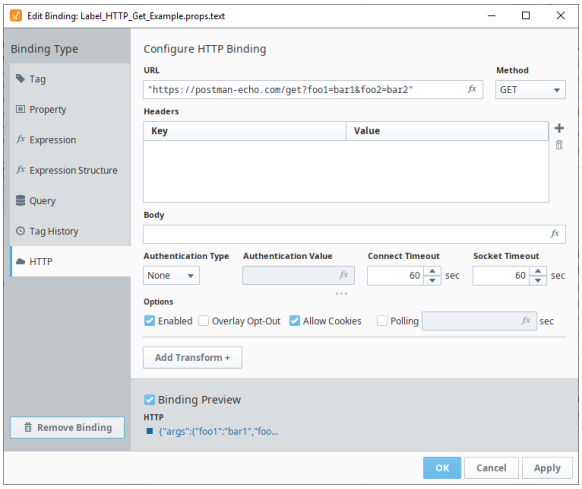
Socket Timeout: Amount of time to wait before failing a connection attempt.

Options:

- **Enabled**: Allows the component to be active/in use /interactive on the screen.
- **Overlay Opt-Out**: Choosing the Overlay Opt-out option will ignore the quality of the chosen Tag (or expression), making it have no effect on the component's quality overlay.
- **Allow Cookies**: Allow cookies from the URL.
- **Polling**: How often to poll the URL (an Expression binding is available to define the poll time (in seconds)).

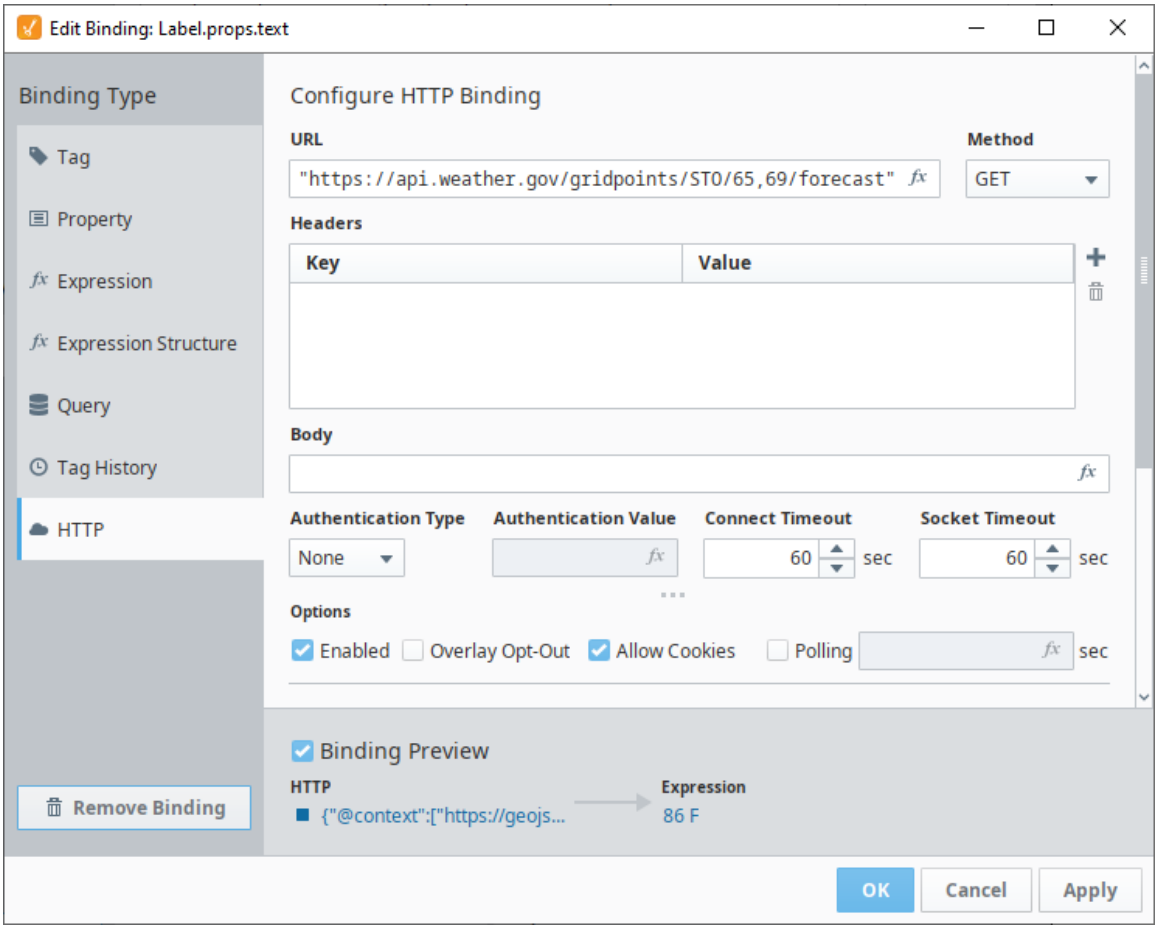
This feature is new in Ignition version **8.0.15**
Click here to check out the other new features

Binding Preview: The Binding Preview checkbox can cancels/stops the binding preview display. Uncheck the Binding Preview option to disable the binding preview. Disabling the binding preview is ideal in cases where you don't want the binding to trigger frequently while configuring it, such as when using a HTTP binding.

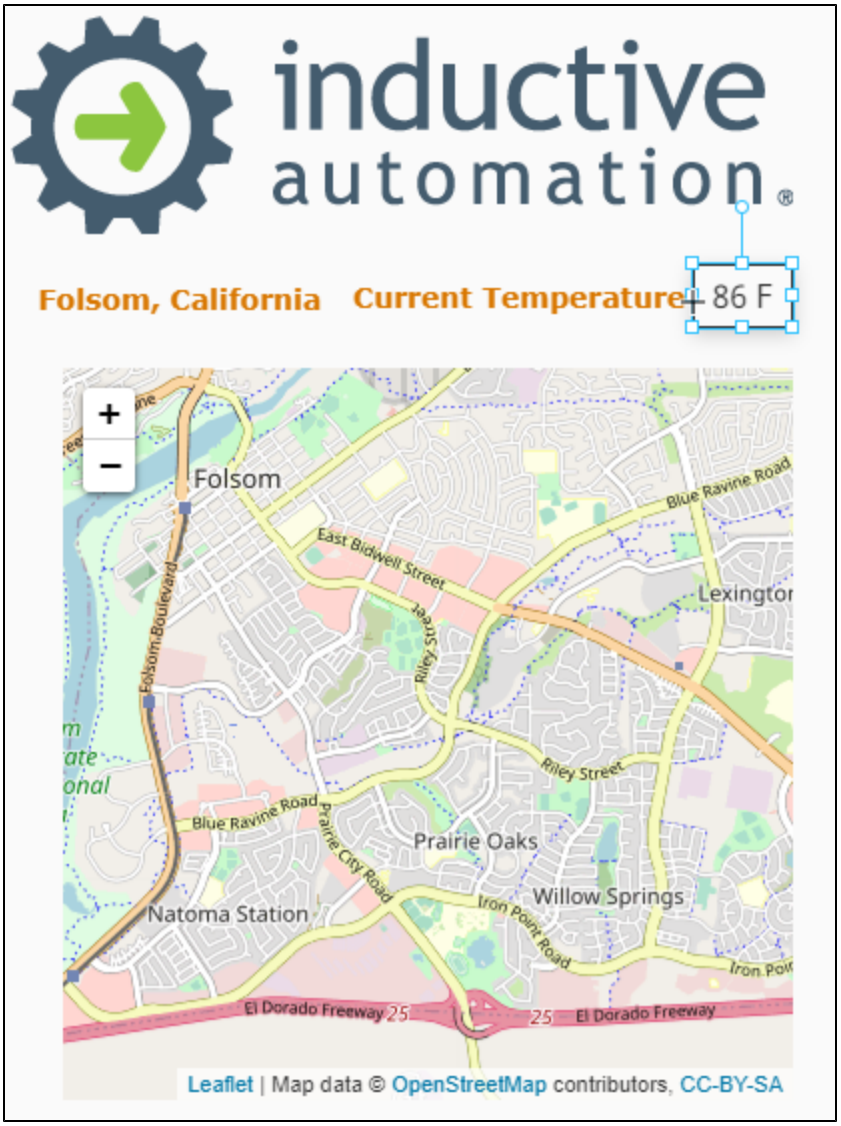


Weather Data Example

The following is an example showing weather data for Inductive Automation headquarters. The API contains detailed forecast information. In this example we use an Expression transform to capture just the current temperature.



The value is bound to a Label component on a view with some other information about Inductive Automation.



In This Section ...