Python - Index a Nested List

```
myList = [1, 2, [3, [4, 5, 6, 7], 8], 9, 10]

print myList[2][1][3]  # Will print out: 7
```

Appending to Lists

You can add values directly to the end of a list with the append() function. You can add anything that is normally allowed in a list.

Python - Appending to a List

```
myList = []

myList.append('Hello')
myList.append('World')

print myList # Will print out ['Hello', 'World]
```

List Functions

Below is a list of common list functions. Some of them are similar to other sequences like strings, while others are unique to lists because lists are mutable.

Function	Description	Example	Output

len(list)	Returns the length of the **list** .	`list = [1, 2, 3, 4, 5]`
		
		`print len(list)`
			5

x in list	Will return **True** if **x** is within the list, **False** if not.
	
	Can also be used to iterate through the list.
		`list = [1, 2, 3, 4, 5]`
		
		`if 4 in list:`
		`  print "There is a 4 in the list"`
		
		`for value in list:`
		`  print value`
			`There is a 4 in the list`
			
			1
			
			2
			
			3
			
			4
			
			5

list.index(x)	Will return the index number of item **x**. Throws an error if item is not found.	`list = [1, 2, 3, 4, 5]`
		
		`print list.index(3)`
			2

min(list)	Returns the smallest item of **list**.	`list = [1, 2, 3, 4, 5]`
		
		`print min(list)`
			1

max(list)	Returns the largest item of **list**.	`list = [1, 2, 3, 4, 5]`
		
		`print max(list)`
			5

list.count(x)	Will return the number of times **x** appears in the list.		2

		`list = [1, 2, 3, 4, 5, 4]` `print list.count(4)`	

list.append (x)	Will add **x** to the end of the list.	`list = [1, 2, 3, 4, 5]`
		`list.append(6)`
		`print list`
			[1, 2, 3, 4, 5, 6]

list.insert(i, x)	Will insert **x** at position **i**.	`list = [1, 2, 3, 4, 5]`
		`list.insert(1, 5)`
		`print list`
			[1, 5, 2, 3, 4, 5]

list.remove (x)	Will remove the first **x** from the list.	`list = [1, 2, 3, 2, 4, 5]`
		`list.remove(2)`
		`print list`
			[1, 3, 2, 4, 5]

list.pop([i])	Will remove the item at index **i**, and return it. If no index is specified, it will remove and return the last item in the list.	`list = [1, 2, 3, 4, 5]`
		`list.pop(2)`
		`print list`
		`list.pop()`
		`print list`
			3
			[1, 2, 4, 5]
			5
			[1, 2, 4]

list.reverse()	Will reverse the items in the list.	`list = [1, 2, 3, 4, 5]`
		`list.reverse()`
		`print list`
			[5, 4, 3, 2, 1]

Tuples

Tuples look similar to lists in that they are defined as a group of comma separated values, but they are enclosed by parenthesis and they are immutable like strings, meaning they can't be altered. Besides that, they are sequences, like lists and strings. This means that tuples have the functionality that other sequences have, such as concatenation, indexing and slicing, and even nesting.

Python - Tuples

```
# Tuples are very simple to create.
myTuple = (1, 2, 3)

print myTuple # Will print out: (1, 2, 3)

# Empty tuples can also be created, to have items added to them later.
myTuple = ()

# Like lists, tuples are not confined to hold values of a single data type either.
```

```
myTuple = (1, "two", 3.3)

# Tuples can even hold other tuples!
myTuple = (1, ("two", 3.3), 4, 'five', (6, (7.7, 'eight'), 9))

a = (1, 2, 3)
b = (4, 5, 6)

# Combine two tuples to make a new tuple
print a + b # Will print out: (1, 2, 3, 4, 5, 6)

myTuple = ('a', 'b', 'c', 'd', 'e', 'f', 'g')

print myTuple[2:5] # Will print out: ('c', 'd', 'e')
```

Lists and tuples can also be nested within each other!

Tuple Functions

All of the functions that work on tuples work on lists, so these should look familiar. However, because they are immutable, not all of the list functions work on tuples. Check out the common tuple functions below.

Function	Description	Example	Output

len(tuple)	Returns the length of the **tuple** .	`tuple = [1, 2, 3, 4, 5]`
		
		`print len(tuple)`
			5

x in tuple	Will return **True** if **x** is within the tuple, **False** if not.
	
	Can also be used to iterate through the tuple.
		`tuple = [1, 2, 3, 4, 5]`
		
		`if 4 in tuple:`
		`  print "There is a 4 in the tuple"`
		
		`for value in tuple:`
		`  print value`
			`There is a 4 in the tuple`
			
			1
			
			2
			
			3
			
			4
			
			5

min(tuple)	Returns the smallest item of **tuple** .	`tuple = [1, 2, 3, 4, 5]`
		
		`print min(tuple)`
			1

max(tuple)	Returns the largest item of **tuple** .	`tuple = [1, 2, 3, 4, 5]`
		
		`print max(tuple)`
			5

Related Topics ...

- Numeric Types
- Strings
- Dictionaries
- Datasets
- Dates

Dictionaries

Mapping Type

A Dictionary is a mapping object. Where sequences are indexed with a numeric index value, dictionaries are indexed using keys. These keys then have a matching value pair that is associated with a particular key. For example, with a list we can extract the object at index 0, which we may have decided is the name, whereas with dictionaries, I can instead extract the a value using a key "name". Because of how they work, dictionaries are sometimes known as associative arrays in other programming languages.

Dictionaries are created using braces { } , where each key/value pair is separated by a comma (,) and keys are separated from their values using a colon (:). In the example below, I created a dictionary with two keys: name, id.

Python - Creating a Dictionary

```
# In this dictionary, I associated the name John Smith to the key "name",
# and the id number 12345 to the key "id".
myDictionary = {"name":"John Smith", "id":12345}
```

On this page ...

- Mapping Type
  - Using a Dictionary
- Dictionary Functions

INDUCTIVE UNIVERSIT

Basic Python - Lists and Dictionaries

Watch the Video

Using a Dictionary

The keys in a dictionary can be numbers, strings, or tuples, but typically a string is used to make a key that best describes the value. Any given key may only appear once in a dictionary, so trying to set another value for a key that already exists will overwrite the previous value for that key. Alternately, attempting to access the value of a key that does not exist will throw an error, while setting a value to a key that does not exist will create a new key/value pair within the dictionary.

To access a value in a dictionary works much like accessing a value in a list; simply place brackets containing the key after the dictionary object.

Python - Accessing Values in a Dictionary

```
# Creates a dictionary with three key/value pairs.
myDictionary = {'Bob': 89.9, 'Joe': 188.72, 'Sally': 21.44}

print myDictionary['Joe'] # Will print out: 188.72

# Adds a key for 'Amir', and alters the value associated with the key 'Sally'.
myDictionary['Amir'] = 45.89
myDictionary['Sally'] = 146.23

print myDictionary # Will print out the whole dictionary: {'Joe': 188.72, 'Amir': 45.89, 'Bob': 89.9,
'Sally': 146.23}
```

It is also easy to loop through all of the values of a dictionary using the keys() function. For example:

Python - Keys Function

```
# The keys() function provides us with a list of keys, which we can iterate through and print out in
addition to using in the value lookup.
for key in myDict.keys():
   print key, myDict[key]
```

There are many use cases for dictionaries, but they are commonly used in Ignition when passing values into a Message Handler or creating a dynamic roster for alarms.

Dictionary Functions

Dictionaries have a few functions that allow for greater control over the dictionary object and the values contained within.

Function	Description	Example	Output

len(dictionary)	Returns the number of items in the dictionary.	`myDictionary = {"name":"John Smith", "id": 12345}`
		
		`print len (myDictionary)`
			2

del dictionary[key]	Will remove the named key.	`myDictionary = {"name":"John Smith", "id": 12345}`
		
		`del myDictionary["id"]`
		
		`print myDictionary`
			{'name': 'John Smith'}

key in dictionary	Will return True if the dictionary has that key.
	
	Can also use "key not in dictionary"
		`myDictionary = {"name":"John Smith", "id": 12345}`
		
		`if "name" in myDictionary:`
		`  print myDictionary["name"]`
			John Smith

dictionary.clear()	Remove all of the items in the dictionary.	`myDictionary = {"name":"John Smith", "id": 12345}`
		
		`myDictionary.clear()`
		
		`print myDictionary`
			{}

dictionary.keys()	Returns a list of the dictionary's keys.	`myDictionary = {"name":"John Smith", "id": 12345}`
		
		`print myDictionary.keys()`
			['name', 'id']

dictionary. values()	Returns a list of the dictionary's values.	`myDictionary = {"name":"John Smith", "id": 12345}`
		
		`print myDictionary.values()`
			['John Smith', 12345]

Related Topics ...

- Client Message Handler
- Gateway Message Handler
- Numeric Types
- Strings
- Lists and Tuples
- Datasets
- Dates

Datasets

Datasets and PyDatasets

A dataset can be thought of as a two dimensional list, or rather a list where each object is another list of objects. Datasets are not normally native to Python, but are built into Ignition because of their usefulness when dealing with data from a database. It is very common to deal with datasets in scripting, as datasets power many of the interesting features in Ignition, like charts and tables.

The main confusion when dealing with datasets is the difference between the dataset object and the PyDataset object. Dataset is the kind of object that Ignition uses internally to represent datasets. When you get the data property out of a component like a Table, you will get a dataset. The PyDataset is a wrapper type that you can use to make datasets more accessible in Python. The biggest differences are seen in how we access the data in the two different objects. However, you can easily convert between the two with system.dataset.toDataSet and system.dataset.toPyDataSet, making it simple to use the object that you find easier to use.

On this page ...

- Datasets and PyDatasets
- Creating Datasets
- Accessing Data in a Dataset
  - Looping Through a Dataset
- Accessing Data in a PyDataset
  - Looping Through a PyDataset
  - PyRow
- Altering a Dataset

Creating Datasets

Because datasets are not native to Python, there is no way to naturally create them within scripting. Instead they must be created using the system.dataset.toDataSet function, which also allows you to convert a PyDataset to a Dataset. It requires a list of headers and a list of each row's data. Each data row list must be the same length as the length of the headers list.

Python - Creating a Dataset

```
# First create a list that contains the headers, in this case there are 4
headers.
headers = ["City", "Population", "Timezone", "GMTOffset"]

# Then create an empty list, this will house our data.
data = []

# Then add each row to the list. Note that each row is also a list object.
data.append(["New York", 8363710, "EST", -5])
data.append(["Los Angeles", 3833995, "PST", -8])
data.append(["Chicago", 2853114, "CST", -6])
data.append(["Houston", 2242193, "CST", -6])
data.append(["Phoenix", 1567924, "MST", -7])

# Finally, both the headers and data lists are used in the function to
create a Dataset object
cities = system.dataset.toDataSet(headers, data)
```

INDUCTIVE UNIVERSITY

Working with Datasets

Watch the Video

Note: All code snippets on this page will reference the cities dataset we created above, so place that code at the beginning of every code snippet.

Accessing Data in a Dataset

To access the data inside of a dataset, each dataset has a few functions that can be called on to access different parts of the dataset. These are listed in the table below.

Function	Description	Example	Output
data.getColumnCount()	Returns the number of columns in the dataset.	`print cities.getColumnCount()`	4
data.getColumnIndex(colName)	Returns the index of the column with the name colName.		2

		`print cities. getColumnIndex ("Timezone")`	
data.getColumnName (colIndex)	Returns the name of the column at the index colIndex.	`print cities. getColumnName(1)`	Population
data. getColumnNames()	Returns a list with the names of all the columns.	`print cities. getColumnNames()`	[City, Population, Timezone, GMTOffset]
data.getColumnType (colIndex)	Returns the type of the column at the index colIndex.	`print cities. getColumnType(3)`	`<type 'java.lang.Integer'>`
data. getColumnTypes()	Returns a list with the types of all the columns.	`print cities. getColumnTypes()`	[class java.lang.String, class java.lang.Integer, class java. lang.String, class java.lang.Integer]
data.getRowCount()	Returns the number of rows in the dataset.	`print cities. getRowCount()`	5
data.getValueAt (rowIndex, colIndex)	Returns the value at the specified row and column indexes.	`print cities.getValueAt (1, 2)`	PST
data.getValueAt (rowIndex, colName)	Returns the value at the specified row index and column name.	`print cities.getValueAt (2, "Population")`	2853114

Looping Through a Dataset

Oftentimes you need to loop through the items in a dataset similar to how you would loop through a list of items. You can use the functions above to do this.

Python - Looping Through a Dataset

```
# We use the same cities dataset from above. Using the range function, we can come up with a range of values
that represents the number of columns.
for row in range(cities.getRowCount()):
        for col in range(cities.getColumnCount()):
                print cities.getValueAt(row, col) # Will print out every item in our cities dataset,
starting on the first row and moving left to right.
```

Accessing Data in a PyDataset

Note: PyDatasets can be accessed in the same ways that Datasets can. This means that all of the above functions (getColumnCount(), getValueAt(), etc) can be used with PyDatasets too.

PyDatasets are special in that they can be handled similarly to other Python sequences. Any dataset object can be converted to a PyDataset using

the function system.dataset.toPyDataSet. All of the functions listed above can be used on a PyDataset, but the data can also be accessed much easier, similar to how you would a list.

Python - Accessing Data in a PyDataset

```
# First convert the cities dataset to a PyDataset.
pyData = system.dataset.toPyDataSet(cities)

# The data can then be accessed using two brackets at the end with row and column indexes. This will print
"PST"
print pyData[1][2]
```

Looping Through a PyDataset

Looping through a PyDataset is also a bit easier to do, working similar to other sequences. The first for loop will pull out each row, which acts like a list and can be used in a second for loop to extract the values.

Python - Looping Through a PyDataset

```
# Convert to a PyDataset
pyData = system.dataset.toPyDataSet(cities)

# The for loop pulls out the whole row, so typically the variable row is used.
for row in pyData:
        # Now that we have a single row, we can loop through the columns just like a list.
        for value in row:
                print value
```

Additionally, a single column of data can be extracted by looping through the PyDataset.

Python - Extract a Column of Data by Looping Through a PyDataset

```
# Convert to a PyDataset
pyData = system.dataset.toPyDataSet(cities)

# Use a for loop to extract out a single row at a time
for row in pyData:
        # Use either the column index or the column name to extract a single value from that row.
        city = row[0]
        population = row["Population"]
        print city, population
```

PyRow

A PyRow is a row in a PyDataset. It works similarly to a Python list.

The examples and outputs are based on the results in the table below. In addition, "print" commands are used, but should be replaced by appropriate logging methods (such as system.util.getLogger) depending on the scope of the script.

A	B
Apple	Orange
Banana	Orange
Apple	Apple

Method	Description	Syntax	Example	Output
index()	Returns the index of first occurrence of the element. Returns a ValueError if the element isn't present in the list.	index (element)	`for row in pyDataset:` `  try:`	

			print row. index("Apple") except: print "No apples in this row"	0 No apples in this row 0
count()	Calculates total occurrence of given element in the row.	count (element)	for row in pyDataset: print row.count ("Apple")	1 0 2

Repeating Elements

You can also have repeating elements in a row:

Example	Output
for row in PyDataset print row * 2	[u'Apple', u'Orange', u'Apple', u'Orange'] [u'Banana', u'Orange', u'Banana', u'Orange'] [u'Apple', u'Apple', u'Apple', u'Apple']

Altering a Dataset

Technically, you cannot alter a dataset. Datasets are immutable, meaning they cannot change. You can, however, create new datasets. To change a dataset, you really create a new one and then replace the old one with the new one. There are system functions that are available that can alter or manipulate datasets in other ways. Any of the functions in the system.dataset section can be used on datasets, the most common ones have been listed below:

- `system.dataset.addRow`
- `system.dataset.deleteRow`
- `system.dataset.setValue`
- `system.dataset.updateRow`

The important thing to realize about all of these datasets is that, again, they do not actually alter the input dataset. They return a new dataset. You need to actually use that returned dataset to do anything useful.

For example, the following code is an example of the setValue function, and would change the population value for Los Angeles.

Python - Altering a Dataset Using the setValue Function

```
# Create a new dataset with the new value.
newData = system.dataset.setValue(cities, 1, "Population", 5000000)

# The cities dataset remains unchanged, and we can see this by looping through both datasets.for row in range
(cities.getRowCount()):
for row in range(cities.getRowCount()):
        for col in range(cities.getColumnCount()):
                print cities.getValueAt(row, col)

for row in range(newData.getRowCount()):
        for col in range(newData.getColumnCount()):
                print newData.getValueAt(row, col)
```

Related Topics ...

- Numeric Types
- Strings
- Lists and Tuples
- Dictionaries
- Dates

Dates

Dates can normally be tricky since they generally require very specific formats. Furthermore, some functions/objects require a date object instead of a string. Fortunately, there are several ways to create and alter date objects with scripting in Ignition.

Python has some built-in libraries to create and manipulate dates and times. However, most users find both Ignition's built-in system functions and even Java's Calendar class easier to use. Regardless, this section will demonstrate some examples from each approach.

On this page ...

- Ignition's System Functions
  - Creating Dates
  - Formatting Dates
  - Date Arithmetic
  - Date Formatting Characters
- Java's Calendar Class
  - Creating Dates
  - Date Arithmetic
- Python's Time and Datetime Libraries
  - Creating Dates - Python's Time Library
  - Creating Dates - Python's Datetime Library
  - Date Arithmetic

Ignition's System Functions

Ignition's system.date library has a large number of functions that provide easy access to datetime creation and manipulation. This page has just a few simple examples. Additional examples and functions can be found in the scripting appendix.

Creating Dates

New datetimes can be created by using either the system.date.now or system.date.getDate. The system.date.getDate function returns a datetime, but the time is set to midnight. However, we can use system.date.setTime to change the time.

Python - System Functions - Creating Dates

```
# Get the current datetime.
print system.date.now()

# Create a date. The time will be set to midnight.
newDate = system.date.getDate(2018, 10, 28)
print newDate

# Change the time on the new date to 11:30 am.
print system.date.setTime(newDate, 11, 30, 0)
```


INDUCTIVE UNIVERSIT

Basic Python - Dates, Colors, and JSON Strings

Watch the Video

Formatting Dates

When printed, datetimes default to a format like the following: **Sun Jan 1 00:00:00 TZ 2018**. However, this can be manipulated by using special characters in the system.date.format function:

Python - System Functions - Date Formatting

```
rightNow = system.date.now()

# Demonstrating the standard format.
print rightNow

# Demonstrating the modified format.
print system.date.format(rightNow, "yyyy-MM-dd HH:mm:ss")
```

Date Arithmetic

The system.date.add* functions can be used to add to or subtract some amount of time from a date. See the system.date.add* functions for more information.

Python - System Functions - Date Arithmetic

```
# Get the current datetime.
newDate = system.date.now()
```

```
# Change the time on the new date to 30 minutes ago.
print system.date.addMinutes(newDate, -30)
```

Date Formatting Characters

The following is a reference of date formatting characters that can be used by system.date.format or Java's DateFormat class. Additionally, there are many other non-scripting uses in Ignition (such as the Vision - Calendar component's Format String property) that can utilize this reference.

Symbol	Description	Presentation	Example	Other Notes
G	Era designator	Text	G=AD	
y	Year	Year	yyyy=1996; yy=96	Lowercase y is the most commonly used year symbol
Y	Week year	Year	YYYY=2009; YY=09	Capital Y gives the year based on weeks (ie. changes to the new year up to a week early)
M	Month in year	Month	MMMM=July; MMM=Jul; MM=07	
w	Week in year	Number	27	If Dec31 is mid-week, it will be in week 1 of the next year
W	Week in month	Number	2	
D	Day in year	Number	189	
d	Day in month	Number	10	
F	Day of week in month	Number	2	2nd Sunday of the month
E	Day name in week	Text	EEEE=Tuesday; E=Tue	
u	Day number of week	Number	1	(1 = Monday, ..., 7 = Sunday)
a	Am/Pm marker	Text	PM	
H	Hour in day (0-23)	Number	0	
h	Hour in am/pm (1-12)	Number	12	
k	Hour in day (1-24)	Number	24	
K	Hour in am/pm (0-11)	Number	0	
m	Minute in hour	Number	30	
s	Second in minute	Number	55	
S	Millisecond	Number	978	
z	Time zone	General time zone	zzzz=Pacific Standard Time ; z=PST	
Z	Time zone	RFC 822 time zone	Z=-0800	
X	Time zone	ISO 8601 time zone	X=-08; XX=-0800; XXX=-08:00	

Java's Calendar Class

While Java's Calendar class if useful, in many cases Ignition's built-in system.date functions are simpler to use. Furthermore, the system.date functions typically use the Calendar class to retrieve the current time, so you are not losing any functionality by using the system functions.

Ignition's System Functions vs Java's Calendar Class
It is highly advisable to use Ignition's system functions to generate and manipulate dates. The information on this page pertaining to the Calendar class is maintained in the interest for legacy installations.

Creating Dates

To create an arbitrary date, you can use the `java.util.Calendar` class. It has various functions to alter the calendar fields, like Calendar.HOUR, Calendar.MONTH, and so on. After you're done manipulating the Calendar, you can use its `getTime()` function to retrieve the Date represented by the calendar. It also has a handy `set()` function that takes the common parameters of a Date. The one major "gotcha" here is that January is month zero, not month one. For example:

Python - Calendar Class - Creating Dates

```
from java.util import Calendar
cal = Calendar.getInstance()

# set year, month, day, hour, minute, second in one call
# This sets it to Feb 25th, 1:05:00 PM, 2010
cal.set(2010, 1, 25, 13, 5, 0)
myDate = cal.getTime()
```

Date Arithmetic

Often you'll have a Date object from a component like the Popup Calendar and want to alter it programmatically. Say, subtracting 8 hours from it, or something like that. The `java.util.Calendar` class is used for this as well. Following the example above, this code would subtract 8 hours from the variable `myDate`.

Python - Calendar Class - Date Arithmetic

```
from java.util import Calendar
cal = Calendar.getInstance()
cal.setTime(myDate)
cal.add(Calendar.HOUR, -8)
myNewDate = cal.getTime()
```

Python's Time and Datetime Libraries

Many components in Ignition that contain a Date property actually expect a Java calendar object. Creating a datetime object using Python's built-in libraries and passing them to a date property on a component will result in an exception.

Ignition's System Functions vs Python's Libraries
It is highly recommended to use Ignition's built-in system.date functions.

Creating Dates - Python's Time Library

The time library can be use to return dates as well as time. Times are created as a tuple of integers. The integers represent the following values: year, month, day of the month, hour, minute, second, weekday, day of the year, daylight savings time)

Check out Python's time library documentation for more information.

Python - Python Library - time

```
import time

# Finds the current local time. The time is returned as a tuple of integers.
myTime = time.localtime()

# Print the time into a 24-character string with the following format: Sun Nov 20 12:00:00 2017
print time.asctime(myTime)

# Alternatively, we can reformat the time in a custom manner, then print it
print time.strftime('%H:%M:%S %b %d %Y', myTime)
```

Creating Dates - Python's Datetime Library

Python's datetime library offers a bit more flexibility since arithmetic can easily be applied. Additional information on the datetime library can be found in Python's official documentation.

Notice the double use of 'datetime' in the example below. This is because the 'datetime' library has a class named 'datetime.'

Python - Python Library - datetime

```
import datetime

# Returns the current datetime.
print datetime.datetime.now()
```

However, we can clean up the above by importing in the datetime class from the library:

Python - Python Library - datetime

```
# Imports the class named 'datetime' from the 'datetime' library, so we don't have to state it twice.
from datetime import datetime

# Returns the current datetime.
print datetime.now()
```

If you need to create a specific datetime, instead of just using the current, you can pass in the values directly when creating an instance of datetime:

Python - Python Library - Creating a New Time

```
from datetime import datetime

# Prints out the following datetime: 2018-01-02 03:04:05.000006
print datetime(2018,1,2,3,4,5,6)
```

Finding the difference between two datetime objects can easily be accomplished by using the '-' character

Python - Python Library - Date Difference

```
from datetime import datetime

rightNow = datetime.now()
someTime = datetime(2018,1,1,1,1,1,1)

# Find the difference between the two dates.
print someTime - rightNow
```

Date Arithmetic

With Python's built-in libraries, the timedelta class provides the simplest way to perform arithmetic on a date: It simply creates an object that effectively represents a duration. The duration can then be applied to a datetime.

Python - Python Library - Date Arithmetic

```
# We're including the timedelta class here
from datetime import datetime, timedelta

rightNow = datetime.now()

# Creating a timedelta object, and setting the hours to 8
offset = timedelta(hours = 8)

# Print the current time, and then print the time minus the offset.
print rightNow
print rightNow - offset
```

Related Topics ...

- Numeric Types
- Strings
- Lists and Tuples
- Dictionaries
- Datasets

Conditions and Loops

If-Statements

The **if** statement should be familiar to anyone with a passing knowledge of programming. The idea of an **if** is that you want your script to execute a block of statements only when a certain condition is true. Python's **if** is simple to use, and has some additional keywords to provide more flexibility.

Simple If-Statement Example

The syntax for **if** is as follows:

Pseudocode - If Statement

```
# Note that 'if' uses lowercase characters.
# Additionally, a colon is placed after the expression.
if expression:

        # The statements that should execute when the expression is true
        # MUST be indented.
        statement
```

Example	Output

`x = 5`
`z = 15`
`if x < 10:`
`  # Since the condition "x < 10" is true,`
`  # the following line will execute`
`  print "'x' is less than 10"`
`if z < 10:`
`  # This condition "z < 10" is false,`
`  # so the following line will not execute`
`  print "this will never show"`
	`'x' is less than 10`

If and Else

You can use the **if...else** form of an **if** statement to do one thing if a condition is true, and something else if the condition is false.

Example	Output

`x = 15`
`if x < 10:`
`  print "x is less than 10"`
`else:`
`  print "x is not less than 10"`
	`x is not less than 10`

Elif (Else If)

Lastly, you can use the **if...elif** form. This form combines multiple condition checks. `elif` stands for "`else if`". This form can optionally have a catch-all `else` clause at the end. For example, this script will print out **`three`**:

Example	Output

`x = 3`
`if x == 1:`
`  print "one"`
`elif x == 2:`
`  print "two"`
	three

On this page ...

- If-Statements
  - Simple If-Statement Example
  - If and Else
  - Elif (Else If)
- For-Loops and While-Loops
  - For-Loop
  - While-Loop
  - The Break and Continue Statements
- The Pass Keyword

INDUCTIVE UNIVERSIT

Control Flow Logic

Watch the Video

```
elif x == 3:
   print "three"
else:
   print "not 1-3"
```

You can use as many elif items as you want, and the else is not required at the end.

For-Loops and While-Loops

For-Loop

Python's **for** loop may be a bit different than what you're used to if you've programmed any C. The **for** loo p is specialized to iterate over the elements of any sequence, like a list. A **for** loop uses an iterator variable to reference each item as it steps through the sequence. This means it's very simple to write a loop!

Note that the syntax of the for-loop requires use of the in-keyword.

Pseudocode - For Loop

```
# In this example, "item" is a variable created specifically by the "for"
loop to act as an iterator.
# The name "item" is not a keyword, and a different variable name may be
used.
# Additionally, note that "for" and "in" are lowercase, and a colon is
present at the end of the line.
for item in sequence:
        # All statements that should execute each iteration must be
indented after the "for" statement
        statement
```

Example	Output

`listOfFruit = ['Apples', 'Oranges', 'Bananas']`
`for fruit in listOfFruit:`
`  print fruit`
	`Apples`
	`Oranges`
	`Bananas`

You don't need to manually create a sequence to repeat a task several times in a for loop. Instead, the built-in function **range()** function can generate a variable-size list of integers starting at zero. For example, calling **range(4)** will return the list [0, 1, 2, 3].

Example	Output

`# Even though this example isn't using the value of "x",`
`# the print statement will still be executed once for each item`
`# in the list returned by range().`
`for x in range(4):`
`  print "this will print 4 times"`
	`this will print 4 times`
	`this will print 4 times`
	`this will print 4 times`
	`this will print 4 times`

While-Loop

A while loop will repeat a block of statements as long as a condition is true. This code will print out the contents of the items in the list.

Pseudocode - While Loop

INDUCTIVE UNIVERSIT

Control Flow Loops

Watch the Video

```
# A while loop simply needs the keyword "while", the condition that
# determines when we should stop iterating, and a colon at the end of the line.
while condition:
        # All statements that should be repeated each iteration must be indented after the "while" statement
        statement.
```

This code uses a function called **len()** , which is a built-in function that returns the length of a sequence.

Example	Output

`listOfFruit = ['Apples', 'Oranges', 'Bananas']`
`x = 0`
`while x < len(listOfFruit):`
`  print listOfFruit[x]`
`  x = x + 1`
	`Apples`
	`Oranges`
	`Bananas`

The Break and Continue Statements

You can stop a loop from repeating in its tracks by using the **break** statement. This code will print out " Loop " exactly two times, and then print " Fini shed ".

Example	Output

`for x in range(10):`
`  if x >= 2:`
`  break`
`  print "Loop"`
`print "Finished"`
	`Loop`
	`Loop`
	`Finished`

You can use the **continue** statement to make a loop stop executing its current iteration and skip to the beginning of the next iteration. The following code will print out the numbers 0-9, skipping 4

Example	Output

`for x in range(10):`
`  if x == 4:`
`  continue`
`  print x`
	`0`
	`1`
	`2`
	`3`
	`5`
	`6`
	`7`
	`8`
	`9`

Infinite Loops

It is incredibly easy to create an infinite loop when using a **while** statement. Depending where the infinite loop was created, it could cause you to lose your work in the Designer, or create a large amount of overhead on the Gateway.

Python - Infinite Loop Created by While Statement

```
x = 0
while x < 10:
        x += 1 # Forgetting to add a way to increment "x" will cause an infinite loop
        print x
```

In many cases, a **for** loop could be used instead of a **while**, but this is not always possible. When using **while**, the best way to avoid an infinite loop is to make sure you always have a way to exit the loop: a simple approach involves using a counter that can eventually trigger a **break** statement, or add the counter as an additional condition to the **while**.

Python - Preventing Infinite Loops Using the Break Keyword

```
###Example 1: using the break keyword
# The counter variable will be used as a guaranteed way out of the While.
counter = 0

# Normally, using True as a condition in a While would be a quick
# way to generate an infinite loop, but the counter helps prevent that.
while (True):

        # Increase the counter
        counter += 1

        # Check the value of the counter. If it's at the point where we can assume we're going to be looping
indefinitely...
        if counter >= 1000:

                # Break out of the loop
                break
```

Python - Preventing Infinite Loops Using an Additional Condition

```
###Example 2: using an additional condition
# Again, the counter variable will be used as a guaranteed way out of the While.
counter = 0

# Instead of using nested logic, we can simply add counter's value as an additional condition with "and"
while (True and counter < 1000):

        # Increase the counter. Once counter >= 1000, the while loop will be forced to end.
        counter += 1
```

The Pass Keyword

When using conditional statements and loops, the pass keyword can be especially useful when writing a new script. When called, the pass keyword does nothing, which may seem useless. However it is great when you find yourself in a situation where you need a line of code to meet a syntax requirement, but don't want the code to do any additional work.

Python - Pass Keyword

```
myVar = system.tag.read(tagPath).value

if myVar == 0:
        firstFunction()
elif myVar == 1:
        secondFunction()
elif myVar == 2:
        # I haven't implemented the thirdFunction() yet, so I can use pass here as a placeholder
        pass.
```

Related Topics ...

- Error Handling
- Getting Started with Scripting in Ignition

Error Handling

What is Error Handling

The concept of error handling is recognizing when an error might occur in a block of code, and instead of throwing the error, handling it gracefully. This can involve giving the user a more distinct error message, letting the user know that their attempt to run the code failed, or even undoing something that your code set it motion so that it can be started again from the same starting point.

Error Handling in Python

Within Python, we can use the **try** and **except** blocks to handle errors. We would first use `try:` and write the code we would like to try indented underneath it. We then must have an `except:` with code that will run if there is an error.

On this page ...

- What is Error Handling
  - Error Handling in Python
  - The Pass Keyword
  - Error Handling Example
- Exception-Specific Failover
- Displaying Error Text
- Determining the Error Object

Pseudocode - Error Handling

```
# With try, we can attempt any amount of code
try:
        some code
        more code
        even more code

# If any of lines in the try block were to throw an error, then we move down and run the block under except.
# The except statement is NOT optional: you must define what your code should do in the event an exception
occurs.
except:
        failover code
```

When running the code above, there is a specific order to the way it executes.

1. The try block is run, and the code within is executed line by line.
    a. If an error occurs, the code in the try block will immediately stop and the except block will begin to execute.
    b. If no error occurs after all code in the try block has been executed, the try block will be finished and the code in the except block will be skipped.
2. After either outcome, any code after the except will then execute.

Because of the way the try and except blocks work, it is very useful on situations that require user input, where something may have been incorrectly entered which would usually result in an error.

The Pass Keyword

The **pass** keyword is unique in that it does nothing except to fill in a spot where code is required. This is useful if we want to handle the exception so that it doesn't throw an error message, but we don't actually want to do anything with it. In this case, we can use pass to tell the script to continue.

Pseudocode - Pass Keyword

```
try:
        some code
        more code
        even more code

except:
        # The error will bring the code to the exception, and then the exception will simply do nothing.
        pass
```

Error Handling Example

An easy way to demonstrate how error handling works is with a division example, since it is easy to cause an error by dividing by 0. Take the code below:

When we run it, we get a printed value of 50. There was no error in the division, and the try block finished successfully. However, if we were to change the value of x to 0, we can see that "An error occurred!" is printed.

Python - Error Handling Division

```
# We start with a value, which could represent some user input.
x = 2

# We use that value in our try block, and have a simple print statement if there is an error.
try:
        value = 100/x
        print value
except:
        print "An error occurred!"
```

Exception-Specific Failover

While each try block must be followed by at least one except block, there can be multiple except blocks to handle different types of errors. This is done by listing the error object after the except keyword and before the colon. Looking back at the example above, I know that my most common error is going to be a divide by zero error. So I can make an exception that is specific to divide by zero errors.

Python - Exception-Specific Failover

```
# We start with a value, which could represent some user input.
x = 0

# Use the user input in division in our try block.
try:
        value = 100/x
        print value

# If the exception that would occur is of type ZeroDivisionError, we can run a specific block of code
except ZeroDivisionError:
        print "Dividing by zero is not allowed. Please stop trying to divide by zero"
# We can then have a second exception without a specific error. This except acts as a catch-all;
# if the user caused an exception we didn't account for above, we can failover to the block of code below.
except:
        print "An error occurred!"
```

The except blocks in the code above cover all possible errors as represented in the table below. Now, we have a more tailored approach to how we handle errors while still catching all of them that may occur.

Inputs (x value)	Output
2	50
0	`Dividing by zero is not allowed. Please stop trying to divide by zero`
'a'	`An error occurred!`

Each try block can have many except blocks, and each except block can also name multiple types of errors as shown below. However, an error that happens within a try block will only ever trigger a single except block.

Pseudocode - Try Block with Except Blocks

```
try:
        some code
except (ZeroDivisionError, RuntimeError, TypeError):
        failover code
```

Displaying Error Text

Sometimes you want to get the actual text from an error in addition to protecting your script. There's an easy way to fetch that information that's built into Python. When you are inside an except section of code, sys.exc_info() gives you access to the error text as a list of values. You can use this to print out your message, display it on the screen, send it to the database, or anything else.

This example should be put on a button, and will write the error text to a Label component that is a sibling to the button. This is useful in Perspective to get error messages out of views and event actions.

```
try:
        # cause an error
        x=[1,2]
        val = x[5]
except:
        # push the error text to a sibling label
        self.getSibling("Label").props.text = sys.exc_info()
```

You can also use the Ignition loggers to push these error messages out to the Gateway console. You can find these in the Gateway Webpage under the Status section, on the **Logs** page.

```
# This code would log an error to the gateway
try:
        # cause an error
        100/0
except:
        # push the error text to the logger
        logger = system.util.getLogger("myLogger")
        # convert the sys.exc_info() to a string and log it
        logger.info(str(sys.exc_info()))
```

Determining the Error Object

To determine the name of the error object that will be thrown from certain errors, we can take a look at the error to figure that out. We already mentioned that dividing by zero gives a ZeroDivisionError, but what about when we divide by a string? If I divide 100 by the letter a without error handling, this is the error I get:

Traceback (most recent call last):
File "<buffer>", line 3, in <module>
TypeError: unsupported operand type(s) for /: 'int' and 'str'

The last line of that error gives the name of the error object "TypeError" followed by the cause of that error. This makes sense, because the string 'a' is the wrong type to be using in division. However, not all errors are so simple and may require a bit more to find the name of the error. For a list of python error names check out this page in the python docs: https://docs.python.org/2.7/library/exceptions.html#Exception

Additionally, some exceptions may be returned by Java. In these cases, Oracle's documentation on the Exception class is more useful: https://docs.oracle.com/javase/8/docs/api/java/lang/Exception.html
A list of common exceptions are listed below.

Exception	Description	Exception Demonstration

ArrayIndexOutOfBoundsException	This typically occurs when a line of code attempts to access an index in a collection,
	but the index specified doesn't exist. This exception is unique to datasets in Ignition. Lists,
	and other built-in Python objects will return the IndexError below.
		`myDataset = system.dataset.toDataSet (["colName"],[[0]])`
		
		`# This will fail because the dataset only has a single row,`
		`# so trying to read the value at row index 5 means we're trying to`
		`# read something that doesn't exist.`
		`print myDataset.getValueAt(0,5)`

AttributeError	Typically seen when a script tries to access a property that doesn't exist.	

	Example: a script tries to get the value of a property on a component, but the property name is misspelled.	`myVar = 10`
		
		`# integers do not natively have a name variable, so this will fail with an AttributeError:`
		`# there isn't an 'attribute' on an integer by the name of 'name'`
		`print myVar.name`

IndexError	Similar to ArrayIndexOutOfBoundsException above, but occurs when Python specific object, such as a list.	`myList = [1,2,3]`
		
		`# There isn't an element in the list at index 4, so this will fail.`
		`print myList[4]`

NameError	Occurs when the local or global name is not found. Typically happens when referencing a variable that hasn't been assigned a value.	`# We haven't given a value to the variable myValue, so it will fail.` `print "The value of the variable is: " , myValue`

TypeError	A TypeError occurs when a function or similar operation is applied to another object of an incorrect type.	`myList = [1,2,3]`
		
		`# The first argument for pop() expects an integer, so passing a string value is innappropriate.`
		`# Passing a string to pop() will return a TypeError.`
		`print myList.pop("0")`

ValueError	A ValueError is returned when the value of an argument is inappropriate. Typically the exact issue is returned in the returned exception.	`myVar = "Hello"`
		`# Strings can be passed to the int() function, but the value needs to be something that`
		`# can be coerced into an integer, like "0".`
		`# Because "Hello" can't be easily converted, the line below will fail.`
		`print int(myVar)`

Related Topics ...

- Conditions and Loops

Built-In Functions

Python Built-In Functions

Functions are code that can be called repeatedly from other places. Functions can have parameters passed into them and may return a resulting value. Some functions, like len(), are built-in. Some functions, like system.gui.messageBox(), are part of the scripting libraries provided by Ignition. Some functions, like math.sqrt(), are provided by the Python Standard Library.

Functions are invoked by using their name followed by an argument list surrounded in parentheses. If there are no arguments, you still need an open and close parenthesis.

This section details several useful Built-in Functions, along with some simple examples. See the official docs for more information.

On this page ...

- Python Built-In Functions
  - Type Casting Functions
  - Checking an Object's Type
  - Generating a Range of Values
  - Rounding Numbers

Type Casting Functions

Python has many functions to convert between data types. Some common type casting functions are listed below

Function	Notes	Example	Output

`bool()`	When casting a numeric value to a boolean, a zero value is false, while all non-zero numbers are True
	
	When casting a String or Unicode value to a boolean, an empty string is False, any other string is True.
		`# Results in False`
		`print bool("")`
		
		`# Results in True`
		`print bool("Test")`
			`False`
			`True`

`int()` and `long()`	When casting a float, rounding will not occur automatically to the decimal value. Instead, the round() function should be called.
	
	When casting a String or Unicode value, the string literal needs to be a valid integer or long: decimal values contained in the string will result in a ValueError
	
	Integers have at least 32 bits of precision, while Longs have unlimited precision.
		`# Float to Integer`
		`print int(123.8)`
		
		`# Float to Long`
		`print long(321.8)`
		
		`# String to Integer`
		`print int("400")`
		
		`# ValueError: the value is not base 10`
		`print int("400.5")`
			`123`
			`321`
			`400`
			`ValueError`

`float()`	When casting a string literal as a float, non-numeric characters in the string will result in an exception, except for a decimal point (".”).	`# Integer to Float`
		`print float(123)`
		
		`# String to Float`
		`print float("400.5")`
			`123.0`
			`400.5`

`str()` and `unicode()`	Most objects can be cast as a string representation of some sort, including sequences.	`print "First line:" + str(80)`
		
		`# Even sequences can be cast as a string,`
		`# making for easy concatenation`
		`myList = [1,2,3]`
		`print str(myList)`
			`80`
			`[1, 2, 3]`

Checking an Object's Type

Checking the data type of an object can easily be done with both the `type()` and `isinstance()` functions.

Function	Description	Example	Output

`type(object)`	When passed a single parameter, this function returns the type of the `object`.	`var = 10`
		`print type(var)`
		`print type(str(var))`
			`type 'int'>`
			`<type 'str'>`

`isinstance (object, classinfo)`	Returns True if the `object` is an instance or subclass of `classinfo`, otherwise, returns false.
	
	If checking for a string or unicode type, a `classinfo` of "basestring", which is the base class for both strings and unicode types, would return True.
		`var = 10`
		`print isinstance (var,int)`
		
		`strVar = "string"`
		`print isinstance (strVar,basestring)`
			`True`
			`True`

Python - Type Validation: type vs isinstance

```
## type() Example
# This example attempts to validate the type of a variable. As written, this will evaluate as True, and thus the print statement would execute.
var = "My String"
if type(var) == type(""):
        print "Variable 'var' is a string"

## isinstance() Example
# The isinstance() function can offer the same functionality as above.
var = "My String"
if isinstance(var, str): # Note the lack of quotation marks around the classinfo parameter. We want to reference the class str, not the string "str".
        print "Variable 'var' is a string"
```

Generating a Range of Values

In some cases, it is useful to generate a range of integers for iteration. Python's `range()` function will return a list of integers.

Function	Description	Example	Output

`range([start,] stop [, step])`	Returns a list of progressively greater integers.
	
	`start` - Integer value denoting the initial value in the list. If omitted, defaults to 0. This parameter is inclusive.
	
	`stop` - Integer value, determines when to cease generating integers. This parameter is exclusive.
	
	`step` - Integer value to increment each new integer by. If omitted, step defaults to 1.
	
	If `step` is positive, integers will be generated as long as (`start + i * step < stop`) is true.
	If `step` is negative, integers will be generated as long as (`start + i * step > stop`) is true.
		`print range(5)`
		`print range(1, 5)`
		`print range(1, 10, 3)`
		`print range(15, 0, -3)`
			`[0, 1, 2, 3, 4]`
			`[1, 2, 3, 4]`
			`[1, 4, 7]`
			`[15, 12, 9, 6, 3]`

Assume we need to read from five separate Tags with a nearly identical Tag path in a single script:

Pseudocode - Tag Path

```
[Provider]Folder/Sub_Folder_1/Tag
[Provider]Folder/Sub_Folder_2/Tag
[Provider]Folder/Sub_Folder_3/Tag
[Provider]Folder/Sub_Folder_4/Tag
[Provider]Folder/Sub_Folder_5/Tag
```

Instead of manually typing each path, we could use `range()` in a for loop that would write the paths automatically.

Python - Range in a For Loop

```
# Initialize an empty list that will ultimately hold all the Tag paths.
tagPaths = []

# Use range to repeatedly append 5 tag paths to the tagPaths list: starting with a value of 1, and ending
with a value of 5.
for num in range(1,6):
        # Use String Formatting to create a Tag path with the iterator's (num) value.
        tagPaths.append("[Provider]Folder/Sub_Folder_%i/Tag" % num)

# Now that tagPaths contains all our tag paths, we can use the list to interact with the tag, such as by
reading their values simultaneously.
tagValues = system.tag.readAll(tagPaths).value
```

Rounding Numbers

You can round numbers inside Python with a few simple functions.

Function	Description	Example	Output

`round(number [, digits])`	When passed a single parameter, this function returns a rounded integer value of a number. If the decimal is greater than or equal to .5, the it rounds up, less than .5 rounds down.
	
	If the optional digits argument is used, then it rounds to that many decimal places.
		`var = 10.236981`
		`print round (var)`
		`print round (var,3)`
			`10`
			`10.237`

`math.floor (number)`	Returns a truncated integer from the number given. The largest integer value less than or equal to *number*.
	
	Note that the example needs to import that math library before being able to call floor().
		`import math`
		
		`var = 100.938`
		`print math. floor(var)`
			`100.0`

math.ceil(number)	Returns the ceiling integer from the number given. The smallest integer value greater than or equal to *number*.
	
	Note that the example needs to import that math library before being able to call ceil().
		`import math`
		
		`var = 100.138`
		`print math. ceil(var)`
			`101.0`

Python - Simple Casting

```
stringVar = "40"

# Without using int(), this line would cause an exception. However int() is able to cast
# the type of stringVar's value to an integer.
print 20 + int(stringVar)

# Type casting is also useful in cases where a string parameter is required, but a numerical value
# should be given, such as the message parameter in system.gui.messageBox().
intVar = 60

# Note that this could also be accomplished with String Formatting instead of using str().
system.gui.messageBox(str(intVar))
```

Related Topics ...

- User Defined Functions
- Libraries

Libraries

The System Library

Ignition comes with a group of system functions, called the System Library. Using a system function is simple. For example, the following code will access the value of a Tag.

Pseudocode - Reading a Tag Value

```
value = system.tag.read("tagPath").value
```

The scripting appendix is full of built-in functions such as this.

Python Libraries

Python Libraries are packages of extra functions that expand the functionality of the code and can be imported into a script. We do this by using the `import` keyword:

Pseudocode - Import a Library

```
# This pseudo code will import a library, and then call a function of that
library.
import myLibrary

myLibrary.specialFunction()
```

The import keyword imports that entire library and allows you to use all of the functions inside of it by calling them off of the imported library. You can also import a piece of a library:

Pseudocode - Import a Function of a Libray

```
# This pseudo code will import a function from a library, and then call
that function.
from myLibrary import specialFunction

specialFunction()
```

Note, that since we are directly importing in the function, we can directly call it instead of having to call it off of the library.

Python Standard Library

Python has an extensive standard library that provides a host of new functionality to the scripting language. The python documentation goes over all of the libraries in its standard library as well as how to use them here: https://docs.python.org/2/library/index.html

Let's take a look at an example of using a common library:

Python - Accessing Files in a Python Standard Library

```
# The csv library provides an easy way to read csv files, regardless of how they are formatted.
import csv

# We first grab our filepath, and feed it into the open function, which opens the file.
filepath = "C:\\test.csv"
csvFile = open(filepath, 'r')

# We then pass our opened csv file object into the csv.reader function, which will read the file.
# This can be looped through in a for loop to print every row of the csv.
```

On this page ...

- The System Library
- Python Libraries
- Python Standard Library
  - Importing 3rd Party Libraries
- Accessing Java
  - Subclassing Java

INDUCTIVE UNIVERSIT

System Library

Watch the Video

```
reader = csv.reader(csvFile)
for row in reader:
        print row
```

Importing 3rd Party Libraries

In addition to the standard libraries, 3rd party libraries can also be imported into Ignition's scripting environment. A Python Library or individual module file will consist of a python file (.py) that contains the code that implements the functions of the library. You can often find python libraries built by other users on the web, or can even create your own. These files can then be placed into a folder within your Ignition server.

- **Windows folder**: C:\Program Files\Inductive Automation\Ignition\user-lib\pylib
- **Linux folder**: /var/lib/ignition/user-lib/pylib
- **Mac OS X folder**: /usr/local/ignition/user-lib/pylib

Once the python file is in that folder, you can then import the library into a script just like any of the standard libraries.

> Ignition uses Python version 2.7. This means that any imported libraries must be compatible with Python 2.7.

Accessing Java

Scripting in Ignition executes in the java based implementation of Python called Jython. (See Python or Jython?). While this doesn't have any great effect on the Python language itself, one of the great side benefits is that your Python code can seamlessly interact with Java code as if it were Python code. This means that your Python code has access to the entire Java standard library.

To use Java classes, you simply import them as if they were Python modules. For example, the following code will print out all of the files in the user's home directory. This code uses the Java classes `java.lang.System` and java.io.`File` to look up the user's home directory and to list the files. Notice that we can even use the Python-style for loop to iterate over a Java sequence.

Python - Accessing Java

```
# Importing the appropriate java libraries.
from java.lang import System
from java.io import File

# Used to look up the files in the users home directory.
homePath = System.getProperty("user.home")
homeDir = File(homePath)

# Loops through the list of files and prints them.
for filename in homeDir.list():
   print filename
```

You can find the reference documentation for the Java standard class library (also known as, the "*JavaDocs*") at: http://docs.oracle.com/javase/8/docs/api/

Subclassing Java

You can also create Python classes that implement Java interfaces. You do need some understanding of Java and object-oriented programming concepts, which are outside the scope of this manual. To create a Python class that implements a Java interface, you simply use the interface as a superclass for your Python class. For example, we could augment the example above to use the overload java.io.File.list(FilenameFilter). To do this, we'll need to create a `FilenameFilter`, which is an interface in Java that defines a single function:

```
boolean accept(File dir, String name)
```

To implement this interface, we create a Python class that has `java.io.FilenameFilter` as its superclass, and implements that Java-style function in a Python-esque way.

Python - Implementing Java Interfaces

```
# Importing the appropriate java libraries.
from java.lang import System
from java.io import *
```

```
# This sets up an extension filter that can check the file extension. Txt is the default.
class ExtensionFilter(FilenameFilter):
   def __init__(self, extension=".txt"):
      self.extension=extension.lower()

   def accept(self, directory, name):
      # make sure that the filename ends in the right extension
      return name.lower().endswith(self.extension)

# Used to look up the files in the users home directory.
homePath = System.getProperty("user.home")
homeDir = File(homePath)

# Prints out all .txt files. Txt is provided if nothing is specified.
for filename in homeDir.list(ExtensionFilter()):
   print filename

# Prints out all .pdf files.
for filename in homeDir.list(ExtensionFilter(".pdf")):
   print filename
```

Related Topics ...

- Built-in Functions

User Defined Functions

Functions

A function is code that can be called repeatedly from other places. Functions can have parameters passed into them, and may return a resulting value. Some functions, like `len`, are built-in. Some functions, like `system.gui.messageBox()`, are part of the scripting libraries provided by Ignition. Some functions, like `math.sqrt()`, are provided by the Python standard library. However, functions can also be defined in a script that can be used later on in the script. In these user defined functions, you give the function a name and some code that will run when the function is called. Then later on in the script, you can call the function by its name and it will run the code specified in the function. This is useful, because it allows you to run a segment of code many times without having to repeat it within the script.

Functions are invoked by using their name followed by an argument list surrounded in parentheses. If there are no arguments, you still need an open and close parenthesis.

On this page ...

- Functions
  - Defining Functions
  - Functions Arguments
  - Keyword Arguments
  - Functions Are Objects
  - Where Can Functions Be Defined
- Function Scope

Defining Functions

Functions are defined using the `def` keyword. A function needs a name and a list of the arguments that it can be passed. For example, this code defines a function that prints "Hello World!".

Python - Defining a Function

```
# First we define our function.
def printHW():
   print "Hello World!"

# We can then call our function.
printHW()
```

INDUCTIVE UNIVERSITY

Functions

Watch the Video

Functions Arguments

When a function accepts arguments, the names of those arguments become variables in the function's namespace. Whatever value was passed to the function when it was invoked becomes the value of those variables. Arguments can have default values, which makes them optional. If an argument is omitted, then its default value will be used. The following code defines a function called **cap**, which will take a number, and make sure it is within an upper and lower limit. The limits default to 0 and 100.

Python - Defining Arugments

```
# We first define our function. Notice that we have 3 different arguments.
# x, min, and max. The min and the max are set to equal 0 and 100 respectively.
def cap(x, min=0, max=100):

        # Check if x is less than the min, return the min if true.
        if x < min:
                return min

        # Check if x is greater than the max, return the max if true.
        elif x > max:
                return max

        # Return the value if it is within the bounds.
        else:
                return x

# We can then see the outcome by running our function with a few different parameters.
# This will print out a 40, since it is within the bounds.
print cap(40)

# This will print out "0", since it is less than the min of 0.
print cap(-1)

# This will print out "100", since it is greater than the max of 100.
print cap(150)
```

```
# This will print out "150", because it uses a max of 200 instead of the default 100.
print cap(150, 0, 200)
```

Keyword Arguments

In Ignition, some complicated script functions are designed to take keyword arguments instead of normal parameters. In the description for those functions, you may see the following info box in this User Manual:

> This function accepts keyword arguments.

Arguments can also be specified by keyword instead of by position. In the example above, the only way someone would know that the 200 in the last call to `cap` specified the max is by its position. This can lead to hard-to-read function invocations for functions with lots of optional arguments. You can use keyword-style invocation to improve readability. The following code is equivalent to the last line above, using 200 for the `max` and the default for the `min`.

Python - Using Keyword Arguments

```
print cap(150, max=200)
```

Because we used a keyword to specify that 200 was the `max`, we were able to omit the `min` argument altogether, using its default. However, using a keyword argument before a non-keyword or positional argument is not allowed.

Python - Non-keyword Argument

```
# This would fail, because the function isn't sure what 150 is being used for.
print cap(max=200, 150)
```

Functions Are Objects

Perhaps one of the most foreign concepts for new Python users is that in Python, functions are first-class objects. This means that functions can be passed around to other functions (this concept is similar to the idea of function pointers in C or C++).

Suppose we wanted a general way to filter a list. Maybe sometimes we want the odd entries, while other times we want even ones. We can define a function called `extract` that takes a list and another function, and returns only entries that "pass" through the other function.

Python - Functions Passed to Other Functions

```
# We define a function that checks if the value passed in is odd.
def isOdd(num):
        return num % 2 == 1

# We define a function that checks if the value passed in is even.
def isEven(num):
        return num % 2 == 0

# We define a function that inserts our list into the appropriate function and returns valid values.
def extract(filterFunction, list):
        newList = []
        for entry in list:
                if filterFunction(entry):
                        newList.append(entry)
        return newList

# Prints out [0, 2, 4, 6, 8]
# Notice that isEven as not invoked, but passed to the filter function.
print extract(isEven, range(10))
```

Where Can Functions Be Defined

User Defined Functions can be defined anywhere that a script is used. As stated before, they are useful to run segments of code multiple times without having to repeat it. They are also used extensively in project scripts where multiple functions can be defined in a single script module. Finally, some special Ignition System functions like system.gui.createPopupMenu or the runScript Expression function use functions as arguments.

Function Scope

The concept of scope is very important in all programming, and Python is no exception. Scope defines what names are directly accessible without any qualifiers. Another way to put this is that the scope determines what variables are defined. In Python, a variable is defined at the time that it is assigned. What scope it belongs to is also defined by where the assignment occurs.

Pseudocode - Defining a Function for Scope

```
# On this line, there is no variable 'x' in scope.
doSomeWork()

# Now 'x' is defined in our scope, because we've assigned a value to it
x = 5

# This will work because x is in scope.
print x
```

When you define a function, that function gets its own scope. Variables that are assigned within that function body will not be available outside of the function.

Python - Variables Defined within a Function Not Available Outside Scope

```
# x is local to myFunction() because this is where it is defined.
def myFunction():
   x = 15
   print x

# This will fail, because x is not available in the outer scope
y = x + 10
```

Related Topics ...

- Built-in Functions
- Libraries

Scripting in Ignition

Where Is Scripting Used?

Python is used in many places in Ignition. Each location has its own events that trigger your scripts to run, and add functionality to your projects in different ways. The most apparent place is in event handlers on components and other objects in Vision Clients and Perspective Sessions.

Script Scope

One important thing to keep in mind before scripting in Ignition, is to understand the concept of scope. Within Ignition, there are different scopes: the Gateway Scope, the Perspective Session scope, and the Client Scope. Where a script is run from determines what scope it is running in. This is important because it determines what system functions can be run, what resources the script can interact with, and where the output will be written to. For example, running a script on a Tag is run in the Gateway Scope and the output is sent to the Gateway console (i.e., wrapper.log file) because Tags are stored in the Gateway. This means that the script will not be able to access any client level resources such as windows or components that you may have open in the Client. Additionally, some of the system functions like system.gui.errorBox only work in the "Client Scope," so you will not be able to use them in the script on the Tag.

"Client Scope" scripts, however, execute on the running client (and also in Designer when testing, but only in Preview Mode). For example, if a component on a window is running a script, its values are isolated to the client, and the output will be displayed on the Designer/Client output console.

System Functions

Ignition comes with a group of system functions, called the System Library. Using a system function is simple. For example, the following code will access the value of a Tag.

Python - Simple Script Using a System Function

```
value = system.tag.read("tagPath").value
```

A complete list of these functions (with their definitions) is available wherever you can add a script. Just type **system.** and then press **Ctrl+Space** to get a list of all the functions available. If you keep typing, the list will even be automatically narrowed down for you! Additionally, the Scripting Functions page in the appendix contains complete documentation for the built-in system functions.

On this page ...

- Where Is Scripting Used?
  - Script Scope
  - System Functions
- Components
- Client, Gateway, and Session Event Scripts
- Project Scripts
- Tag Scripts
- Reporting
- Alarming
- Sequential Function Charts

INDUCTIVE UNIVERSIT

Scripting in Ignition

Watch the Video

INDUCTIVE UNIVERSIT

System Library

Watch the Video

Components

Both Perspective and Vision offer component based scripting triggers, providing a means to execute a script under a number of different situations, such as a user interacting with a component or a component property value changing. For more information on how both module handle component based scripts, take a look at the Scripting in Perspective and Scripting in Vision sections.

Client, Gateway, and Session Event Scripts

Scripts can be set to activate on specific events that occur during runtime. For example, you can triggering a script to run when a vision client starts, or on certain time intervals.

More information on these events can be found on the Client Event Scripts, Gateway Event Scripts, and Perspective Session Event Scripts pages.

Project Scripts

You can create your own reusable blocks of code in the Project Library. Once configured, these functions can be called from anywhere in a project, just like our system.* functions.

Tag Scripts

Once Enabled, these scripts are fired whenever a Tag value changes or an alarm event happens. You can use them for additional diagnostics, to set additional Tags, or to react to an alarm event. Because these events are on Tags, they are Gateway Scoped.

Reporting

Reporting uses scripting in many different ways to help increase the effectiveness of the report. Scripting in Reports is used to create and modify data sources, manipulate charts, and set up a script as a scheduled report action.

Alarming

The Alarm Notification system can also use scripting to great effect. A script block allows a script to be run within the pipeline, allowing data to be manipulated as the alarm event travels through the pipeline. Additionally, scripting can be used to generate a custom roster of users at runtime, giving full customization to who gets notified by the alarm event.

Sequential Function Charts

Sequential Function Charts (SFCs) are a flowchart of blocks that run scripts. They are executed in a specific sequential order along with some logic to potentially loop or call other charts. The scripts here can interact with the Gateway, and provide greater control when each step needs to complete before the next one can begin in multi-step processes.

Related Topics ...

- Scripting Data Source
- Sequential Function Charts
- Notification Block
- Tag Event Scripts
- Client Event Scripts
- Gateway Event Scripts
- Perspective Component Methods
- Perspective Session Event Scripts
- Project Library

In This Section ...

Getting Started with Scripting in Ignition

Overview

The best way to get started with scripting in Ignition is to write some very simple scripts and print them within Ignition. Let's take the phrase, "Hello World," and print it within Ignition. To show how flexibile Ignition is, there are actually a few ways to easily do this. One way is to use the Scripting Console, and another way is to use a Button component. Both examples are shown below.

On this page ...

- Overview
- Example - "Hello World" Using the Script Console
- Example - "Hello World" on a Button Component
- Example - Using a Message Box

Example - "Hello World" Using the Script Console

When testing or writing a new script, the Scripting Console is very useful since you can immediately get some feedback on the results of the script. When learning Python, it is a great place to start since you don't have to create a window or component before you begin writing your code.

1. In the **Designer**, open the Script Console from the Menu Bar: **Tools > Script Console.**

2. The Script Console will appear. Type the following code, or simply copy and paste it into the text area in the Multiline Buffer on the left side of the Script Console:

Python - Simple Print

```
print 'Hello World'
```

3. Click the **Execute** button at the bottom of the Script Console. You should see the message "Hello World" appear in the Interactive Interpreter on the right side of the Script Console.

Example - "Hello World" on a Button Component

Scripts are commonly located on components in a window. In this example, let's add a script on a Button component, and print out "Hello World" when the Button is pressed.

1. In the **Designer**, create a new Main Window. Give it a meaningful name if you like. (We won't need to reference the name of the window in any of these examples).

2. Drag a standard **Button** component to your window.

3. Let's add a script that triggers when the Button is pressed. With the Button component selected, right click on **Scripting** to open the **Component Scripting** window. Alternatively, you can double-click the Button to open the scripting window.

4. Under Event Handlers, select the **action > actionPerformed** option, and click the **Script Editor** tab. Make sure the **actionPerformed** event is highlighted. If an event is not selected, all of the remaining features in this window will be disabled, so we will not be able to write a script. Additionally, if the wrong event is selected, then our script will not trigger when we expect it to.

5. In the Script Editor, you will see a large editable Text Area. This is where we will type our script.

6. Let's generate a message that shows "Hello World!" Use the following code to get started. Make sure the word "print" lines up exactly to the left edge. Indention in Python means something, so we need to avoid starting our lines with a space or tab unless we're denoting a block of code under something like an if-statement or function definition. Click **OK** to close the **Component Scripting** window.

Python - Simple Print

```
print "Hello World!"
```

Notice the **actionPerformed** event is blue and bold. This means there is a script on this event. This is useful to know in situations where a component has scripts on multiple events.

Additionally, an asterisk character (*) is next to the event. This means you have not applied/saved the changes to the script. The asterisk will disappear when you press either **OK** or **Apply**, and reappear whenever you make a change to the script. If you see this, then it means you may want to save any changes you made.

The **Script Editor** tab also has a blue color, denoting where the script is. An event will only ever have a script located on a single tab at any time. If a new tab is selected and configured, it will wipe out the work on the prior tab, for example, writing a script on **Script Editor** and then

configuring the **Navigation** tab will erase the script on the **Script Editor** tab.

7. Now we can test the script. Place the **Designer** into Preview Mode, and press the Button. If everything is working as intended, then it should appear as if nothing happened. This is because we used `print` in our script, which always outputs to a console, as opposed to popping up at the user. This means we need to open the Designer's **Console** to see the results of our script. At the Designer's menu bar, select **Tools > Console**. This will make the console appear.

8. There may be a large amount of text in the Output Console. The Designer logs many different types of activities and events here, including polling events from components on other windows that are currently open in the Designer. However, the most recent events should be towards the bottom. As a quick tip, you can also click the eraser icon to clear out the console, and then press your **Button** again to generate a new entry like we did in this example.

We can now see where print statements go when called from a component, but this isn't too useful, as we don't want our users to open the console to see messages. Print statements are very useful when troubleshooting a problematic script, or even when testing a new script. Keep this in mind as you start to delve into scripting more.

Example - Using a Message Box

Now that we've seen how to print to the Output Console, lets make our message appear when we call it. This time, we will modify the script on the Button component to bring up a window that the user will see.

1. Open the **Component Scripting** window again, and find the script on the Button.

2. We will use one of Ignition's built-in functions, called system.gui.messageBox, to display the message. This will make a message box appear, which is a modal window that we can pass a string to. Remove your old code on the Button, as we will be replacing it. Start by typing the following:

Pseudocode - Built-in Functions Begin with `system.`
`system.`

3. With the Text Cursor just to the right of the ".", hold The **Ctrl/Cmd** key and press the **Spacebar**. This will make the Autocompletion popup appear. This lists all of the available system functions. Start by clicking on **gui**, and then **messageBox**. Note, that you can still type while the popup is open to filter the results in the list.

4. Once you've selected **messageBox**, click on it, and it should start some of the code for you automatically.

5. Complete the code by placing some parentheses and a message as a string. Alternatively, you can copy the example below.

Python - Using a System Function to Write to a Message Box

```
system.gui.messageBox('Hello World')
```

6. Place the **Designer** into **Preview Mode**, and click your Button. You will see a Message Box appear displaying the message we passed to the function.

Related Topics ...

- Python Scripting
- Scripting Functions
- Component Events

Gateway Event Scripts

Gateway Event Scripts Overview

Gateway Event Scripts are scripts that run directly on the Gateway. They are useful because they always run, regardless if any sessions or clients are open. They also provide a guaranteed way to make sure only a single execution of a particular script occurs at a time, as opposed to placing a script in a window, as there could be multiple instances of the window open at a given point of time.

Note that even though Gateway Event Scripts run on the Gateway, they're still considered a project resource. Project backups will include any Gateway Event Scripts.

The Gateway Event scripting workspace is located in the Scripting menu of the Designer or in the Project Browser under **Scripting > Gateway Events**.

Other Event Scripts

The content on this page will focus primarily on Gateway Event Scripts. However, there is some overlap with Client Event Scripts, as they have similar events. More information can be found on the Client Event Scripts and Perspective Session Event Scripts pages.

Note: System functions are available for both Client Event Scripts and Gateway Event Scripts, but some system functions are specific to either one or the other. When you're writing event scripts, it's important to remember the scope of where you're writing the script: Client or Gateway. You can check Scripting Functions in the Appendix to see list of all system functions, their descriptions, and what scope they run in.

On this page ...

- Gateway Event Scripts Overview
  - Other Event Scripts
- Startup Script
  - Gateway Startup Behavior
- Update Script
- Shutdown Script
  - Gateway Shutdown Behavior
- Timer Scripts
  - Timer Script Settings
- Tag Change Scripts
  - Tag Change Script Interface
- Message Scripts
  - Gateway Message Handler Settings
  - The Payload
  - Calling Message Handlers
- Troubleshooting Gateway Scripts

INDUCTIVE UNIVERSITY

Gateway vs Client Event Scripts

Watch the Video

Startup Script

The Startup Script event runs at the startup of the Gateway. Additionally, if the project is restarted in someway, such as by making a change to a Gateway Event Script and saving, then the Startup Script will be called. This means that while editing scripts frequently in the Designer, the startup and shutdown events may happen frequently.

INDUCTIVE UNIVERSIT

Startup Scripts

Watch the Video

Gateway Startup Behavior

There is a specific order to when the various startup scripts are run. When troubleshooting your Gateway startup times, consider the following:

1. **Gateway starts** - The Gateway will start as an OS service, and start the context. No startup scripts can run before this is complete.
2. **Projects are started** - This includes all of the Gateway scoped items in the projects such as Transaction Groups, SFCs, etc. This does not refer to launching clients, and no clients can be automatically launched at this time. All **Gateway Startup Scripts** are run at this time for each project. Note: if you copied a project, always check for Gateway scoped events such as these. You generally don't want a Gateway Startup Script to run twice because it is in two projects.

Update Script

The Update Script event runs after a project is saved or updated on the Gateway. This enables you to insert a script that will run every time a project is saved.

Shutdown Script

The Shutdown Script event runs at the shutdown of the project, which means it can be used as a way to trigger a script when the Gateway has to be restarted. It allows you to run a piece of code as the shutdown is occurring. After the script completes, the shutdown will finish.

Note that the Shutdown Script event only gets called if the Gateway is requested to shut down: if the computer power is lost abruptly (power outage, hard restart, etc.) this shutdown script will not run.

Gateway Shutdown Behavior

Similar to how a Startup Script behaves, "Shutdown" in the context of this event means "project shutdown", so shutting down the Gateway would trigger this event, as well as disabling the project containing a script on this event. Additionally, making a change to a Gateway Event Script in the Designer, and then saving the project will cause the project to restart, which means this event can get called by simply making changes in the Designer and saving.

Timer Scripts

The Timer Scripts execute periodically on a timer at a fixed delay or rate. This allows you to set up a sort of heartbeat that can run on the Gateway. This is the ideal event to use if you need the Gateway to periodically perform some scripting task.

INDUCTIVE UNIVERSITY

Timer Scripts

Watch the Video

Since multiple Timer Scripts can be added, there are separate buttons that allow you to manage each Timer Script.

- **Add Timer Script** - Adds a new Timer Script.
- **Remove Timer Script** - Will delete the selected Timer Script.
- **Modify Timer Script** - Will modify the settings for the selected Timer Script.

Timer Script Settings

Below is an overview of the settings for a Timer Script.

- **Name**: The name of the Timer script. Names must be unique per project, so two timer scripts in the same project cannot have the same name.
- **Delay**: The delay period in milliseconds. The meaning of this setting is dependent on the **Delay Type** setting.
- **Enabled**: Allows you disable the Timer Script when set to false.
- **Delay Type**: Determines how the **Delay** setting is utilized.
    - A **Fixed Delay** timer script (the default) waits for the given **Delay** between each script invocation. This means that the script's rate will actually be the delay plus the amount of time it takes to execute the script. This is the safest option since it prevents a script from mistakenly running continuously because it takes longer to execute the script than the delay.
    - **Fixed Rate** scripts attempt to run the script at a fixed rate relative to the first execution. If the script takes too long, or there is too much background process, this may not be possible. See the documentation for java.util.Timer.scheduleAtFixedRate() for more details.
- **Threading**: Determines which thread this script should run in. In other words, this setting allows you to specify if you want this timer script to share execution resources or not. The rule of thumb here is that quick-running tasks should run in the shared thread, and long-running tasks should get their own dedicated thread.
    - The **Shared** setting means that all timer scripts will share a thread. This is usually desirable, as it prevents creating lots of unnecessary threads: threads have some overhead, so a small amount of resources are used per thread. However, if your script takes a long time to run, it will block other timer tasks on the shared thread.
    - The **Dedicated** setting will create a separate thread specifically for the timer script to use. This setting is desirable when your scripts executions must be as consistent as possible, as other timer scripts can't slowdown or otherwise impact the execution of a script in a separate thread.

Tag Change Scripts

The Tag Change Script event allows you to specify any number of Tags, and trigger a script when one of them change. Since these execute based on a Tag changing value, Tag Change Scripts are ideal when you need a script to run based on some some signal from a PLC.

Having the Tag Change Scripts run in the Gateway Scope means that the scripts are active as long as the Gateway is running. Thus, you do not need a client or session to be open for a Gateway Tag Change Script to execute. When executing, each Tag Change Script runs in a separate thread. This prevents long running scripts from blocking the execution of other Tag Change Scripts.

Note: Due to their nature, Client Tags will not trigger Gateway Tag Change Scripts. However, there are other similar events, such as a Client Event Script, that enable you to trigger a script based on a Client Tag changing value.

INDUCTIVE UNIVERSIT

Tag Change Scripts

Watch the Video

Tag Change Script Interface

Tag List

Lists all of the available Tag Change scripts in the project. Two icons are available below the list:

- **Add Script** - Adds a new Tag Change Script to the list.
- **Remove Script** - Removes the currently selected script from the list.

Tags Tab

The **Tags** tab contains settings for the script. See the **Script Settings** description below

- **Script Name** - The name of the script. Script names must be unique per project.
- **Enabled** - Determines if the script is active or not. Set to false to disable the script.
- **Change Triggers** - When the Tag changes, the script can trigger based on the **Value**, **Quality**, and/or **Timestamp**. Note that regardless of how many triggers changed, the script only executes once per tag, so leaving all triggers enabled will not trigger three executions each time the Tag changes.
- **Tag Paths** - A list of Tag paths to monitor. When any of the Tags listed in this area change, the script will trigger. Note that the list is not comma separated: new paths are specified each line.
    - **Tag Browser** - Opens a Tag Browser window, allowing you to quickly lookup and add Tag paths to the Tag Change Script.
    - **Path Diagnostic** - Click this icon to verify the paths specified under the **Tag Paths** list. This is useful for checking for typos in the list.
    - Wildcards can be used at the end of configured tag paths. For example, [provider]folder/*.

Script Tab

The **Script** tab is where the Python script associated with this event will be placed. Two icons are available below the list:

- **Add Script** - Adds a new Tag Change Script to the list.
- **Remove Script** - Removes the currently selected script from the list.

Tag Change Objects

Tag Change Scripts contain several built-in objects that are useful for inspecting the event, such as seeing what value the Tag changed to. These objects are listed below:

The initialChange Value

A variable that is a flag (0 or 1) which indicates whether or not the event is due to the initial subscription or not. This is useful as you can filter out the event that is the inital subscription, preventing a script from running when the values haven't actually changed.

```
if not initialChange:

    # Do something useful here
```

The newValue Object

A QualifiedValue object that contains the following methods

Method	Description	Usage Example
`getValue()`	Returns the new value on the tag	`newValue.getValue()`
`getQuality()`	Returns the new quality on the tag	`newValue.getQuality()`
`getTimestamp()`	Returns the new timestamp value	`newValue.getTimestamp()`

The event Object

This object offers some additional utility, such as accessing the previous values on the tag.

Method	Description	Usage Example
`getCurrentValue()`	Returns a QualifiedValue object (similar to the newValue object), representing the new values on the tag.	`event.getCurrentValue().getValue()`
`getPreviousValue()`	Returns a QualifiedValue object, representing the values on the tag before the change.	`event.getPreviousValue().getValue()`
`getTagPath()`	Returns a TagPath object, that can be further examened for details about the path on the tag that changed. See the TagPath Object table below. Additionally, the TagPath object can easily be turned into a string, providing quick access to the path of the tag that changed value.	`event.getTagPath()`
`getValue()`	Returns the new value on the tag, similar to getCurrentValue(). This method is a convenient way to retrieve just the new value, without needing to interact with the QualifiedValue object returned by getCurrentValue().	`event.getValue()`

The TagPath Object

These methods are available on the TagPath object returned by `event.getTagPath()` calls

Method	Description	Usage Example
`getItemName()`	Returns the name of the item at the end of the path, which can be used to get the name of the tag that changed.	`event.getTagPath().getItemName()`
`getParentPath()`	Returns the path to the Tag's parent folder.	`event.getTagPath().getParentPath()`

Message Scripts

Message Handlers allow you to write a script that will run in the scope they are located in, but they can be invoked by making a call from other projects or even other Gateways. They can be called using three different scripting functions: system.util.sendMessage, system.util.sendRequest, and system.util.sendRequestAsync.

Under the list of handlers, three small buttons allow you to add, remove and manage your handlers.

- **Add Message Handler** - Will add a message handler.
- **Remove Message Handler** - Will delete the highlighted message handler.
- **Modify Message Handler** - Will modify the settings for the highlighted message handler.

Gateway Message Handler Settings

When adding or modifying a message handler, a Message Handler settings window will popup.

The following settings are available:

- **Name** - The name of the message handler. Each message handler must have a unique name per project.
- **Threading** - Determines the threading for the message handler. Contains the following options:
  - **Shared** - The default way of running a message handler. Will execute the handler on a shared pool of threads in the order that they are invoked. If too many message handlers are called all at once and they take long periods of time to execute, there may be delays before each message handler gets to execute.
  - **Dedicated** - The message handler will run on its own dedicated thread. This is useful when a message handler will take a long time to execute, so that it does not hinder the execution of other message handlers. Threads have a bit of overhead, so this option uses more of the Gateway's resources, but is desirable if you want the message handler to not be impeded by the execution of other message handlers.
- **Security** - Allows you to specify security zone and role combinations that are allow to request this message handler.

The Payload

Inside the message handler is your script. The script will have a single object available to it, the **payload**. The payload is a dictionary containing the objects that were passed into it. In essence, the payload is the mechanism that allows you to pass the message handler values.

The payload is simply a python dictionary, so extracting values involves specifying the key:

Pseudocode - Payload Values

```
value1 = payload["MyFirstValue"] #"MyFirstValue" is the key that is
associated with a value. We are taking the value associated with
MyFirstValue, and assigning it to value1.
value2 = payload["MySecondValue"] #Similarly, we are taking the value
associated with MySecondValue and assigning it to value2.
```

Calling Message Handlers

Once you have your message handlers created, you can then call them from a script using one of three scripting functions: system.util.sendMessage, system.util.sendRequest, and system.util.sendRequestAsync. These functions allow you to call a message handler in any project, even if the project that the message handler resides on is different from the one you are calling it from. The message handler will then execute in the scope in which it was created, and will use any parameters that you pass in through the payload.

Pseudocode - Calling a Message Handler

```
project="test"
messageHandler="My Message Handler"
myDict = {'MyFirstValue': "Hello", 'MySecondValue': "World"}
results=system.util.sendMessage(project, messageHandler, myDict)
```

Troubleshooting Gateway Scripts

While they are technically project resources, remember that Gateway Event Scripts technically run on the Gateway. Thus the Status section of the Gateway is useful for diagnosing issues with Gateway Event Scripts.

Related Topics ...

- Client Event Scripts
- Perspective Session Event Scripts

Project Library

Scripts under the Project Library are a project-based resource that allows user created Python scripts to be configured. Objects and functions created in a Project Library script can be called from anywhere in the project. Project Library scripts are accessible from the Project Browser, under the **Scripting** item.

Additionally, a single project can be designated as the **Gateway Scripting Project**, meaning that scripts defined in the stated project can be called from the Gateway scope.

On this page ...

- Add a Script
  - Scripts and Packages
  - Usage Example
- Gateway Scripting Project

Add a Script

To add a a Project Library script, right click the **Project Library** item and click the **New Script** option.

Scripts and Packages

There are two main types of resources under the Project Library.

- **Scripts** - Each script resource can contain many functions and objects.
- **Packages** - Each package effectively acts as a folder, allowing you to better organize each script resource.

Usage Example

For example, let's suppose you added the following script module named **`myFuncs`**, whose body is shown below.

Python

```
def hello():
        return "Hi there!"
```

Once we **save** our project, we can now call this function from anywhere within the project using the following syntax

Python - Calling the Project Script

```
myFuncs.hello()
```

Note: Project Library scripts are not accessible to the other resources until the project is saved.

For example, we could open the Script Console (Tools menu > Script Console), write the following, and execute the script.

Each script resource can contain multiple functions and objects. As you add new function definitions, the list on the right will populate giving you a quick way to navigate through long scripts.

Gateway Scripting Project

Project Library scripts are normally only accessible from the project they were defined in. Thus objects that exist in other scopes, such as Tags that exist in the Gateway scope, are unable to call Project Library scripts. Attempting to do so will result in Gateway log errors stating that "global name 'yourScript' is not defined'.

The exception to this rule is the Gateway Scripting Project. This project is specified by the **Gateway Scripting Project** property, which is set in the Con fig section of the Gateway Webpage under Gateway Settings. Entering in the name of a project under this property allows the Gateway access to Project Library scripts configured in the specified project.

Thus, if the **myFuncs** library in the prior section was configured in a project named "Tester"

We could use that name in the Gateway Scripting Project.

After we save, Tags and other Gateway-scoped resources can then start calling any of the scripts from our Project Library in the "Tester" project.

Related Topics ...

- User Defined Functions
- Scripting in Ignition

Web Services, SUDS, and REST

Web Services Overview

Web services are software solutions that allow for interacting with machines residing on a network. In short, web services are nothing more than web pages for machines. They provide a standard way for a third party to request and receive data from a piece of hardware on the network without having to know anything about how that machine works.

On this page ...

- Web Services Overview
  - Protocols
  - Can Ignition Make RESTful Calls?
- Common Web Services Workflow

Protocols

There are two common approaches to Web Services in Ignition: making a HTTP (HyperText Transfer Protocol) method call, or making a SOAP (Simple Object Access Protocol) call.

- HTTP methods, such as GET and POST, are called using the built-in system functions, such as system.net.httpGet(). More information on HTTP calls can be found on the HTTP Methods page.
- SOAP is XML based, and typically requires a third party Python library. There are many third party Python libraries that utilize SOAP, however, documenting them goes outside the scope of this manual (as do all third party libraries).

The approach you choose typically depends on the server you're trying to make calls to: more specifically, the protocol(s) it supports.

> **What About the SUDS Library?**
>
> The SUDS library, a library that used to come included with the Python Standard Library, offered SOAP based functionality. However, SUDS development has been halted, and is no longer included in the standard library.
>
> In the interest of posterity, the legacy SUDS documentation has been condensed and can be found on the SUDS - Library Overview page. Note that the legacy documentation should be considered deprecated.

Can Ignition Make RESTful Calls?

Yes it can! However, it is important to understand that REST is an architecture, **NOT** a protocol. Instead, REST utilizes and describes how a protocol should be used. Thus, a RESTful architecture could use both of the protocols mentioned on this page, although HTTP is far more common.

Common Web Services Workflow

While all Web Services follow the same standards, they all do different things. They wouldn't be worth anything if you didn't get the information you need, or if they contained a lot of excess data. If you are unfamiliar with a particular Web Service, there are a few things that you can do to figure out what data is available and how to get it.

1. Identify a Web Service that you will be using. Usually the Web Service has an API somewhere documenting how requests should be made.
2. Write a script to pull some information from the Web Service. If using HTTP, this could mean starting with a GET call, where as SOAP would involve retrieving the WSDL (Web Services Description Language). In both cases, you may need to find a way to authenticate against the server (usually with some user credentials or an auth token, the API for the service would have more details).
3. Once you have the results from the GET/WSDL, identify the information or functions you want to use.
4. Write a script to use that function and return your values.
5. Parse the results and use them. This can be for display, saving to a database, or anything else you need.

Note: Web Services sometimes take a lot of time to return results, especially the first time they are called. If you put your Web Services script in a button, the client will freeze until the call is complete (this is because the event handlers are run on the GUI thread). It's a good idea to use system.util.invokeAsynchronous() or add a waiting image to your screen to let the user know Ignition is working as expected.

Related Topics ...

- WebDev Module

In This Section ...

HTTP Methods

Overview

Web services calls typically require some protocol to make requests. HTTP is an incredibly common protocol, so this page will introduce how to incorporate these calls in a python script. Note that all of the examples on this page can be easily called with the Script Console, but can be utilized through some other means (the actionPerformed event on a Button).

On this page ...

- Overview
- Finding an Endpoint
- Making the Call
- Parsing the Results
- Make the Results Human Readable
- Troubleshooting HTTP Methods
  - HTTP Response Codes

Finding an Endpoint

Ignition doesn't natively expose an endpoint for web services calls, so we'll have to utilize a service of some sort for the examples on this page. Fortunately, there are many public services we can utilize, such as Yahoo Weather. From here we can generate an endpoint to use. At the time of this writing, we're using the following endpoint to retrieve the sunset time in Maui,HI:

Pseudocode - Finding a Resource

```
https://query.yahooapis.com/v1/public/yql?q=select%20astronomy.sunset%
20from%20weather.forecast%20where%20woeid%20in%20(select%20woeid%20from%
20geo.places(1)%20where%20text%3D%22maui%2C%20hi%22)&format=json&env=store%
3A%2F%2Fdatatables.org%2Falltableswithkeys
```

Making the Call

To retrieve the results of this information in Ignition, we can use system.net.httpGet() to fetch the results of this call. We can try the following script in the Scripting Console:

Python - Creates a variable to Store the Endpoint and Retrieves Results

```
# Create a variable to store the endpoint.
myEndpoint = "https://query.yahooapis.com/v1/public/yql?q=select%20astronomy.sunset%20from%20weather."
myEndpoint += "forecast%20where%20woeid%20in%20(select%20woeid%20from%20geo.places(1)%20where%20text%3D"
myEndpoint += "%22maui%2C%20hi%22)&format=json&env=store%3A%2F%2Fdatatables.org%2Falltableswithkeys"

print system.net.httpGet(myEndpoint)
```

Printing this results in the following Python string:

Python - Results

```
{"query":{"count":1,"created":"2017-10-31T16:44:19Z","lang":"en-US","results":{"channel":{"astronomy":
{"sunset":"5:50 pm"}}}}}
```

Parsing the Results

If we wanted to extract a single value ouf of the results, we have a number of approaches. One useful approach would be to turn this JSON string into a Python Dictionary. This way we can single out a key instead of using regex or looking for substrings (both valid approaches in their own regard).

When presented with a JSON string, we can call system.util.jsonDecode() to turn a JSON string into a native Python Object. Thus, we can modify our code to the following:

Python - Parsing the Results Code

```
# Create a variable to store the endpoint.
myEndpoint = "https://query.yahooapis.com/v1/public/yql?q=select%20astronomy.sunset%20from%20weather."
myEndpoint += "forecast%20where%20woeid%20in%20(select%20woeid%20from%20geo.places(1)%20where%20text%3D"
myEndpoint += "%22maui%2C%20hi%22)&format=json&env=store%3A%2F%2Fdatatables.org%2Falltableswithkeys"
```

```
# Make the GET call.
results = system.net.httpGet(myEndpoint)

# Convert the JSON string into a Python object. In this case, it results in a Dictionary.
decodedDict = system.util.jsonDecode(results)

# Now we can treat the results like a nested dictionary, thus we can specify the "query" key,
# and then the nested "created" key to return just the date.
print decodedDict.get("query").get("created")
```

Now we can easily retrieve a single value bay specifying key names on the results.

Make the Results Human Readable

Now that we know how to extract the results, we should clean up the output of the GET call. The JSON string returned by the endpoint could potentially be long and cumbersome to read through for a human, but we can use Python's built-in **pprint** library to pretty print the results.

Python - Now with Pretty Print

```
# Import the pprint library
import pprint

# We'll instantiate an instance of PrettyPrinter, and store it in a variable named pp.
pp = pprint.PrettyPrinter(indent=4)

# Create a variable to store the endpoint.
myEndpoint = "https://query.yahooapis.com/v1/public/yql?q=select%20astronomy.sunset%20from%20weather."
myEndpoint += "forecast%20where%20woeid%20in%20(select%20woeid%20from%20geo.places(1)%20where%20text%3D"
myEndpoint += "%22maui%2C%20hi%22)&format=json&env=store%3A%2F%2Fdatatables.org%2Falltableswithkeys"

# Make the GET call.
results = system.net.httpGet(myEndpoint)

# Convert the JSON string into a Python object. In this case, it results in a Dictionary.
decodedDict = system.util.jsonDecode(results)


# Print out the dictionary in an easy to read format.
print pp.pprint(decodedDict)
```

The resulting output, which is much easier to read, looks like the following:

Python - Results

```
{   u'query': {   u'count': 1,
                  u'created': '2017-10-31T16:44:19Z',
                  u'lang': 'en-US',
                  u'results': {   u'channel': {   u'astronomy': {   u'sunset': '5:50 pm'}}}}}
None
```

From here we can see all of the keys that lead to our final value:

```
Interactive Interpreter
Jython 2.5.3 (v2.5.3:3d2dbae23c52+, Nov 17 2012, 11:51:23)
[Java HotSpot(TM) 64-Bit Server VM (Oracle Corporation)] on java1.8.0_131

>>>
{   u'query': {   u'count': 1,
                  u'created': '2017-10-31T16:44:19Z',
                  u'lang': 'en-US',
                  u'results': {   u'channel': {   u'astronomy': {   u'sunset': '5:50 pm'}}}}}
None
>>> |
```

To get to the value for the sunset key, we simply need to address each key along the way:

Python - To Get Value for the Sunset Key

```
print decodedDict.get("query").get("results").get("channel").get("astronomy").get("sunset")
```

Troubleshooting HTTP Methods

When making HTTP calls, it is helpful to be familiar with the status codes that are returned by errors. To demonstrate, we could modify an earlier example:

Python - Returns Error Status Codes

```
# Create a variable to store the endpoint.
myEndpoint = "https://query.yahooapis.com/v1/public/yql?q=select%20astronomy.sunset%20from%20weather."


### Note that the following two lines are commented out, meaning we're about to make a GET call with a bad
/incomplete address.
# myEndpoint += "forecast%20where%20woeid%20in%20(select%20woeid%20from%20geo.places(1)%20where%20text%3D"
# myEndpoint += "%22maui%2C%20hi%22)&format=json&env=store%3A%2F%2Fdatatables.org%2Falltableswithkeys"

print system.net.httpGet(myEndpoint)
```

This will return an error, which looks like the following:

Python - IOError: Response Code 400

```
Traceback (most recent call last):
  File "<buffer>", line 8, in <module>
IOError: Server returned HTTP response code: 400 for URL: https://query.yahooapis.com/v1/public/yql?q=select%
20astronomy.sunset%20from%20weather.
```

Note that HTTP response code **400**, which means bad request, was referenced. This error code is correct because we intentionally used an incomplete address!

HTTP Response Codes

The World Wide Web Consortium has a page dedicated to HTTP response codes, which details all possible error codes. However, several common codes are listed below:

Response	Description

Code	
400	**Bad Request** - The server could not understand the request due to malformed syntax.
401	**Unauthorized** - The request requires some sort of authentication. Potentially some user credentials or an auth token of some kind.
403	**Forbidden -** The server understood what you requested, but is intentionally refusing the request. In some cases, the error message may include a reason why the request was not fulfilled (but not always). Typically, if the server doesn't include a reason, they'll use a 404 error code instead
404	**Not Found** - Your syntax was correct, but the server could not find the resource you were asking for. This could mean a typo, or missing portion of the URL you are using. In this case, double check the address you're specifying. Depending on the configuration, this could also mean that the server does actually have the resource you requested, but doesn't want to confirm its existence (typically due to a security policy. See error code 403 above).

Related Topics ...

- system.net

SUDS - Library Overview

The SUDS Library

The SUDS library is a SOAP-based web services client developed for Python. It is extremely simple to use and practically eliminates the need for the user to understand or even view the WSDL of a web service.

Disclaimer
The SUDS library used to be included in the Python Standard Library. However it has since been removed, meaning you may not have access to it when performing a fresh install of Ignition. Additionally, development on the library has mostly ceased, so any copies you find online may be drastically outdated.

The information on this page will be maintained for legacy users that need to be familiar with the old SUDS library. As a result, this page and its contents should be considered deprecated.

The SUDS library interprets the WSDL file for you and through a couple simple function calls allows you to get a list of the available methods provided to you by the web service. You can then invoke these methods in Ignition through event scripting to send and receive data in your projects. You will have to familiarize yourself with the SUDS library in order to make use of it.

On this page ...

- The SUDS Library
  - Simple Example
  - Beyond the Example
- Complex Arguments

Simple Example

If you read through the SUDS documentation, you'll see that the Client object is the primary interface for most users. It is extremely simple using this object and a few print statements to view a list of available methods provided by the web service you are connecting to. This example will illustrate how to make an initial connect to a web service, print out the list of available methods, and then call one of these methods and display the resulting value in a label on an Ignition window at the push of a button. The below example uses a public web service and is available for anyone to test against.

1. First, we can use the script playground to test out some scripting calls and see the output. The below example shows how to get a reference to a client object. By printing this client object to the console we get an output of all the available methods, types, and other information about the web service as defined in the WSDL file.

Python - W3Schools WSDL

```
from suds.client import Client
client = Client("http://www.w3schools.com/xml/tempconvert.asmx?WSDL")
print client
```

This WSDL defines two functions: CelsiusToFahrenheit(string tempCelsius) and FahrenheitToCelsius(string tempFahrenheit). These are the functions that this web service makes available to you. Don't worry about the fact that the methods are listed twice. This is just because the WSDL has two definitions of the functions that are formatted for different SOAP version standards. To call these functions in Ignition

scripting, you have to make use of the "service" member of the client object. You can see printing the returned results shows the conversion.

2. To make a simple conversion window in an Ignition project you can add a button, a numeric textbox, and a label to a window. Then on the button to calculate a Fahrenheit to Celsius calculation, you would place something like the following:

Python - Fahrenheit to Celsius

```
from suds.client import Client
client = Client("http://www.w3schools.com/xml/tempconvert.asmx?WSDL")

far_value = event.source.parent.getComponent('Numeric Text Field').floatValue
cels_value = client.service.FahrenheitToCelsius(far_value)

event.source.parent.getComponent('Label').text = cels_value
```

3. Then you can make a second button to do the opposite: calculate Celsius to Fahrenheit.

Python - Celsius to Fahrenheit

```
from suds.client import Client
client = Client("http://www.w3schools.com/xml/tempconvert.asmx?WSDL")

cels_value = event.source.parent.getComponent('Numeric Text Field').floatValue
far_value = client.service.CelsiusToFahrenheit(cels_value)

event.source.parent.getComponent('Label').text = far_value
```

4. With your scripts in place your window should now function as a simple temperature conversion tool!

Beyond the Example

While the example is relatively simple, it can easily be expanded upon. However, always keep the general workflow in mind when using the SUDS library:

Pseudocode - WSDL Workflow

```
#Import the SUDS Client object
from suds.client import Client

#Instantiate a new Client Object
client = Client("url_to_your_wsdl")

#Call the desired method using the service instance variable
client.service.MyMethod(myArgument)
```

Complex Arguments

In the overview, the methods provided by the web service were very simple and took simple argument types. Sometimes however, the web service will describe complex types and allow you create instances of these types that can then be added to the system/machine that the web service is providing an interface for.

A simple, hypothetical example of this would be a system that stores contact information of clients and can be used as an address book of sorts by other systems on the network. It may provide not only a way to pull contact information for a certain individual out, but also a way to insert new contacts. We'll keep the example simple and say that contacts have only a name and a phone number.

Note: This example is completely hypothetical. It is intended to give insight into complex types. It does not make use of an an actual functional web service.

For example, say we create and print the client object we associated with our web service in the following manner:

Pseudocode - Client Object

```
from suds.client import Client
url = 'http://localhost:7575/webservices/hypothetical_webservice?wsdl'
client = Client(url)
print client
```

And the resulting output is the following:

Python - Results

```
Suds ( https://fedorahosted.org/suds/ )  version: 0.4 GA  build: R699-20100913

Service (hypothetical_webservice)
   Prefixes (0):
   Ports (1):
```

```
    (Soap)
      Methods:
        addContact(Contact contact, )
        getContactList(xs:string str, xs:int length, )
        getContactByName(Name name, )
  Types (3):
    Contact
    Name
    Phone
```

Here you can see that, while not too complicated, the web service defines more than just methods that take simple type arguments and return the same. Under the Types section, you can see there are three "complex" types. These are basically just objects like one creates in an object oriented programming language like java. The SUDS Client object has an instance variable called "factory" that allows you to create these complex types so you can use them to invoke methods defined by your web service that take complex arguments.

If we wanted to add a contact using the addContact() method, we have to create a contact object first:

Pseudocode - Using a Method

```
contact = client.factory.create('Contact')
print contact
```

The create function creates a new contact object that knows its own structure. We can view this structure by calling print on this new object and see that it prints the following:

Python - Structure

```
(Contact)=
  {
    phone = []
    age = NONE
    name(Name) =
        {
            last = NONE
            first = NONE
        }
   }
```

By examining the Contact type object, we can see its structure and know what we need to create in order to have a valid Contact to add to the address book. We could then do the following to supply the necessary information for the Contact object and then call our addContact function.

Pseudocode - Adding a Contact

```
contact = client.factory.create('Contact')

phone= client.factory.create('Phone')
phone.areacode = '916'
phone.number = '5557777'

name = client.factory.create('Name')
name.first = 'John'
name.last = 'Doe'

contact.name = name
contact.phone = phone
contact.age = 30

client.service.addContact(contact)
```

After execution a new contact will have been added via the web service!

Steps to remember when using complex types:

Pseudocode - Complex Type Reminders

```
#Create a new type object using the factory instance variable of the Client object
my_type = client.factory.create('MyType')

#If you don't know the structure of the newly created object then print it to the console
print my_type
```

Related Topics ...

- Web Services, SUDS, and REST

JSON Format

About JSON

JavaScript Object Notation (JSON) is a language-independent data format. It's a lightweight format for storing and transporting data (i.e., when data is sent from a server to a web page). JSON is easy for users to read and write and for machines to parse and generate.

JSON Rules

JSON has a few simple rules:

- Data is in name/value pairs.
- Data is separated by commas.
- Curly braces hold objects.
- Square brackets hold arrays.

On this page ...

- About JSON
- JSON Rules
- How JSON Works
  - JSON Data - Name and a Value
  - JSON Objects
  - JSON Arrays
- Where JSON Is Used in Ignition
  - Perspective Component Properties
  - Tags
  - JSON in Tag UDTs
  - Looping through JSON Objects with Scripting

How JSON Works

JSON Data - Name and a Value

JSON data is written as name/value pairs. A name/value pair consists of a field name in double quotes, followed by a colon, followed by a value.

```
{
        "companyName":"Inductive Automation"
}
```

JSON Objects

JSON objects are written inside curly braces.

Note: The properties of a JSON object have no defined order. If you need a defined order, use a dataset or an array instead.

```
{
        "firstName":"Sally",
        "lastName":"Smith"
}
```

JSON Arrays

JSON Arrays are written inside square brackets. An array can contain objects. In the following example, the object "companies" is an array and contains three objects.

```
{
   "companies":[
      {
         "companyName":"Inductive Automation",
         "cityName":"Folsom",
         "stateName":"CA"
      },
      {
         "companyName":"Hewlett Packard",
         "cityName":"Palo Alto",
```

```
        "stateName":"CA"
      },
      {
        "companyName":"Apple",
        "cityName":"Cupertino",
        "stateName":"CA"
      }
   ]
}
```

Where JSON Is Used in Ignition

Ignition uses the JSON format to store much of its data internally, including Tags and Perspective component properties.

Perspective Component Properties

Components have properties (props), which are simply named values. These properties are arranged in a tree structure following the structure and data model of the common JSON document format. Component properties are defined as a JSON structure, and are variable according to the type of component that the config object represents. All components registered in the module have a set of default properties included. These defaults are provided to the instantiated component at runtime, and so default props are not saved. Instead, only those which have a value that differs from the default are stored, serialized and sent to the client during loading.

Example - Sample Data from Table Component

```
[ { "city": "Helsinki", "country": "Finland", "population": 635591 }, { "city": "Jakarta", "country":
"Indonesia", "population": 10187595 }, { "city": "Madrid", "country": "Spain", "population": 3233527 },
{ "city": "Prague", "country": "Czech Republic", "population": 1241664 }, { "city": "San Diego",
"country": "United States", "population": 1406630 }, { "city": "Tunis", "country": "Tunisia",
"population": 1056247 } ]
```

Tags

Ignition exports and imports Tag configurations to and from JSON. Tags are defined as JSON objects, which consist of properties, arrays, and sub-objects. The system.tag.configure function can take either a String document definition, or a JSON object that defines one or more Tags. Overrides for UDTs are created by simple redefinition of properties, and complex structures like Event Scripts and Alarm configurations will be merged with inherited definitions.

You can copy the JSON or one or or more Tags in the Tag Browser. This copies them into the system clipboard. In addition, pasting the JSON into a different provider/designer will create or overwrite tags.

Example 1 - Tag Export

```
{
  "name": "Tank Instance",
  "typeId": "Tank UDT",
  "tagType": "UdtInstance",
  "tags": [
    {
      "value": "80",
      "name": "Tank Level",
      "tagType": "AtomicTag"
    },
    {
      "value": 80,
      "name": "sliderValue",
      "tagType": "AtomicTag"
    }
  ]
}
```

Example 2 - Tag Export

```
{
  "tags": [
    {
      "valueSource": "memory",
      "dataType": "Boolean",
      "alarms": [
        {
          "setpointA": 1,
          "name": "Above Normal"
        }
      ],
      "name": "Boolean Tag",
      "value": false,
      "tagType": "AtomicTag"
    },
    {
      "valueSource": "memory",
      "dataType": "Boolean",
      "name": "One Shot Trigger",
      "tagGroup": "Driven One Shot",
      "value": true,
      "tagType": "AtomicTag",
      "enabled": true
    },
    {
      "valueSource": "opc",
      "opcItemPath": "ns\u003d1;s\u003d[Generic]_Meta:Random/RandomDouble1",
      "dataType": "Float8",
      "name": "Pressure3",
      "tagGroup": "Driven One Shot",
      "tagType": "AtomicTag",
      "enabled": true,
      "opcServer": "Ignition OPC UA Server"
    },
    {
      "valueSource": "opc",
      "opcItemPath": "ns\u003d1;s\u003d[Generic]_Meta:Random/RandomDouble2",
      "dataType": "Float8",
      "name": "Thickness3",
      "tagGroup": "Driven One Shot",
      "tagType": "AtomicTag",
      "opcServer": "Ignition OPC UA Server"
    }
  ]
}
```

JSON in Tag UDTs

You can also set JSON strings as properties in an Ignition Tag. Any properties of a Tag can be set to a string that represents a JSON object.

Note: In a UDT, the { } braces are used to denote a property reference. Make sure you don't have any properties with names that look like JSON objects so there is no overlap. Strings will appear as black text, parameter references will appear as grey and italicized.

Looping through JSON Objects with Scripting

Traversing a JSON object in scripting is simple as long as the structure is known. If there are objects within objects, you can use multiple loops to get through it all.

Let's use the JSON Array object above for a simple example. You can loop through the list to get the repeating items by name.

```
# loop through the JSON data
# fetch the data, this will change depending on where the script is in relation to the table
json = self.items
# access the companies object (which is a list)
```

```
companies = json["companies"]

# loop through the companies list
for company in companies :
        # get each item out of the row object
        name = company["companyName"]
        city = company["cityName"]
        state= company["stateName"]
        # now do something with the data
```

Related Topics ...

- JSON Functions

Basic Python Troubleshooting

When learning how to code in Python, most Ignition users tend to place most of their learning efforts on memorizing syntax or other aspects of the language. While being comfortable with the language is useful, there are plenty of references available: countless books and websites that describe syntax and usage already exist.

In truth, the best thing you can do to make yourself a better programmer is to learn some basic troubleshooting behaviors. While syntax examples are all over the Internet, examples detailing exactly what you want your code to do, to the extent you want it to, will be difficult if not impossible to find.

This section details how to troubleshoot a script in Ignition. You won't walk away from this section having exact answers to specific problems, but rather examples and concepts that you can apply to your own scripts.

On this page ...

- Coding Best Practices
  - Look for Errors
  - If You Find an Error, Read It!
  - Use Print Statements
  - Add Comments
  - Test Early, Test Often
  - Avoid Hard-Coding Arguments: Use Variables Instead
  - Decide on a Naming Convention
  - The Simplest Approach Really Is the Best Approach

Coding Best Practices

The following are some general and helpful best practices that can help minimize the amount of time you spend troubleshooting, as they can help better direct you to a problem.

Look for Errors

When code fails mid-execution, it always generates an error message. Where the message appears depends on where the script executed:

Scope	Print Command	Print Output Location
Gateway *Note: anything that isn't in the Client or Session scopes listed below uses this scope*	system.util.getLogger	Scripts on Gateway-scoped resources (Tags, Alarm Pipelines, SFCs, etc.,) will appear on the Logs page of the Status section on the Gateway. Additionally, the wrapper.log file in Ignition's installation directory will have these messages. Here are the default wrapper.log file paths for each operating system: • **Windows:** Program Files/Inductive Automation/Ignition/logs • **Lunix:** /var/log/ignition • **Mac OS X:** /Users/UserName/Documents/Ignition-osx-x.x.x/logs
Vision	Python's `print` command, or system.util.getLogger	The Client Console will contain any errors generated in the client: press **Ctrl + Shift + F7** to open the console, or using the menubar in the client to go to **Help > Diagnostics** then click on the **Console Tab**. Additionally, a red Error Box should appear with details on the error if a Component Event Handler threw the exception: Extension Functions do not generate the Error Box. If you don't see the error box then it might be minimized, or open in the background (behind the Client).
Perspective	system.perspective.print	Web browsers generally offer a way to inspect a page, which usually contains a console of some sort.
Designer	Depends. See the output location.	The Designer reports errors in a similar manner to Clients: errors appear in the Designer's Console (**Ctrl + Shift + c**), Component Event Handlers will generate a red Error Message. Note that events some events, such as Client Startup Scripts, will not trigger in the Designer, so get in the habit of launching a Client when testing non-component scripting events. While in the Designer, and working on Perspective resources, calls to system.perspective.print will appear in the Designer's Console as well.

What happens if I don't see any errors?

If there truly isn't an error message somewhere, but your code isn't doing what you expected, then ask yourself the following questions:

1. What is your script supposed to do?
2. What is it actually doing?

Question #2 is harder to answer: if you knew what it was doing, you wouldn't be stuck! The best way to answer this question is by adding **`print`** statements to your code.

When a script doesn't perform to expectations, it can suggest a problem with the script's workflow. Some of the pages in this section can offer some suggestions on what to do.

If You Find an Error, Read It!

It is common for users that are new to scripting to see an error message, immediately close it without much thought, and then stare at their code as if the problem will politely make it self known. While looking over your code line-by-line will eventually lead you to the problem, the error messages can provide you with a shortcut to the issue. Check out some of the pages in this section for more information on reading an error message.

Use Print Statements

When testing your scripts, the print command can help you verify that your code is behaving the way it should. This allows you to reconstruct what your code did when it executed. Don't be afraid to add helpful print statements to your code.

Once your code works as expected, remember to either remove or comment out the print statements, so they don't flood the console during normal use as this makes troubleshooting other issue more difficult. Just be mindful of the scope of the script, as that determines how you generate print messages, as well as where the console is.

Python - Using print to Troubleshoot

```
myVar = system.tag.read("folder/tag")

# printing out variables you are going to use in an if-statement later allows you to confirm that the values
are what you expect them to be.
print "myVar is set to: " + str(myVar)

# Sometimes viewing the data type of the variable can prove helpful.
print "myVar is a: " + type(myVar)

# Should your code have multiple if-statements, adding a print statement before and after can show you where
the flow of the script went.
print "Starting if-statement"

if myVar > 100:

        # If you don't see this print statement, then your if-statement evaluated to False.
        print "Inside if-statement"
        doWork()

# Printing the end of your script doesn't give you any useful troubleshooting information, but if you need
to trigger the script multiple times, it helps delineate each execution.
print "Script Ended"
```

Add Comments

While comments are useful to remind you how your code works, you can also use them to plan your script before you write any code. Break down what you want the code to do into several smaller steps, and then leave comments describing those steps in order. This provides you a chance to review the script's workflow before worrying about syntax. It also provides natural points to stop and test your code, to make sure it is doing what you think it should do.

Test Early, Test Often

Unless the script is very simple, avoid writing the entire script and then testing at the end. You may have missed an important line early on, so now you have to adjust all of the code below that line to make the script run. As mentioned above, add print statements, and run your script to make sure it is doing what you think it should.

Additionally, stopping to test your code provides an excellent opportunity to **save your project**. Get into the habit of saving before you execute any new code.

Avoid Hard-Coding Arguments: Use Variables Instead

Instead of doing this:

Python - Hard Coded Message Box

```
system.gui.messageBox("Hello you. Glad to see you")
```

Try to get into the habit of doing this:

Python - Message Box Variable

```
message = "Hello you. Glad to see you"
system.gui.messageBox(message)
```

Simple examples like the above don't make for the best use case, but when you have a large script that references a value multiple times, it is easier to declare the value once in a variable, and then just reference the variable throughout your code. If you need to change the value later, then you can simply change it once where you initialize the variable, and you don't have to search every line of your code looking for the value.

Decide on a Naming Convention

When creating variables, try to adopt a naming convention that comes natural to you, and stick to it. If you consistently write variables using the same conventions, you will be less likely to end up with a typo when referencing that variable later in your code. Remember that code is case-sensitive, so something as simple as forgetting to capitalize a letter will cause an error.

This is especially important if you are working in a group on the same project. It's better to get together with your colleges and agree upon some naming conventions before you write any code.

Ignition's **system.* functions** use the camelCase naming convention. That is, the first letter of each word is capitalized except for the very first letter. We recommend that you use it for variable names because It is easy to remember to use camelCase for both functions and variable names instead of for just one.

The Simplest Approach Really Is the Best Approach

When learning how to code, it's not uncommon to run into multiple issues that require you to find a workaround. However, these workarounds can cascade into other issues and make your script more complicated. Consider the following:

> "The goal of my script is to access A and then output B...
> Shoot, B requires C, so I'll add C...
> Wait, C requires that D exists, so I'll create that...
> Oh, D needs interfaces E, F, and G, so let's add those in...
> Hmm. E needs H and I, F needs J and K, while G needs Y, Z and...A **again**?!?"

Take a step back and ask yourself "what is this script doing"? If you can't do that in a sentence or two, you may want to rethink the script. If the scope of the script is too large, then it considerably increases the complexity of the code, which in turn could add a plethora of problems later.

If you keep adding workarounds, but you're code is not getting any closer to achieving its end goal, there may be an easier way to accomplish what you're trying to do with a different approach.

Related Topics ...

- Python Scripting

In This Section ...

Reading Error Messages

When an error occurs in the execution of a script, an error message box will pop up. The popup box appears in front of any open Designer windows, and will remain in view until you close it or click on something behind it.

There are two modes for the Error box selectable by the **Send to Front** and **Send to Back** icons.

- **Send to Front** means additional error messages will cause the popup to reappear on top.
- **Send to Back** will cause the errors to remain hidden below the Designer.

On this page ...

- Error Message Box Overview
- Troubleshooting Errors Using the Error Message Box
  - Broken Example - Incorrect Attribute
  - Broken Example - Undefined
  - Broken Example - Type Error

Error Message Box Overview

Exceptions usually include a line number. Take note of the number in the Details tab, and start your search for the problem there. Be aware, however, that the line reported may not be the cause of the issue. The actual problem may be higher up in the code due to a faulty initialization, or some other issue. When troubleshooting, always start looking at the line reported, and work your way back towards the top.

When testing a script, you may come across an Error Message Box like the following. Your first inclination may be to close the error without closer examination: resist it!

The error messages generated from failed script executions are incredibly helpful. The **Message** tab on the error describes the Event Handler that encountered the exception; in this case, the actionPerformed event handler. This is important in cases where multiple Event Handlers on the same component have scripts. Without knowing which Event Handler generated the script, we could waste time trying to troubleshoot the wrong script.

Additionally, it shows the name of the component, which is "NavButton_MyWindow". Again, this is helpful if multiple scripts from multiple components are triggering in quick succession. The **Message** tab will clearly point you towards the source of the exception. This is another reason to give meaningful names to your components: doing so makes the process of tracking an error much easier.

The Details tab has even more information, specifically the line number that the exception occurred on, as well as the error message. The message here states that the window at path "MyWindow" does not exist, so we can check the Project Browser to see if we simply mistyped the name of the window.

Troubleshooting Errors Using the Error Message Box

When we don't get our expected results from our script, as in the following examples, always read both the Message and Details tabs in the Error Message Box. They are pretty good about pointing you to the root cause of your error. From there, you can easily find and fix any errors in your script.

The following examples show how to troubleshoot some of these error messages from the Error Message Box. Keep in mind that every script is unique, but at least you'll become familiar with what some of the error messages mean, and gain a little insight of the troubleshooting process.

Broken Example - Incorrect Attribute

In the following script, when a Button is pressed a Message Box is supposed to open on a window and display "Hello World".

Python - Broken: Incorrect Attribute

```
system.gui.messagebox("Hello World")
```

From the error message box, you know two important things right away:

- The script is using the actionPerformed event on the Hello Button.
- Line 1 displays the description of the error.

Check to be sure you used the correct attribute. Then check the spelling, case sensitive letters, spacing, and operators in your script. If you are familiar with Ignition's Built-in Scripting Functions, you can probably spot the error immediately. If not, you might want to use the autocompletion popup feature to retype your 'messageBox' scripting function. This will automatically fix any syntax errors.

The error in this example is in the attribute name. It uses case sensitive letters (i.e., **messageBox**).

In this case, "messagebox" must be spelled with a capital "B" since we're trying to use Ignition's system.gui.messageBox function. We corrected the script by changing our code to the following:

Python - Corrected: Incorrect Attribute

```
system.gui.messageBox("Hello World")
```

Broken Example - Undefined

Here is a similar script. When a Button is pressed, it's supposed to open a Message Box on a window and display "Testing".

Python - Broken: Undefined

```
system.gui.messageBox(Testing)
```

The Error Message Box displays an error on the actionPerformed script on the button component in line 1 with an **undefined** name. The Details tab is simply telling us that something is not defined in the script. You can either define a variable or create a string by putting "Testing" in quotes to correct the error.

For this example, we corrected the code by turning the argument passed to system.gui.messageBox into a string literal with quotation marks:

Python - Corrected: Undefined

```
system.gui.messageBox("Testing")
```

Broken Example - Type Error

Similar to the other scripts, this script opens a Message Box and is supposed to display "100" when a Button is pressed.

Python - Broken: TypeError

```
system.gui.messageBox(100)
```

Since this script failed to execute successfully, the Error Message Box popped up and displayed an error on the actionPerformed script on the button component in line 1 with a **TypeError**. This error tells us that the first parameter is expecting a string, not an integer.

For this example, we corrected the code by changing the argument passed to system.gui.messageBox into a string literal with quotation marks.

Python - Corrected: TypeError

```
system.gui.messageBox("100")
```

You can see from the examples above, that the Error Message Box provides quite a bit of information that points to the root cause of an error on an Event Handler script. As you learn from the Error Message Boxes about what these messages mean, you'll be able to quickly spot them in your script and quickly correct any errors you encounter.

Related Topics ...

- Basic Python Troubleshooting
- Scripting in Ignition
- Getting Started with Scripting in Ignition

Troubleshooting - Nothing Happened

Did it Work?

When testing a script, you may find yourself in a situation where the script appears to be running, but doesn't seem to work. Furthermore, you may not see any error messages stating there is a problem.

When a script doesn't perform to expectations and doesn't throw an exception, it can suggest a problem with the script's workflow. This page describes a couple of common scenarios.

Before You Begin...

Make **absolutely** certain that there isn't an Error Message Box hiding somewhere, otherwise you'll waste time applying the troubleshooting tips outlined below, when the real the error message was hiding in the background behind another window the whole time.

If a button component on a window is running a script, the output will be displayed on the Designer/Client Output Console. You can check for any error messages by navigating to the Console. From the **Client** menu bar, go to **Help > Diagnostics** and select the **Console tab**. From the **Designer** menu bar, select **T ools > Console**.

On this page ...

- Did it Work?
  - Before You Begin...
- Common Scenarios when Nothing Happens
  - An Important Line Is Missing
  - The Script Is not Being Called
  - Important Lines Are Being Skipped
- Determining the Cause

Common Scenarios when Nothing Happens

An Important Line Is Missing

Some critical part of your code is missing or commented out. It could be something simple, like your code is supposed to increment a value, but you forgot to write the line that increments the value.

Pseudocode - Missing Code

```
myVar = 0

def doSomething():
        print "Doing work!"

# We're not modifying the value of myVar directly after it is initialized,
# so the current implementation of this code means the expression in our if-statement will never return True.
# Perhaps doSomething() returns a value that needs to be assigned to myVar.
if myVar == 10:
        doMoreStuff()
```

A quick way to test your code is by running your script in the Script Console before attaching it to a scripting event or specific component. You can see that running the code above in the Script Console that nothing happened, thus, suggesting an error.

The Script Is not Being Called

The script doesn't appear to be working because the mechanism that is supposed to be triggered hasn't been called. This can be caused by using the wrong event: (i.e., perhaps you placed the code on a Button's **propertyChange** event, when you meant to place it on **actionPerformed**).

Alternatively, perhaps you defined a function in your script, but you forgot to call the function.

Additionally, if you're testing the script in the Designer, make sure it is in **Preview Mode**. Event based scripts will not run in the Designer unless it is in Preview Mode.

Important Lines Are Being Skipped

This can be caused by incorrect indentation, or a misconfigured condition in an if-statement's expression. For example, you used "**==**" when you meant to use "**!=**". Verify that you are using the correct operators in if-statements.

Pseudocode - Skipping Code

```
myVar = 0

if myVar == 10:
        print "Hello, the value of myVar matches the condition in the if-statement"
```

Test your code for any errors by running your script in the Script Console. You can see nothing happened by running the code above in the Script Editor, once again suggesting an error.

Determining the Cause

The easier step to take when troubleshooting your script is to start adding print statements to your code. From here, you can start piecing together what the code is doing:

- **Print** the value of variables to make sure they are coming in the way you expect.
- Place **print** statements at the start and end of your code. This allows you to determine when your script is being called, and when it finishes.
- Adding a **print** statement before and after an if-statement can show you the flow of your script went as expected, or was filtered out by the if-statement.

Related Topics ...

- Script Console
- Basic Python Troubleshooting
- Reading Error Messages

Troubleshooting Workflow

Troubleshooting a script is an iterative process. Since the script stops executing at the first error, we won't see if there are any other errors on later lines unless, of course, we spot them ourselves while writing the code. As a result, we will have to keep trying our script until it executes successfully.

It is important to understand that scripts always execute from top to bottom, and each line must complete before the next line may move on. In the event, our code returns an exception, we can assume that our troubleshooting process should always start at the line reported in the error, and then work up until we find the problem.

On this page, we will take an indepth look at a script with multiple problems, and work through each one. Note, that this is not a comprehensive list of all possible types of exceptions that could occur when writing a script, but instead, this page attempts to demonstrate the troubleshooting process.

To learn more about troubleshooting specific script errors, refer to the following sections:

- Reading Error Messages
- Troubleshooting - Nothing Happened

On this page ...

- Scenario Overview
- First Error - NoneType Object
  - Exception Error Explained
  - What Should We Look For?
  - Solution
- Second Error - Checking Attributes
  - Exception Error Explained
  - What Should We Look For?
  - Solution
- Third Error - ArrayIndexOutOfBounds
  - Exception Error Explained
  - What Should We Look For?
  - Solution

Scenario Overview

Here we have a window with two components: a **Button** and **Power Table**. The purpose of the button is to find which cell in the Power Table the user selected, and write the value to a Tag.

The code on the button is listed below.

Python - Sample Code Block

```
# Create a variable that references the Power Table
table = event.source.parent.getComponent('Power Table')

# Find the cell the user currently has selected, and store the value in a
variable
userSelectedValue = table.data.getValueAt(table.selectedrow, table.
selectedColumn)

# Write the User Selected Value to a tag
system.tag.write("Scripting/ButtonError/WriteTarget", userSelectedValue)
```

First Error - NoneType Object

When the button is pressed, we are presented with an error. The **Message** tab in the Error Box describes the Event Handler that encountered the exception.

The **Details** tab tells us where to start looking, specifically the line number where the exception occurred as well as the error message. Line 5 is referenced, so the troubleshooting process should start there.

Exception Error Explained

When an error message refers to a **NoneType** object, that simply means a null, or None in Python. The word '**attribute**' is used to reference a property on an object. In this case, this means the script was trying to access the **'data'** property on nothing. The message is telling us that it couldn't find a property named **'data'** on a nothing, which is correct since NoneType objects don't have any properties named 'data'.

If we were to reword this message to something a bit more straightforward, it would say the following: "I tried to access the 'data' property on 'null', but it doesn't have a property by that name."

What Should We Look For?

If your exception is referring to a NoneType object, then some line of code probably tried to reference something else (i.e., Tag value, property, another variable, etc.), but couldn't find anything at the location you specified. It is also helpful to note what attribute was mentioned in the error, which was **data**. If we look at our code, we see the following on line 5.

Python - Line 5

```
userSelectedValue  = table.data.getValueAt(table.selectedrow, table.selectedColumn)
```

Based on this information, we should take a closer look at the table variable since our code is specifying **'data'** as an attribute of the table. The error states that it could not find the data attribute on a NoneType object, but our code only references data in regards to the table object, so maybe there is something wrong with how that table object was referenced or initialized. If we look further up in our code, we see that `table` was initialized on line 2:

Python - Line 2

```
table = event.source.parent.getComponent('Power Table')
```

Since this script was placed on the Button component's **actionPerformed** event, we can trace back to the source of the issue:

event = actionPerformed event
source = Button this script is placed on
parent = Container that the Button was placed in. Based on the Button's position in the window, this appears to be the Root Container.
getComponent('**Power Table**') = Should return a reference to a component named '**Power Table**' directly inside of the Root Container, assuming one exists. If we take a look at the Project Browser, we see the following:

There is no component named '**Power Table**' in the Root Container. However, there is a Power Table component named '**My Table**', so it appears someone renamed the component, which caused our script to initialize the variable **`table`** as a '**NoneType**' instead of a reference to a component.

Solution

To fix this issue, we can simply update our code to use the new name of the Power Table, which is '**My Table**'. We can type this in manually, but since it is case sensitive, it must match exactly. It may be easier to find the **Name** property on the component, copy the name to the system clipboard (**Ctrl-C**), and paste it (**Ctrl-V**) into the script. Our code now looks like the following:

Python - Updated Code Block

```
# Create a variable that references the Power Table
table = event.source.parent.getComponent('My Table')

# Find the cell the user currently has selected, and store the value in a variable
userSelectedValue = table.data.getValueAt(table.selectedrow, table.selectedColumn)

# Write the User Selected Value to a tag
system.tag.write("Scripting/ButtonError/WriteTarget", userSelectedValue)
```

Second Error - Checking Attributes

If we try our button again, we get the following exception error.

Exception Error Explained

Like the last exception, the problem has to do with our code trying to access an attribute (property) on an object, but the given attribute doesn't exist on the object.

What Should We Look For?

Again, the exception refers to line 5, but note that the error mentions a different object and attribute this time. Without knowing what a **`'com.inductiveautomation.factorypmi.application.com'`** object is, we can tell that our script is trying to reference an attribute named **`'selectedrow'`**. This gives us a starting point for troubleshooting the error. If we check line 5, we see the following:

Python - Line 5

```
userSelectedValue = table.data.getValueAt(table.selectedrow, table.selectedColumn)
```

We see **`table.selectedrow`** as the first parameter being passed to the getValueAt() function. We checked into the `table` variable earlier, so this doesn't necessarily mean that the variable is the problem, although there is no harm in double checking, especially if you have not personally verified the variable. Assuming that the table variable is correct, let's check the attribute. We can check the Property Editor in the Designer to find a matching property, or head over to the Power Table component page in the manual and check the property reference. We can see that there is a **Selected Row** property on the Power Table, but it should be spelled **`"selectedRow"`** with a capital **`"R"`**.

We also know that the getValueAt() function takes a row index as the first parameter, so it would make sense that our code has a typo.

Solution

We should fix the typo in the attribute name.

Python - Line 5 Updated

```
userSelectedValue = table.data.getValueAt(table.selectedRow, table.selectedColumn)
```

Third Error - ArrayIndexOutOfBounds

After making our most recent change, the button works great! Users are able to select a cell, and write to the Tag. The one exception, is if the user does not have a cell selected before clicking the button. In this case, the following error occurs:

Exception Error Explained

Our code attempted to look up something in a collection of some sort (i.e., sequence, dataset, etc.,) by index, but was told to check index -1, which is out of bounds. Indexes in Ignition are typically zero-based, meaning they start at 0, and increment from there, so an index of -1 doesn't exist. Commonly, when a property in Ignition refers to index -1, that means nothing is selected, so our code failed because nothing was selected.

What Should We Look For?

Fortunately, the exception told us what the issue is: the user did not have anything selected before pressing the button, so we don't have to search further.

Solution

The solution to this issue is more open ended, as there are many ways to suggest to our user that they need to select a cell in the table first. The **Selected Row** and **Selected Column** properties on the Power Table will have -1 values if none of the cells are selected, so we could use an **'if'** statement checking the values of one of those properties. Our new code could look like the following:

Python - Code Block Update #3

```
# Create a variable that references the Power Table
table = event.source.parent.getComponent('My Table')

# Make sure a cell in the Power Table is selected first
if table.selectedRow != -1:
# Find the cell the user currently has selected, and store the value in a variable
        userSelectedValue = table.data.getValueAt(table.selectedRow, table.selectedColumn)
```

```
        # Write the User Selected Value to a tag
        system.tag.write("Scripting/ButtonError/WriteTarget", userSelectedValue)

# If a cell isn't selected, then let the user know
else:
        system.gui.messageBox("Please select a cell in the table first!")
```

Related Topics ...

- Reading Error Messages
- Troubleshooting - Nothing Happened

Scripting Vs. SQL Vs. Expressions

Expression Language & SQL Queries vs Scripting

There are three major languages in Ignition, the Expression language, the SQL Queries, and Python Scripting. It is important to understand the differences between the three and to know where each is used. Scripting is used in the event handlers that are available all over Ignition, but Expressions and SQL are in the Property Binding locations shown here.

Spot the Difference - Comments

When starting out with Ignition, it can be difficult to know which syntax you should be using for a particular area of text. One little trick that may help is to be familiar with how each language handles comments, and then utilize the **Ctrl + /** command which automatically adds the characters that comments out a line of code in the language are you are typing in. Once you remember to use **Ctrl + /**, you simply need to be familiar with the characters that each language uses.

Expression Language - Comment

```
// The Expression Language uses double forward slashes.
```

SQL - Comment

```
-- Areas in Ignition that accept SQL syntax use double dashes.
```

Python - Comment

```
# Python uses the pound/number/hash character.
```

In addition to comments, interfaces for each of the languages usually contain other signs or reminders about the language. These will be covered in their respective sections on this page.

Python Scripts

Overview

Python is featured prominently throughout Ignition and many different resources can contain a Python Script.

How Can I Tell If I'm Writing a Python Script?

Python Scripts typically use the words **Script** or **Event** in the interface. Additionally, Python requires a particular Event to be selected so if you see event Handlers on the left, you know you are looking at a Python script.

On this page ...

- Expression Language & SQL Queries vs Scripting
- Spot the Difference - Comments
- Python Scripts
  - Overview
  - How Can I Tell If I'm Writing a Python Script?
  - Where Are Python Scripts Used?
- SQL Queries
  - Overview
  - How Can I Tell If I Should Be Using SQL Syntax?
  - Where Is SQL Used in Ignition?
  - SQL in Python
- Expression Language
  - Overview
  - How Can I Tell If I'm Writing an Expression?
  - Where Is The Expression Language Used?

Where Are Python Scripts Used?

Python Scripts are used all throughout Ignition. Some resources, such as components, even have multiple places to write scripts! Below are some common locations for scripts:

- Scripting in Vision
- Extension Functions
- Tag Event Scripts
- Client Event Scripts
- Gateway Event Scripts

SQL Queries

Overview

The SQL language is used for selecting information from a database. It can be used in a variety of ways, but most of them will either make modifications to the database or return a set of values (with a few notable exceptions like Stored Procedures). The majority of users will be returning large chunks of data into a Table or Report in Ignition. This means a complete dataset will be returned based on a user selection, a time range, or any combination of factors.

How Can I Tell If I Should Be Using SQL Syntax?

Typically, the words **Query** and **Database** appear in the interface somewhere. Additionally, there is usually a way to specify a **Database Connection**.

Where Is SQL Used in Ignition?

Below is a reference of the most common areas in Ignition where SQL queries may be used.

- SQL Query Bindings
- Named Queries
- Database Query Browser
- Reporting Data Source: SQL Query Data Source and Basic SQL Query
- Query Tags
- Python Scripts - See the SQL in Python header below for more details

SQL in Python

SQL queries can be called from a **Python** script. There are several system functions in Ignition that allow a script to run a query against the database, such as system.db.runPrepQuery. This is a more advanced technique, as you need to adhere to both language's syntax. Furthermore, when typing a SQL query in a Python script interface, the syntax highlighting can not help with the SQL portions. The syntax highlighting in a Scripting Window is only looking for Python syntax, not SQL.

In cases where you plan on calling a SQL query from a Python script, it is highly recommended to write the query in the **Database Query Browser** first (substituting parameters with static values for testing purposes), and then move the query over to the script once the query executes successfully on its own. This approach can save you some time troubleshooting, as there will be less ambiguity when an error occurs since you know the query runs.

Below we see an example of calling a SQL query in a script. Line 5 creates a variable called "query", and assigns it a string consisting of a prepared statement (using SQL). The query is then executed with the system.db.runPrepUpdate function.

For more examples of using a query in a Python Script, check out the system.db.* functions.

Expression Language

Overview

The Expression Language is a simple programming language that we invented (very similar to many other existing expression languages), and is different from the Python scripting you will find in Ignition. The expression language is a very simple kind of language where everything is an expression - which is a single piece of code that returns a value. This means that there are no statements, and no variables, just operators, literals, and functions.

The most common expression language that most people are familiar with is the one found in Microsoft Excel. You can have Excel calculated a cell's value dynamically by typing an expression like `=SUM(C5:C10)`. Our expression language has similar functionality, but different syntax. It is mainly used to define dynamic values for **Tags** and **component properties**. In Ignition's Expression Language, you will use component properties and functions like if({Root Container.type}="Type C",True,False).

How Can I Tell If I'm Writing an Expression?

Typically interfaces that expect the Expression Language use the word **Expression**. Additionally, you'll commonly find the **Expression Function** button $f(x)$ towards the right side of the text area.

Where Is The Expression Language Used?

Below is a list of resources that commonly utilize the Expression Language

- Expression Bindings on Property Bindings and Alarm Bindings
- Expression Tags and Derived Tags
- Several Alarm Pipeline Blocks, including the Expression Block, Switch Block, and Notification Blocks.
- Expression Items in Transaction Groups
- Parameters on Reports
- SFC Transitions

Related Topics ...

- Expression Language and Syntax
- SQL in Ignition
- Scripting
- Python Scripting

Scripting Examples

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 ...

- Reading and Writing to Tags
- Importing and Exporting a CSV
- Reading a Cell from a Table
- Adding a Delay to a Script
- Export Tag Historian to CSV
- Parsing XML with the Etree Library

Reading and Writing to Tags

There are simple interfaces in Ignition that allow you to easily write to a Tag on some Event, and Reading can be as simple as creating a Tag binding. Sometimes, however, the built-in approaches can be too simplistic or limiting. The Reading and Writing to Tags page details how to better interact with Tags from scripting.

Importing and Exporting a CSV

CSV files are used by many software programs to export data so that other systems may utilize the information contained within. The Importing and Exporting a CSV page demonstrates how to both import a CSV into Ignition, as well as export data from Ignition into a CSV.

Reading a Cell from a Table

Once data is populated into a Table component, it's useful to know how to read and extract a data from a cell in a Table, particularly if users can select a row in a Table. The Read a Cell from a Table page has some good examples for retrieving data from a single cell and multiple cells in a Table.

Adding a Delay to a Script

In some cases, being able to halt execution of a script can be helpful while waiting for some other event to occur. Adding a Delay to a Script page details some common approaches, as well as approaches to avoid.

Export Tag Historian to CSV

The Tag Historian Data can be great, but it's sometimes difficult to view it outside of Ignition. The Export Tag Historian to CSV page details how to pull out a subset of Tag history data and export it to a CSV file.

Parsing XML with the Etree Library

XML files are a great way to share information between two systems, but parsing them can seem like a daunting task. Parsing XML with the Etree Library shows how to use Python's Etree Library to read through an XML document.

Related Topics ...

- Basic Python Troubleshooting

In This Section ...

Location Based Vision Startup Scripts

Client Startup Scripts

Security can be created on a Vision Client using a Startup Script that checks certain user information and then customizes the level of access within the project based on that information. We can pull user information from places like System Tags for Vision Clients or Session Properties for Perspective Sessions. Using that information, the script can then customize which windows or views the user sees, provide some additional information to the user, or even prevent the user from accessing the project completely. An additional example is located here.

Location Based Vision Client Restrictions

Each Client has access to distinct hostname and IP addresses. These can be used in Client startup scripts that evaluate the Client's information and compares it to a list of acceptable host names or IP addresses. This information can come from a database or a set of Tags. In this example, we will prevent access to the client if the user is logging in from an incorrect location.

1. On the Project tab, choose Client Events.

2. Select the Startup icon.
3. Add the following script:

```
# Prevent access if user is not logging in from the correct location

# Grab the hostname that the user is logging in from
hostname = system.tag.readBlocking(["[System]Client/Network
/Hostname"])[0].value

if hostname != "Machine A Computer":
        # If the user logs in on a computer that is not called
Machine A Computer,
        # inform them that the project can only be accessed from the
Machine A computer,
        # and then log them out.
        system.gui.messageBox("This project can only be accessed
from the 'Machine A Computer'.")
        system.util.exit(1)
```

On this page ...

- Client Startup Scripts
- Location Based Vision Client Restrictions
- Location Based Startup Display

INDUCTIVE UNIVERSIT

Custom Security

Watch the Video

4. Click **OK** to save the script.
5. Save your project and launch the client. You will get a popup that informs you that the project must be launched using the Machine A Computer.

Location Based Startup Display

Another common use for startup scripts on Vision Clients is to open a specific window depending on the location of the log in. In this example, if the user logs in on Machine A Computer, then the Machine A Details window will be displayed. If they log in on a different computer, the Overview window is displayed.

1. On the Project tab, choose Client Events.

2. Select the Startup icon.
3. Add the following script:

```
# Open a window based on location

# Grab the hostname that the user is logging in from
hostname = system.tag.readBlocking(["[System]Client/Network/Hostname"])[0].value

if hostname == "Machine A Computer":
        # If the user logs in on a computer that is called Machine A Computer,
        # then open the Machine A details window.
        system.nav.openWindow("Machine A Details")
else:
        # Otherwise, open the overview
        system.nav.openWindow("Overview")
```

4. Click **OK** to save the script.

> **Note:** Any windows that are set to Open on Startup will still open in addition to the window specified in the startup script. You should disable any main windows from opening on startup if using this method.

5. Save your project.

Related Topics ...

- Client Tags for Indirection

Reading and Writing to Tags

In many cases, the binding system is the most appropriate way to display a Tag value on the screen. However, you may wish to access a Tag's value in a script. Using the system functions, you can read from a Tag and write to a Tag in Ignition.

On this page ...

- Script Builder
- Know Your Scope
- Manual Tag Reads
  - Reading from a Single Tag
  - Reading from Multiple Tags
- Relative Tag Paths
- Manual Tag Writes
  - Writing to a Single Tag
  - Writing to Multiple Tags

Script Builder

If you simply need to write to value to a Tag from a script, you can use the Set Tag Value tab of the Script Editor. To learn more, refer to the sections on Script Builders for more information. However, if you need to do more than just send a single write, see the section below.

Know Your Scope

Whenever a script from a Shared resource (such an Alarm Pipeline) attempts to interact with a Tag, the script must specify the Tag Provider, otherwise, the script could return an exception. Tag Providers are always included at the start of the Tag Path and look like the following: `[tagProvider]`

When interacting with a Tag from the Project scope (such as a script on a component) you may optionally include the Tag provider. If omitted, the Project default Tag provider will be used.

Pseudocode - Tag Read Scope

```
# Reading a Tag without the Tag Provider. You would not want to use this
from the Shared scope.
system.tag.readBlocking(["My/Tag/Path"])

# Reading from the same Tag, but specifying the Tag Provider. This format
may be used safely in either scope.
system.tag.readBlocking(["[default]My/Tag/Path"])
```

INDUCTIVE UNIVERSIT

Reading and Writing Tags

Watch the Video

Manual Tag Reads

Reading from a Single Tag

Reading a Tag from a script is accomplished with the `system.tag.readBlocking()` function, which requires the Tag path you wish to read from. This function returns a 'Qualified Value'; this object is more than just the value. A Qualified Value is the Tag value that has three attributes; Value, Quality, and TimeStamp.

The function normally expects a list of Tag paths, but can be used with a single Tag: simply provide a list of only a single Tag path.

When handling the results of the read, the results will be returned in a list of qualified values, so you'll need to specify which qualified value you're interested in via Python slicing, even if there is only a single qualified value in the list.

Pseudocode - Reading Tag Attributes

```
# get the Tag value
value = system.tag.readBlocking(["tagPath"])[0].value

# get all three attributes
tag = system.tag.readBlocking(["tagPath"])[0]
value = tag.value
quality = tag.quality
timestamp= tag.timestamp
```

Reading from Multiple Tags

The `system.tag.readBlocking()` function can easily be used to read multiple Tags in a single call:

Python - Reading Multiple Tags and Printing the Values

```
# Create a List of Tag Paths to read
paths = ["Scripting/Tags/Alarm_1", "Scripting/Tags/Alarm_2", "Scripting/Tags/Alarm_3"]

# Read the Tags, and store the complex results in a variable
values = system.tag.readBlocking(paths)

# For each Tag Path, iterate through our results...
for index in range(len(paths)):

        # ...and do something with the individual values
        print values[index].value
```

Relative Tag Paths

Like elsewhere in Ignition, relative paths may be used from within a script. This is especially useful when writing a Tag Event script inside of a UDT, as you can specify relative members in the same UDT with "[.]".

Pseudocode - Reading with Relative Tag Paths

```
# Assuming a UDT with two sub-members, a script from one can read from one member to the other using the
following code:
system.tag.readBlocking(["[.]otherMember"]).value
```

Manual Tag Writes

Writing to a Single Tag

Much like reading, there is a function you can use to write to Tags: `system.tag.writeBlocking`. It requires a list of Tag paths, as well as a list of values to write to those Tags.

Pseudocode - Tag Write

```
system.tag.writeBlocking(["tagPath"],["Hello World"])
```

Writing to Multiple Tags

The `system.tag.writeBlocking` function can also write to multiple Tags, again by providing multiple paths and values.

Python - Multiple Tag Writes

```
# Create a List of Tag Paths to write to
paths = ["Scripting/Tags/Alarm_Setpoint_1", "Scripting/Tags/Alarm_Setpoint_2"]

# Create a List of values to write
values = [72, 72]

# Send of the write requests
system.tag.writeBlocking(paths, values)
```

Related Topics ...

- Tags

Exporting and Importing a CSV

A CSV file, (comma separated values) is one of the most simple structured formats used for exporting and importing datasets. It is a convenient and flexibe way to edit and share data across applications. Ignition has a built-in function to convert a dataset to CSV data called system.dataset.toCSV. You can even convert the contents of a CSV to a script and move it to an Ignition component, such as a Power Table.

This section contains examples for exporting and importing data to a CSV as well as converting the contents of a CSV to a script.

> The examples on this page make use of Vision components, and run in the Vision Client scope.

On this page ...

- Exporting Data to a CSV
- Importing Data from a CSV
- Converting the Data into a Dataset
  - Calling the system.dataset.fromCSV Function
  - Calling the csv.reader Function

Exporting Data to a CSV

You can export a dataset from a query or table to a CSV file.

1. Identify the dataset that you want to export to a CSV file. In this case, we can generate some data on a Power Table and export that data. Drag a **Power Table** component on to your window and toggle its **TestData** property to generate some data.

2. Drag a **Button** component on the window, and double click on the Button to open the **Component Scripting** window.

3. Next, let's add our script on the Button component's actionPerformed event. Select the **actionPerformed** event, and click on the **Script Editor** tab.

4. Copy the contents from one of the examples below, and paste the contents to the **Script Editor**. Notice the system.dataset.exportCSV scripting function is used in the first example to export the dataset to a CSV file:

Python - Hard Coded Filepath

```
# Create a variable that references our Power Table. You could
modify this part
# of the example to point to a different component in the window.
component = event.source.parent.getComponent('Power Table')

# Use system.dataset.toCSV to turn the dataset into a CSV string
csv = system.dataset.toCSV(component.data)

# Write to local file system. Note the "r" character right before
the directory path.
# This denotes a raw string literal, meaning we ignore escape
sequences (like the "/")
system.file.writeFile(r"C:\myExports\myExport.csv", csv)
```


INDUCTIVE UNIVERSITY

Exporting Data to CSV

Watch the Video

5. Instead of hardcoding the path as we did in the above example, we could ask the user to select a directory on the local system with system.file.saveFile:

Python - User Selected Directory

```
# Create a variable that references our Power Table. You could
modify this part
# of the example to point to a different component in the window.
component = event.source.parent.getComponent('Power Table')

# Use system.dataset.toCSV to turn the dataset into a CSV string.
csv = system.dataset.toCSV(component.data)

# Use system.file.saveFile to have the user find a directory to
write to.
filePath = system.file.saveFile("myExport.csv", "csv", "Comma
Separated Values")

# We can check the value of filePath to make sure the user picked a
path before
# attempting to write.
if filePath:
        system.file.writeFile(filePath, csv)
```

6. To test your scripts, put the Designer in **Preview Mode**, and press the **Button**. Open your **myExport.csv** file and check your data.

	A	B	C	D	E
10	84	0.56352293	1B827F36	FALSE	11/6/2017 9:25
11	92	0.53782463	05D9E3D9	FALSE	11/6/2017 9:25
12	13	0.17691666	66E05608	TRUE	11/6/2017 9:25
13	91	0.90138406	23A77B71	FALSE	11/6/2017 9:25
14	97	0.5005872	089B981E	TRUE	11/6/2017 9:25
15	18	0.010637939	B98CAFF8	FALSE	11/6/2017 9:25
16	55	0.18030751	2469BC91	TRUE	11/6/2017 9:25
17	14	0.7256514	F252FB1D	FALSE	11/6/2017 9:25
18	15	0.2902785	197496C1	TRUE	11/6/2017 9:25
19	99	0.8383941	916B3310	FALSE	11/6/2017 9:25
20	38	0.03875667	C8081AD1	FALSE	11/6/2017 9:25

Importing Data from a CSV

There are several ways to import data from a CSV file. First, we could use system.file.readFileAsString to read the entire file as a string. Note, that this will read the file as is, meaning `"\n"` can be used to denote new lines.

Python - Using system.file.readFileAsString()

```
# Ask the user to find the CSV in the local file system.
path = system.file.openFile("csv")

# Use readFileAsString to read the contents of the file as a string.
# This string will be the parameter we pass to fromCSV below
stringData = system.file.readFileAsString(path)
```

```
# Split stringData into a List of strings, delimited by the new line character
stringData = stringData.split("\n")

# Iterate through the list, and do something with each line
for i in range(len(stringData)):

        # We're printing the row here, but you could do something more useful with the data.
        print stringData[i]
```

Alternatively, Python's CSV Library could be use to read in the contents of a CSV. In some cases, this is the easier approach, as the reader object is ready to be iterated over. Note, that this approach does read in each row as a List of strings:

Python - Using csv.reader()

```
# Import Python's built-in csv library
import csv

# Ask the user to find the CSV in the local file system.
path = system.file.openFile("csv")

# Create a reader object that will iterate over the lines of a CSV.
# We're using Python's built-in open() function to open the file.
csvData = csv.reader(open(path))

# Iterate through the reader object, and do something with each line
for row in csvData:

        # We're printing the row here, but you could do something more useful with the data.
        print row
```

Converting the Data into a Dataset

Once you've read in the contents of a CSV into a script, you may wish to move it elsewhere. It is not uncommon to move the data to a **Power Table**, or other components on the screen. The main difficulty with this is converting the CSV data into a dataset so that it fits into the **Power Table** componen t's **Data** property. We have a couple of approaches listed below.

Calling the system.dataset.fromCSV Function

The system.dataset.fromCSV function can take a string and convert it to a dataset. Note, that the function expects a very specific format:

CSV File Content

```
#NAMES
Col 1,Col 2,Col 3
#TYPES
I,str,D
#ROWS,6
44,Test Row 2,1.8713151369491254
86,Test Row 3,97.4913421614675
0,Test Row 8,20.39722542161364
25,Test Row 4,20.39722542444222
33,Test Row 5,20.39722542232323
62,Test Row 6,20.39722542111999
```

Example

1. Create a Text file on your local system named "**example.csv**."
2. Open the Text file, copy the contents of the "**CSV file Content**" box above, and paste the contents to the file. Save the changes to the **examp le.csv** file.

3. Let's move the contents of a CSV file to a Power Table. Add a **Power Table** and a **Button** component to your window. Double click on the **B utton** component, and paste the following code into the **Script Editor** of the **actionPerformed** event:

Python - Using system.dataset.fromCSV()

```
# Ask the user to find the CSV in the local file system.
path = system.file.openFile("csv")

# Use readFileAsString to read the contents of the file as a string.
stringData = system.file.readFileAsString(path)

# Convert the string into a dataset
data = system.dataset.fromCSV(stringData)

# Pass the dataset to the data property on the Power Table.
event.source.parent.getComponent('Power Table').data = data
```

4. To execute your script, put the Designer into **Preview Mode**, and press the **Button**. A window will open for you for you to navigate and choose your CSV file, then click Open.

5. Your data will be displayed in the Power Table as shown below.

Col 1	Col 2	Col 3
44	Test Row 2	1.87
86	Test Row 3	97.49
0	Test Row 8	20.4
25	Test Row 4	20.4
33	Test Row 5	20.4
62	Test Row 6	20.4

Calling the csv.reader Function

As mentioned, system.dataset.fromCSV() requires a specific format, which may not match the format of your file. In this case, we can use Python's CSV Library to parse the file and convert it to a dataset.

CSV File Content

```
Col 1,Col 2,Col 3
44,Test Row 2,1.8713151369491254
86,Test Row 3,97.4913421614675
0,Test Row 8,20.39722542161364
25,Test Row 4,20.39722542444222
33,Test Row 5,20.39722542232323
62,Test Row 6,20.39722542111999
```

Here is the code to import the above CSV data.

Python - Using Python's csv Library

```
# Import Python's built-in csv library.
import csv

# Ask the user to find the CSV in the local file system.
path = system.file.openFile("csv")

# Create a reader object that will iterate over the lines of a CSV.
# We're using Python's built-in open() function to open the file.
csvData = csv.reader(open(path))

# Create a List of strings to use as a header for the dataset. Note that the number
# of headers must match the number of columns in the CSV, otherwise an error will occur.
# The simplest approach would be to use next() to read the first line in the file, and
# store that at the header.
header = csvData.next()

# Create a dataset with the header and the rest of our CSV.
dataset = system.dataset.toDataSet(header ,list(csvData))

# Store it into the table.
event.source.parent.getComponent('Power Table').data = dataset
```

Related Topics ...

- system.dataset.exportCSV

Adding a Delay to a Script

Overview

In some cases, having a script execute after a delay is preferable. A common use case is waiting for some event elsewhere in the system to finish: a Tag change script executes that needs to wait for a new value from a separate Tag. One approach to this is to trigger our script, and then hold or wait until the other event occurs. On this page, we'll take a look at a couple of different approaches to this problem.

It is important to note that pausing a script can cause your client to lock. It is often preferred to look for another event to trigger the script you need. For the example above where we are waiting on a Tag to change, you might be able to use the Tag change to fire your script instead of waiting in the original script. It is up to you to determine the best trigger based on what exactly your script does.

On this page ...

- Overview
- Using the system.util.invokeLater Function
- Using a Timer from Python's Threading Library
- Calling time.sleep from Python's Time Library
- Using a While Loop
- Approaches to Avoid - Locking the Client
  - Reasons to Avoid time.sleep and While Loops
  - Recommended Alternative
- Demonstration - Executing a Delay Between User Keyboard Input
  - Workflow for a Delay Using Two Scripts

Using the system.util.invokeLater Function

The system.util.invokeLater function is a great way to add a delay mid-way through the script. Simply create a function that represents all of the work that should occur after the delay, and pass the function to invokeLater, along with a delay period.

The example below calls two Message Boxes: once initially when pressed, and the other after a three second delay.

Python - Two Message Boxes

```
message = "All Done!"

# Create a function that will be called by invokeLater.
def runThisLater():
        system.gui.messageBox(message)

# Call invokeLater with a 3000ms delay. Note that our function will not
run immediately
# because invokeLater always executes once the rest of this script is
complete.
system.util.invokeLater(runThisLater, 3000)

# Bring up another Message Box. This will appear before the "All Done!"
message, because of invokeLater.
system.gui.messageBox("Waiting...")
```

One of the main limitations with invokeLater is that you can not pass parameters to the function that will be called. Parameters need to be initialized and determined elsewhere in your script, usually in the function definition.

Using a Timer from Python's Threading Library

The Threading Library has a Timer function that works in a very similar fashion to invokeLater. The main difference is that you can pass parameters to the function parameter when calling the Timer. Take a look at Python's official documentation for more details on the Timer object.

The example below will again call two Message Boxes with a three second delay between them. However, the text that is defined in the second Message Box is specified when starting the Timer.

Python - Python Threading Timer

```
from threading import Timer

# Create a function that will be called by the Timer.
def runThisLater(param):
        system.gui.messageBox(str(param))

# Constructs a Timer object that runs a function after a specified interval of time has passed.
# Note the start() at the end: this is required to start the Timer. Don't forget this part!
Timer(3.0, runThisLater, ["Stop Waiting, I'm done"]).start()
```

```
# Bring up another Message Box. This will appear before the "All Done!" message, because of the Timer
system.gui.messageBox("Waiting")
```

Another benefit to using the Timer, is you can cancel the execution of the Timer's action using the cancel() function, but it only works if the Timer is still in its waiting stage.

Calling time.sleep from Python's Time Library

The simplest approach to pausing a script is to use the sleep() function in Python's Time Library:

Python - Sleeping the Code

```
from time import sleep

# This will pause execution of the script for 3 seconds. After that time, the script will continue.
sleep(3)

print "I'm awake!"
```

Using a While Loop

The While Loop is another simple approach to adding a delay: simply keep looping until the other event occurs. As always, you will want to take steps to ensure that an infinite loop never occurs: easiest by initializing a counter variable before iterating in the loop, and breaking out if the counter reaches a certain value.

Python - While Loop Safeguard

```
counter = 0

while not otherEvent:

        checkOtherEvent()
        counter += 1

        # Use this to break out if the event takes too long. You'll need the rest of your script to account
for this possibility.
        if  counter >= 10000:
                print "Took too long. Leaving the While Loop"
                break
```

Approaches to Avoid - Locking the Client

While the sleep() and while loop functions are simple to use, they can cause problems by locking up the client and because of this, we generally recommend avoiding them if possible. Typically, system.util.invokeLater on a delay can accomplish the same task.

Note, that there are use cases for both approaches outlined below especially so in regard to the While Loop. However, neither function should be used to force a delay in a Client.

> Using either of these methods to pause or delay a script can lock up the entire client, which may be very dangerous.

Reasons to Avoid time.sleep and While Loops

If used on a component, these approaches could lock up the Client or Designer. Scripts called from a component run in the same thread that the Client and Designer use to refresh the screen. Whenever a script on a component triggers, the screen is unable to refresh until the script finishes. In most cases, this is so fast that no one notices. If your script is intentionally calling sleep() or using a long running While Loop, then the Client will lock up for the duration.

Additionally, these approaches run an extended period of time, and block other component based scripts from executing. They do not yield to other scripts while waiting.

Because of these two reasons, the sleep() and While Loop functions can cause your clients to appear unresponsive, when really they are just running a script that prevents the screen from refreshing.

Recommended Alternative

As mentioned earlier, the system.util.invokeLater function (also mentioned on this page) can provide the same functionality. If a sleep() or While Loop function must be used, then they should be called from system.util.invokeAsynchronous to prevent blocking any other scripts.

Demonstration - Executing a Delay Between User Keyboard Input

In many cases, you may need to show your users a large number of entries on a Table or Template Repeater. As more entries are added to the system, adding a search field that can filter results becomes more appealing. Furthermore, being able to filter as the user types (as opposed to forcing them to hit enter every time) adds some polish to the window. However, if the entries are backed by the database, you will not want to run a query every time a user presses a key. Instead, it would be preferable to wait for a delay in input, and run one query to limit strain on the database.

The following is not a traditional example that you would find here in the User Manual, and assumes you are comfortable with many concepts in Ignition.

Workflow for a Delay Using Two Scripts

One approach to adding a delay involves two scripts:

- **Pre-Delay Script**: This script fires before the delay. In this case, we will use a script on the Text Field's keyReleased event. This will call the script everytime a new key is entered into the Text Field, and calls the Post-Delay Script on a delay. It also passes the text that is currently in the Text Field, and increments an edit count (could be configured as a custom property on the same component).
- **Post-Delay Script**: This script runs after the delay, so it needs to check if there were any new keystrokes that occurred during the delay. If there were no new keystrokes, then it can run the query. In this demonstration, we will use a Custom Method not the Text Field component.

Additionally, we will need some criteria that our script can use to determine if it is safe to run our query. You can have multiple criteria here, but for the sake of simplicity we will use the **Text Property on the Text Field**. This way we will know what the user typed before the delay, and then can compare it back to the Text property after the delay. If the Text on the component after the delay is different, then we don't want to run the query as the user may still be typing.

The keyReleased script is shown below. It is using a .05 second delay, but could of course can be modified. We're creating a new Timer object every time a key is released, so there will be many of these scripts firing as the user types. However, they are fairly lightweight, and don't directly interact with the database, so having many executions of this script isn't taxing on the system.

Python - Pre-Delay Script

```
# We're using the Timer object for this because we want to pass parameters to the function that will run
after the delay.
from threading import Timer

# Ignoring arrow keys.
if (event.keyCode < 37 or event.keyCode > 40):

        # Grab the text in the component currently.
        currentText = event.source.text

        # Calls the function after 0.5 seconds, and pass the currentText.
        Timer (0.5, event.source.sendingQuery,[currentText]).start()
```

Below is the sendingQuery script. As mentioned, this script will determine if the user ceased keyboard activity before running a query against the database. This example is using a Custom Method on the Text Field, so it uses the `self` argument to reference the source component.

Python - Post-Delay Script

```
def sendingQuery (self, oldText):

        # Read the value of the text property, since it may have changed during the delay.
        currentSearchText = self.text

        # If the text before the delay is the same as the after the delay, there has been a pause in
keyboard activity, so we should run the query.
        if (currentSearchText == oldText):

                # If the text field isn't empty, then we need to filter the results with a WHERE clause and
our criteria.
                if(currentSearchText != ""):
                        newQuery = "SELECT * FROM employees WHERE CONCAT(firstname, ' ', lastname) like '%"
+ currentSearchText + "%'"
                # If the Text Field Is blank, then we should run the query again, but this time without a
filter, so all results appear.
                else:
                        newQuery = "SELECT * FROM employees"
```

We can now run our query in between user keyboard activity.

Related Topics ...

- system.util.invokeLater
- Timer Scripts

Export Tag Historian to CSV

On this page ...

- Obtaining Historian Data
- Exporting to CSV
- History Tag Search and Export

Obtaining Historian Data

Sometimes, it may be useful to export the values from the Tag Historian. There are many ways that this can be done, but the easiest way is to use the built in system.tag.queryTagHistory function. This allows you to specify a list of Tag paths to query history for.

Python - Grabbing History Data

```
# First we start by creating a start and end time that we will be query
history for.
endTime = system.date.now()
startTime = system.date.addMinutes(endTime, -30)

# Next we call our queryTagHistory function, using the start and end dates
as well as a Tag path.
# The other parameters listed for this function can be altered to fit your
need.
data = system.tag.queryTagHistory(paths=["[default]myTag"],
startDate=startTime, endDate=endTime, returnSize=10, aggregationMode="
Average", returnFormat='Wide')
```

This simple script will query Tag History for a Tag called myTag. It will query the last 30 mins of data, return 10 records, each record will be an average of the time slice, and the return format will be wide. For more information on the function parameters and the different options available, see the queryTagHistory function page. The data returned is then being stored in a variable **data**, and we can use it later in the script in anyway we want.

Exporting to CSV

Now that we have our data, it is really easy to export it to a CSV file. Ignition has a built-in function to convert a dataset to CSV data called system.dataset.toCSV. This CSV data can then be written to an actual file using system.file.writeFile.

Python - Export History to CSV

```
# First we start by creating a start and end time that we will be query history for.
endTime = system.date.now()
startTime = system.date.addMinutes(endTime, -30)

# Next we call our queryTagHistory function, using the start and end dates as well as a Tag path.
# The other parameters listed for this function can be altered to fit your need.
data = system.tag.queryTagHistory(paths=["[default]myTag"],
startDate=startTime, endDate=endTime, returnSize=10, aggregationMode="Average", returnFormat='Wide')

# Turn that history data into a CSV.
csv = system.dataset.toCSV(data)

# Export that CSV to a specific file path. The r forces it to use the raw path and not require double
backslashes.
system.file.writeFile(r"C:\myExports\myExport.csv", csv)
```

History Tag Search and Export

If you have a lot of Tags storing history, it can be difficult to list them all out in the function above. This example makes things easier by specifying a folder of Tags instead of individual Tags. The script first grabs all of the historical Tags from a particular folder and gets the Tag paths using the system.tag.browseHistoricalTags function, runs those through the system.tag.queryTagHistory function, and then exports the results to a CSV file. This script is great, because it can be run from the scripting console, or from a button on the client. However, everything in the script is hardcoded to work with a specific system, though that can be easily changed to fit any system. Below are the list of parts that can be changed with what each of them is used for.

Part of Script	Description

`path='histprov:myDB:/drv:myIgnitionGateway:myTagProvider:/tag:myFolderOfTags'`	The path to the parent folder that we are searching through for history Tags. Each part of the string corresponds to a different part of your system. • histprov:myDB:/ - This is the name of your Historical Tag Provider that you are searching in. Replace **myDB** with the name of your historical Tag Provider. • drv:myIgnitionGateway:myTagProvider:/ - The first thing is the name of your Ignition Gateway. Replace **myIgnition Gateway** with the name of your Ignition Gateway, which can be found by going to the Gateway Configure Webpage and navigating to **System > Gateway Settings**. It is the **System Name**. The second thing is the name of the Realtime Tag Provider where the Tags reside. Replace **myTagProvider** with the name of your Tag Provider. • tag:myFolderOfTags - This is the name of the folder you want to search for historical Tags. Replace **myFolderOfTags** with the name of a folder in your Tag Browser to search through. **Note:** Paths to Internal Historian providers (such as those used by Edge Gateways) do not need to include the Gateway name.
`system.tag.queryTagHistory(...)`	This is what will actually query for historical data. While the path parameter should be kept the same, the other parameters can be altered to fit your needs. Currently, the function is querying the last 30 mins of history, has a return size of 10, an aggregation mode of Average, and a return format of wide. Of course, all of this can be changed. See system.tag.queryTagHistory for more information on the parameters that can be used here.
`r"C:\myExports\myExport.csv"`	This is the path that the CSV file will save at. This can be changed to be any location you want. You can even use system.file.saveFile to first ask the user for a save location and filename, and then passing the path returned from that function into the writeFile function. The r at the beginning of the string is important as it allows us to not have to use double backslashes.

Caution:

This script can be dangerous! It recursively looks through a folder to find all historical Tags, and then uses all of those Tags to look up Tag history. Looking through a folder that has too many history Tags, or querying history for too large a period of time can potentially result in locking up your system. Use caution when using this function, and never search through too large a set of Tags or query too much history.

Python - Recursively Browse for History and Export to CSV

```
# Our browse function that will browse for historical tags.
# By setting this up as a function, it allows us to recursively call it to dig down through the specified folder.
# Pass in an empty list that we can add historical paths to, and the path to the top level folder.
def browse(t, path):

        # Loop through the results of the historical tag browse, and append the path to the empty list.
        for result in system.tag.browseHistoricalTags(path).getResults():
                t.append(result.getPath())

                # If the result is a folder, run it through the browse function as well.
                # This will continue until we are as deep as possible.
                if result.hasChildren():
                        browse(t, result.getPath())

# Start with an empty list to store our historical paths in.
historyPaths = []

# Call the browse function, passing in an empty list, and the folder that we want to browse for historical tags.
# This path is a placeholder. It should be replace with your valid path.
browse(historyPaths, path='histprov:myDB:/drv:myIgnitionGateway:myTagProvider:/tag:myFolderOfTags')

# Create another empty list to store our tag paths that we will pull out of the historical paths.
tagPaths = []

# Loop through the list of historical tag paths, split out just the tag path part,
# and push it into our tag path list.
for tag in historyPaths:
        tagPaths.append("[myTagProvider]" + str(tag).split("tag:")[1])
```

```
# Now that we have a list of tag paths, we need to grab the historical data from them.
# Start by creating a start and end time.
endTime = system.date.now()
startTime = system.date.addMinutes(endTime, -30)

# Then we can make our query to tag history, specifying the various parameters.
# The parameters listed for this function can be altered to fit your need.
data = system.tag.queryTagHistory(paths=tagPaths, startDate=startTime, endDate=endTime, returnSize=10,
aggregationMode="Average", returnFormat='Wide')

# Turn that history data into a CSV.
csv = system.dataset.toCSV(data)

# Export that CSV to a specific file path. The r forces it to use the raw path with backslashes.
system.file.writeFile(r"C:\myExports\myExport.csv", csv)
```

Related Topics ...

- Exporting and Importing a CSV

Parsing XML with the Etree Library

What is the xml.etree Library?

The etree (ElementTree) library is a part of the python standard library, and contains many tools that make it simple to parse through and pull information out of an XML document. There are other libraries that can parse through XML documents, but etree is commonly used and very easy to get started with. The etree library will break up the XML into easily accessible elements, each representing a single node in the entire XML tree. For more information on using the etree library beyond the scope of this page, see the python documentation.

On this page ...

- What is the xml.etree Library?
  - Using the xml.etree Library
- A Simple Book Example

Using the xml.etree Library

There are a couple of different ways to import the XML data from etree, depending on how it is being stored. It can pull the data in from an XML file using the filepath, or it can read a string. Notice how regardless of how we import the XML, we end up with a root object.

Python - Reading a File

```
# The library must first be imported no matter how we pull in the data.
import xml.etree.ElementTree as ET

# Here we can grab the filepath using Ignition's built in openFile function, parse that into a tree, then
grab the root element.
filepath = system.file.openFile()
tree = ET.parse(filepath)
root = tree.getroot()
```

Python - Reading from a String

```
# The library must first be imported no matter how we pull in the data.
import xml.etree.ElementTree as ET

# Alternately, we can start with a string of the xml data.
xmlString = """
<employee id="1234">
        <name>John Smith</name>
        <start_date>2010-11-26</start_date>
        <department>IT</department>
        <title>Tech Support</title>
</employee>
"""

# Then parse through the string using a different function that takes us straight to the root element.
root = ET.fromstring(xmlString)
```

Each Tag is considered an element object. In the example above, the root element would be the employee Tag. Elements can also have attributes, which are within the Tag itself. In the example above, the employee element has an attribute id with a value 1234. Finally, each element can also have additional data, typically in the form of a string. This additional data is usually placed in between the element's start and end Tags. In the example above, the employee element has no additional data, but its children do. The name element would have an additional data value of John Smith. All of this data can be accessed using the Element object's built-in functionality. The major functions are listed below, and each example uses the reading from a string root XML example from above.

Function	Description	Example	Output
Element.tag	Returns the name of the Element's Tag.	`print root.tag`	employee
Element.attrib	Returns a dictionary of the Element's attributes.		{'id':'1234'}

		print root. attrib	
Element. text	Returns the additional data of the Element. The example here will return nothing because the root does not have any text. The next example uses children which do have text.	print root. text	
for child in Element	Will iterate through the Element's children. Each child is then its own element, complete with Tag, attrib, and text properties.	for child in root: print child. tag, child. text	name John Smith start_date 2010-11-26 departme nt IT title Tech Support
Element [index]	Allows direct reference to an Element's children by index. Since Tags can be nested many times, further nested children can be accessed by adding an additional index in square brackets as many times as necessary: Element[1][4][0] From the original element, we would go to the child located in the first position, that child's fourth position child, and that child's zero position child. When direct referencing child elements in this way, they still have access to the Tag, attrib, and text properties.	root[2]. tag root[3]. text	departme nt Tech Support

A Simple Book Example

Using the functions above, we can now easily parse through an XML file and use the results for something. Lets keep it simple, and parse through a document and then place the values into a table. First we need to start with an XML document. We have one below for you to test with in a string form, which would need to be pasted at the top of the script.

XML String

```
document = """
<catalog>
        <book id="bk101">
                <author>Gambardella, Matthew</author>
                <title>XML Developer's Guide</title>
                <genre>Computer</genre>
                <price>44.95</price>
                <publish_date>2000-10-01</publish_date>
                <description>An in-depth look at creating applications
                with XML.</description>
        </book>
        <book id="bk102">
                <author>Ralls, Kim</author>
                <title>Midnight Rain</title>
                <genre>Fantasy</genre>
                <price>5.95</price>
                <publish_date>2000-12-16</publish_date>
                <description>A former architect battles corporate zombies,
                an evil sorceress, and her own childhood to become queen
                of the world.</description>
        </book>
        <book id="bk103">
                <author>Corets, Eva</author>
                <title>Maeve Ascendant</title>
                <genre>Fantasy</genre>
                <price>5.95</price>
                <publish_date>2000-11-17</publish_date>
```

```
        <description>After the collapse of a nanotechnology
        society in England, the young survivors lay the
        foundation for a new society.</description>
</book>
<book id="bk104">
        <author>Corets, Eva</author>
        <title>Oberon's Legacy</title>
        <genre>Fantasy</genre>
        <price>5.95</price>
        <publish_date>2001-03-10</publish_date>
        <description>In post-apocalypse England, the mysterious
        agent known only as Oberon helps to create a new life
        for the inhabitants of London. Sequel to Maeve
        Ascendant.</description>
</book>
<book id="bk105">
        <author>Corets, Eva</author>
        <title>The Sundered Grail</title>
        <genre>Fantasy</genre>
        <price>5.95</price>
        <publish_date>2001-09-10</publish_date>
        <description>The two daughters of Maeve, half-sisters,
        battle one another for control of England. Sequel to
        Oberon's Legacy.</description>
</book>
<book id="bk106">
        <author>Randall, Cynthia</author>
        <title>Lover Birds</title>
        <genre>Romance</genre>
        <price>4.95</price>
        <publish_date>2000-09-02</publish_date>
        <description>When Carla meets Paul at an ornithology
        conference, tempers fly as feathers get ruffled.</description>
</book>
<book id="bk107">
        <author>Thurman, Paula</author>
        <title>Splish Splash</title>
        <genre>Romance</genre>
        <price>4.95</price>
        <publish_date>2000-11-02</publish_date>
        <description>A deep sea diver finds true love twenty
        thousand leagues beneath the sea.</description>
</book>
<book id="bk108">
        <author>Knorr, Stefan</author>
        <title>Creepy Crawlies</title>
        <genre>Horror</genre>
        <price>4.95</price>
        <publish_date>2000-12-06</publish_date>
        <description>An anthology of horror stories about roaches,
        centipedes, scorpions  and other insects.</description>
</book>
<book id="bk109">
        <author>Kress, Peter</author>
        <title>Paradox Lost</title>
        <genre>Science Fiction</genre>
        <price>6.95</price>
        <publish_date>2000-11-02</publish_date>
        <description>After an inadvertant trip through a Heisenberg
        Uncertainty Device, James Salway discovers the problems
        of being quantum.</description>
</book>
<book id="bk110">
        <author>O'Brien, Tim</author>
        <title>Microsoft .NET: The Programming Bible</title>
        <genre>Computer</genre>
        <price>36.95</price>
        <publish_date>2000-12-09</publish_date>
        <description>Microsoft's .NET initiative is explored in
        detail in this deep programmer's reference.</description>
</book>
```

```
        <book id="bk111">
                <author>O'Brien, Tim</author>
                <title>MSXML3: A Comprehensive Guide</title>
                <genre>Computer</genre>
                <price>36.95</price>
                <publish_date>2000-12-01</publish_date>
                <description>The Microsoft MSXML3 parser is covered in
                detail, with attention to XML DOM interfaces, XSLT processing,
                SAX and more.</description>
        </book>
        <book id="bk112">
                <author>Galos, Mike</author>
                <title>Visual Studio 7: A Comprehensive Guide</title>
                <genre>Computer</genre>
                <price>49.95</price>
                <publish_date>2001-04-16</publish_date>
                <description>Microsoft Visual Studio 7 is explored in depth,
                looking at how Visual Basic, Visual C++, C#, and ASP+ are
                integrated into a comprehensive development
                environment.</description>
        </book>
</catalog>
"""
```

We can then place a Table component and a Button component on the window, and place this script on the Button's actionPerformed event.

Python - Complete XML Parsing

```
# Start by importing the library
import xml.etree.ElementTree as ET

######
# Here is where you would paste in the document string.
# Simply remove this comment, and paste in the document string.
######

# We can then parse the string into useable elements.
root = ET.fromstring(document)

# This creates empty header and row lists that we will add to later.
# These are used to create the dataset that will go into the Table.
# We could fill in the names of the headers beforehand, since we know what each will be.
# However, this allows us to add or remove children keys, and the script will automatically adjust.
headers = []
rows = []

# Now we can loop through each child of the root.
# Since the root is catalog, each child element is an individual book.
# We also create a single row empty list. We can add all of the data for a single book to this list.
for child in root:
        oneRow = []

        # Check if the book has any attributes.
        if child.attrib != {}:

                # If it does contain attributes, we want to loop through all of them.
                for key in child.attrib:

                        # Since we only want to add the attributes to our header list once, first check if
it is there.
                        # If it isn't add it.
                        if key not in headers:
                                headers.append(key)

                        # Add the attribute value to the oneRow list.
                        oneRow.append(child.attrib[key])

        # Loop through the children of the book.
        for child2 in child:
```

```
                # Similar to above, we check if the tag is present in the header list before adding it.
                if child2.tag not in headers:
                        headers.append(child2.tag)

                # We can then add the text of the Element to the oneRow list.
                oneRow.append(child2.text)

        # Finally, we want to add the oneRow list to our list of rows.
        rows.append(oneRow)

# Once the loop is complete, this will print out the headers and rows list so we can manually check them in
the console.
print headers
print rows

# Convert to a dataset, and insert into the Table.
data = system.dataset.toDataSet(headers, rows)
event.source.parent.getComponent('Table').data = data
```

Related Topics ...

- system.file.openFile
- Libraries

Audit Log and Profiles

Ignition's built-in auditing system automatically records certain actions that occur in the system, such as a Tag writes or User Source authentication, into a SQL database table. Utilizing the system involves creating an Audit Profile, followed by enabling auditing in a project. Once both prerequisites have been met, the Gateway will automatically create a database table named AUDIT_EVENTS, and use the table to start tracking user actions.

The Remote Audit Log configuration option allows audit events to be automatically sent to a remote Gateway's audit log. The remote Gateway you plan to connect to must have a Audit Profile created. To learn more about sending audit events to a remote Gateway, refer to section Creating a Remote Gateway Audit Profile on this page.

Auditing Actions

For a list of actions that are recorded by an audit profile, see the Auditing Actions Reference page.

On this page ...

- Create a Database Audit Profile
- Create an Internal Audit Profile
- Creating a Remote Gateway Audit Profile
- Enabling Auditing in a Project
- Viewing Information in an Audit Log

Create a Database Audit Profile

1. Go to the **Config** section of the Gateway Webpage.
2. Scroll down to the **Security > Auditing** from the menu on the left. The Audit Profiles page is displayed.
3. Click the **Create a new Audit Profile** link.
4. You have the option of storing audit logs into an external database or sending them to a remote Gateway. For this example, select **Database**. (Configuring audit events to be sent to a remote Gateway's audit log is addressed in Creating a Remote Gateway Audit Profile section on this page).

5. Enter the **Name** of the audit log and **Description** (optional).
6. In the Retention field, set a value in days for how long you want audit records kept. (The default is 90 days.)
7. Under the **Database Settings**, select the **Database** where the table will be stored, select the **Auto Create** check box, and enter the desired **T able Name**.
8. Click **Create New Audit Profile**.

Once some changes have been made to a Tag or a Database table, Ignition will begin recording.

AUDIT_EVENTS_ID	EVENT_TIMESTAMP	ACTOR	ACTOR_HOST	ACTION	ACTION_TARGET	ACTION_VALUE
1	2016-07-25 17:50:09	admin	IU-WorkStation	tag write	B Tags/B3:1	1.0
2	2016-07-25 17:50:51	admin	IU-WorkStation	tag write	B Tags/B3:1	100.0
3	2016-07-25 17:50:53	admin	IU-WorkStation	tag write	B Tags/B3:1	2.0
4	2016-07-25 17:50:56	admin	IU-WorkStation	tag write	B Tags/B3:1	8.0
5	2016-07-25 17:51:20	admin	IU-WorkStation	query	update audit_events set acto...	4
6	2016-07-25 17:51:51	admin	IU-WorkStation	query	UPDATE audit_events SET `A...	1

Database Audit Profile Properties Table

Main	
Name	The default name, is the name of the Audit Profile.
Description	Description of the audit profile. Optional.

Retention	This feature is new in Ignition version **8.1.1** Click here to check out the other new features How long (in days) should audit records be kept? Values less than or equal to 0 will disable pruning. Default is 90 days.
Database Settings	
Database	The database connection to use to store audit events.
Auto Create	If true (selected), the table schema specified here will be automatically verified and created if necessary. Default is true.
Pruning Enabled	This feature is new in Ignition version **8.1.3** Click here to check out the other new features If false, this audit profile will never prune records, regardless of the retention field. Otherwise, the retention field will be followed. Default is false.
Table Name	The name of the table to store audit events. Default is AUDIT_EVENTS.

Create an Internal Audit Profile

The Internal Audit Profile option allows an Ignition Gateway to store audit records without an external SQL database. The only way to interact with the Internal Audit Profile is via the Status page of the Gateway webpage.

1. Go to the **Config** section of the Gateway Webpage.
2. Scroll down to the **Security > Auditing** from the menu on the left. The Audit Profiles page is displayed.
3. Click the **Create a new Audit Profile** link.
4. Select **Internal**.

5. Enter a name for the audit log and a description (optional).
6. In the Retention field, set a value in days for how long you want audit records kept. (The default is 90 days.)
7. Click **Create New Audit Profile**.

Internal Audit Profile Properties Table

Main	
Name	The default name, is the name of the Audit Profile.
Description	Description of the audit profile. Optional.
Retention	This feature is new in Ignition version **8.1.1** Click here to check out the other new features

	Value in days for how long you want audit records kept. (The default is 90 days.)

Creating a Remote Gateway Audit Profile

Just like configuring audit events to be logged into an external database, it is done from the Gateway Webpage, **Config > Security > Auditing.**

1. To have your audit events automatically sent to a remote Gateway's audit profile, select **Remote**, and click **Next**.

2. A list of known Gateways will be displayed. If you don't see a Gateway that you expected to see, check your Gateway Network settings to verify that the connections are valid. You also have the option to specify a Gateway manually. This example selects a valid Gateway. Click **Next**.

3. If an Audit profile exists, the fields will auto-populate. The name of the Gateway will appear in the **Name** field prefaced with the audit profile name (i.e., Ignition_Test_Auditing), as shown in the following example. Click **Create New Audit Profile**.

4. You will receive a successful message stating your new Audit Profile was created.

Remote Gateway Audit Profile Properties Table

Main	
Name	The default name, is the name of the Remote Gateway and Audit Profile.
Description	Description of the audit profile. Optional.
Enabled	By default. the journal profile is enabled.
Remote Settings	
Target System	The remote system to send audit events to over the Gateway network.
Target Profile	The audit profile on the remote system to log events into.

Enabling Auditing in a Project

1. Go to the Designer, open the project that you want to enable auditing on, then go to **Project > Properties**.

2. Go to the General section, select the **Enable Auditing** check box, and select your **Audit Profile** from the drop-down menu. The audit profile is used to record audit actions for your project. If the new audit profile does not show up, click **Refresh**.
3. Click **OK**.
4. **Save** your Project.

Viewing Information in an Audit Log

There are a few ways to view audit information: using a Table component, interface on the Gateway, or the Database Query Browser. Here is one example of viewing an Audit Log using the Database Query Browser.

1. In the Designer, go to **Tools > Database Query Browser.**
2. Under the **Schema** area, double click on a **table**, and it will expand the query in the Database Query Browser area.
3. Click **Execute**. All the audit log data will be displayed in the Resultset1 area.

Related Topics ...

- Database Query Browser
- Audit Log Display

In This Section ...

Alarm Notification Auditing

Alarm Notification profiles can be set to store information in an Audit Profile.

How to Store Alarm Notifications in an Audit Profile

Once you have an audit profile created in Ignition, you can configure your Alarm Notification Profile to start using it.

1. Go to the **Config** section of the Gateway.
2. Choose **Alarming > Notification** from the menu on the left.
   The Alarm Notification Profiles page is displayed.
3. Edit the appropriate notification profile by clicking the **edit** link.

4. The Edit Alarm Notification Profile page is displayed.
   Scroll down to the **Auditing** section, and select an **Audit Profile** from the drop-down menu.
   Click **Save Changes**, and Ignition will automatically begin storing information.

Now that the Alarm Notification Profile is storing data into the Audit system, you have a complete log of all alarm emails and acknowledgements that you can review.
See Audit Log Display for more info on how to retrieve information from the Audit Log.

On this page ...

- How to Store Alarm Notifications in an Audit Profile

INDUCTIVE UNIVERSITY

Alarm Notification Auditing

Watch the Video

Related Topics ...

- Alarm Notification
- Database Connections
- Audit Log Display

Audit Log Display

This page documents several ways to view results from the Audit Log System. For more information on how the Audit Log works, see the Audit Log and Profiles page.

On this page ...

- Access the Audit Log with Table Functions
- Access the Audit Log Using the Database Query Browser
- Access the Audit Log on the Gateway

INDUCTIVE UNIVERSITY

View Audit Information

Watch the Video

Access the Audit Log with Table Functions

Since Ignition makes accessing data from databases seamless, it is possible to bind a data property of a table directly to database table. Alternatively, it is possible to access the contents of the audit log with table functions.

Once you have an Audit Log setup and attached to a project, you can go back to that project and see what information is in that audit log. This example uses the Table Functions to extract data from the Audit Log.

1. In **Designer**, drag a **Table** component on to a window.
2. In the **Property Editor**, click on the binding icon from the table's **Data** property. The Property Binding window is displayed.
3. Select the **Functions** Binding Type.
4. Select **Functions** from the various binding options.
   a. **Binding Function**, select the **Audit Log** from the drop-down menu.
   b. **Audit Profile Name**, select **Audit** or the name of your Audit Profile.
   c. **Start Date**, enter the appropriate start date.
   d. **End Date**, enter the appropriate end date.
   e. **Polling Mode**, select **Relative**.
   f. Select any other appropriate function options from the menu, and click **OK**.

5. The table will populate with all the information stored in the audit log based on the Table Functions you selected. You can use the Table Customizer to configure how you want the table to look by reorganizing and hiding columns, making columns sortable, assigning meaningful headers, and much more.

Timestamp	Actor	Action	Action Target	Action Value	Result...	Result	System	Context ...	Context
Apr 27, 2018 11:39 AM	admin	login			0	AuditStatus...	project=Parameters	4	Client
Apr 27, 2018 3:48 PM	admin	logout			0	AuditStatus...	project=Parameters	4	Client
Apr 27, 2018 3:48 PM	admin	project save	Parameters		0	AuditStatus...	project=Parameters	2	Designer
Apr 27, 2018 3:49 PM	admin	project save	Parameters		0	AuditStatus...	project=Parameters	2	Designer
Apr 27, 2018 3:49 PM	admin	logout			0	AuditStatus...	project=Parameters	2	Designer
Apr 28, 2018 12:00 AM	Scheduled Report	report executed	WorkOrder Report	Failed Action -...	-2,147,4...	AuditStatus...	project=Parameters	1	Gateway
Apr 28, 2018 12:00 AM	Scheduled Report	report executed	Test Report	Failed Action -...	-2,147,4...	AuditStatus...	project=Parameters	1	Gateway
Apr 28, 2018 12:00 AM	Scheduled Report	report executed	Veggie Report 2		0	AuditStatus...	project=Parameters	1	Gateway
Apr 28, 2018 12:00 AM	Scheduled Report	report executed	Car Inventory Report	Failed Action -...	-2,147,4...	AuditStatus...	project=Parameters	1	Gateway

Access the Audit Log Using the Database Query Browser

The Database Query Browser makes it easy to search your database tables to view audit information.

1. In the **Designer** under the **Tools** menubar, select **Database Query Browser**. The Database Query Browser opens.

2. On the right side of Browser window under **Schema**, will be a list of tables from the currently selected database. Double click on the **audit_events** table, and click **Execute.**
   The data from the audit_events table will appear in the Resultset 1 tab.

Refer to the Database Query Browser to learn more about how it works.

Access the Audit Log on the Gateway

Ignition provides a simple interface to view Audit Logs on the Gateway.

1. On the Gateway webpage in the **Config** section, scroll down to **Security > Auditing**.
2. The Audit Profile page will be displayed. Select the Audit Profile where your information is stored, and click **More > view log**.

3. Choose the parameter settings if you're looking for something specific, otherwise, enter a **Start Date** and **End Date**, and click **Search**.

Related Topics ...

- Scripting
- Database Query Browser

- Table Customizer
- Security

Auditing Actions Reference

The auditing system in Ignition records actions originating from the Gateway, Perspective and Vision projects. This page lists which actions are logged by the auditing system.

On this page ...

- Gateway Audit Actions
  - Project System
  - Modules
  - Gateway - General
  - Gateway - Restore
  - Licensing Changes
  - Redundancy
  - Web Server Page
  - Identity Providers - User Authorization
  - Gateway Network
  - Email Settings
  - Audit Profile
  - User Sources
  - Service Security
  - Identity Providers
  - Security Levels
  - Security Zones
  - Database - Connections
  - Database - Drivers
  - Store and Forward
  - Alarming - General
  - Alarming - Alarm Journal
  - Alarming - Notification
  - Schedules
  - Tags - Realtime
  - Tags - Historical
  - OPC Client Connections
  - OPC UA - Device Connections
  - OPC UA - Server Settings
  - Enterprise Administration
  - Enterprise Administration - Event Thresholds
  - Enterprise Administration - Controller Settings
  - Enterprise Administration - Agent Settings
  - Enterprise Administration - Agent Management
  - Enterprise Administration - License Management
  - Enterprise Administration - Agent Tasks
  - Sequential Function Charts
  - Add a Record Manually
- Perspective Auditing Actions
- Vision Auditing Actions
  - Tags
  - Vision Tag Writes
  - Vision Component Database Writes
  - Vision User Login/logout
  - Database Query Browser
  - Vision Scripting
- Designer
  - Designer Login and Closing
  - Database Query Browser
- Alarm Notification
  - Alarm Notification Attempts
- Reporting Module
  - Report Execution
- Audit Table Definition

Gateway Audit Actions

The following actions are recorded in an audit log when the Gateway has a Gateway Audit Profile is configured.

Project System

The following project-based actions are tracked by the auditing system.

- Project Property changes made from the Designer.
- Project setting changes made from the gateway's web interface.
- Creating or deleting a project.
- Saving a project (action recorded as "project update").

In addition, project files on the Gateway's file system are closely monitored. If a user or third-party system modifies any of the project files, an entry will be recorded in the auditing system.

The following Gateway-level actions are recorded in an audit log when the Gateway has a Gateway Audit Profile is configured.

Modules

- Installing modules on the Gateway.
- Restarting a module.
- Deleting a module.

Gateway - General

- Gateway startup
- Gateway shutdown (assuming the gateway was requested to shutdown: unintended shutdowns from power failures and such will not be recorded).

Gateway - Restore

- Restoring the Gateway from a Gateway backup. Specifically, the Gateway will log that it was asked to before a restore, then perform the restoration.

Licensing Changes

- Activating a license.
- Unactivating a license.
- Updating a license.

Redundancy

- Saving after making any changes to the Redundancy Settings page.

Web Server Page

- Installing or removing a security certificate.
- Making changes to the Web Server Settings page.

Identity Providers - User Authorization

- Login requests and responses.
- Logout requests and responses.

Gateway Network

- Saving changes to Gateway Network General Settings .
- Creating, editing, or deleting outgoing connections.

- Approving incoming connections.

Email Settings

- Creating, editing, or deleting an SMTP Profile.

Audit Profile

- Creating, editing, or deleting an Audit Profile.

User Sources

- Creating, editing, or deleting a User Source.
- Creating, editing, or deleting a user.
- Creating, editing, or deleting a role.

Service Security

- Editing and saving a policy.

Identity Providers

- Creating, editing, or deleting an Identity Provider configuration.
- Making changes to a User Attribute Mapping.
- Creating, editing, or deleting a User Grant.
- Saving changes on a Security Level Rule.

Security Levels

- Creating, editing, and deleting security zones

Security Zones

- Creating, editing, and deleting security zones

Database - Connections

- Creating, editing or deleting a database connection.

Database - Drivers

- Creating, editing or deleting a JDBC driver
- Creating, editing or deleting a Translator

Store and Forward

- Creating, editing or deleting a Store and Forward engine.

Alarming - General

- Saving changes on the Alarming General settings page.

Alarming - Alarm Journal

- Creating, editing or deleting an Alarm Journal Profile

Alarming - Notification

- Creating, editing, or deleting an Alarm Notification Profile.

Schedules

- Creating, editing or deleting a schedule.
- Creating, editing or deleting a holiday.

Tags - Realtime

- Creating, editing or deleting a Realtime Tag Provider

Tags - Historical

- Creating, editing or deleting a Historical Tag Provider.

OPC Client Connections

- Creating, editing or deleting an OPC connection.

OPC UA - Device Connections

- Creating, editing, or deleting a device connection (editing/saving a device connection configuration without making any changes will be recorded as an edit).
- Editing a Modbus address mapping via gateway interface on Modbus device connections.
- Editing DNP3 Aliased Points via gateway interface on DNP3 device connections.
- Editing tags via gateway interface on Omron NJ device connections.
- Adding FINS tags via gateway interface on Omron FIN device connections.

OPC UA - Server Settings

- Editing the OPC UA Settings page.

Enterprise Administration

- Configuring a gateway to be either an Agent or Controller.

Enterprise Administration - Event Thresholds

- Changes made to Event thresholds.

Enterprise Administration - Controller Settings

- Making changes to the Controller Settings page, including uninstalling the controller.

Enterprise Administration - Agent Settings

- Making changes to the Agent Settings page, including uninstalling the agent.

Enterprise Administration - Agent Management

- Creating, editing, deleting an Agent Group.

Enterprise Administration - License Management

- Adding or removing a license from the License Management page.

Enterprise Administration - Agent Tasks

- Creating, editing, or deleting an agent task
- Separate records are taken each time a task executes.

Sequential Function Charts

- Changes made to the SFC Settings page.

Add a Record Manually

- You can also add a record into the audit profile using the function system.util.audit.

Perspective Auditing Actions

Perspective Sessions generate entries in an assigned audit profile. The following actions are recorded in the Audit Profile:

- Tag changes from a component binding.
- Session login.
- Session logout.
- Authentication level changes (a user's security level changes).

Vision Auditing Actions

The Vision project needs an audit profile configured and auditing enabled. Vision Clients will then log records to an assigned audit profile. Here is a list of audit actions that will be tracked in the Ignition auditing system:

Tags

The following Tag related actions generate entries in the audit log. Note that the functions below must originate from the Tag Browser, the Designer's Scripting Console, or Vision component-based scripts.

- Tag Creation - Including tags created with the Tag Editor and the system.tag.configure function.
- Tag Deletion - Including those deleted from the Tag Browser's UI and the system.tag.deleteTags function.
- Tag Edits - Including edits made to tags from the Tag Editor and the system.tag.configure function.
- Moving Tags - Including moves made by drag-and-drop in the Tag Browser or by calling the system.tag.move function.
- Tag Renames - Renaming a tag generates an entry.

Vision Tag Writes

Write requests sent from a tag either through a standard Tag Binding, Indirect Tag Binding, or manual entry from the Tag Browser.

Vision Component Database Writes

The system explicitly captures modifications made to database tables through the following methods:

- SQL Query Bindings - modifications from the UPDATE Query will be recorded.
- DB Browse Binding - modifications made with the Enable Database Writeback area will be recorded.

Vision User Login/logout

- Logging into a Vision Client will generate an entry in the auditing system, as will logging out of a client.
- Closing the client while logged in is treated as a logout, and will generate a "logout" entry. Note that a logout entry is only recorded if the client is aware that it is closing: entries aren't generated if the designer closed unexpectedly ("crashes").

Database Query Browser

- If the project opened in the Designer has an assigned Audit Profile, then changes made to database tables using the database query browser are automatically recorded to the audit log. "Changes" in this case refer to UPDATE, DELETE, or INSERT statements manually typed and executed from the database query browser.
- Enabling edit mode and applying changes, including typing in new values, adding rows, removing rows, and clearing out fields, are recorded as queries called from the project.

Vision Scripting

The following functions generate entries in the audit log if called from Vision component-based scripts, or from the Designer's Scripting Console.

- system.db.execSProcCall
- system.db.runPrepUpdate
- system.db.runUpdateQuery
- system.tag.writeBlocking
- system.tag.writeAsync
- system.report.executeAndDistribute
- system.report.executeReport

Designer

Designer Login and Closing

- Opening a project in the Designer that has auditing enabled will also generate a login entry in the auditing system. Note that this occurs when the user opens the project, not when they log in using the Designer's login screen: auditing is project-based, so the user has to select a project that is being edited first.
- Closing the Designer effectively counts as logging off, and will generate a "logout" entry. Similar to vision, should the designer close unexpectedly, then an entry will not be recorded.

Database Query Browser

If the project opened in the Designer has an assigned Audit Profile, then changes made to database tables using the database query browser are automatically recorded to the audit log. "Changes" in this case refer to UPDATE, DELETE, or INSERT statements manually typed and executed from the database query browser.

Enabling edit mode and applying changes, including typing in new values, adding rows, removing rows, and clearing out fields, are recorded as queries called from the project.

Alarm Notification

Alarm Notification Attempts

Attempts to send out alarm notifications are recorded in the auditing system. Specifically, the Gateway will record when it attempted to send out a notification, as well as if the attempt failed (such as the SMTP server refusing the request). It is important to note that the auditing system can not report failures that occur outside of the Gateway. Thus, if a voice notification fails to send due to some error in the VOIP system, it's possible that the Gateway won't report the VOIP error, but the audit log will have an entry stating that the Gateway attempted to send the notification.

Reporting Module

Report Execution

Reporting Module Reports generate an entry in the auditing system when a report is executed. Thus:

- Reports running on a schedule will generate an entry.
- Report schedules executed on demand will generate an entry.
- Navigating to a Vision window (in either the Designer or a Vision Client) will trigger a report execution, generating an entry in the auditing system.

Audit Table Definition

The following table describes the audit table as it exist in the database:

Column Name	Description
AUDIT_EVENTS_ID	The id of the row.
ACTION	Brief description of the action.
ACTION_TARGET	The target of the action.
ACTION_VALUE	The value acted upon the action target.
ACTOR	The logged in user when the action occurred or a description of the system that generated the action.
ACTOR_HOST	The host computer where the action occurred.
EVENT_TIMESTAMP	The time when the action occurred.
ORIGINATING_CONTEXT	A numerical description of the origin of the originating system. gateway = 1, designer = 2, client = 4
ORIGINATING_SYSTEM	The name of the project or system where this action occurred.
STATUS_CODE	The quality code where (where applicable). 192 signifies success. 0 signifies bad or failure.

Related Topics ...

- Audit Log and Profiles
- Audit Log Display

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 and Vision Modules come with a host of useful components out of the box, many of which are specialized for industrial controls use. Other modules, like the Reporting module, add more components for specialty purposes.

Configuring components is usually the bulk of a designer's work when designing a project. The basic workflow is to take a component from the palette and drop it into a container or a window. From there, you can use the mouse to drag and resize the component into the correct position. While the component is selected, you can use the component's properties to alter the component's appearance and behavior.

In this section, you can view all the components in the following modules, their properties along with an example of how to configure it.

In This Section ...

Perspective Components

This section covers all the built-in Perspective components. While the component is selected, you can use the Property Editor to alter the component's properties, which changes the component's appearance and behavior. Components are the building blocks of the Designer Interface that when combined create the visual part of a view to do something useful, like display dynamic information or control a device. This section contains all the built-in Perspective components, many of which are specialized for industrial controls use. When the component is selected, you can change the component's properties to alter its appearance and behavior.

Perspective components also have Meta Properties, Position Properties (depending on the container they are in), and the option for Custom Properties.

Here is a complete list of Perspective components and a link pointing to a page containing the component's description, properties, and usage examples.

INDUCTIVE UNIVERSITY

Component Overview

Watch the Video

Perspective - Chart Palette

Perspective - Container Palette

Perspective - Embedding Palette

Perspective - Display Palette

Perspective - Navigation Palette

Perspective - View Object

Perspective - Report Viewer

Perspective - Input Palette

Perspective - Symbols Palette

Perspective - Chart Palette

Chart Components

Charts allow you to display and show off your data in a graphical way.

The following is a complete list of Chart components that give you various options for displaying data, and a link pointing to a page containing the component's description, properties, and an example of how to configure it.

In This Section ...

Perspective - Chart Range Selector

General

Component Palette Icon:

INDUCTIVE UNIVERSITY

Chart Range Selector

Watch the Video

Description

The Chart Range Selector component is a small recreation of a chart that operators can use to intelligently select a time range based on seeing the existing data. This component complements the Time Series Chart component, and should always be used with a chart. Its features include:

- Zoom and pan in/out via mouse wheel interaction.
- Click-and-drag brush range selection and panning.
- Start and End property values that are updated as the brush range changes. These properties can govern the start/end points of data queries to return a dataset.
- Time range showing the overall range of the data being displayed by the brush. (The range updates as the brush is updated.)
- Simple display customization for the axes, baselines, markers, and the overall chart data appearance.
- Label properties have their own dedicated styling properties, such as color and size.

User Interaction

The Chart Range Selector component properties have impact on the way a user can interact with a chart in the runtime.

Interaction	Description
Zoom	The user can zoom in and out on the Chart Range Selector, but can not zoom out past its standard level of zoom.
Pan	The user can pan across the Chart Range Selector.
Refresh	The Chart Range Selector will not refresh its time range if it is zoomed in.
Pinch Zoom	On a mobile device, the user can now pinch-zoom the Chart Range Selector. Zooms must originate from within the boundaries of the displaying chart data.
Brushes	On a mobile device, the user can draw brushes in the Chart Range Selector via a single touch point (multiple touch points will allow zooming to occur). Brushes can be moved in the Chart Range Selector via a single touch point.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description

enablePanZoom	Allow the chart to be panned and zoomed. The chart cannot be zoomed out past its base range.
data	Objects that are the data source for the chart (required). Data can be an object containing a time entry and value entries (all must be numbers) (required). Each value entry must be labeled with the column name to which it corresponds. Data can also be an array containing value entries (all must be numbers). Each value entry consists of a timestamp (which must be the first value) and one or more values that were captured at that time. Finally, data can also be in the form of a dataset, for example, the data property can be bound to a Tag History binding to display either realtime data, or historical data (via start and end dates).
selectedRange	The start and end points of the selected range (required). This property is updated as you interact with the brush.

Name	Description	Property Type
start	A Unix timestamp in milliseconds.	value: numeric
end	A Unix timestamp in milliseconds.	value: numeric

brushRange	An object used to control the display of the date/time range values at the bottom of the component.

Name	Description	Property Type
visible	Whether the brush range is visible.	value: boolean
dateFormat	The date format of the range using a MomentJS date string.	value: string dropdown
timeFormat	The time format of the range using a MomentJS time string.	value: string dropdown

timeAxis	This property provides settings for the X Axis. This property uses the same configuration as the timeAxis property of the Time Series Component.

Name	Description	Property Type
visible	The visible state of the the axis.	value: boolean
tickCount	The number of ticks on the axis (as a multiple of 2, 5, or 10).	value: numeric
height	The height of the axis.	value: numeric
color	The color of the axis. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	value: string
tick	The configuration of the ticks on the axis.	object

Name	Description	Property Type
color	The color of the ticks. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	value: string
label	The configuration of the label drawn on the tick.	object

Name	Description	Property Type
angled	If set to true, the tick labels will be angled rather than horizontal.	value: boolean
format	The date/time format displayed by each tick using a MomentjS date string (https://momentjs.com/). Options are: Auto, Millisecond [638], Second [:35], Hour Minute [8:15], Hour with Meridiem [8 AM], Day of Week, Month, and Hour [Monday 2nd, 08 AM], Abbreviated Day of Week and Month [Mon 2nd], Abbreviated Month and Day of Month [Jan 2nd], Full Month [January], Abbreviated Month and Year [Jan 20], Full Year [2020], [3-2-2020	value: string

	8:15:35], [2020-3-2 8:15:35], Unix Millisecond Timestamp [1563464737269] , Unix Timestamp [1563464737]. Default is “Auto”, where the property attempts to figure out the best format. For a listing of suggested formats, refer to https://momentjs.com/docs/#/parsing/string-format/ Go to the Example at the bottom of this page to see the "Unit [Example] : Notation"	
font	The settings for the label's font. **Name** / **Description** / **Property Type** color / Color of the label text. / value: string size / The font size, in pixels, of the label text. / value: numeric	object
style	Custom CSS styles to apply to the tick labels. Any style that applies to an SVG text element can be used. See also style options.	object

style	Custom CSS styles to apply to the ticks. Any style that applies to an SVG line element can be used. See also style options.	object

style	A style object containing properties which are applied to the horizontal line of the axis. Any property that would apply to an SVG line element can be used here. See also style options.	object

yAxis	An object used to control the display of the Y Axis. This component was built with the assumption that this axis may be secondary information, so it can be de-emphasized, if desired.

Name	Description	Property Type
visible	The visible state of the the axis.	value: boolean
width	The width of the axis, in pixels (required).	value: numeric
label	The configuration of the Y axis label.	object

Name	Description	Property Type
visible	Whether or not the label is visible.	value: boolean
text	The text for the label.	value: string
offset	Offset the Y axis label from its default position. This allows you to fine tune the label location, which may be necessary depending on the scale and how much room the tick labels take up. This may be positive or negative.	value: numeric
font	The settings for the label's font.	object

Name	Description	Property Type
color	The color of the label text.	value: string

	size	The font size, in pixels, of the label text. value: numeric
style	Custom CSS styles to apply to the Y axis label. Any style that applies to an SVG text element can be used. See also style options.	object

tick	The configuration for the ticks drawn on the axis.	object

Name	Description	Property Type
color	The color of the ticks. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	value: string
label	The settings for the label on the tick.	object

Name	Description	Property Type
font	The font style for the label.	object

Name	Description	Property Type
color	The color of the label text.	value: string
size	The font size, in pixels, of the label text.	value: numeric

style	Custom CSS styles to apply to the baseline label. Any style that applies to an SVG `text` element can be used. See also style options.	object

style	Custom CSS styles to apply to the ticks. Any style that applies to an SVG `text` element can be used. See also style options.	object

areaStyles	An object providing default style to the chart trends as a whole.

Name	Description	Property Type
colorScheme	A Color Brewer color scheme to use on the series. See ColorBrewer2.org for available color schemes.	value: string
colors	A list of colors to apply to the columns (in order) for each trend. If these values are provided, they will override the value provided for the colorScheme.	value: string

style	Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a style class .

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example 1 - Using a Format Property in a prop.timeAxis.tick.label

The new `format` property represents the preferred date/time format for the `timeAxis` property. You can enter any preferred date/time format as defined by https://momentjs.com/docs/#/parsing/string-format/ . Below is a listing of suggested formats and how they can be used.

Unix Example : Notation

```
"Millisecond [638]": "SSS",
"Second [:35]": ":ss",
"Hour Minute [8:15]": "h:mm",
"Hour with Meridiem [8 AM]": "h A",
"Day of Week, Month, and Hour [Monday 2nd, 08 AM]": "dddd Do, hh A",
"Abbreviated Day of Week and Month [Mon 2nd]": "ddd Do",
"Abbreviated Month and Day of Month [Jan 2nd]": "MMM Do",
"Full Month [January]": "MMMM",
"Abbreviated Month and Year [Jan 20]": "MMM YY",
"Full Year [2020]": "YYYY",
"[3-2-2020 8:15:35]": "M-D-YYYY h:mm:ss",
"[2020-3-2 8:15:35]": "YYYY-M-D h:mm:ss",
"Unix Millisecond Timestamp [1563464737269]": "x",
"Unix Timestamp [1563464737]": "X"
```

Example 2 - Using the Chart Range Selector

To begin using the Chart Range Selector, a Time Series Chart with trend data will be needed. This example shows how to configure the Chart Range Selector.

1. Begin by configuring tag history on a Tag of your choice.
2. From the Perspective section of the **Project Browser** on your Designer, right click on the **Views** folder and select **New View...** to create a new view.
3. This will bring up the New View window. Give your view a name and select the **Flex** Root Container Type. The Page URL setting will remain unchecked for this example.

4. From the PerspectiveComponent Palette, drag and drop a **Time Series Chart** onto your newly created view. In the **Position** category of the Property Editor, set the **position.grow** property to **'1'**.
5. Drag and drop a Chart Range Selector component onto your view. Go to the **Postion** category and set the Chart Range Selector's **position.grow** property to **'1'**.
6. With the Chart Range Selector selected, go to the **data** property, click on the binding icon to bring up the Binding Editor window and select the **Tag History** binding type as shown in the image below.
7. Set the **Time Range** to **Historical.** We'll configure the binding to span the last one hour of historical data by making the following changes:
   Start Date: `dateArithmetic(now(0), -1, 'hour')`
   End Date: `now(0)`

8. Under the Select Tags section, click on the **Tag** icon and use the Tag browser to drill down to the Tag you created in Step 1
9. Click **OK**.

10. Click **OK** on the Binding Editor window to accept the binding changes. You should now have a Chart Range Selector displaying the last 1 hour of historical data for your Tag created in Step 1.

11. Now select your Time Series Chart and from the Perspective Property Editor, and click on the **binding** icon for the **series[0].data** property to open the Binding Editor window.
12. Select the **Tag History** binding type as shown in the following image.

13. Set the Time Range to **Historical.**
14. The Start Date needs to have a property binding configured pointing to the Chart Range Selector's **props.selectedRange.start** property as in the image below.

15. Similarly, the **End Date** needs to have a property binding configured pointing to the Chart Range Selector's **props.selectedRange.end**.
16. For the Tag History Binding configuration, click on the **Tag** icon on the right of the Select Tags table and drill down to the Tag from Step 1, then click **OK**.

17. After clicking OK and accepting the binding configurations on the Time Series Chart, you will be able to use your Chart Range Selector to select what data you want on your Time Series Chart to display. Simply drag and re-size the Chart Range Selector's brush section as shown below.

2000
1800
1600
1400
1200
1000
800
600
400
200
0
1:09
1:11
1:14
1:16
1:19
6-10-2020 1:07:55 to 1:20:20
2000
1500
1000
500
0
12:46
12:58
1:10
1:21
1:33
6-10-2020 1:07:55 to 1:20:27

Perspective - Pie Chart

General

Component Palette Icon:

Description

A Pie Chart displays a list of named items, each of which has a value that is part of a total. The total is the sum of the value of each item. The key to the Pie Chart component is the **data** property, which contains the items that will be displayed as pie wedges. It is fully customizable in its appearance, from colors, line widths, text styles, and more.

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

The Pie Chart component had two pre-configured variants:

- Flat Chart - Default component described above.
- Three-Dimensional Chart - Component pre-set with three-dimensional pie wedges.

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Pie Chart Example

The Pie Chart component can be used for things like inventory tracking. Below is an example that uses a Pie Chart to display the inventory of a local ice cream shop:

1. From the Perspective section of the Project Browser in your Designer, right click on the Views folder and select **New View...** to create a new view.
2. This will bring up the New View window. Give your view a name and select the Coordinate Root Container type. The Page URL setting will remain unchecked for this example.

3. From the Perspective Component Palette, drag and drop a Pie Chart onto your newly created view.
4. Set the Pie Chart's **title** property to "Ice Cream Inventory".
5. Copy the array below and paste it on the Pie Chart's **data** property.

```
[
  {
    "flavor": "Vanilla",
    "count": "50"
  },
  {
    "flavor": "Chocolate",
    "count": "15"
  },
  {
    "flavor": "Strawberry",
    "count": "68"
  },
  {
    "flavor": "Cookies and Cream",
    "count": "19"
  },
  {
    "flavor": "Mint Chocolate Chip",
    "count": "9"
  },
  {
    "flavor": "Chocolate Chip",
    "count": "53"
  },
  {
    "flavor": "Pistachio",
    "count": "48"
  }
]
```

The array above is a an array of objects. Each object is a dictionary containing key/value pairs where the keys of each dictionary represent ice cream flavors and ice cream inventory counts with their respective values for each. Value types can vary as you can pass both "50" and 50 as counts and the Pie Chart will still be able to render the data correctly. The Pie Chart can have various forms of data sources. The data source array can be built via scripting following the above format or it can be built using a query binding on the Pie Chart's **data** property. The query used must return two columns in any order where each column represents a string and a numeric value to be rendered by the chart.

6. Once you have done this, your Pie Chart should look like this which is an accurate representation of the inventory data for this ice cream shop:

Properties

Name	Description
data	Data source for the chart. Each object within an array defines the name and value for a single pie section.
colors	Colors that correspond to each pie section, respective of order in data. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.
title	Name to display for this chart.
tileColor	Color of the title. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.
valueFormat	Label and legend value format configuration.

Name	Description	Property Type
showPercentSymbol	Whether to show the percent symbol next to the percent value.	value: boolean
showValueAsPercent	Whether to show the value as percent.	value: boolean

Name	Description
showLabels	Whether to show labels for each section of this chart. Default is true (show).
labels	Settings for the labels.

Name	Description	Property Type
showName	Whether to show the name on the label.	value: boolean
showValue	Whether to show the value on the label. Hiding values will disable any value formats set.	value: boolean
bent	Bend labels around chart slices. Default is false.	value: boolean
align	Whether the labels should be aligned in vertical columns.	value: boolean
inside	This feature is new in Ignition version **8.1.2** Click here to check out the other new features	object

Settings for showing labels inside of the chart slices instead of outside.

Name	Description	Property Type
enabled	Value that determines whether to show the labels inside of the chart slices instead of outside, based on if the value percentage is below the percentLimit threshold.	value: boolean
radius	Distance in percentage towards center of Pie Chart while inside is enabled. 0 represents outside edge while 100 would be directly in the middle.	value: numeric
color	Label color for labels while they are displayed inside the chart.	value: color
percentLimit	Value that determines at what value percentage to place label on outside of chart instead of inside.	value: numeric

wrap	Label text wrapping configuration. Ability to wrap long labels.	object

Name	Description	Property Type
enabled	Enables label text wrapping. Default is false.	value: boolean
maxWidth	The maximum allowable label width, in pixels.	value: numeric

color	Color of the labels. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	color

showLegend	Whether to show a legend for this chart. Default is true (show).
legend	

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

Settings for legend.

Name	Description	Property Type
fontSize	Font size for legend labels.	value: numeric
icon	Settings for the icon on entries in the legend.	object

Name	Description	Property Type
height	Height value of legend icon.	value: numeric
width	Width value of legend icon.	value: numeric
enabled	Value that determines whether to show the legend icons or hide them.	value: boolean

position	Aligns legend to specified direction.	value: string

legendLabelColor	Color of the legend labels.
cutoutRadius	Percent of total radius to cut out of center of chart. If greater than zero, the chart becomes ring-style instead of pie.
selectOutline	Outline for each section of the pie. Options are as follows:

Name	Description	Property Type
width	Width of the border (in pixels) around the pie section.	value: numeric
color	Color of border around each pie section. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	color
opacity	Opacity of border around each pie section. 0 is fully transparent, 1 is fully opaque.	value: numeric

enableTransitions	Whether the chart has visual transition effects for changes in chart data.
threeDi	Whether the chart has depth effect to look three-dimensional.

mension al	
style	Use styles to customize the visual style of the component. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a style class.

Perspective - Power Chart

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

General

Component Palette Icon:

INDUCTIVE UNIVERSITY

Power Chart - Overview

Watch the Video

Description

The Power Chart collects and displays data based on the pens that have been configured on the chart. Users can add or remove pens from the chart, which in turn changes the underlying data. It is fully customizable in its appearance, from labels, colors, line widths, legend, scroll bars, text styles, and more.

Note: The Power Chart utilizes functionality provided by the Tag Historian module, and requires a Tag Historian license to function.

The Power Chart has a responsive design and a mobile-optimized display that is different than the standard display. It has a mobile breakpoint so it fits better on mobile devices. The mobile breakpoint is 750px, and is configurable on the 'config' property, `responsiveDesignWidth`, which is d escribed in the Property Table on this page. The chart can change how it's rendered when viewed on smaller devices. The image below shows how the Power Chart renders on a smaller device.

Component Anatomy

Aside from the trending area, the component features several additional areas that provide additional functionality. The diagram below identifies these areas on a standard device.

Browse Tags

The Browse Tags panel allows you to browse for any available historical data, and add it to the chart. It is similar to the Designer's Tag Browser, but this tree reports any records that are accessible from the Tag Historian system, including tables provided by the DB Table Historian Provider. There are two ways to add Tags to the chart's display:

- Select any nodes (entries with the Tag icon), and click **Add Selected Tags** , which will add a pen to the chart that represents the node that was selected.
- Drag selected nodes onto the chart. You can select multiple items by using Ctrl-click. You'll see a prompt indicating how many Tags are selected, i.e., 4 Tags.)

If there's more than one plot in the chart configuration, you will be prompted to choose which plot to add the pen to.

Pen Table

This table shows each pen that's currently on the component, offers some aggregates based on the chart's current range, and provides some quick actions such as hiding the pen and changing it's color.

Chart Settings

This panel allows users to add new objects to the chart, such as new axes and plots.

User Interaction

The top of the component features many interactive toggles and elements. Many of the properties available in the Property Editor are also available through a user interface in the runtime.

Interaction		Description
Browse Tags		Toggles the Tag Browser panel.
Date Range Selector		Allows you to set the range on the chart. There are two modes: • **Realtime**: Shows the most recent chart data based on the given timeframe. Useful in cases where you want to display r

- **Historical**: Allows you to pick at start datetime and end datetime using a popup calendar.

When the chart is in Historical mode, an additional time axis appears. On this access you can narrow down what the c
clicking and dragging the mouse along the axis.

Pan and Zoom		In this mode you can drag or swipe to pan forward and backward in time. On desktop device, clicking and dragging will pan a mobile/touchscreen device, tapping and dragging will pan. The "pinch" and "spread" gestures will zoom.
Zoom reset		To reset the zoom to default, click the zoom reset icon.
X Trace		Tap to place a vertical line, which shows an interpolated value for each pen on the plot. To clear the X Trace values, select **C**

Range Brush		Allows you to select a range of data on the chart. While a brush is active, the Pen Control Table will show aggregate summar brushes will create multiple aggregation summaries.

Individual selections can be removed by clicking the trashcan icon in the Pen Control Table, or by selecting **Clear Range Bru**

Annotate		Click near a trend, line, or data point and you'll have the opportunity to add an annotation. The annotation is stored with the T Annotations can edited by clicking the **Edit** icon and deleted by clicking the **Delete** icon, which appear when hoverin

Full Screen		Puts the chart into full screen mode.
Settings		Opens the Chart settings panel, allowing users to modify various aspects of the chart from the session. There are four tabs: A

Tab	Description
Axes	You can add or delete Axes here, or click the **Edit** icon to edit an existing axis. The options are the same on
Pens	You can add or delete pens here or click the **Edit** icon to edit an existing pen. The options are the same on t

Plots	You can add or delete plots here or click the **Edit** icon to edit an existing plot. The options are the same on tl

Columns	The Columns Chart Settings tab has options for datapoints to display on the Pen Control Panel and datapoints t

Chart Settings
Axes
Pens
Plots
Columns
Select datapoints to display on the Pen Control Panel
Current Value
Minimum
Maximum
Average
X Trace
Axis
Plot
Select datapoints to display on the Range Brush Table
First
Last
Minimum
Maximum
Median
Average
UCL

More Menu	⋮	Provides additional contextual options, depending on the current state of the chart. The button to open the More Menu *only* a following options are available:

Option	Description
Export	Takes the datapoints visible on the various plots, and exports them to a CSV.
Print	Opens print dialog box so user can print the chart.
Clear X Traces	Clears all X Traces on the chart. Only appears when there are X Traces on the chart.
Clear Range Brushes	Clears all range brushes on the chart. Only appears when there are brush selections on the chart.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description
config	Configuration for the data feeding the chart.

Name	Description

tagBrowserStart Path	A path to a nested Tag History provider structure from which browsing will start. The path is expected to contain characters. Example:
	`histprov:Sample_SQLite_Database:/drv:My_Gateway:My_Tag_Provider:/tag:My_Folder/An`
	• histprov - The name of the Tag Historian Provider
	• drv - The historian driver, which is typically a combination of a gateway name and tag provider name separ `tagProvider`
	• tag - A path to a node that has children. Typically should lead to either a folder or the root node of a UDT ir to the root of the drv,
	While providing a path to this property, the tag component can be omitted, which will set the starting path for the particular driver. For example:
	`histprov:Sample_SQLite_Database:/drv:My_Gateway:My_Tag_Provider:/`

mode	The type of query that is being made against the data source. Options are realtime or historical.
refreshRate	Duration (in milliseconds) that data will be queried for updated results. (realtime mode only)

pointCount	Number of data points returned for the selected time range.
startDate	Start date for a historical data query. (historical mode only)
endDate	End date for a historical data query. (historical mode only)
dateFormat	The date format displayed when in historical mode using a MomentJS date string (https://momentjs.com). (histo
timeFormat	The time format displayed when in historical mode using using a MomentJS time string (https://momentjs.com).
rangeSelectorPen	The pen that will drive the data display of the range selector. (historical mode only)
unitOfTime	Time unit used for a realtime data query. (realtime mode only)
measureOfTime	Time measurement used for a realtime data query. Options are seconds, minutes, hours, days, weeks, months,
responsiveDesignWidth	A number (in pixels) that will be used as the switch over width to the responsive design for the chart so it fits bet breakpoint is 750px, and is configurable.
visibility	This feature is new in Ignition version **8.1.2** Click here to check out the other new features Settings to show/hide elements within the component interface.

Name	Description
showTagBrowser	Flag representing the visible state of the Tag Browser. Toggling this property will sho Browser.
showDateRangeSelector	Flag representing the visible state of the Date Range Selector. Toggling this property Range Selector.
showPenControlDisplay	Flag representing the visible state of the Pen Control display. Toggling this property w Control display.
buttons	Settings to show/hide the buttons used in the interface

Name	Description
showTagBrowserButton	Flag representing the visible state of the "Open Tag Browse "Close Tag Browser" buttons.
showPanZoomButton	Flag representing the visible state of the "Pan/Zoom" toggle
showXTraceButton	Flag representing the visible state of the "X Trace" toggle bu
showRangeBrushButton	Flag representing the visible state of the "Range Brush" tog
showAnnotationButton	Flag representing the visible state of the "Annotation" toggle
showFullscreenButton	Flag representing the visible state of the "Fullscreen" toggle
showSettingsButton	Flag representing the visible state of the "Settings" toggle b
showMoreButton	Flag representing the visible state of the "Show More" toggl

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

Settings to control the format of data exported from the chart via the More button's "Export" option.

Name	Description	Property Ty
dateFormat	The date format of the exported data. See https://numeraljs.com for formats.	value: string
timeFormat	The time format of the exported data. See https://momentjs.com for formats.	value: string

interaction

Configuration for the presentation of, and interaction with, chart data.

Name	Description
mode	Current user interaction mode of the chart. Options are panAndZoom, xTrace, rangeBrush, or annotation.
fullscreen	Flag representing the full screen presentation mode of the chart.
xTrace	Configuration to build the xTrace display (mode must be set to xTrace).

Name	Description
values	An array of read-only timestamp values representing the visible x-trace positions.
infoBox	Configuration to build the box portion of the x-trace display.

Name	Description
visible	Whether or not the box is visible.
showTime	Whether or not the time value above the box is visible.
width	The width of the box.
dataFormat	A numeral.js data format for displaying the data for this axis. See https://numeraljs.com for f This feature is new in Ignition version **8.1.2** Click here to check out the other new features As of Ignition 8.1.2, setting the dataFormat property to an empty string will result in no forma applied to the value. Useful in cases where a binding is returning the data in a preformatted
dateFormat	The date format displayed when in historical mode using a MomentJS date string (https://mo com).
timeFormat	The time format displayed when in historical mode using using a MomentJS time string (http js.com).
stroke	A configuration object describing the properties that will be applied to the stroke of the box.
fill	The fill configuration.

Name	Description	Prop Type
color	The color to apply to the line stroke, if applicable.	color
width	The width to apply to the line stroke, if applicable.	value:
opacity	The opacity to apply to the line stroke, if applicable.	value:
dashArray	The spacing to apply between dashes of the line stroke, if applicable.	value:

Name	Description	Property Type
color	The color to apply to the trend fill, if applicable.	color
opacity	The opacity to apply to the trend fill, if applicable.	value: numeric

style	Style for the box. Full menu of style options is available. You can also specify a style class.

line	Configuration to build the vertical line portion of the x-trace display.

Name	Description
visible	Whether or not the line is visible.
color	The color to apply to the line stroke, if applicable.
width	The width to apply to the line stroke, if applicable.
opacity	The opacity to apply to the line stroke, if applicable.
dashArray	The spacing to apply between dashes of the line stroke, if applicable.
style	Style for the box. Full menu of style options is available. You can also specify a style class

rangeBrush	Configuration to build the range brush display (mode must be set to rangeBrush).

Name	Description
values	An array of config objects to build each range brush.

Name	Description	Property Type
start	The start timestamp position.	value: string
end	The end timestamp position.	value: string

active	Configuration to build the active range brush display.

Name	Description
color	The color of the active range brush.
opacity	The opacity of the active range brush.
style	Style settings for the active range brush. Full menu of style options is available. You can also sp a style class.

inactive	Configuration to build the inactive range brush displays.

Name	Description
color	The color of the inactive range brush.
opacity	The opacity of the inactive range brush.
style	Style settings for the inactive range brush. Full menu of style options is available. You can also specify a style class.

annotation	Configuration to build the annotation display (mode must be set to annotation).

infoBox	Configuration to build the box portion of the annotion display.

Name	Description
visible	Whether or not the box is visible.
showTime	Whether or not the time value above the box is visible.
width	The width of the box.
dateFormat	The date format displayed when in historical mode using a MomentJS date string (https://m
timeFormat	The time format displayed when in historical mode using using a MomentJS time string (htt com).
stroke	A configuration object describing the properties that will be applied to the stroke of the box.

Name	Description	Pro
color	The color to apply to the line stroke, if applicable.	color
width	The width to apply to the line stroke, if applicable.	value
opacity	The opacity to apply to the line stroke, if applicable.	value
dashArray	The spacing to apply between dashes of the line stroke, if applicable.	value

fill	The fill configuration.

Name	Description	Property Type
color	The color to apply to the trend fill, if applicable.	color
opacity	The opacity to apply to the trend fill, if applicable.	value: numeric

style	Style for the box. Full menu of style options is available. You can also specify a style class.

line	Configuration to build the connecting line portion of the annotation display.

Name	Description
visible	Whether or not the line is visible.
color	The color to apply to the line stroke, if applicable.
width	The width to apply to the line stroke, if applicable.
opacity	The opacity to apply to the line stroke, if applicable.
dashArray	The spacing to apply between dashes of the line stroke, if applicable.
style	Style for the box. Full menu of style options is available. You can also specify a style class.

dot	Configuration to build the dot portion of the annotation display.

Name	Description
visible	Whether or not the dot portion is visible.
color	The color to apply to the dot.
radius	The radius of the dot.
opacity	The opacity of the dot.
styke	The style settings for the dot. Full menu of style options is available. You can also specify a style c

axes	Collection of predefined axes against which the data visualizations can be drawn.

Name	Description
name	The name of the axis.
position	The side of the plot upon which the axis should be drawn. Options are left or right.
width	The width of the axis.
color	The color of the Y axis vertical bar.
dataFor mat	A numeral.js data format for displaying the data displayed in the pen control portion of the chart for this axis. See https://r
range	Configuration for the upper and lower limits of the axis.

Name	Description
auto	If true, the minimum and maximum displaying values for the axis will be auto calculated.
min	Minimum range value. If no value is provided, a minimum value will be calculated from the data bound to this
max	Maximum range value. If no value is provided, a maximum value will be calculated from the data bound to th

Name	Description
label	The label configuration for the Y axis.

Name	Description
text	The text of the Y axis label.
offset	Offset the Y axis label from its default position. This enables you to fine tune the label location, which may be the scale and how much room the tick labels take up. Value may be positive or negative. Default is 0.
font	Font configuration for the Y access label. Font size and color options for the font.
style	Style for the Y access label. Full menu of style options is available. You can also specify a style class.

Name	Description
tick	Tick configuration.

Name	Description
color	The color of the ticks.
label	Tick label configuration

Name	Description
font	Label font configuration. Font size and color options for the font.
style	Style for the tick label. Full menu of style options is available. You can also specify a style class.

Name	Description
style	Style for the tick. Full menu of style options is available. You can also specify a style class.

Name	Description
style	Style for the display. Full menu of style options is available. You can also specify a style class.

pens	Visual representation of each active item

Name	Description
name	Name of the pen.

visible	Whether or not the pen is visible on the chart.
enabled	Availability of the pen on the chart and pen configuration panel.
selectable	Flag to allow the pen to be responsive to user selection.
axis	Name of an axis in the "axes" array to plot against. If left blank, a default axis will be created based on data values.
plot	The plot to which this pen is bound.
display	Configuration that drives the display of the pen.

Name	Description
type	The type of chart to be built. Options are line, area, bar, or scatter.
interpolation	Type of curve that should be used to draw the line portion of the chart. Options are: curveBasis, curveBasisC curveCardinalOpen, curveCatmullRom, curveCatmullRomOpen, curveLinear, curveMonotoneX, curveMonot curveStep, curveStepAfter, or curveStepBefore. More information on the interpolation methods above can be found in D3's documentation.
breakLine	If true, the line will be broken on either side of bad/missing data values. If false, bad/missing data values are points are connected.
styles	Settings for the display when it is normal, highlighted, selected, or muted.

Name	Description
normal	An object providing style configuration for the “normal” state (no user interaction) of a column, or in a trend. Any color values specified here will override values set in the colorScheme or colors p
highlighted	An object providing style configuration for the “highlighted” state (mouse hover) of a column, or d in a trend. Any color values specified here will override values set in the colorScheme or colors p The highlighted property uses the same configuration properties as the `'normal'` property abov
selected	An object providing style configuration for the “selected” state (mouse click) of a column, or data trend. Any color values specified here will override values set in the colorScheme or colors prope
muted	An object providing style configuration for the “muted” state (non selected) of a column, or data e trend. Any color values specified here will override values set in the colorScheme or colors prope

Each of these four display settings has the same set of properties.

Name	Description
stroke	A configuration object describing the properties that will be applied to the stroke of the trend displayed (if applicable). The `line`, and `area` trend types will have these styles applied to t

Name	Description	Prop Type
color	The color to apply to the line stroke, if applicable.	color
width	The width to apply to the line stroke, if applicable.	value:
opacity	The opacity to apply to the line stroke, if applicable.	value:
dashArray	The spacing to apply between dashes of the line stroke, if applicable.	value:

Name	Description
fill	The fill configuration.

Name	Description	Property Type

color	The color to apply to the trend fill, if applicable.	color
opacity	The opacity to apply to the trend fill, if applicable.	value: numeric

data	

Name	Description
source	Source of the data behind the pen.
aggregateMode	Mode to use to group the data. Options are: default, Average, MinMax, SimpleAverage, Sum, Minimum, Ma DurationOff, CountOn, CountOff, Count, Range, Variance, StdDev, PctGood, or PctBad.

plots	A plot represents a row containing one or more pens.

Name	Description
relativeWeight	Ratio between all plots.
color	Background color of the plot.
markers	An array of markers that can be added to the plot to better visualize the data being displayed.

Name	Description
type	Type of marker to add to the plot. Options are line or band.
axis	Name of the axis against which the marker should be drawn. This must be specified for the marker to be d
value	Value where the line marker should be drawn.
display	Configuration for the display of the marker.

Name	Description
color	Color of the marker
width	Width of the line.
opacity	Opacity of the marker
dashArray	Dashed appearance of the marker.
label	Configuration for the label of the marker.

Name	Description	Prop Type
text	Text for the label.	value
position	The position of the label relative to the line. Options are right or left.	value
font	Label font configuration. Font size and color options for the font.	objec
style	Style for the label. Full menu of style options is available. You can also specify a style class	objec

style	Style for the display. Full menu of style options is available. You can also specify a style c

style	Style for the individual plot. Full menu of style options is available for text, background, margin and padding, border, sh specify a style class

dataColumns	Configuration for the data columns that can be shown in tabular displays throughout the chart.

Name	Description
penControl	Configuration for the data columns that can display for pens.

Name	Description	Property Type
currentValue	Show the "current value" column for the pen based on the time range.	value: boolean
minimum	Show the "minimum" column for the pen based on the time range.	value: boolean
maximum	Show the "minimum" column for the pen based on the time range.	value: boolean
average	Show the "average" column for the pen based on the time range.	value: boolean
axis	Show the "axis" column for the pen.	value: boolean
plot	Show the "plot" column for the pen.	value: boolean
xTrace	Show the "xTrace" column for the pen based on the time range.	value: boolean

Name	Description
rangeSelection	Configuration for the data columns that can display for the range brush.

Name	Description	Property Typ
first	Show the "first" column for the pen based on the time range.	value: boolean
last	Show the "last" column for the pen based on the time range.	value: boolean
average	Show the "average" column for the pen based on the time range.	value: boolean
minimum	Show the "minimum" column for the pen based on the time range.	value: boolean
maximum	Show the "minimum" column for the pen based on the time range.	value: boolean
median	Show the "median" column for the pen based on the time range.	value: boolean
delta	Show the "delta" column for the pen based on the time range.	value: boolean
sum	Show the "median" column for the pen based on the time range.	value: boolean
ucl	Show the "UCL" column for the pen based on the time range.	value: boolean
lcl	Show the "LCL" column for the pen based on the time range.	value: boolean
standardDeviation	Show the "standardDeviation" column for the pen based on the time range.	value: boolean

title	Configuration for the title of the chart.

Name	Description	Property Type
visible	Whether or not the title is visible.	value: boolean
text	The text for the title.	value: string
font	Title font configuration. Font size and color options for the font.	object
style	Style for the display. Full menu of style options is available. You can also specify a style class.	object

timeAxis	Configuration for the time axis (X axis) of the chart.

Name	Description
visible	Whether or not the time axis is visible.
tickCount	The number of ticks.
height	The height of the time axis.

	color	The color of the axis.
	tick	Tick configuration.

Name	Description
color	The color of the ticks.
label	Tick label configuration

Name	Description	I
angled	Whether or not the tick labels are angled.	va
format	Date/time format displayed by each tick using a MomentJS data string (https://momentjs.com).	va
font	Label font configuration. Font size and color options for the font.	ol
style	Style for the tick label. Full menu of style options is available. You can also specify a style class.	ol

style	Style for the tick. Full menu of style options is available. You can also specify a style class.

style	Sets a style for this timeAxis. Full menu of style options is available for text, background, margin and padding, border, shape specify a style class.

legend	Configuration for the display of the legend for the chart.

Name	Description	Property Type
visible	Whether or not the legend is visible.	value: boolean

style	Sets a style for this chart. Full menu of style options is available for text, background, margin and padding, border, shape and miscellanec

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Perspective - Simple Gauge

General

Component Palette Icon:

Description

The Simple Gauge component in Perspective provides a way to show realtime values in a range as they change. This gauge is a less complicated version of the Gauge component. It has just one axis, is easy to configure and is customizable in its appearance.

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

The Simple Gauge component had three pre-configured variants:

- Half Circle - Default layout with a half-circle gauge.
- 3/4 Circle - Layout with a 3/4 circle gauge.
- Full Axis - Layout with a full axis gauge.

Properties

> Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
> This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
value	Numeric value for the gauge to display. Default is 0.	value: numeric
minValue	Minimum gauge value for this gauge. Default is 0.	value: numeric
maxValue	Maximum gauge value for this gauge. Default is 100.	value: numeric
startAngle	Radial position for the start of the gauge's arc. Default is 180.	value: numeric
endAngle	Radial position for the end of the gauge's arc. Default is 360.	value: numeric
arc	The arc is a radial band that displays the gauge's value.	object

Name	Description	Property Type
width	Width of the line (in pixels) that represents the arc. Default is 20.	value: numeric
color	Color of the arc line showing the gauge's value. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	color
cornerRadius	Amount to round the edges of the arc. Default is 0.	value: numeric

arcBackground	Background or 'track' for the gauge arc. Shows shape and total potential value behind the arc.	value: numeric

Name	Description	Property Type
color	Color of the arc background. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value.	color
opacity	Opacity of the arc. 0 is fully transparent, 1 is fully opaque.	value: numeric

label	The label for the gauge is displayed as text, with optional units.	object

Name	Description	Property Type
visible	Indicates whether or not the label is visible. Default is true (visible).	value: boolean
size	Font size to display label text. Default is 25.	value: numeric
units	Any unit information to append to the value on the label.	value: string
maxDecimal	Maximum number of digits after decimal to display in the label. If null, full value will display.	value: numeric

animate	Whether needle should be animated in a sweeping motion when value changes. Default is false.	value: boolean
style	Sets a style for this gauge. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

Property	Value
Value	32.4567
props.arc.width	15
props.arc.cornerRadius	25
props.arc.color	#FF8C00
props.arcBackground.opacity	0.4
props.label.units	psi
props.label.maxDecimal	1

Perspective - Time Series Chart

General

Component Palette Icon:

INDUCTIVE UNIVERSITY

Time Series Chart

Watch the Video

Description

The Time Series component provides an efficient way to visualize data from a variety of different data sources as chart data. Time series chart requires that the X axis of the chart represents time and the Y axis represents values. The Time Series Chart includes the following features:

- Zoom or pan in and out via mouse wheel interaction.
- X-Trace display showing data at the hovered time position.
- Multiple chart display types (Area, Bar, Line, and Scatter).
- Multiple Y axes with the ability to align to the left or right side of the chart.
- Multiple plots as well as multiple trends per plot.
- Baselines and markers.
- Custom axes.
- Time range showing the overall range of the data being displayed in the chart
- Simple display customization for the axes, different trend display types, baselines, and markers.
- Label and Title properties have their own dedicate styling properties, such as color and size.

By default, the charts contain example data, but typically a tag history binding or named query binding will be used to feed data to the charts.

The Chart Range Selector provides a complement to this chart. The Time Series Chart and Chart Range Selector components are most powerful when paired together.

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

The Time Series Chart component had four pre-configured variants:

- Line chart - Default layout with appearance of a line chart.
- Area chart - Layout set up as an area chart.
- Bar chart - Layout set up as a bar chart.
- Scatter chart - Layout set up as a scatter chart.

User Interaction

The Time Series component properties have impact on the way a user can interact with a chart in the runtime.

Interaction	Description
Zoom	The user can zoom in and out on the Time Series chart. When zoomed to any level past its base time range, the Time Series chart will display a **zoom reset** icon in the upper right corner. Click on the icon to return the chart to its base range. Note that the range will not refresh while zoomed in.
Pan	The user can pan across the Time Series chart. When panned past its base time range, the Time Series chart will display a **pan reset** icon in the upper right corner. Click on the icon to return the chart to its base range. Note that the range will not refresh while panning.
Pinch Zoom	On a mobile device, the user can pinch-zoom the Time Series chart. Zooms must originate from within the boundaries of the displaying chart data.
Tracker Position	On a mobile device, the user can move the tracker position on the Time Series chart via touch.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description
enablePanZoom	Allows the chart to be panned and zoomed. The chart cannot be zoomed out past its base range.
series	A list of series entries used as the base data source for the chart display. Each series will be a new line drawn on the chart. At a minimum component requires at least one complete series entry to display data. With no other configuration provided, an initial display will be creat data (required). Each series entry requires the following properties:

Name	Description
name	The name of the series. A name must be provided or one will be added. This will also be used as the label of the auto-generated Y axis in the chart (required).
data	Data can be an object containing a time entry and value entries (all must be numbers) (required). Each value entry must be labeled with the column name to which it corresponds. Data can also be an array containing va entries (all must be numbers). Each value entry consists of a timestamp (which must be the first value) and one or more values that were captured at that time. Finally, data can also be in the form of a DataSet, for example, the data property c be bound to a Tag History binding to display either realtime data, or historical data (via start and end dates). **Note:** When using a dataset, the Time column should be the first column.

Name	Description
plots	A list of plots (subplots) for the chart. At least one entry is required. Plot entries contain properties that allow much finer control over the w data can be displayed. Each plot will be represented by its own row in the component (plots always stretch to fill the width of the chart). A following properties:

Name	Description
trends	If specified, a trend will become the display mechanism for the plot in which it resides. It describes the way that the data shou look, and provides different display types as opposed to the `line` display type that is provided when building from series data Multiple trends will be built on top of each other in the same `plot`. An axis will be generated for each `trend`, unless they sha common axis (either custom created, or generated from `series` data). A trend contains the following properties: (optional)

Name	Description	Prope Type
visible	If specified, this is the visible state of the trend. If not visible, the trend data will be hidden, but the time range of the trend will still be represented in the overall time range of the plot.	value: boolea
type	The type of chart to create (required). Options are area, bar, line, and scatter.	value: string
series	The series used to feed data to this trend (required).	value: string
interpolation	The type of curve that should be used to draw the line portion of the chart. Options are: curveBasis, curveBasisOpen, curveCardinal, curveCardinalOpen, curveCatmullRom, curveCatmullRomOpen, curveLinear, curveMonotoneX, curve MonotoneY, curveNatural, curveStep, curveStepAfter, curveStepBefore. More information on the interpolation methods above can be found in D3's documentation.	value: string
breakLine	This property will be available when a trend of type **line** is being used. If true, the line will be broken on either side of bad/missing data values. If false, bad/missing data values are removed and the adjoining points are connected.	value: boolea
stack	This property will be available when a trend of type **area** or bar is being used. If true, the multiple columns of the chart will stack on top of each other.	value: boolea
axis	If specified, the name of an axis that is described in the axesproperty of the plot to which this trend belongs. This axis will be used as the Y axis for the trend.	value: string
radius	This property will be available when a trend of type **scatter** is being used. A number specifying the radius (in pixels) of the displaying points.	value: numeri
columns	If provided, only the columns in this list will be shown. Any style properties provided here will also override any existing style currently in place. A column contains the following properties:	array

Name	Description	Property Type
key	This needs to match a column name from the series to which this trend is bound (required). Once in place, this will allow the style configuration provided here to override the styles provided in the `defaultStyles` property.	value: string
color	If provided, this value will override any previous color values for the column (both stroke and fill). Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	value: string
styles	If provided, the styles for the state values listed here will override any previous state values. The state value options are:	object

Name	Description	Property Type
normal	Takes the same configuration options as the same named state value under the defaultStyles property.	object
highlighted	Takes the same configuration options as the same named state value under the defaultStyles property.	object
selected	Takes the same configuration options as the same named state value under the defaultStyles property.	object
muted	Takes the same configuration options as the same named state value under the defaultStyles property.	object

baselines	If specified, a line will be drawn on the trend based upon a given type of functionality. Options as follows:	

Name	Description	Property Type

visible	Whether the baseline is visible or not.	value: boolean
function	The type of baseline that will be drawn (required). Options are min, max, avg, ucl (upper control limit), and lcl (lower control limit).	value: string
column	The column against which the baseline should be calculated. If not specified, the first column in the series to which the trend is bound will be used.	value: string
axis	The axis against which the baseline should be calculated. The trend must be bound to an axis for this to work correctly.	value: string
color	The color of the line. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	value: string
width	The width of the line, in pixels.	value: numeric
opacity	The opacity of the line, ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque.	value: numeric
dashArray	The dashed appearance (SVG dashed array) of the line. The pattern of dashes and gaps used to paint the stroke. It's a list of space separated lengths (in pixels) and percentages (percentage of the total stroke length) that specify the lengths of alternating dashes and gaps. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. Thus, "5,3,2" is equivalent to "5,3,2,5,3,2".	value: numeric
label	The configuration used for the label drawn on the baseline.	object

Name	Description	Property Type
text	The label text.	value: string
position	The position of the label relative to the baseline.	value: string
font	The font style for the label.	object

Name	Description	Property Type
color	The color of the label text.	value: string
size	The font size, in pixels, of the label text.	value: numeric

Name	Description	Property Type
style	Custom CSS styles to apply to the baseline label. Any style that applies to an SVG text element can be used. See also style options.	object

style	Custom CSS styles to apply to the line portion of the baseline. Any style that applies to an SVG `line` element can be used. See also style options.	object

axes

A list of axis entries for that can be used for the plot. These can be used to override the Y Axis that is generated based on dat the series to which each trend is bound. An axis entry has the following properties.

Name	Description	Prope Type
name	The name of the axis (required). Also used as the display label of the axis.	value: string
min	The minimum value of the axis (required). If no value is specified, auto range will be used. A minimum value will be calculated from the data bound to this axis.	value: numeri

max	The maximum value of the axis (required). The maximum range value of the axis. If no value is specified, auto range will be used. A maximum value will be calculated from the data bound to this axis	value: numeri
alignment	The side of the trend upon which the axis should be presented (required).	value: string
width	The width of the axis, in pixels (required).	value: numeri
label	The configuration of the Y axis label.	object

Name	Description	Property Type
visible	Whether or not the label is visible.	value: boolean
text	The text for the label.	value: string
offset	Offset the Y axis label from its default position. This allows you to fine tune the label location, which may be necessary depending on the scale and how much room the tick labels take up. This may be positive or negative.	value: numeric
font	The settings for the label's font.	object

Name	Description	Property Type
color	The color of the label text.	value: string
size	The font size, in pixels, of the label text.	value: numeric

Name	Description	Property Type
style	Custom CSS styles to apply to the Y axis label. Any style that applies to an SVG `text` element can be used. See also style options.	object

tick	The configuration for the ticks drawn on the axis.	object

Name	Description	Property Type
color	The color of the ticks. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	value: string
label	The settings for the label on the tick.	object

Name	Description	Property Type
font	The font style for the label.	object

Name	Description	Property Type
color	The color of the label text.	value: string
size	The font size, in pixels, of the label text.	value: numeric

Name	Description	Property Type
style	Custom CSS styles to apply to the baseline label. Any style that applies to an SVG `text` element can be used. See also style options.	object

Name	Description	Property Type
style	Custom CSS styles to apply to the ticks. Any style that applies to an SVG `text` element can be used. See also style options.	object

markers	Settings for the markers, a list of visual indicators that can be added to the plot (optional). These are meant to draw emphasis the data. In this first release, the line marker will be the only available option.

Name	Description	Prope Type
visible	Indicates whether or not the marker is **visible**.	value: boolea
value	The numeric value represented by the marker.	value: numeri
type	The type of marker. Currently the only option is line.	value: string
axis	The axis against which the marker should be drawn (required).	value: string
line	The configuration for the line portion of the marker.	object

Name	Description	Property Type
width	The width of the marker line, in pixels.	value: numeric
color	The color of the marker line. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	value: string
opacity	The opacity of the marker line, ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque.	value: numeric
dashArray	The pattern of dashes and gaps (SVG dashed array) used to paint the marker line. It's a list of space separated lengths (in pixels) and percentages (percentage of the total stroke length) that specify the lengths of alternating dashes and gaps. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. Thus, "5 3 2" is equivalent to "5 3 2 5 3 2".	value: numeric
label	The configuration for marker's label.	object

Name	Description	Property Type
text	The marker label text.	value: string
position	The position of the label relative to the line.	value: string
font	The font settings for the label.	object

Name	Description	Property Type
color	The color of the label text. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL v alue. See Col or Selector.	value: string
size	The font size, in pixels, of the label text.	value: numeric
style	Custom CSS styles to apply to the marker label. Any style that	object

	applies to an SVG `text` element can be used. See also style options.	
style	Custom CSS styles to apply to the marker line. Any style that applies to an SVG `line` element can be used. See also style options.	object
style	Custom CSS styles to apply to the marker line. Any style that applies to an SVG line element can be used. See also style options.	object

title

Settings for the title of the chart.

Name	Description
visible	Indicates whether or not the title is visible. Default is false.
text	Text for the title of the chart.
height	The vertical space taken up by the title.
font	Title font configuration. Options as follows:
style	Sets a style for the title. Full menu of style options is available for text, background, margin and padding, border, shape a miscellaneous. You can also specify a style class.

Name	Description	Prope Type
color	Color of the title text. Can be chosen from color wheel, chosen from color palette, or entered as R GB or HSL value. See Color Selector.	value:
size	The font size, in pixels, of the title text.	value: numeri

timeAxis

This property provides settings for the X Axis. Note that multiple plots share the same axis. (required)

Name	Description
visible	The visible state of the the axis.
tickCount	The number of ticks on the axis (as a multiple of 2, 5, or 10).
height	The height of the axis.
color	The color of the axis. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Colo Selector.
tick	The configuration of the ticks on the axis.

Name	Description	Prope Type
color	The color of the ticks. Can be chosen from color wheel, chosen from color palette, or entered as R GB or HSL value. See Color Selector.	value: string
label	The configuration of the label drawn on the tick.	

Name	Description	Property Type
angled	If set to true, the tick labels will be angled rather than horizontal.	value: boolean
format	The date/time format displayed by each tick using a MomentjS date string (https://momentjs.com/). Default is “Auto”, where the property attempts to figure out the best format. For a listing of suggested formats, refer to https://momentjs.com/docs/#/parsing/string-format/	value: string
font	The settings for the label's font. **Name** / **Description** / **Property Type** color — Color of the label text. — value: string size — The font size, in pixels, of the label text. — value: numeric	object
style	Custom CSS styles to apply to the tick labels. Any style that applies to an SVG text element can be used. See also style options.	object

style	A style object containing properties which are applied to the horizontal line of the axis. Any property that would apply to an SVG line element can be used here. See also style options.

timeRange

An object describing the presentation of the time range display below the chart. The following properties are available:

Name	Description	Property Type
visible	The visible state of the time range display as a whole.	value: boolean
dateFormat	The date format of the range using a MomentJS date string.	value: string
timeFormat	The time format of the range using a MomentJS date string.	value: string

xTrace

Configuration to build the x-trace display when hovering over the chart.

Name	Description
value	A read only timestamp representing the current x-trace position. If there is no active x-trace position, this value will be an empty string.
visible	The visible state of the x-trace display.
infoBox	Configuration to build the box portion of the x-trace display.

Name	Description	Prope Type
visible	The visible state of the box.	value: boolea
showTime	Whether to display the timestamp of the current X Trace value above the info box.	value: boolea
width	Width of the info box, in pixels.	value: numeri
dateFormat	The date format of the xtrace date/time display using a Momentjs date string (https://momentjscom/). Options are: [7-18-2019], [2019-7-18], [07-18-2019], [2019-07-18], [Jul 18th 19], [Jul 18th 2019], or none.	value: string
timeFormat	The time format of the xtrace date/time display using a MomentJS time string (https://momentjs.com/). Options are: 12 hour [8:41:06], 12 hour w/ period [8:41:06 AM], 24 hour [08:41:06], 24 hour	value: string

	w/milliseconds [08:41:06:269], Unix Millisecond Timestamp [1563464737269], Unix Timestamp [1563464737], or none.	
dataFormat	A NumeralJS value used to format the data found at the current timestamp of the X Trace display. See numeral.js for a list of available data formats. Options are: number [1,000.12], integer [1,200], four decimal precision [1.1200], percent [10.12%], scientific [1.01E+03], accounting [$(1,000.12)], financial [(1,000.12)], currency [$1,000.12], currency (rounded) [$1,012], duration [24:01:00], abbreviation [1.2k], or ordinal [100th].	value: string
stroke	A configuration object describing the properties that will be applied to the stroke of the box display.	object

Name	Description	Property Type
color	The color of the box stroke. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	value: string
width	The width of the box stroke, in pixels.	value: numeric
opacity	The opacity of the box stroke, ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque.	value: numeric
dashArray	The dashed appearance (SVG dashed array) of the box stroke. It's a list of space separated lengths (in pixels) and percentages (percentage of the total stroke length) that specify the lengths of alternating dashes and gaps. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. Thus, "5,3,2" is equivalent to "5,3,2,5,3,2".	value: numeric

fill	A configuration object describing the properties that will be applied to the fill of the box display.	object

Name	Description	Property Type
color	The color of the box fill. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	value: string
opacity	The opacity of the box fill, ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque.	value: numeric

style	Custom CSS styles to apply to the info box. Any style that applies to an SVG `line` element can be used. See also style options.	object

line

Configuration to build the vertical line portion of the x-trace display.

Name	Description	Prope Type
visible	The visible state of the line.	value: boolea
color	The color of the line. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	value: string
width	The width of the line, in pixels.	value: numeri
opacity	The opacity of the line, ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque.	value: numeri
dashArray	The dashed appearance (SVG dashed array) of the line. It's a list of space separated lengths (in pixels) and percentages (percentage of the total stroke length) that specify the lengths of alternating dashes and gaps. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. Thus, "5,3,2" is equivalent to "5,3,2,5,3,2".	value: numeri
style	Custom CSS styles to apply to the line. Any style that applies to an SVG line element can be used. See also style options.	object

legend

Configuration for the display and position of the legend for the Time Series Chart.

Name	Description

visible	The visible state of the legend. Default is false.	v
position	The position of the legend. Options are top, right, bottom, or left.	v s
style	A style object containing properties which are applied to the legend. See also style options. You can also specify a style class.	c

defaultStyles

An object providing style settings to the chart trends as a whole.

Name	Description
colorScheme	Specifies a Color Brewer color scheme to use on the series. See ColorBrewer2.org for available color schemes.
colors	A list of colors to apply to the columns (in order) for each trend. If these values are provided, they will override the value provided for the colorScheme.
normal	An object providing style configuration for the “normal” state (no user interaction) of a column, or data entry, in a trend. A color values specified here will override values set in the colorScheme or colors properties.

Name	Description	Prope Type
stroke	A configuration object describing the properties that will be applied to the stroke of the trend type being displayed (if applicable). The `line`, and `area` trend types will have these styles applied to them.	object

Name	Description	Property Type
width	The width of the trend stroke, in pixels.	value: numeric
opacity	The opacity of the trend stroke, ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque.	value: numeric
dashArray	The dashed appearance (SVG dashed array) of the trend stroke. It's a list of space separated lengths (in pixels) and percentages (percentage of the total stroke length) that specify the lengths of alternating dashes and gaps. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. Thus, "5,3,2" is equivalent to "5,3,2,5,3,2".	value: numeric

Name	Description	Prope Type
fill	A configuration object describing the properties that will be applied to the fill of the trend type being displayed (if applicable). area, and box, and scatter trend types will have these styles applied to them.	object

Name	Description	Property Type
opacity	The opacity of the trend fill, ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque.	value: numeric

Name	Description
highlighted	An object providing style configuration for the “highlighted” state (mouse hover) of a column, or data entry, in a trend. Any color values specified here will override values set in the colorScheme or colors properties. The highlighted property uses same configuration properties as the `'normal'` property above.

Name	Description	Prope Type
stroke	A configuration object describing the properties that will be applied to the stroke for the “highlighted” state (mouse hover) of a column, or data entry, in a trend. The `line`, and `area` trend types will have these styles applied to them.	object

Name	Description	Property Type
opacity	The opacity of the trend stroke, ranging from 0 to 1. 0 is fully	value:

	transparent, 1 is fully opaque.	numeric

fill	A configuration object describing the properties that will be applied to the stroke for the “highlighted” state (mouse hover) of a column, or data entry, in a trend.The area, and box, and scatter trend types will have these styles applied to them.	object

Name	Description	Property Type
opacity	The opacity of the trend fill, ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque.	value: numeric

selected	An object providing style configuration for the “selected” state (mouse click) of a column, or data entry, in a trend. Any co values specified here will override values set in the colorScheme or colors properties.	

Name	Description	Prope Type
stroke	A configuration object describing the properties that will be applied to the stroke for the “selected” state (mouse hover) of a column, or data entry, in a trend. The `line`, and `area` trend types will have these styles applied to them.	object

Name	Description	Property Type
opacity	The opacity of the trend stroke, ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque.	value: numeric

fill	A configuration object describing the properties that will be applied to the stroke for the “selected” state (mouse hover) of a column, or data entry, in a trend. The area, and box, and scatter trend types will have these styles applied to them.	object

Name	Description	Property Type
opacity	The opacity of the trend fill, ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque.	value: numeric

muted	An object providing style configuration for the “muted” state (non selected) of a column, or data entry, in a trend. Any colo values specified here will override values set in the colorScheme or colors properties.	

Name	Description	Prope Type
stroke	A configuration object describing the properties that will be applied to the stroke for the “muted” state (mouse hover) of a column, or data entry, in a trend. The `line`, and `area` trend types will have these styles applied to them.	object

Name	Description	Property Type
opacity	The opacity of the trend stroke, ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque.	value: numeric

fill	A configuration object describing the properties that will be applied to the stroke for the “muted” state (mouse hover) of a column, or data entry, in a trend.The area, and box, and scatter trend types will have these styles applied to them.	object

Name	Description	Property Type
opacity	The opacity of the trend fill, ranging from 0 to 1. 0 is fully	value:

					transparent, 1 is fully opaque.	numeric
style	Sets a style for this chart. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneo specify a style class.					

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example - Displaying History in a Time Series Chart

This example shows a Time Series Chart displaying Temperature and Pressure values for Tank 100.

1. Create a new view, and drag an **Time Series Chart** component on to your view.

2. With the Time Series Chart selected, click on the chain link **Binding icon** under the **series > 0 > data** property.

3. This will open the Edit Binding window. Select the **Tag History** binding type.
4. From the **Select Tags** section, click on the Tag Browse icon on the far right. Browse to select your Tag path. We selected **Realistic1** and **Realistic2** from the Generic > Realistic folder.

5. Now let's configure the following Tag History binding settings:
    a. Double click in the Alias column for each Tag to add an Alias
        i. Add the **Alias** for **Realistic1** as **temp.**
        ii. Add the **Alias** for **Realistic2** as **pressure.**
    b. Set the **Point** count to **100.**
    c. Under **Most Recent,** select **MIN** from the dropdown.
    d. Set **Polling** to **2.**
    e. The Tag History binding returns a **Dataset**. You will see the format type in the Binding Preview in the lower left.

6. Click **OK**.

7. Now you have Tag History data in your Time Series Chart.
8. To add a title, scroll to the **Title** object in the PROPS section and set the following values:

    a. **title.visible** - Set to **true**.
    b. **title.text** - Enter a title: **Tank 100 Pressure and Temp Tracking**.
    c. **legend.visible** - Set to true.

9. **Save** your project.
10. In **Preview Mode**, when you hover over the chart you will see a timestamp and Temp and Pressure values representing the current x-trace position.

Tank 100 Temperature and Pressure
12-9-2020 2:57:57
temp: -26.98
pressure: -41.92
0
-5
-10
-15
-20
-25
-30
-35
-40
-45
:23
:35
:46
:58
:10
12-9-2020 2:57:17 to 2:58:16
temp
pressure

Perspective - XY Chart

General

Component Palette Icon:

INDUCTIVE UNIVERSITY

XY Chart

Watch the Video

Description

The XY Chart displays data trends. It provides a flexible way to display either timeseries or X-Y data by entering data in the **dataSources** property. I t is fully customizable in its appearance, from labels, colors, line widths, legend, scroll bars, text styles, and more.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description
dataSo urces	Objects that are the data source for the chart. When configured to show a date or time on the x-axis, the chart expects that each entry in a order. It is highly advised that you sort the contents of any given data source.
title	Chart title configuration.

Name	Description
text	The chart title text.
appeara nce	Appearance related title options.

Name	Description
color	Color for the title.
font	Font settings.

Name	Description
size	Specifies the size of the font in pixels (px) or points (pt). If you enter just a number, Perspective assumes the value is in pixels.
weight	Sets how thick or thin characters in the text are displayed.

padding	Padding space around the title. Values can be set for the top, bottom, left, and right.

subtitle	Chart subtitle configuration.

Name	Description
text	The chart subtitle text.
appearance	Appearance related subtitle settings.

Name	Description
color	Color for the title.
font	Font settings.

Name	Description
size	Specifies the size of the font in pixels (px) or points (pt). If you enter just a number, Perspective assumes the value is in pixels.
weight	Sets how thick or thin characters in the text are displayed.

padding	Padding space around the subtitle. Values can be set for the top, bottom, left, and right.

legend	Settings for the chart legend.

Name	Description
enabled	Whether the legend is enabled for this chart. Default is true (enabled).
position	Where the legend is located in relation to the chart (left, right, top, bottom, or absolute). Default is bottom.
absolute	Settings used to position the legend when the position property is set to absolute.Values can entered for x and y.
markers	Settings for the markers.

Name	Description
enabled	Enables all series markers used in the legend.
width	The width of each marker, in pixels.
height	The height of each marker, in pixels.
mirrorLookOfSeries	Markers will be drawn to mirror the look of the series elements. Important: Customizing markers beyond wi this value is set to false.
stroke	Settings for the marker line stroke.

Name	Description	Property Type
color	Marker line stroke color.	value: string

	width — Marker line stroke width. — value: numeric
	opacity — Marker line stroke opacity. 0 is fully transparent, 1 is fully opaque. — value: numeric
cornerRadius	Corner radius applied to the rectangle marker. Values can be set for the topLeft, topRight, bottomLeft, and l
icon	Settings for the icon on the legend.

Name	Description
path	Shorthand path to the icon source, in this format: library/iconName. The materials icon library is primary source for icons in Ignition, see https://fonts.google.com/icons?selected=Material+Icon
color	Color of the icon.
verticalCenter	The vertical anchor point for the icon. Options are none, top, middle, bottom
horizontalCenter	The horizontal anchor point for the icon. Options are none, left, middle, right
width	Width of the icon.
height	Height of the icon.

Name	Description	Property Type
labels	Settings for the labels on the legend.	
text	A template string which is applied to all the labels. Default is [bold]{name}[/].	value: string
font	Font settings for the labels.	object

Name	Description	Property Type
weight	The weight of the font.	value: numeric
color	The color of the font.	value: string
size	The font size.	value: numeric

Name	Description
cursor	Settings for the chart cursor.
enabled	Enables all chart cursors that are set to show.
series	Binds the chart cursor to a specified series' data source.
lineX	The chart cursor configuration for the line that intersects the X axis. Options as follows:

Name	Description
enabled	The chart cursor configuration for the line that intersects the X axis.
stroke	Settings for the stroke. Options as follows:

Name	Description

color	Cursor line stroke color. See Color Selector.
width	Cursor line stroke width
opacity	Cursor line opacity. 0 is fully transparent, 1 is fully opaque.
dashArray	SVG dashed array. The pattern of dashes and gaps used to paint the stroke. It's a list of comma lengths (in pixels) and percentages (percentage of the total stroke length) that specify the length alternating dashes and gaps. If an odd number of values is provided, then the list of values is rep yield an even number of values. Thus, "5,3,2" is equivalent to "5,3,2,5,3,2".

lineY	The chart cursor configuration for the line that intersects the Y axis. Options as follows:

Name	Description
enabled	The chart cursor configuration for the line that intersects the Y axis.
stroke	Settings for the stroke. Options as follows:

Name	Description
color	Cursor line stroke color. See Color Selector.
width	Cursor line stroke width
opacity	Cursor line opacity. 0 is fully transparent, 1 is fully opaque.
dashArray	SVG dashed array. The pattern of dashes and gaps used to paint the stroke. It's a list of comma lengths (in pixels) and percentages (percentage of the total stroke length) that specify the length alternating dashes and gaps. If an odd number of values is provided, then the list of values is rep yield an even number of values. Thus, "5,3,2" is equivalent to "5,3,2,5,3,2".

enableTransitions	Whether the transition animations are enabled for this chart. Default is false (disabled).
scrollBars	Configuration for the scroll bars on the chart. Options as follows:

Name	Description	
horizontal	Settings for horizontal scrollbars. Options as follows:	o

Name	Description	Property Type
enabled	Whether the horizontal scrollbar is enabled for this chart. Default is true (enabled).	value: boolean
series	Binds the horizontal scroll bar to a series.	array

vertical	Settings for vertical scrollbars. Options as follows:	o

Name	Description	Property Type
enabled	Whether the vertical scrollbar is enabled for this chart. Default is true (enabled).	value: boolean
series	Binds the vertical scroll bar to a series.	array

background	Configuration for the background of the chart.

Name	Description
render	Sets the render mode for the chart background. Options are none, gradient, or color. Default is none.
gradient	Sets the gradient configuration for the chart background. Options are:

Name	Description	Property Type
direction	Sets the direction of the gradient. Options are linear or radial.	value: string dropdown
rotation	Gradient rotation. Applies only to the linear gradient. Default is 0.	value: numeric
colors	Colors to be used in the gradient.	array

color	Color to be used in background if color property is set under render. Can be chosen from color wheel, chosen from color value.
opacity	Opacity of background of the chart. 0 is fully transparent, 1 is fully opaque.

xAxes	Configuration properties for the X Axes of the chart. Options as follows:

Name	Description
name	A unique name to identify this axis configuration object. This field is required in order to configure the series.
label	Enables or disables a label for the x axis.

Name	Description
enabled	Enables a label drawn alongside this axis.
text	Label text.
color	Label color. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. S

inversed	Indicates if the scale of the axis should be flipped.
visible	Make the label visible, if label is enabled.
tooltip	Tool tip configuration for the axis. Options as follows:

Name	Description	Property
enabled	Enables the tooltip. Default is true.	value: boo
text	Tooltip text.	value: stri
cornerRadius	Radius for the corner of tooltip.	value: nu
pointerLength	Length (in pixels) for the pointer on the tooltip.	value: nu
background	Configuration for the color and opacity of the background of the tooltip. See Color Selector.	color

render	Sets the axis type to render. Options are category, date, or value. Default is date.
category	Category axis configuration. Applied when render is set to category. Groups data items into categories and allots equal s category axis to remove a certain range from its scale. Options as follows:

Name	Description	Property Type
enabled	Enables a break range. Default is false.	value: boolean
startCategory	Start point of the break.	value: string

endCategory	End point of the break.	value: string
size	The break size as a decimal percentage of the removed values.	value: numeric

date

Date axis configuration. Applied when render is set to date. Uses data and time scale. Options as follows:

Name	Description
baseInterval	Adjusts the granularity of the time scale. Otherwise will adjust intelligently by default. Options as follows:
range	You can optionally adjust the date range. Otherwise will auto adjust by default. Options as follows:
break	Tells the date axis to remove a certain range from its scale. Options as follows:

baseInterval

Name	Description
enabled	Enables or disables baseInteval.
timeUnit	Specifies the base time to apply to this time scale. Options are minute, second, hour, day, week, or year.
count	Specifies how many time units each data item was collected.
skipEmptyPeriods	Removes empty time units from display. Using this feature affects performance. Will reset use of axis breaks if true.

range

Name	Description	Property Type
max	Maximum date in this range.	value: string
min	Minimum date in this range.	value: string
useStrict	Strictly enforces start and end values.	value: boolean

break

Name	Description	Property Type
inputFormat	Sets the date format of the axis value from the data source.	value: string dropdown
format	Sets the date format for the axis labels.	value: string dropdown

value

Value axis configuration. Applied when render is set to date. Uses data and time scale. Options as follows:

Name	Description
range	You can optionally adjust the date range. Otherwise will auto adjust by default. Options as follows:
logarithmic	Use logarithmic scale. Useful if data varies greatly within the relevant series. Default is false.
break	Tells the value axis to remove a certain range from its scale. Options as follows:

range

Name	Description	Property Type
max	Maximum date in this range.	value: string
min	Minimum date in this range.	value: string
useStrict	Strictly enforces start and end values.	value: boolean

break

Name	Description	Property Type
enabled	Enables a break range. Default is false.	value: boolean
startValue	Start point of the break.	value: numeric

	endValue: End point of the break. (value: numeric) size: The break size as a decimal percentage of the removed values. (value: numeric)
format	A number format string to be applied against numbers if in number rendering mode. Options are number, inte precision, percent, scientific, currency, currency (rounded), or abbreviation.
appearance	Appearance options for the x axis. Options as follows:

Name	Description
opposite	Renders the axis on the opposite side. Default is false
inside	Renders the axis labels on the inside of the axis. Default is false.
labels	Axis label configuration:

Name	Description	Property Type
color	Axis label color.	color
opacity	Axis label opacity.	value: numeric
rotation	Rotation of the label. Default is 0.	value: numeric
wrap	Whether or not to wrap the label text.	value: boolean

grid — Configures the color, opacity and SVG dashed array of the grid lines.

Name	Description
color	Axis grid color.
opacity	Axis grid opacity.
dashArray	SVG dashed array. The pattern of dashes and gaps used to paint the stroke. It's a list of comma lengths (in pixels) and percentages (percentage of the total stroke length) that specify the length alternating dashes and gaps. If an odd number of values is provided, then the list of values is rep yield an even number of values. Thus, "5,3,2" is equivalent to "5,3,2,5,3,2".
minDistance	The minimum distance between grid lines.
position	Defines the grid's relative position within the chart. A range from 0 to 1, with 0 meaning start and meaning end.

font — Configures the axis font size and weight.

Name	Description	Property Type
size	Axis font size.	value: numeric
weight	Axis font weight.	value: numeric

yAxes — Configuration properties for the Y Axes of the chart.

Name	Description
name	A unique name to identify this axis configuration object. This field is required in order to configure the series.

label	Enables or disables a label for the y axis. Options as follows:

Name	Description
enabled	Enables a label drawn alongside this axis.
text	Label text.
color	Label color. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. S

inversed	Indicates if the scale of the axis should be flipped.
visible	Make the label visible, if label is enabled.
tooltip	Tool tip configuration for the axis. Options as follows:

Name	Description	Property Type
enabled	Enables the tooltip. Default is true.	value: boolean
text	Tooltip text.	value: string
cornerRadius	Radius for the corner of tooltip.	value: numeric
pointerLength	Length (in pixels) for the pointer on the tooltip.	value: numeric
background	Configuration for the color and opacity of the background of the tooltip.	color

render	Sets the axis type to render. Options are category, date, or value. Default is date.
category	Applied when **render** is set to **category**. Category axis configuration. Groups data items into categories and allots equal

Name	Description	Property Type
enabled	Enables a break range. Default is false.	value: boolean
startCategory	Start point of the break.	value: string
endCategory	End point of the break.	value: string
size	The break size as a decimal percentage of the removed values.	value: numeric

date	Applied when **render** is set to **date**. Date axis configuration. Uses data and time scale. Options as follows:

Name	Description
baseInterval	Adjust the granularity of the time scale. Otherwise will adjust intelligently by default.

Name	Description
enabled	Enables or disables baseInteval.
timeUnit	Specifies the base time to apply to this time scale. Options are minute, second, hour, day, week, or year.
count	Specifies how many time units each data item was collected.
skipEmptyPeriods	Removes empty time units from display. Using this feature affects performance. Will reset the use of axis breaks if true.

range	You can optionally adjust the date range. Otherwise will auto adjust by default. Options as follows:

Name	Description	Property Type
max	Maximum date in this range.	value: string
min	Minimum date in this range.	value: string

	useStrict: Strictly enforces start and end values. (value: boolean)
break	Tells the date axis to remove a certain range from its scale. Options as follows:

Name	Description	Property Type
enabled	Enables a break range. Default is false.	value: boolean
startDate	Start date of the break.	value: string
endDate	End date of the break.	value: string
size	The break size as a decimal percentage of the removed values.	value: numeric

inputFormat	Sets the date format of the axis value from the data source.
format	Sets the date format for the axis labels. Options are date, time, or date and time.
value	Applied when **render** is set to **value**. Value axis configuration. Uses data and time scale. Options as follows:

Name	Description
range	You can optionally adjust the date range. Otherwise will auto adjust by default. Options as follows:

Name	Description	Property Type
max	Maximum date in this range.	value: string
min	Minimum date in this range.	value: string
useStrict	Strictly enforces start and end values.	value: boolean

logarithmic	Use logarithmic scale. Useful if data varies greatly within the relevant series.
break	Tells the value axis to remove a certain range from its scale. Options as follows:

Name	Description	Property Type
enabled	Enables a break range. Default is false.	value: boolean
startValue	Start point of the break.	value: numeric
endValue	End point of the break.	value: numeric
size	The break size as a decimal percentage of the removed values.	value: numeric

format	A number format string to be applied against numbers if in number rendering mode. Options are number, inte precision, percent, scientific, currency, currency (rounded), or abbreviation.
appearance	Appearance options for the y axis. Options as follows:

Name	Description
opposite	Renders the axis on the opposite side. Default is false
inside	Renders the axis labels on the inside of the axis. Default is false.
labels	Axis label configuration:

Name	Description	Property Type
color	Axis label color.	color
opacity	Axis label opacity.	value: numeric
rotation	Rotation of the label. Default is 0.	value: numeric

wrap	Whether or not to wrap the label text. value: boolean
grid	Configures the color, opacity and SVG dashed array of the grid lines.

Name	Description
color	Axis grid color.
opacity	Axis grid opacity.
dashArray	SVG dashed array. The pattern of dashes and gaps used to paint the grid. It's a list of comma se lengths (in pixels) and percentages (percentage of the total grid length) that specify the lengths c alternating dashes and gaps. If an odd number of values is provided, then the list of values is rep yield an even number of values. Thus, "5,3,2" is equivalent to "5,3,2,5,3,2".
minDistance	The minimum distance between grid lines.
position	Defines the grid's relative position within the chart. A range from 0 to 1, with 0 meaning start and meaning end.

font — Configures the font size and weight.

Name	Description	Property Type
size	Axis font size.	value: numeric
weight	Axis font weight.	value: numeric

series — An array of series configurations to apply to this chart.

Name	Description
name	A unique name to identify this application of this series.
label	Series label to use with legend.

Name	Description	Property Type
text	Label text.	value: string

visible	Enables series visibility. Default is true.
hiddenInLegend	Hides the series in the legend. Default is false.
defaultState	Series default state configuration. Default is true (visible).
data	Data settings for the series. Options as follows:

Name	Description
source	Name of the data source to bind to this series.
x	The y value key from the specified data source to be used in this series. Value is a string value if the data so array of objects. Value is an index if the data source is an array of arrays.
y	The y value key from the specified data source to be used in this series. Value is a string value if the data so array of objects. Value is an index if the data source is an array of arrays.

xAxis	Name of the x axis configuration object to be used with this series.

yAxis	Name of the y axis configuration object to be used with this series.
zIndex	Sets the series stack order relative to other series.
tooltip	Tool tip configuration for the series. Options as follows:

Name	Description	Property Type
enabled	Enables the tooltip. Default is true.	value: boolean
text	A format string to apply to the tooltip text.	value: string
cornerRadius	Radius for the corner of tooltip.	value: numeric
pointerLength	Length (in pixels) for the pointer on the tooltip.	value: numeric
background	Configuration for the color and opacity of the background of the tooltip.	object

Name	Description	Property Type
color	Background color. See Color Selector.	color
opacity	Background opacity.	value: numeric

render	The series render mode. Options are **candlestick**, **column**, **line**, **stepLine**. Default is line.
candlestick	When **render** is set to **candlestick**. These are the candlestick settings. Options as follows:

Name	Description
open	Open data settings

Name	Description	Property Type
x	Name of the field that holds the open data for the horizontal axis.	value: string
y	Name of the field that holds the open data for the vertical axis.	value: string

high	High data settings.

Name	Description	Property Type
x	Name of the field that holds the high data for the horizontal axis.	value: string
y	Name of the field that holds the high data for the vertical axis.	value: string

low	Low data settings.

Name	Description	Property Type
x	Name of the field that holds the low data for the horizontal axis.	value: string
y	Name of the field that holds the low data for the vertical axis.	value: string

appearance	Appearance settings.

Name	Description
fill	

Name	Description	Property Type
color	Cursor line stroke color.	color
opacity	Opacity of the stroke.	value: numeric

stroke	Series stroke configuration.

Name	Description	Property Type
color	Cursor line stroke color.	color
opacity	Opacity of the stroke.	value: numeric
width	Width of the stroke, in pixels.	value: numeric

stacked	Stacks this column series.
deriveFieldsFromData	

fill	

Name	Description	Property Type
color	Cursor line stroke color.	color
opacity	Opacity of the stroke.	value: numeric

stroke	

Name	Description	Property Type
color	Cursor line stroke color.	color
opacity	Opacity of the stroke.	value: numeric
width	Width of the stroke, in pixels.	value: numeric

heatRules	These heat rules apply to the fill of the columns.

Name	Description	Property Type
enabled	Whether or not heat rules are enabled. Default is false.	value: boolea
max	Color for max.	value: string
min	Color for min.	value: string
dataField		value: string

column	When **render** is set to **column**. These are the column settings. Options as follows:

Name	Description
open	Open settings.

Name	Description	Property Type
x	Name of the field that holds the open data for the horizontal axis.	value: string
y	Name of the field that holds the open data for the vertical axis.	value: string

appearance	Appearance settings.

Name	Description
fill	Fill settings.

Name	Description	Property Type
color	Cursor line stroke color.	color
opacity	Opacity of the stroke.	value: numeric

stroke	Stroke settings.

Name	Description	Property Type
color	Cursor line stroke color.	color
opacity	Opacity of the stroke.	value: numeric
width	Width of the stroke, in pixels.	value: numeric

width	Percent
height	Percent
stacked	Stacks this column series.
deriveFieldsFromData	

Name	Description	Property Type
fill	Fill settings.	object

Name	Description	Property Type
color	Cursor line stroke color.	color
opacity	Opacity of the stroke.	value: numeric

Name	Description	Property Type
stroke	Stroke settings.	object

Name	Description	Property Type
color	Cursor line stroke color.	color
opacity	Opacity of the stroke.	value: numeric
width	Width of the stroke, in pixels.	value: numeric

heatRules

These heat rules apply to the fill of the columns.

Name	Description	Property Type
enabled	Whether or not heat rules are enabled. Default is false.	value: boole
max	Color for max.	value: string
min	Color for min.	value: string
dataField		value: string

line

When **render** is set to **line**. These are the line settings. Options as follows:

Name	Description
open	Configures the open data.

Name	Description	Property Type
x	Name of the field that holds the open data for the horizontal axis	value:string
y	Name of the field that holds the open data for the vertical axis	value:string

appearance	Configures the appearance of the line series. Options as follows:

Name	Description
connect	Connects the lines over empty data points. Default is true.
tensionX	Horizontal tension setting of the line. Range is 0 to 1. A 1 value indicates high tension, so th maximally attracted to the points it connects (i.e. straight line). A 0 value means the opposite is 1.
tensionY	Vertical tension setting of the line. Range is 0 to 1. A 1 value indicates high tension, so the li maximally attracted to the points it connects (i.e. straight line). A 0 value means the opposite is 1.
minDistance	The minimum distance (in pixels) between two points. Default is 0.5.
stroke	Series stroke configuration. Options as follows:

Name	Description	P
width	Width of the stroke, in pixels. Default is 3.	va nu
opacity	Opacity of the stroke.	va nu
color	Cursor line stroke color.	co
dashArray	SVG dash array. The pattern of dashes and gaps used to paint the stroke. It's a list of comma separated lengths (in pixels) and percentages (percentage of the total stroke length) that specify the lengths of alternating dashes and gaps. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. Thus, "5,3,2" is equivalent to "5,3,2,5,3,2".	ar

fill	Series color configuration.

Name	Description	Property
color	The color to draw this series.	value: colo
opacity	Opacity as a percentage from 0 to 1. 0 is transparent, 1 is opaque.	value: floa

bullets	Series bullet configuration. Type of bullet to render (circle or rectangle) for each pen.

Name	Description	P
enabled	Enables bullets.	va be
render	Type of bullet to render. Options are circle, rectangle, or icon.	va st du
width	Bullet width.	va nu
height	Bullet height.	va nu
label	Label properties.	ob

Name	Description	Property Type
text	Label text.	value: string

position.dx	Label x position.	value: numeric
position.dy	Label y position.	value: numeric

tooltip	Tooltip configuration.	ol

Name	Description	Property Type
enabled	Enables the tooltip.	value: boolean
text	Tooltip text. Can be a format string.	value: string
cornerRadius	The corner radius.	value: numeric
pointerLength	The pointer length.	value: numeric
background	Background color and opacity for the tool tip.	object

fill	Fill settings.	ol

Name	Description	Property Type
color	The fill color for the bullets in this series. See Color Selector.	color
opacity	The bullet opacity.	value: numeric

stroke	Stroke settings.	ol

Name	Description	Property Type
width	Width of the stroke, in pixels. Default is 3.	value: numeric
opacity	Opacity of the stroke.	value: numeric
color	Cursor line stroke color.	color

rotation	Rotation of the bullet.	va nu
deriveFieldsFromData	Settings for derived fields.	ol

Name	Description	Property Type
fill	Fill settings.	object

Name	Description	Property Type
color	The fill color for the derived fields in this series. See Color Selector.	color
opacity	The derived field opacity.	value: numeric

Name	Description	Property Type
stroke	Stroke settings.	object

Name	Description	Property Type
color	Cursor line stroke color.	color

	opacity	Opacity of the stroke.	value: numeric
	width	Width of the stroke, in pixels.	value: numeric
rotation	Derived field rotation (0-360).	value: numeric

heatRules	These heat rules apply to the radius of a circular bullet.	ol

Name	Description	Property Type
enabled	Whether or not heat rules are enabled. Default is false.	value: boolean
max	Color for max.	value: string
min	Color for min.	value: string
dataField	The data field.	value: string

stepLine	When **render** is set to **stepLine**. These are the stepLine settings. Options as follows:

Name	Description
open	Configures the open data.

Name	Description	Property Type
x	Name of the field that holds the open data for the horizontal axis	value:string
y	Name of the field that holds the open data for the vertical axis	value:string

Name	Description
appearance	Configures the following options:

Name	Description
connect	Connects the lines over empty data points. Default is true.
tensionX	Horizontal tension setting of the line. Used to create smooth or sharp lines Range is 0 to 1. A indicates high tension, so the line is maximally attracted to the points it connects (i.e. straigh value means the opposite. Default is 1.
tensionY	Vertical tension setting of the line. Used to create smooth or sharp lines Range is 0 to 1. A 1 indicates high tension, so the line is maximally attracted to the points it connects (i.e. straigh value means the opposite. Default is 1.
minDistance	The minimum distance (in pixels) between two points. Default is 0.5.
stroke	Series stroke configuration. Options are as follows:

Name	Description	
width	Width of the stroke, in pixels. Default is 3.	va nu
opacity	Opacity of the stroke. 0 is fully transparent, 1 is fully opaque. Default is 1.	va nu
color	Color of border around each pie section. See Color Selector.	co
dashArray	SVG dashed array. The pattern of dashes and gaps used to paint the stroke. It's a list of comma separated lengths (in pixels) and percentages	ar

	(percentage of the total stroke length) that specify the lengths of alternating dashes and gaps. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. Thus, "5,3,2" is equivalent to "5,3,2,5,3,2".

fill	Fill settings.

Name	Description	Property Typ
color	The fill color for the columns in this series. See Color Selector.	color
opacity	The column opacity.	value: numeric

bullets	Series bullet configuration.

Name	Description	
enabled	Enables bullets.	va b
render	Type of bullet to render. Options are circle or label.	va st dr
width	Bullet width.	va nu
height	Bullet height.	va nu
label	Label settings.	ol

Name	Description	Property Type
text	Label text.	value: string
position	Label position.	object

tooltip	Tooltip settings.

Name	Description	Property Type
enabled	Enables the tooltip.	value: boolean
text	Tooltip text. Can be a format string.	value: string
cornerRadius	The corner radius.	value: numeric
pointerLength	The pointer length.	value: numeric
background	Background color and opacity for the tool tip.	object

fill	Fill settings.

Name	Description	Property Type
color	The fill color for the columns in this series. See Color Selector.	color
opacity	The column opacity.	value: numeric

stroke	Stroke settings.

Name	Description	Property Type

color	Cursor line stroke color. color
opacity	Opacity of the stroke. value: numeric
width	Width of the stroke, in pixels. value: numeric

rotation	Bullet rotation. Value can be 0 to 360.	va nu
deriveFieldsFromData	Settings for derived fields.	ol

Name	Description	Property Type
fill	Fill settings.	object
stroke	Stroke settings.	object
rotation	Bullet rotation (0-360).	value: numeric

Name	Description	Property Type
color	The fill color for the columns in this series. See Color Selector.	color
opacity	The column opacity.	value: numeric

Name	Description	Property Type
color	Cursor line stroke color.	color
opacity	Opacity of the stroke.	value: numeric
width	Width of the stroke, in pixels.	value: numeric

heatRules	These heat rules apply to the radius of a circular bullet.	ol

Name	Description	Property Type
enabled	Whether or not heat rules are enabled. Default is false.	value: boolean
max	Color for max.	value: string
min	Color for min.	value: string
dataField		value: string

style	Sets a style for this chart. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneo

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1

This example demonstrates how to configure a Heatmap, where the value of each cross axis is represented as a color along a gradient. This style of chart is largely accomplished by setting the series "render" property to "column", and enabling "heatRules".

The data used in this example is shown in the code block below: The x-axis is set to the "year" key, while the y-axis is set to the "fruit" key. The "dataField" property under "heatRules" is set to the "count" key.

year - the x-axis

fruit - the y-axis

count - determines the color used in the block.

```
[
  {
    "year": 2015,
    "fruit": "apple",
```

```
    "count": 20
  },
  {
    "year": 2015,
    "fruit": "orange",
    "count": 400
  },
  {
    "year": 2015,
    "fruit": "banana",
    "count": 150
  },
  {
    "year": 2016,
    "fruit": "apple",
    "count": 200
  },
  {
    "year": 2016,
    "fruit": "orange",
    "count": 300
  },
  {
    "year": 2016,
    "fruit": "banana",
    "count": 400
  },
  {
    "year": 2017,
    "fruit": "apple",
    "count": 200
  },
  {
    "year": 2017,
    "fruit": "orange",
    "count": 400
  },
  {
    "year": 2017,
    "fruit": "banana",
    "count": 30
  }
]
```

Notable Property Configurations

The example requires both an X axis and a Y axis. In addition to a series

PROPS Path	Property Description	Value
xAxes.0.render	Makes the chart group values in the X Axis, and provides equal padding between each category. Partially responsible for rendering boxes on the chart.	category
yAxes.0.render	Makes the chart group values in the Y Axis, and provides equal padding between each category. Partially responsible for rendering boxes on the chart.	category
series.0.render	When combined with "category" renders for the X and Y axes, allows the categories boxes to be rendered on the chart.	column
series.0.column. appearance.heatRules. enabled	Makes the chart change the color on each category based on dataField key ("count", in our example), applying the min and max colors.	true
series.0.column. appearance.heatRules. max	The property represents which color to use for higher values.	#26E3B1
series.0.column. appearance.heatRules. min	The property represents which color to use for lower values.	#E5DBDB (pick a color you want to represent low colors)
series.0.column.	Determines which key in the underlying data should be used to determine the	count

appearance.heatRules.dataField	color of th	

Example Configuration

The JSON string below can be used to replicate the heatmap example. Simply copy the contents of the code block below (**double click** on any part of the JSON to select all of it) , and paste it into a container in your designer.

```
[
  {
    "type": "ia.chart.xy",
    "version": 0,
    "props": {
      "legend": {
        "enabled": false
      },
      "xAxes": [
        {
          "name": "year",
          "label": {
            "enabled": true,
            "text": "year",
            "color": ""
          },
          "visible": true,
          "tooltip": {
            "enabled": true,
            "text": "",
            "cornerRadius": 3,
            "pointerLength": 4,
            "background": {
              "color": "",
              "opacity": 1
            }
          },
          "inversed": false,
          "render": "category",
          "category": {
            "break": {
              "enabled": false,
              "startCategory": "",
              "endCategory": "",
              "size": 0.05
            }
          },
          "date": {
            "baseInterval": {
              "enabled": false,
              "timeUnit": "hour",
              "count": 1,
              "skipEmptyPeriods": false
            },
            "range": {
              "max": "",
              "min": "",
              "useStrict": false
            },
            "break": {
              "enabled": false,
              "startDate": "",
              "endDate": "",
              "size": 0.05
            },
            "inputFormat": "yyyy-MM-dd kk:mm:ss",
            "format": "M/d"
          },
          "value": {
            "range": {
              "max": "",
              "min": "",
              "useStrict": false
```

```
            },
            "logarithmic": false,
            "break": {
              "enabled": false,
              "startValue": 0,
              "endValue": 100,
              "size": 0.05
            },
            "format": "#,###.##"
          },
          "appearance": {
            "opposite": true,
            "inside": false,
            "labels": {
              "color": "",
              "opacity": 1
            },
            "grid": {
              "color": "",
              "opacity": 1,
              "dashArray": "",
              "minDistance": 60,
              "position": 0.5
            },
            "font": {
              "size": "",
              "weight": 500
            }
          }
        }
      ],
      "yAxes": [
        {
          "name": "fruit",
          "label": {
            "enabled": true,
            "text": "fruit",
            "color": ""
          },
          "visible": true,
          "tooltip": {
            "enabled": true,
            "text": "",
            "cornerRadius": 3,
            "pointerLength": 4,
            "background": {
              "color": "",
              "opacity": 1
            }
          },
          "inversed": false,
          "render": "category",
          "category": {
            "break": {
              "enabled": false,
              "startCategory": "",
              "endCategory": "",
              "size": 0.05
            }
          },
          "date": {
            "baseInterval": {
              "enabled": false,
              "timeUnit": "hour",
              "count": 1,
              "skipEmptyPeriods": false
            },
            "range": {
              "max": "",
              "min": "",
              "useStrict": false
```

```
            },
            "break": {
              "enabled": false,
              "startDate": "",
              "endDate": "",
              "size": 0.05
            },
            "inputFormat": "yyyy-MM-dd kk:mm:ss",
            "format": "M/d/yyyy HH:mm:ss"
          },
          "value": {
            "range": {
              "max": "",
              "min": "",
              "useStrict": false
            },
            "logarithmic": false,
            "break": {
              "enabled": false,
              "startValue": 0,
              "endValue": 100,
              "size": 0.05
            },
            "format": "#,###.##"
          },
          "appearance": {
            "opposite": false,
            "inside": false,
            "labels": {
              "color": "",
              "opacity": 1
            },
            "grid": {
              "color": "",
              "opacity": 1,
              "dashArray": "",
              "minDistance": null,
              "position": 0.5
            },
            "font": {
              "size": "",
              "weight": 500
            }
          }
        }
      ],
      "series": [
        {
          "name": "count",
          "label": {
            "text": "Process Temp"
          },
          "visible": true,
          "hiddenInLegend": false,
          "defaultState": {
            "visible": true
          },
          "data": {
            "source": "data",
            "x": "year",
            "y": "fruit"
          },
          "xAxis": "year",
          "yAxis": "fruit",
          "zIndex": 0,
          "tooltip": {
            "enabled": true,
            "text": "{name}: [bold]{valueY}[/]",
            "cornerRadius": 3,
            "pointerLength": 4,
            "background": {
```

```
            "color": "",
            "opacity": 1
          }
        },
        "render": "column",
        "candlestick": {
          "open": {
            "x": "",
            "y": ""
          },
          "high": {
            "x": "",
            "y": ""
          },
          "low": {
            "x": "",
            "y": ""
          },
          "appearance": {
            "fill": {
              "color": "",
              "opacity": 1
            },
            "stroke": {
              "color": "",
              "opacity": 1,
              "width": 1
            },
            "stacked": false,
            "deriveFieldsFromData": {
              "fill": {
                "color": "",
                "opacity": ""
              },
              "stroke": {
                "color": "",
                "opacity": "",
                "width": ""
              }
            },
            "heatRules": {
              "enabled": false,
              "max": "",
              "min": "",
              "dataField": ""
            }
          }
        },
        "column": {
          "open": {
            "x": "",
            "y": ""
          },
          "appearance": {
            "fill": {
              "color": "",
              "opacity": 1
            },
            "stroke": {
              "color": "",
              "opacity": 1,
              "width": 1
            },
            "stacked": false,
            "width": null,
            "height": null,
            "deriveFieldsFromData": {
              "fill": {
                "color": "",
                "opacity": ""
              },
```

```
            "stroke": {
              "color": "",
              "opacity": "",
              "width": ""
            }
          },
          "heatRules": {
            "enabled": true,
            "max": "#26E3B1",
            "min": "#E5DBDB",
            "dataField": "count"
          }
        }
      },
      "line": {
        "open": {
          "x": "",
          "y": ""
        },
        "appearance": {
          "connect": true,
          "tensionX": 1,
          "tensionY": 1,
          "minDistance": 0.5,
          "stroke": {
            "width": 3,
            "opacity": 1,
            "color": "",
            "dashArray": ""
          },
          "fill": {
            "opacity": 0,
            "color": ""
          },
          "bullets": [
            {
              "enabled": false,
              "render": "circle",
              "width": 10,
              "height": 10,
              "label": {
                "text": "{value}",
                "position": {
                  "dx": 0,
                  "dy": 0
                }
              },
              "fill": {
                "color": "",
                "opacity": 1
              },
              "stroke": {
                "color": "",
                "opacity": 1,
                "width": 1
              },
              "rotation": 0,
              "tooltip": {
                "enabled": true,
                "text": "{name}: [bold]{valueY}[/]",
                "cornerRadius": 3,
                "pointerLength": 4,
                "background": {
                  "color": "",
                  "opacity": 1
                }
              },
              "deriveFieldsFromData": {
                "fill": {
                  "color": "",
                  "opacity": ""
```

```
                },
                "stroke": {
                  "color": "",
                  "opacity": "",
                  "width": ""
                },
                "rotation": ""
              },
              "heatRules": {
                "enabled": false,
                "max": 100,
                "min": 2,
                "dataField": ""
              }
            }
          ]
        }
      },
      "stepLine": {
        "open": {
          "x": "",
          "y": ""
        },
        "appearance": {
          "connect": true,
          "tensionX": 1,
          "tensionY": 1,
          "minDistance": 0.5,
          "stroke": {
            "width": 3,
            "opacity": 1,
            "color": "",
            "dashArray": ""
          },
          "fill": {
            "opacity": 0,
            "color": ""
          },
          "bullets": [
            {
              "enabled": true,
              "render": "circle",
              "width": 10,
              "height": 10,
              "label": {
                "text": "{value}",
                "position": {
                  "dx": 0,
                  "dy": 0
                }
              },
              "fill": {
                "color": "",
                "opacity": 1
              },
              "stroke": {
                "color": "",
                "opacity": 1,
                "width": 1
              },
              "rotation": 0,
              "tooltip": {
                "enabled": true,
                "text": "{name}: [bold]{valueY}[/]",
                "cornerRadius": 3,
                "pointerLength": 4,
                "background": {
                  "color": "",
                  "opacity": 1
                }
              },
```

```
                "deriveFieldsFromData": {
                  "fill": {
                    "color": "",
                    "opacity": ""
                  },
                  "stroke": {
                    "color": "",
                    "opacity": "",
                    "width": ""
                  },
                  "rotation": ""
                },
                "heatRules": {
                  "enabled": false,
                  "max": 100,
                  "min": 2,
                  "dataField": ""
                }
              }
            ]
          }
        }
      }
    ],
    "style": {
      "marginRight": "100px",
      "paddingRight": "100px"
    },
    "dataSources": {
      "data": [
        {
          "year": 2015,
          "fruit": "apple",
          "count": 20
        },
        {
          "year": 2015,
          "fruit": "orange",
          "count": 400
        },
        {
          "year": 2015,
          "fruit": "banana",
          "count": 150
        },
        {
          "year": 2016,
          "fruit": "apple",
          "count": 200
        },
        {
          "year": 2016,
          "fruit": "orange",
          "count": 300
        },
        {
          "year": 2016,
          "fruit": "banana",
          "count": 400
        },
        {
          "year": 2017,
          "fruit": "apple",
          "count": 200
        },
        {
          "year": 2017,
          "fruit": "orange",
          "count": 400
        },
        {
```

```
            "year": 2017,
            "fruit": "banana",
            "count": 30
          }
        ]
      }
    },
    "meta": {
      "name": "XYChart"
    },
    "position": {
      "basis": "536px"
    },
    "custom": {}
  }
]
```

Example 2

XY Chart Gantt Style Example

In addition to basic data plotting, an XY Chart can be used to plot horizontal bar across different lanes, in this fashion:

This can useful when it comes to machine downtime scheduling, shift scheduling, and maintenance scheduling. To achieve this functionality with the XY Chart, follow the example below:

1. From the Perspective section of the Project Browser on your Designer, right click on the Views folder and select **New View...** to create a new view.
2. This will bring up the New View window. Give your view a name and select the Coordinate Root Container Type. The Page URL setting will remain unchecked for this example.

3. From the Perspective Component Palette, drag and drop a XY Chart onto your newly created view.
4. Click to select your newly added XY Chart and from the property editor, click on the + icon next to the **dataSources** property and select Array to create a new data source array and name it schedule_data.

5. Right click on your newly created schedule_data dataSource and paste the following:

```
[
  {
    "name": "John",
    "fromDate": "2018-01-01 08:00",
    "toDate": "2018-01-01 10:00",
    "color": "colorSet.getIndex(0).brighten(0)"
  },
  {
    "name": "John",
    "fromDate": "2018-01-01 12:00",
    "toDate": "2018-01-01 15:00",
    "color": "colorSet.getIndex(0).brighten(0.4)"
  },
  {
    "name": "John",
    "fromDate": "2018-01-01 15:30",
    "toDate": "2018-01-01 21:30",
    "color": "colorSet.getIndex(0).brighten(0.8)"
  },
  {
    "name": "Jane",
    "fromDate": "2018-01-01 09:00",
    "toDate": "2018-01-01 12:00",
    "color": "colorSet.getIndex(2).brighten(0)"
  },
  {
    "name": "Jane",
    "fromDate": "2018-01-01 13:00",
    "toDate": "2018-01-01 17:00",
    "color": "colorSet.getIndex(2).brighten(0.4)"
  },
  {
    "name": "Peter",
    "fromDate": "2018-01-01 11:00",
```

```
    "toDate": "2018-01-01 16:00",
    "color": "colorSet.getIndex(4).brighten(0)"
  },
  {
    "name": "Peter",
    "fromDate": "2018-01-01 16:00",
    "toDate": "2018-01-01 19:00",
    "color": "colorSet.getIndex(4).brighten(0.4)"
  }
]
```

Note: Any data that you wish to plot on the XY Chart to build a Gantt Chart needs to come in the format specified by the **schedule_data** property above.

6. Set XY Chart's variables **cursor.series** and **scrollbars.horizontal.series** to "schedule data".
7. Set the X axes properties as follows:
    a. Configure the X axes date format by setting the property **xAxes[0].date.format** to be a date format that works for you. For this example, we chose **date time** from the dropdown, which sets it to "M/d/yyyy HH:mm:ss".
    b. Set the properties **xAxes[0].appearance.grid.minDistance** to NULL.

8. Set the Y axes properties as follows:
    a. Delete the **yAxes[1]** property since it will not be used.
    b. Set the property **yAxes[0].name** to "Operator".
    c. Disable the Y axes label by setting the **yAxes[0].label.enabled** property to false.
    d. Set the **yAxes[0].label.text** property to "Schedule Data".
    e. Set the property **yAxes[0].render** to "category".
    f. Set your **yAxes[0].appearance.grid.position** to "0".

9. Set the series properties as follows:
    a. Delete the **series[1]** property since it will not be used.
    b. Set your **series[0].name** to "schedule data". This links your Horizontal Scroll bar from step 6 to your chart's series.
    c. Set your **series[0].data.source** to "schedule_data". This links your chart's series to your data source from step 4.
    d. Configure your **series[0].data.x** to be "toDate" and your **series[0].data.y** to be "name".
    e. Configure your **series[0].xAxis** to be "time" and your **series[0].yAxis** to be "Operator".
    f. Set your **series[0].render** property to "column".
    g. Your **series[0].column.open.x** property must be set to "fromDate".
    h. Finally, to help with data visualization, set your **series[0].tooltip.text** to "{name}: [bold]{fromDate} - {toDate}[/]".

10. Save your project.
11. Put the Designer into Preview mode to see the chart in action.

Following these steps should result in the chart below:

Perspective - Container Palette

Container Components

Container components provide a way of laying out and organizing components within a View. The different container types support different layout strategies. Containers are essential to creating responsive applications, and they allow your applications to gracefully display information across a wide variety of screen sizes and orientations.

The following is a complete list of Container components, and a link pointing to a page containing the component's description, properties, and an example of how to configure it.

In This Section ...

Perspective - Breakpoint Container

General

Component Palette Icon:

Breakpoint Container

Watch the Video

Description

The Breakpoint Container is a simple container whose purpose is to swap subcontainers based on a 'breakpoint' in pixels. This allows for a very simple responsive design that removes one container and replaces it with another when the breakpoint (typically viewport width) is reached. With the Breakpoint container, you can define completely different content to render for each breakpoint. This is in contrast to a container such as the column container, where the content for each breakpoint is identical but the layout of it changes according to screen size. See also Breakpoint Containers

Breakpoint Containers are ideal to use in cases where you want completely different components available between a mobile user or a desktop user, since the components in each breakpoint are completely separate instances. Thus, if you're attempting to make a responsive view, and a Flex Container isn't quite giving you the result you're looking for, a Breakpoint Container may be a better fit.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property 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	Use Style to customize the visual style of the component. The Style menu contains all the tools for modifying text, background, margins, borders, and more. You can also specify a style class.	object

Scripting Functions

- Description

  Returns an ArrayList, which contains references to all components inside of the container.

- Parameters

  none

- Return

  Array List - An ArrayList of components in the container. The resulting ArrayList can be iterated over via a for-loop.

Perspective - Column Container

General

Component Palette Icon:

INDUCTIVE UNIVERSITY

Column Container

Watch the Video

Description

The Column Container is a 12 column grid layout, where components can be organized into columns which alter their layout depending on screen viewport size. A Column container provides a way to create a single set of components that can be arranged up to three ways depending on the width of the session. See also Layouts for Column Containers.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
breakpoints	Width breakpoint declarations for child layouts. When the container is sized below minWidth, child position rules will fall back to the next set breakpoint rules. **Name** / **Description** / **Property Type** name / Name of the breakpoint. / value: string minWidth / Minimum width for this breakpoint. / value: numeric	array
gutters	Settings for the gutters, which are the amount of space, in pixels, to place between child components. **Name** / **Description** / **Property Type** vertical / Vertical gutter setting between child components. / value: numeric horizontal / Horizontal gutter setting between child components. / value: numeric	object
style	Sets a style for this component. The Style menu contains all the tools for modifying text, background, margins, borders, and more. You can also specify a style class.	object

Scripting Functions

- Description

  Returns an ArrayList, which contains references to all components inside of the container.

- Parameters

  none

- Return

  Array List - An ArrayList of components in the container. The resulting ArrayList can be iterated over via a for-loop.

Example

For an example of using Column containers, see Column Containers.

Perspective - Coordinate Container

General

Component Palette Icon:

INDUCTIVE UNIVERSITY

Coordinate Container

Watch the Video

Description

The Coordinate Container makes a component's size and location relative to its parent's size and location. Components can be fixed size, or optionally grow/shrink proportionally when the view is stretched. See also Coordinate Containers.

Coordinate Containers are ideal to use in cases where you need components to overlap each other, such as adding a component on top of another (z-axis) to act as an overlay. They're also useful in cases where you do **not** want components within to resize - for example, building a diagram where each element is a separate component.

Components place on coordinate containers can be rotated. The Rotate property has been moved to the Position Properties section of the Perspective Property Editor. For more information, see Working with Perspective Components.

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

The Coordinate Container component has two pre-configured variants:

- Fixed - Child layouts will be in fixed coordinate space.
- Percent - Child layout will be stretched to different size containers.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Props

Name	Description	Property Type
mode	Whether child layouts should always be in fixed coordinate space, or stretched relative to different container sizes: fixed or percent.	value: string dropdown
aspectRatio	Only applied in percent mode. Optional dimensions, 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 component. 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

Scripting Functions

- Description

  Returns an ArrayList, which contains references to all components inside of the container.

- Parameters

  none

- Return

  Array List - An ArrayList of components in the container. The resulting ArrayList can be iterated over via a for-loop.

Example

In this example, we have a Coordinate container with a Cylindrical Tank component and a Slider component. We've set the container property to percent so that the components will grow and shrink with the container size.

Container properties:

Property	Value
props.mode	percent

Cylindrical Tank properties:

Property	Value
position.x	0.05
position.y	0
position.width	0.9
position.height	0.9

Slider properties:

Property	Value
position.x	0.05
position.y	0.9
position.width	0.9
position.height	0.09

For more information and examples for Coordinate Containers, refer to Coordinate Containers.

Perspective - Flex Container

General

Component Palette Icon:

INDUCTIVE UNIVERSITY

Flex Container

Watch the Video

Description

The Flex Container provides an efficient way to lay out, align, and distribute space among components in the container particularly when their size is unknown or dynamic. The Flex Container can alter a component's width and height to best fill the available space to accommodate all types of devices and screen sizes. It expands components to fill available free space, or shrinks them to prevent overflow.
For more information, see Flex Containers.

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

The Flex Container component has two pre-configured variants:

- Row - Child layout will be in rows.
- Column - Child layout will be in columns.

Properties

> Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
> This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Props

Name	Description	Property Type
direction	Direction of the child layout. Options are row, row-reverse, column, column-reverse.	value: string
wrap	Whether the container should allow children to wrap to the next line if space has run out. Options are nowrap, wrap, wrap-reverse.	value: string

justify	Adjusts placement of children along the main axis when there is extra space, which may be used to fill areas before, after, or in-between: flex-start, flex-end, center, space-between, space-around, space-evenly.	value: string
alignItems	Adjusts placement of children along the cross axis when there is extra space: flex-start, flex-end, center, baseline, stretch.	value: string
alignContent	Adjusts alignment of wrapped content when there is free space in the cross axis: flex-start, flex-end, center, space-between, space-around, stretch.	value: string
style	Use Style and Classes to customize the visual style of the component. The Style menu contains all the tools for modifying text, background, margins, borders, and more. You can also specify a style class.	object

Property User Interface

When a Flex container is deep selected, there is a Graphical User Interface (GUI) at the top of the Perspective Property Editor that enables you to set the container's properties. Functionality is similar to that of the properties in the Props Tree, but you may find the visual interface easier or quicker to use.

Direction

This sets the direction for the child layout. Options are **Row** or **Column**. When the **Reverse** icon is selected, the contents of this container are displayed in reverse order.

Direction: Row

The following table shows the icons and properties they represent when **Direction: Row** is selected. The icon that's displayed if Reverse order is selected is also shown.

Note: Left/right/top/bottom notes in the descriptions refer to non-reversed directions.
The phrase "when there is extra space" means when no components have are stretching to fill the space. ie: when no components have "grow" greater than 0.

Items			
Row Icon	**Row Reversed Icon**	**Property**	**Description**
		Flex Start	Child items are placed along the start (top) of the container when there is extra space.
		Center	Child items are placed along the center of the container when there is extra space.
		Flex End	Child items are placed along the end (bottom) of the container when there is extra space.
		Stretch	Child items are stretched from top to bottom of the container.
		Baseline	Child items are placed so the baseline of the text matches for all of them when there is extra space.
Justify			
Row Icon	**Row Reversed Icon**	**Property**	**Description**
		Flex start	Adjusts placement of children to the start (left) of the container when there is extra space. If reversed, children are placed along the right.
		Center	Adjusts placement of children along the center of the container when there is extra space.
		Flex End	Adjusts placement of children along the end (right) of the container when there is extra space. If reversed, children are placed along the left.
		Space Between	Adjusts placement of children with space in between them reaching to the edges of the container when there is extra space.
		Space Around	Adjusts placement of children with even spacing in between them with some space along the edges when there is extra space.
		Space Evenly	Adjusts placement of children with even spacing in between them and the edges of the container when there is extra space.
Children			
Icon		**Property**	**Description**
N/A		Don't Wrap	If there are more components than the width allows, shrink them.
N/A		Wrap	If there are more components than the width allows, wrap onto the next line.
		Reverse Wrap	Toggle to reverse the direction of wrap from top-down to bottom-up

Content (Only applicable when Children:Wrap is selected.)			
Row Icon	**Row Reversed Icon**	**Property**	**Description**
		Flex start	Adjusts placement of wrapped content to the start (top) of the container when there is free space.
		Center	Adjusts placement of wrapped content to the middle of the container when there is free space.
		Flex End	Adjusts placement of wrapped content to the end (bottom) of the container when there is free space.
		Space Between	Adjusts placement of wrapped content evenly with space in between each wrapped line, reaching to the edges (top and bottom) of the container when there is extra space.
		Space Around	Adjusts placement of wrapped content evenly with space in between each wrapped line and the edges (top and bottom) of the container when there is extra space.
		Stretch	Adjusts placement of wrapped content evenly with space in between each wrapped line and after the last line (bottom) of the container when there is extra space.

Direction: Column

The following table shows the icons and properties they represent when **Direction: Column** is selected. The icon that's displayed if Reverse order is selected is also shown.

Items			
Column Icon	**Column Reversed Icon**	**Property**	**Description**
		Flex start	Child items are placed along the start (left) of the container when there is extra space.
		Center	Child items are placed along the center of the container when there is extra space.
		Flex End	Child items are placed along the end (right) of the container when there is extra space.
		Stretch	Child items are stretched from left to right of the container.
		Baseline	Child items are placed so the baseline of the text matches for all of them when there is extra space.
Justify			
Column Icon	**Column Reversed Icon**	**Property**	**Description**

		Flex start	Adjusts placement of children to the start (top) of the container when there is extra space. If reversed, children are placed along the bottom.
		Center	Adjusts placement of children along the center of the container when there is extra space.
		Flex End	Adjusts placement of children along the end (bottom) of the container when there is extra space. If reversed, children are placed along the top.
		Space Between	Adjusts placement of children with space in between them reaching to the edges of the container when there is extra space.
		Space Around	Adjusts placement of children with even spacing in between them with some space along the edges when there is extra space.
		Space Evenly	Adjusts placement of children with even spacing in between them and the edges of the container when there is extra space.
Children			
Icon		**Description**	
N/A		Don't Wrap	If there are more components than the width allows, shrink them.
N/A		Wrap	If there are more components than the width allows, wrap onto the next line.
		Reverse	Toggle to reverse the direction of wrap from top-down to bottom-up
Content (Only applicable when Children:Wrap is selected.)			
Column Icon	**Column Reversed Icon**	**Property**	**Description**
		Flex start	Adjusts placement of wrapped content to the start (left) of the container when there is free space.
		Center	Adjusts placement of wrapped content to the middle of the container when there is free space.
		Flex End	Adjusts placement of wrapped content to the end (bottom) of the container when there is free space.
		Space Between	Adjusts placement of wrapped content evenly with space in between each wrapped line, reaching to the edges (left and right) of the container when there is extra space.
		Space Around	Adjusts placement of wrapped content evenly with space in between each wrapped line and the edges (left and right) of the container when there is extra space.
		Stretch	Adjusts placement of wrapped content evenly with space in between each wrapped line and after the last line of the container when there is extra space.

Scripting Functions

- Description

  Returns an ArrayList, which contains references to all components inside of the container.

- Parameters

  none

- Return

  Array List - An ArrayList of components in the container. The resulting ArrayList can be iterated over via a for-loop.

Example

For more information and examples for Flex Containers, refer to Flex Containers.

In the following example, we have three components inside a Flex container: Button, Thermometer, and LED Display. The position properties for the Button and LED Display are set so that their size will not change if the Flex container is resized. However, the Thermometer will shrink or grow depending on the Flex container size.

The following properties are set for the Flex Container:

Property	Value
props.direction	row
props.wrap	nowrap
props.justify	space-between
props.alignItems	center
props.alignContent	flex-start

The following properties are set for the Button component within the container:

Property	Value
props.text	Next Tank
position.grow	0
position.shrink	0
position.basis	80px

The following properties are set for the Thermometer component within the container:

Property	Value
props.direction	row
position.grow	1
position.shrink	1
position.basis	50%

The following properties are set for the LED Display component within the container:

Property	Value
position.grow	0
position.shrink	0
position.basis	80px

Perspective - Tab Container

General

Component Palette Icon:

INDUCTIVE UNIVERSITY

Tab Container

Watch the Video

Description

The Tab Container uses tabs as navigation buttons arranged together with the selected tab highlighted. Only one component can be displayed in each tab. See also Tab Containers.

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

The Tab Container component has two pre-configured variants:

- Classic - Layout is a traditional menu with boxed tabs.
- Modern - Layout has no borders around each tab and shows selection with an underline.

Properties

> Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
> This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description
tabs	Responsible for the number of tabs in the container. Adding additional elements to this array will result in an additional tab being rendered In addition, each element is responsible for determining the content of the tab: either text or an embedded view. By default, the componer array, set to a value type. Providing a string to the element will set the text on the tab.

If the element is changed to an object data type, then the text on the tab can be set by adding a **text** value member to the object and prov below. Alternatively, **viewPath** (value data type) and **viewParams** (object data type) can be added to the element, which allows you to re pass parameters to it. Element 1 below demonstrates the idea.

current TabIndex	Which index in tabs array is currently active.
menuTy pe	If the type is 'classic', a traditional menu with boxed tabs is shown. If the type is 'modern', it has no borders around each tab and shows se
tabSize	Default size allotted to a single tab. If a container width does not allow, tab width will shrink from this size accordingly.

Name	Description	Property Type
width	Width in pixels for the tab size.	value: numeric
height	Height in pixels for the tab size.	value: numeric

menuSt yle	Opens the Style menu to change Tab properties: Text, Background, Margin and Padding, Border, and Misc.
content Style	Sets a style for the content frame component. Full menu of style options is available. You can also specify a style class.
tabStyle	Additional styling to apply to all tabs depending active (selected) or inactive state.

Name	Description
active	Sets a style for a tab when it is the **active** tab. The Style menumenu contains all the tools for modifying text, background, mar more. You can also specify a style class.
inactive	Sets a style for tabs that are **inactive**. The Style menu contains all the tools for modifying text, background, margins, borders, also specify a style class.

style	Sets a style for this component. Full menu of style options is available. You can also specify a style class.

User Interaction

The Tab container has some additional ways to interact with it besides just with the Property Editor. The Tab container has two tabs preconfigured. To add or remove tabs, you can do so by interacting with the tabs themselves, or by adding or removing elements in the **tabs** array.

Adding Components to Tabs

To add a component to a tab, deep select the Tab container, then drag a component onto it.

Adding and Deleting Tabs

To delete a tab, click on the "X" to the right of the tab itself.

To add a tab, click on the Add icon to the right of the tabs:

You can also use the right-click menu in the Property Editor. Just right click on the tab you want to work with. You'll see options for copying, pasting,

Scripting Functions

- Description

  Returns an ArrayList, which contains references to all components inside of the container.

- Parameters

  none

- Return

  Array List - An ArrayList of components in the container. The resulting ArrayList can be iterated over via a for-loop.

Example

In this example, the default (Classic variant) Tab Container is used. Three tabs are set up in the Tab Container component. Tab 3 is active and contains a Map component.

Property	Value
props.tabs.0	Tab 1
props.tabs.1	Tab 2
props.tabs.2	Tab 3
props.menuStyle.backgroundColor	#D5D5D5
tabStyle.active.backgroundColor	#00FFFF
tabStyle.active.fontWeight	bold
tabStyle.inactive.backgroundColor	#CCFFFF
tabStyle.inactive.fontWeight	lighter

Example 2

In this example, the default (Modern variant) Tab Container is used. Two tabs are set up in the Tab Container component, each with a Cylindrical Tank component.

Property	Value
props.tabs.0	Tank 1
props.tabs.1	Tank 2
props.menuType	modern

Perspective - Display Palette

Display Components

Perspective offers a variety of components that display static and dynamic information.

Here is a complete list of Display components, and a link pointing to a page containing the component's description, properties, and an example of how to configure it.

Display Components

Watch the Video

In This Section ...

Perspective - Alarm Journal Table

General

Component Palette Icon:

Description

The Perspective Alarm Journal Table displays the history of the alarm system. It can be configured to show active, cleared, and acknowledged events.

The Perspective Alarm Journal Table has a number of configuration options that can be used to filter on realtime and historical alarm data, and change how the component displays those alarms. When you first drag the Alarm Journal Table into the Designer, by default, the table will show you the last 8 hours of alarm journal data. You can interface with the journal table in the Designer, in Preview Mode of the Designer, and in a Perspective Session.

The Alarm Journal Table provides a host of filtering properties that allow you to filter on various parts of alarms and view the details. All the alarm states are visible by clicking the Filter button on the table. There is also a search bar where you can enter text to further refine your filter criteria so you have less alarm events to scroll through. The Alarm Journal Table can filter for alarm events in either Realtime or Historical using the Date Range feature.

You can change the columns that are displayed, the order of the columns, and/or the column width in Preview Mode and in a Perspective Session. Right-click on the table header to show/hide columns. Click and drag the margins of the columns to resize their width. You can also sort table columns in ascending or descending order by simply clicking the up or down arrows next to each column header.

Note: The Alarm Journal must first be set up with a valid database connection for the Alarm Journal Table to see alarm history from the database.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
name	The name of the alarm journal to query. Default is "Journal".	value: string
refresh Rate		value: numeric

	This feature is new in Ignition version **8.1.0** Click here to check out the other new features The rate at which the table will poll for updates in milliseconds.	
enable Header	Enables the table header. Default is true.	value: boolean
enable Details	Enables the details action. Default is true.	value: boolean
toolbar	Settings for the toolbar. **Name** / **Description** / **Property Type** enabled / Enables the visibility of the table toolbar. Default is true. / value: boolean enableDateRange / Enables the visibility of the date range toggle. Default is true. / value: boolean enableFilter / Enables the visibility of the text filter toggle. Default is true. / value: boolean enableFilterResults / Enables the visibility of the filters results count message. Default is true. / value: boolean enablePreFilters / Enables the visibility of the prefilter toggle. Default is true. / value: boolean enableConfiguration / Enables the visibility of the configuration toggle. Default is true. / value: boolean	object
dateFormat	A date format string to be applied against dates. Options are none, date in the format "10/15/2018", time in format "3:59:00 PM", or date time in format "10/15/208 15:59:00".	value: string
responsive	Responsive layout configuration. Rows are converted to cards. While in responsive layout, disables or removes certain table features that are no longer applicable. **Name** / **Description** / **Property Type** enabled / Enables responsive layout. Default is false. / value: boolean breakpoint / Width in pixels that triggers change in responsive layout. / value: numeric	object
dateRange	Settings for date range state. **Name** / **Description** / **Property Type** mode / The current mode: realtime or historical. / value: string realtime / Realtime mode state settings. / object **Name** / **Description** / **Property Type** interval / The realtime interval as an integer / value: numeric unit / The realtime interval unit: hours, days, months and years. / value: string	object
filter	Filter settings. **Name** / **Description** / **Property Type** text / The alarm events filter text. / value: string events / Alarm event types. / object **Name** / **Description** / **Property Type**	object

active	Whether to display alarms with active events. Default is true.	value: boolean
acked	Whether to display alarms with acked events. Default is true.	value: boolean
cleared	Whether to display alarms with cleared events. Default is true.	value: boolean
system	Whether to display alarms with system events. Default is false.	value: boolean

priorities — object

Alarm event priority pre-filters.

Name	Description	Property Type
diagnostic	Whether to display alarms with diagnostic priority. Default is false.	value: boolean
low	Whether to display alarms with low priority. Default is true.	value: boolean
medium	Whether to display alarms with medium priority. Default is true.	value: boolean
high	Whether to display alarms with high priority. Default is true.	value: boolean
critical	Whether to display alarms with critical priority. Default is true.	value: boolean

conditions — object

Gateway side alarm query conditions.

Name	Description	Property Type
source	Filter alarms by alarm source path. Specify multiple paths by separating them with commas. Supports the wildcard *.	value: string
displayPath	Filter alarms by display path, falling back to the source path if a custom display path in not set. Specify multiple paths by separating them with commas. Supports the wildcard *.	value: string
provider	Filter alarms by provider.	value: string

results — object

Alarm event filtering results configuration and data.

Name	Description	Property Type
enabled	Enable filtering results to be written back to props. **Caution:** Enabling this property may cause a performance decline.	value: boolean
data	An array of objects representing the current filtered data, if enabled and active.	array

rowStyles — object

Styles to apply to rows given their alarm event and designated priority.

Name	Description	Property Type
active	Style settings for rows with active alarms.	object

Name	Description	Property Type
base	Base style settings for active alarms. 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

priorities	Style settings for the alarm row based on priority.	object

Name	Description	Property Type
diagnostic	Style for active alarms with diagnostic priority. Full menu of style options is available. You can also specify a style class	object
low	Style for active alarms with low priority. Full menu of style options is available. You can also specify a style class.	object
medium	Style for active alarms with medium priority. Full menu of style options is available. You can also specify a style class.	object
high	Style for active alarms with high priority. Full menu of style options is available. You can also specify a style class.	object
critical	Style for active alarms with critical priority. Full menu of style options is available. You can also specify a style class.	object

acked	Style settings for rows with acked alarms.	object

Name	Description	Property Type
base	Base style settings for acked alarms.	object
priorities	Style settings for the alarm row based on priority.	object

Name	Description	Property Type
diagnostic	Style for acked alarms with diagnostic priority. Full menu of style options is available. You can also specify a style class .	object
low	Style for acked alarms with low priority. Full menu of style options is available. You can also specify a style class.	object

medium	Style for acked alarms with medium priority. Full menu of style options is available. You can also specify a style class.	object
high	Style for acked alarms with high priority. Full menu of style options is available. You can also specify a style class.	object
critical	Style for acked alarms with critical priority. Full menu of style options is available. You can also specify a style class.	object

cleared	Style settings for rows with cleared alarms.	object

Name	Description	Property Type
base	Base style settings for cleared alarms. 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
priorities	Style settings for the alarm row based on priority.	object

Name	Description	Property Type
diagnostic	Style for cleared alarms with diagnostic priority. Full menu of style options is available. You can also specify a style class.	object
low	Style for cleared alarms with low priority. Full menu of style options is available. You can also specify a style class.	object
medium	Style for cleared alarms with medium priority. Full menu of style options is available. You can also specify a style class.	object
high	Style for cleared alarms with high priority. Full menu of style options is available. You can also specify a style class.	object
critical	Style for cleared alarms with critical priority. Full menu of style options is available. You can	object

	also specify a style class.	

system	Style settings for rows with system alarms.	object

Name	Description	Property Type
base	Base style settings for system alarms. 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
priorities	Style settings for the alarm row based on priority.	object

Name	Description	Property Type
diagnostic	Style for system alarms with diagnostic priority. Full menu of style options is available. You can also specify a style class.	object
low	Style for system alarms with low priority. Full menu of style options is available. You can also specify a style class.	object
medium	Style for system alarms with medium priority. Full menu of style options is available. You can also specify a style class.	object
high	Style for system alarms with high priority. Full menu of style options is available. You can also specify a style class.	object
critical	Style for system alarms with critical priority. Full menu of style options is available. You can also specify a style class.	object

columns	Used for determining what column properties to display on the Alarm Journal Table.	object

Name	Description	Property Type
eventTime	Settings for the eventTime column.	object

Name	Description	Property Type

	enabled	Whether the column is enabled. Default is true.	value: boolean
	width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
	strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
	sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

eventId	Settings for the eventId column.	object

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

name	Settings for the name column.	object

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

source	Settings for the source column.	object

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

eventState	Settings for the eventState column.	object

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

priority — Settings for the priority column. — object

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

state — Settings for the state column. — object

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

displayPath — Settings for the displayPath column. — object

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

label	Settings for the label column. **Name** / **Description** / **Property Type** enabled — Whether the column is enabled. Default is true. — value: boolean width — The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static. — value: numeric strictWidth — If enabled, the width of the column (set with the width property) becomes static. Default is false. — value: boolean sort — Default sort order of the column. Options are none, ascending, or descending. — value: string	object
eventValue	Settings for the eventValue column. **Name** / **Description** / **Property Type** enabled — Whether the column is enabled. Default is true. — value: boolean width — The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static. — value: numeric strictWidth — If enabled, the width of the column (set with the width property) becomes static. Default is false. — value: boolean sort — Default sort order of the column. Options are none, ascending, or descending. — value: string	object
isSystemEvent	Settings for the isSystemEvent column. **Name** / **Description** / **Property Type** enabled — Whether the column is enabled. Default is true. — value: boolean width — The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static. — value: numeric strictWidth — If enabled, the width of the column (set with the width property) becomes static. Default is false. — value: boolean sort — Default sort order of the column. Options are none, ascending, or descending. — value: string	object
ackUser	Settings for the ackUser column. **Name** / **Description** / **Property Type** enabled — Whether the column is enabled. Default is true. — value: boolean width — The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static. — value: numeric strictWidth — If enabled, the width of the column (set with the width property) becomes static. Default is false. — value: boolean	object

	sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

ackNotes	Settings for the ackNotes column.	object

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

sortOrder	The default weighted order in which columns and their contents are sorted relative to other columns and their contents. Only works if used when the component loads. Columns need to have sort configured in order for this to work.	array
pager	Settings for the pager.	object

Name	Description	Property Type
enabled	Enables the pager to be displayed. Default is true.	value: boolean
hide	Visually hides the pager from view. Useful when pager is manipulated in a controlled fashion via the activePage property. Default is false.	value: boolean
options	Rows to show per pager option.	array
initialOption	The initial option to use when the table first loads. It must exist as an available option.	value: numeric
activePage	Represents the current active page and corresponds to the value of the page jump input field.	value: numeric

style	Sets a style that applies to the component. Full menu of style options is available. You can also specify a style class.	object

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Extension Functions

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

- Description

  Called for each event before it is displayed in the table, allowing you to hide or show each alarm event (row) in the table. Provides an opportunity to write a more complex filter than what's normally provided to the component. Return False to exclude an alarm event from the table.

- Parameters

ComponentModelScriptWrapper.SafetyWrapper self- A reference to the component that is invoking this function.

PyAlarmEvent alarmEvent - The alarm event itself. Call `alarmEvent.get('propertyName')` to inspect properties on the event. Common properties: 'name', 'source', 'priority'.

- Return

  Boolean - The function must return either a True or False for every alarm event in the table. True will show the alarm. False will hide the alarm.

- Scope

  Session

Examples

With the built-in alarmEvent object all alarm event properties are accessible to this function, and can be used to help determine if any given event should appear on the table. Furthermore, Associated Data (also known as custom alarm properties) can be examined from the same event.

```
# Replace Property Name below with a the name of the property you wish to filter on.
if alarmEvent.get('Property Name'):
    return True
# It's important that you return a False value for the events you don't want to see in the table.
else:
        return False
```

You could also condense the code example above by using something like the following:

```
return "Low"== str(alarmEvent.get('priority'))
```

Examples

For examples and additional information on usage of the Alarm Journal Table, see Alarming in Perspective.

Perspective - Alarm Status Table

General

Component Palette Icon:

Description

The Perspective Alarm Status Table displays the current state of the alarm system. It can be configured to show active, unacknowledged, cleared, and acknowledged alarm states. The Alarm Status Table allows you to view currently active alarm events in the system, providing an easy way to inspect the alarm details, shelve alarms , and acknowledge them.

Acknowledgement is handled by selecting (checking) alarms and pressing the "Acknowledge" button. If any of the selected alarms require acknowledge notes, then a popup window will be presented in which the operator must add notes to the acknowledgement.

The Alarm Status Table provides a host of filtering properties that allow you to filter on various parts of alarms and view the details. All the alarm states are visible by clicking the Filter button on the table. There is also a search bar where you can enter text to further refine your filter criteria so you have less alarm events to scroll through.

Shelving is supported by pressing the "Shelve" button when an alarm is selected and choosing a time duration. This temporarily removes the alarm from the entire alarm system (not just the local client) while you are working to resolve the issue. When the duration time is up, if the alarm is still active, it will pop back into the alarm system. The duration times shown to the user are customizable by editing the values inside the "shelvingTimes" property in the Property Editor. The alarms that have been shelved can be un-shelved by pressing the Unshelve Selected Alarms button at the bottom of the table or the modal in the row of the unshelved alarm.

You can change the columns that are displayed and the column width in Preview Mode and in a Perspective Session. Right-click on the table header to show/hide columns. Click and drag the margins of the columns to resize their width. You can also sort table columns in ascending or descending order by simply clicking the up or down arrows next to each column header. Sorting on alarm State and Priority in the Alarm Status Tabl e sorts in descending order. All the other columns the sort order is alphanumerical.

Note: For information on how to configure the Vision Alarm Status Table refer to Alarming in Perspective.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
refresh Rate	This feature is new in Ignition version **8.1.0** Click here to check out the other new features	value: numeric

	The rate at which the table will poll for updates in milliseconds.	
enable Header	Enable table header. Default is true.	value: boolean
enable Details	Enable active events table details action. Default is true.	value: boolean
enableAcknowledge	Enable acknowledge action. Default is true.	value: boolean
enableShelve	Enable shelve action. Default is true.	value: boolean
enable Unshelve	Enable unshelve action. Default is true.	value: boolean
toolbar	Settings for the toolbar. **Name** / **Description** / **Property Type** enabled / Enables the visibility of the table toolbar. Default is true. / value: boolean enableActiveTab / Enables the visibility of the Active Events tab. / value: boolean enableShelvedTab / Enables the visibility of the Shelved Events tab. / value: boolean enableFilter / Enables the visibility of the text filter toggle. Default is true. / value: boolean enableFilterResults / Enables the visibility of the filters results count message. Default is true. / value: boolean enablePreFilters / Enables the visibility of the pre-filter toggle. Default is true. / value: boolean enableConfiguration / Enables the visibility of the configuration toggle. Default is true. / value: boolean	object
shelving Times	Available alarming shelving times in seconds. Shelving times are customizable by editing values for this property in the Property Editor.	array
responsive	Responsive layout configuration. Rows are converted to cards. While in responsive layout, disables or removes certain table features that are no longer applicable. **Name** / **Description** / **Property Type** enabled / Enables responsive layout. Default is false. / value: boolean breakpoint / Width in pixels that triggers change in responsive layout. / value: numeric	object
filters	This is where you configure filtering properties for displaying alarm data in the Alarm Status Table. **Name** / **Description** / **Property Type** active / Settings for active alarms. / object **Name** / **Description** / **Property Type** text / The active alarm events filter text. / value: string states / Pre-filters for filter active alarm events: / object **Name** / **Description** / **Property Type** activeUnacked / Active and unacknowledged. Default is true. / value: boolean Active and / value:	object

activeAcked	acknowledged. Default is true.	boolean
clearUnacked	Cleared and unacknowledged. Default is true.	value: boolean
clearAcked	Active and acknowledged. Default is false.	value: boolean

priorities	Alarm state priority pre-filters.	object

Name	Description	Property Type
diagnostic	Whether to display alarms with diagnostic priority. Default is false.	value: boolean
low	Whether to display alarms with low priority. Default is true.	value: boolean
medium	Whether to display alarms with medium priority. Default is true.	value: boolean
high	Whether to display alarms with high priority. Default is true.	value: boolean
critical	Whether to display alarms with critical priority. Default is true.	value: boolean

conditions	Gateway side alarm query conditions.	object

Name	Description	Property Type
source	Filter alarms by alarm source path. Specify multiple paths by separating them with commas. Supports the wildcard *.	value: string
displayPath	Filters alarms by display path, falling back to the source path if a custom display path isn't set. Specify multiple paths by separating them with commas. Supports the wildcard *.	value: string
provider	Filter alarms by alarm provider.	value: string

results	Active alarm filtering results configuration and data.	object

Name	Description	Property Type
enabled	Enable filter results to be written back to props. Warning: Doing so may cause performance decline. Default is false.	value: boolean
data	An array of objects representing the current filtered data if enabled	array

	and active.	

shelved	Temporarily silence an alarm for a fixed period of time while the alarm event issue is worked on.	object

Name	Description	Property Type
text	The filter text for shelved alarms.	value: string
results	Shelved alarm filtering results configuration and data.	object

Name	Description	Property Type
enabled	Enable filter results to be written back to props. Warning: Doing so may cause performance decline. Default is false.	value: boolean
data	An array of objects representing the current filtered data if enabled and active.	array

selection	This feature is new in Ignition version **8.1.4** Click here to check out the other new features Currently selected alarms and alarm selection configuration.	object

Name	Description	Property Type
active	Active alarm selection configuration and read-only list of currently selected active alarms.	object

Name	Description	Property Type
mode	Active alarm selection configuration. Options are multiple, single, or none.	value: string
data	A read-only list of currently selected active alarms.	value: array

Name	Description	Property Type
shelved	Shelved alarm configuration and read-only list of currently selected shelved alarms	object

Name	Description	Property Type
mode	Shelved alarm selection configuration. Options are multiple, single, or none.	value: string
data	A read-only list of currently selected active alarms.	value: array

rowStyles	Styles to apply to rows given their alarm state and designated priority.	object

Name	Description	Property Type
activeUnacked	Style settings for rows with activeUnacked alarms.	object

Name	Description	Property Type

base	Base style settings for activeUnacked alarms. Full menu of style options is available. You can also specify a style class.	object
priorities	Style settings for the alarm row based on priority. **Name** / **Description** / **Property Type** diagnostic — Style for activeUnacked alarms with diagnostic priority. Full menu of style options is available. You can also specify a style class. — object low — Style for activeUnacked alarms with low priority. Full menu of style options is available. You can also specify a style class. — object medium — Style for activeUnacked alarms with medium priority. Full menu of style options is available. You can also specify a style class. — object high — Style for activeUnacked alarms with high priority. Full menu of style options is available. You can also specify a style class. — object critical — Style for activeUnacked alarms with critical priority. Full menu of style options is available. You can also specify a style class. — object	object

activeAcked	Style settings for rows with activeAcked alarms. **Name** / **Description** / **Property Type** base — Base style settings for activeAcked alarms. Full menu of style options is available. You can also specify a style class. — object priorities — Style settings for the alarm row based on priority. — object **Name** / **Description** / **Property Type** diagnostic — Style for activeAcked alarms with diagnostic priority. Full menu of style options is available. You can also specify a style class. — object	object

low	Style for activeAcked alarms with low priority. Full menu of style options is available. You can also specify a style class.	object
medium	Style for activeAcked alarms with medium priority. Full menu of style options is available. You can also specify a style class.	object
high	Style for activeAcked alarms with high priority. Full menu of style options is available. You can also specify a style class.	object
critical	Style for activeAcked alarms with critical priority. Full menu of style options is available. You can also specify a style class.	object

clearUnacked	Style settings for rows with clearUnacked alarms.	object

Name	Description	Property Type
base	Base style settings for clearUnacked alarms. Full menu of style options is available. You can also specify a style class.	object
priorities	Style settings for the alarm row based on priority.	object

Name	Description	Property Type
diagnostic	Style for clearUnacked alarms with diagnostic priority. Full menu of style options is available. You can also specify a style class.	object
low	Style for clearUnacked alarms with low priority. Full menu of style options is available. You can also specify a style class.	object
medium	Style for clearUnacked alarms with medium priority. Full menu of style options is available. You can also specify a style class.	object
high	Style for clearUnacked alarms with high priority. Full	object

	menu of style options is available. You can also specify a style class.	
critical	Style for clearUnacked alarms with critical priority. Full menu of style options is available. You can also specify a style class.	object

clearAcked	Style settings for rows with clearAcked alarms.	object

Name	Description	Property Type
base	Base style settings for clearAcked alarms. Full menu of style options is available. You can also specify a style class.	object
priorities	Style settings for the alarm row based on priority.	object

Name	Description	Property Type
diagnostic	Style for clearAcked alarms with diagnostic priority. Full menu of style options is available. You can also specify a style class.	object
low	Style for clearAcked alarms with low priority. Full menu of style options is available. You can also specify a style class.	object
medium	Style for clearAcked alarms with medium priority. Full menu of style options is available. You can also specify a style class.	object
high	Style for clearUnacked alarms with high priority. Full menu of style options is available. You can also specify a style class.	object
critical	Style for clearAcked alarms with critical priority.Full menu of style options is available. You can also specify a style class.	object

dateFormat	A date format string to be applied against dates.	value: string

array

activeSortOrder	The default weighted order in which columns and their contents are sorted relative to other columns and their contents. Used when the component loads. Active event columns need to have sort configured in order for this to work.	
shelvedSortOrder	The default weighted order in which columns and their contents are sorted relative to other columns and their contents. Used when the component loads. Shelved event columns need to have sort configured in order for this to work.	array
columns	Used only for determining what columns to show on load.	object

Name	Description	Property Type
active	Active alarm even columns to display on load	object

Name	Description	Property Type
activeTime	Settings for the activeTime column.	object

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

Name	Description	Property Type
displayPath	Settings for the displayPath column.	object

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

Name	Description	Property Type
priority	Settings for the priority column.	object

Name	Description	Property Type

enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

state	Settings for the state column.	object

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

source	Settings for the source column.	object

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

label	Settings for the label column.	object

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

name	Settings for the name column.	object

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

eventId	Settings for the eventId column.	object

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

eventValue	Settings for the eventValue column. **Name** / **Description** / **Property Type** enabled / Whether the column is enabled. Default is true. / value: boolean width / The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static. / value: numeric strictWidth / If enabled, the width of the column (set with the width property) becomes static. Default is false. / value: boolean sort / Default sort order of the column. Options are none, ascending, or descending. / value: string	object
notes	Settings for the notes column. **Name** / **Description** / **Property Type** enabled / Whether the column is enabled. Default is true. / value: boolean width / The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static. / value: numeric strictWidth / If enabled, the width of the column (set with the width property) becomes static. Default is false. / value: boolean sort / Default sort order of the column. Options are none, ascending, or descending. / value: string	object
isActive	Settings for the isActive column. **Name** / **Description** / **Property Type** enabled / Whether the column is enabled. Default is true. / value: boolean width / The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static. / value: numeric strictWidth / If enabled, the width of the column (set with the width property) becomes static. Default is false. / value: boolean sort / Default sort order of the / value:	object

	column. Options are none, ascending, or descending.	string

isAcked	Settings for the isAcked column. **Name** / **Description** / **Property Type** enabled — Whether the column is enabled. Default is true. — value: boolean width — The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static. — value: numeric strictWidth — If enabled, the width of the column (set with the width property) becomes static. Default is false. — value: boolean sort — Default sort order of the column. Options are none, ascending, or descending. — value: string	object
isClear	Settings for the isClear column. **Name** / **Description** / **Property Type** enabled — Whether the column is enabled. Default is true. — value: boolean width — The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static. — value: numeric strictWidth — If enabled, the width of the column (set with the width property) becomes static. Default is false. — value: boolean sort — Default sort order of the column. Options are none, ascending, or descending. — value: string	object
ackTime	Settings for the ackTime column. **Name** / **Description** / **Property Type** enabled — Whether the column is enabled. Default is true. — value: boolean width — The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static. — value: numeric strictWidth — If enabled, the width of the column (set with the — value: boolean	object

	width property) becomes static. Default is false.	
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

ackUser	Settings for the ackUser column.	object

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

ackNotes	Settings for the ackNotes column.	object

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

ackPipeline	Settings for the ackPipeline column.	object

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e.,	value: numeric

	flex grow. If strictWidth is enabled, the column will be fixed and static.	
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

activePipeline	Settings for the activePipeline column.	object

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

clearTime	Settings for the clearTime column.	object

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

clearPipeline	Settings for the clearPipeline column.	object

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean

width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

deadband	Settings for the deadband column.	object

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

shelved	Shelved alarm columns to display on load.	object

Name	Description	Property Type
expires	Settings for the expires column.	

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

			shelved By	Settings for the shelvedBy column. **Name** / **Description** / **Property Type** enabled — Whether the column is enabled. Default is true. — value: boolean width — The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static. — value: numeric strictWidth — If enabled, the width of the column (set with the width property) becomes static. Default is false. — value: boolean sort — Default sort order of the column. Options are none, ascending, or descending. — value: string			
			sourcePath	Settings for the sourcePath expires column. **Name** / **Description** / **Property Type** enabled — Whether the column is enabled. Default is true. — value: boolean width — The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static. — value: numeric strictWidth — If enabled, the width of the column (set with the width property) becomes static. Default is false. — value: boolean sort — Default sort order of the column. Options are none, ascending, or descending. — value: string			
pager	Settings for the pager. **Name** / **Description** / **Property Type** enabled — Enables the pager to be displayed. Default is true. — value: boolean hide — Visually hides the pager from view. Useful when pager is manipulated in a controlled fashion via the activePage property. Default is false. — value: boolean options — Rows to show per pager option. — array initialOption — The initial option to use when the table first loads. It must exist as an available option. — value: numeric activePage — Represents the current active page and corresponds to the value of the page jump input field. — value: numeric shelvedPage — Represents the current shelved page and corresponds to the value of the page jump input field. — value: numeric							object
style	Sets a style that applies to the component. Full menu of style options is available. You can also specify a style class.						object

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Extension Functions

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

- Description

  Called for each event before it is displayed in the table, allowing you to hide or show each alarm event (row) in the table. Provides an opportunity to write a more complex filter than what's normally provided to the component. Return False to exclude an alarm event from the table.

- Parameters

  ComponentModelScriptWrapper.SafetyWrapper self- A reference to the component that is invoking this function.

  PyAlarmEvent alarmEvent - The alarm event itself. Call `alarmEvent.get('propertyName')` to inspect properties on the event. Common properties: 'name', 'source', 'priority'.

- Return

  Boolean - The function must return either a True or False for every alarm event in the table. True will show the alarm. False will hide the alarm.

- Scope

  Session

Examples

With the built-in alarmEvent object all alarm event properties are accessible to this function, and can be used to help determine if any given event should appear on the table. Furthermore, Associated Data (also known as custom alarm properties) can be examined from the same event.

```
# Replace Property Name below with a the name of the property you wish to filter on.
if alarmEvent.get('Property Name'):
    return True
# It's important that you return a False value for the events you don't want to see in the table.
else:
        return False
```

You could also condense the code example above by using something like the following:

```
return "Low"== str(alarmEvent.get('priority'))
```

Examples

Example 1

In a Perspective Session, click on the **Filter** button to filter on alarm states and/or use the **Search Bar** to target more specific alarm events. Select from the **Configuration Settings** to change the column headers to display the alarm event data you're interested in.

This example uses the Search Bar to find any text with 'High Wind Speed' in the Active,Unacknowledge alarm state.

Example 2 Alarm Status Table Row Styles

In the Designer, you can change row styles to be different colors for the different priorities for each alarm state. In this example, the rowStyle for the Critical priority for the activeAcked alarm state was changed to green.

Perspective - Barcode

General

Component Palette Icon:

Description

The Barcode component enables you to display text as a barcode. The component supports 105 different barcode types including Code 128, QR code, EAN-8, and ISBN.

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

The Barcode component has two pre-configured variants: Code 128 and QR Code.

Properties

> Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
> This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
value	Value to be encoded as a barcode.	value: numeric
type	What barcode specification to use, currently there are 105 formats supported including Code 128, QR code, EAN-8, and ISBN.	value: string

A-B	C	D	E-G
auspost	channelcode,	daft	ean13
azteccode	codablockf,	databarexpanded	ean13composite
azteccodecompact	code11	databarexpandedcomposite	ean14
aztecrune	code128	databarexpandedstacked	ean2
bc412	code16k	databarexpandedstackedcomposite	ean5
	code2of5	databarlimited	ean8
	code32	databarlimitedcomposite	ean8composite
	code39	databaromni	flattermarken

	code39ext	databaromnicomposite	gs1-128	
	code49	databarstacked	GS1-128	
	code93	databarstackedcomposite	gs1-128composite	
	code93ext	databarstackedomni	gs1-cc	
	codeone	databarstackedomnicomposite	gs1datamatrix	
	coop2of5	databartruncated	gs1datamatrixrectangular	
		databaretruncstedcomposite	gs1northamericancoupon	
		datalogic2of5	gs1qrcode	
		datamatrix		
		datamatrixrectangular		
		datamatrixrectangularextension		
		dotcode		
H-J	**K-P**	**Q-Z**		
hanxin	kix	qrcode		
hibcazteccode	leitcode	rationalizedCodabar		
hibccodeablockf	mailmark	raw		
hibccode128	matrix2of5	royalmail		
hibccode39	maxicode	sscc18		
hibcdatamatrix	micropdf417	symbol		
hibcdatamatrixrectangular	microqrcode	telepen		
hibcmicropdf417	msi	telepennumeric		
hibcpdf417	onecode	ultracode		
hibcqrcode	pdf417	upca		
iata2of5	pdf417compact	upcacomposite		
identcode	pharmacode	upce		
industrial2of5	pharmacode2	upcecomposite		
interleaved2of5	planet			
isbn	plessey			
ismn	posicode			
issn	postnet			
itf14	pzn			
japanpost				

displayValue	If true, the barcode's value will be displayed as text.	value: boolean
valuePosition	If displayValue is true, this property determines where the value should be displayed. Options are top or bottom; default is bottom.	value: string
valueStyle	Sets a style for the display value for this component. 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
errorStyle	Sets a style for this component. 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
style	Sets an overall style for this component. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1

Property	Value
props.value	014113910613
props.type	upca
props.displayValue	true
props.valuePosition	bottom
props.valueStyle.fontFamily	Verdana
props.valueStyle.fontSize	18px

Example 2

Property	Value
props.value	http://inductiveautomation.com
props.type	qrcode
props.displayValue	true
props.valuePosition	top
props.valueStyle.color	#2747C7
props.valueStyle.fontFamily	sans-serif

props.valueStyle.fontSize	14px
props.valueStyle.fontWeight	bold
props.style.paddingTop	12px
props.style.borderColor	D97700

Perspective - Cylindrical Tank

General

Component Palette Icon:

Description

A component that looks like a 3D cylindrical tank with some liquid inside. Component can be configured so that the "liquid" rises and falls as the 'value' property changes. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a style class.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
value	Numeric value of the tank's level.	value: numeric
capacity	Total capacity of the tank. Default is 100.	value: numeric
liquidColor	Color used to render the filled part of the tank. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	color
tankColor	Color of the non-filled tank section. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	color
liquidOpacity	The opacity of the liquid in the tank. 0 is fully transparent, 1 is fully opaque. Default is 0.5.	value: numeric
liquidWarningColor	The warning color of the liquid in the tank. See Color Selector.	color
tankWarningColor	The tank warning color. See Color Selector.	color
warning	The warning threshold indicates by turning a pale red when its reached or exceeded its set value. Default is 100.	value:

Threshold		numeric
strokeWidth	The stroke width, in pixels, for the outside of the tank. Default is 1.	value: numeric
valueDisplay	Value display configuration. Renders and styles a value overlay in the tank. **Name** / **Description** / **Property Type** enabled — Whether valueDisplay is shown. Default is true. — value: boolean style — Modify the valueDisplay style using the style properties. 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	object
style	Sets a style for this cylindrical tank. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

Property	Value
props.value	85
props.capacity	100
props.warningThreshold	80

Perspective - Dashboard

General

Component Palette Icon:

Description

The Dashboard component exposes layout capabilities to end users in a Perspective session so they have the ability to customize their dashboard layout for their individual needs. Widgets are configured in the Designer by designers and made available to Perspective session users. The Dashboard component uses a grid system based off of CSS grid specifications to position and place widgets.The Property Editor of the Dashboard component is where the designer controls the general layout of the grid by specifying the responsive mode: fixed or stretch, if the dashboard is editable, and if each widget is configurable and available in a Perspective session.

End users can choose from a list of pre-configured widgets to configure their dashboards in a Perspective session. They can add, remove, resize, move around, and configure widgets, including the ability to interact with widgets in a session such as entering text in a text field, displaying/hiding components in a widget, and even use parameters to pass a property to a specified view.

To learn more, refer to Configuring a Dashboard.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description

pack	This feature is new in Ignition version **8.1.4** Click here to check out the other new features Enables widget packing algorithm. When disabled, widgets can be placed anywhere on the Dashboard and the component will not try to rearrange them in an optimal layout.
grid	The grid layout mode defines the responsive behavior of the grid and its cells: fixed and stretch.

Name	Description	Propert Type
fixed	In fixed mode, the grid's dimensions can be greater or less than the full dimensions of its containing element, and its cells are given a static size, effectively creating a scrollable grid when cells overflow beyond the containing elements dimensions.	value: string
stretch	In stretch mode, the grid's dimensions are restricted to the full dimensions of its containing element, and its cells consume one free unit of space, effectively growing and shrinking with the containing element.	value: string

isEditing	Controls the runtime edit mode of the dashboard component. Stays in sync with the edit/play toggle control located at the bottom of the component.
editingToggle	Whether to display the dashboard editing toggle option. When disabled, hides the built in edit/play toggle control located at the bottom of t component. Disable this if you'd like to implement your own toggle that updates the `isEditing` prop in a controlled fashion. Default is tr
fixed	Visible when the grid mode is `fixed` .

Name	Description	Property Type
cellSize	Width and height of a grid cell. Exclusively for fixed mode.	numeric
rowCount	The number of rows in the grid.	numeric
columnCount	The number of columns in the grid.	numeric
rowGuttersize	The gap size between grid rows.	numeric
columnGutterSize	The gap size between grid columns.	numeric

stretch	Visible when the grid mode is `stretch` .

Name	Description	Property Type
rowCount	The number of rows in the grid.	numeric
columnCount	The number of columns in the grid.	numeric
rowGutterSize	The gap size between grid rows.	numeric
columnGutterSize	The gap size between grid columns.	numeric

widgets	An array of configuration objects for widgets currently in use the dashboard display.

Name	Description	Propert Type
name	The unique widget name.	value: stri
viewPath	The current configuration view path of the widget.	value: stri
viewParams	Parameters being passed to the view. This feature is new in Ignition version **8.1.4** Click here to check out the other new features	object

	As of 8.1.4 a dropdown list of parameters is available when the user clicks the **Add Object Member** icon. This makes it easy to add parameters from the rendered view. 	
isConfigurable	Whether this widget is configurable during runtime. If enabled, dashboard is in edit mode, the toggle becomes available when the widget is selected which is used to configure the widgets view. When toggled on, the configuring view parameter will be true.	value: boolean
header	Configuration object for the widget header. **Name** / **Description** / **Property Type** enabled / When enabled, renders the widget header. / value: boolean title / The header title to display. / value: string style / Style to be applied to the widget header. Full menu of style options is available. You can also specify a style class. / object	object
body	Configuration object for the widget body. **Name** / **Description** / **Property Type** style / Style to be applied the widget body. Full menu of style options is available. You can also specify a style class. / object	object
minSize	Specifies the widgets minimum allowable size when determining widget layout. Users may not resize widgets below these dimensions. **Name** / **Description** / **Property Type** columnSpan / The minimum allowable columns that this widget may span. / value: numeric rowSpan / The minimum allowable rows that this widget may span. / value: numeric	object
position	The widget position in the dashboard. Whenever a widget is added, resized, or moved the widget position object is automatically updated. **Name** / **Description** / **Property Type** rowStart / The top position of the widget. / value: numeric rowEnd / The bottom position of the widget. / value: numeric columnStart / The left position of the widget. / value: numeric columnEnd / The right position of the widget. / value: numeric	object
style	Style to be applied the widget. Full menu of style options is available. You can also specify a style class.	object

availableWidgets	An array of widgets as configuration objects that are available to the user. When a widget is added to the dashboard via the add widget modal, this configuration object is copied to the widgets in use array, and act as the widgets defaults.

Name	Description	Property Type
viewPath	The current configuration view path of the widget.	string
viewParams	Parameters being passed to the view at the specified path. This feature is new in Ignition version **8.1.4** Click here to check out the other new features As of 8.1.4 a dropdown list of parameters is available when the user clicks the **Add Object Member** icon. This makes it easy to add parameters from the rendered view.	object
isConfigurable	Whether this widget is configurable during runtime. If enabled and the dashboard is in edit mode, the toggle becomes available when the widget is selected which is used to configure the widgets view. When toggled on, the configuring view parameter will be 'true.'	value: boolean
defaultSize	Specifies the widgets default size adding a widget with no size specified.	object

Name	Description	Property Type
columnSpan	The default columns that this widget will span.	value: numeric
rowSpan	The default rows that this widget will span.	value: numeric

Name	Description	Property Type
minSize	Specifies the widgets minimum size used when determining widget layout.	object

Name	Description	Property Type
columnSpan	The minimum allowable columns that this widget may span.	value: numeric
rowSpan	The minimum allowable rows that this widget may span.	value: numeric

Name	Description	Property Type
category	A category in which to group this widget when displayed in the add widgets modal.	value: string
name	A unique name to provide this widget. This is used in the add widget modal. If no name is specified, its value will be blank. This is a required property.	value: string
header	Widget header configuration.	object

Name	Description	Property Type
enabled	Whether the widget header should show.	value: boolean
title	The header title to display.	value: string
style	Style to be applied the widget. Full menu of style options is available. You can also specify a style class.	object

Name	Description	Property Type
body	Widget body configuration.	object

Name	Description	Property Type
style	Style to be applied the widget. Full menu of style options is available. You can also specify a style class.	object

Name	Description	Property Type
style	Style to be applied to the widget. Full menu of style options is available. You can also specify a style class.	object

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

In addition to the demonstration below, learn more about the Dashboard component on the Configuring a Dashboard page.

Demonstration

This image shows the dashboard in a Perspective Session with one widget.

Property	Value
props.grid	stretch
props.isEditing	true
props.editingToggle	true
props.stretch.rowCount	6
props.stretch.columnCount	14
props.widgets.0.name	Widget 3
props.widgets.0.viewPath	Widget 3
props.widgets.0.isConfigurable	true
props.widgets.0.header.enabled	true
props.widgets.0.header.title	Widget 3
props.availableWidgets.2.viewPath	Widget 3
props.availableWidgets.2.isConfigurable	true

props.availableWidgets.2.name	Widget
props.availableWidgets.2.header.enabled	true
props.availableWidgets.2.header.title	Widget 3

This same example also allows the user to interact with the widget in Edit mode when the `'isConfigurable'` property is set to 'true.' The view uses a `'configuring'` parameter to go into 'configuring' mode allowing users to configure the widget. It allows the user to show/hide the Thermometer showing the temperature of the Tank by simply checking/unchecking the ShowTemp checkbox.

By unchecking Show Temp, the Thermometer component is removed from the widget on the dashboard in a Perspective Session.

Widget 3
12%
Show Temp
Widget 3
50%

Configuring a Dashboard

The Dashboard exposes widgets to end users in a Perspective Session so they can customize their dashboard layout for their individual needs. Widgets are views that are pre-configured in the Designer and made available to Perspective Session users. End users have the flexibility to add, remove, resize, move around, and even configure widgets in the dashboard of their Perspective Session without having access to the Designer. Users can interact with widgets in a session on both desktop and mobile devices. There may be some minor variances in how a user can interact with their dashboard between desktop and mobile devices, but the principle is still the same.

On this page ...

- Configuring a Dashboard Component
- Setting Up a User Dashboard
  - Adding a Widget
  - Removing a Widget
  - Moving a Widget
  - Resizing a Widget
- Configuring a Widget
  - Setting a Widget as Configurable in the Designer
  - Creating a Configurable View in the Designer
- Saving Perspective Session Edits and Populating Widgets

Configuring a Dashboard Component

Configuring a Dashboard starts with designing widgets and having a selection of pre-configured widgets for users to choose from to configure their individual dashboards. Designers create the widgets and make them available for end users to use in their individual dashboards. By making the widgets available using the 'availableWidgets' property, the widget overlay modal is populated with a searchable list of all the available widgets a user can add to their dashboard. The dashboard component contains a host of addtional properties that can be configured based on the end-user requirements.

The Dashboard component uses a grid system based off of CSS grid specifications to position and place your widgets. The Property Editor settings of the Dashboard component control the general layout of the grid. They specify the responsive mode: fixed or stretch, if the dashboard is editable, and if each widget is configurable and available in a Perspective session. The image below shows one widget on a dashboard in the Designer along with some of its properties.

To learn more about Dashboard properties, refer to the Dashboard component page.

Setting Up a User Dashboard

Setting up a dashboard starts with users choosing from a list of pre-configured widgets to configure their dashboards in a Perspective Session based on their individual needs.

To edit the dashboard in a Perspective Session, the user can put the dashboard into Edit mode by clicking the Edit icon at the bottom of the dashboard and deciding what widgets they want, where they want them, and how they want them configured. They can add, remove, resize and configure widgets, including the ability to interact with widgets such entering text in a text field or displaying/hiding components in a widget. You can also remove this control entirely and implement your own by configuring the `'editingToggle'` property on the component. Refer to the Dashboard component properties for more details.

The following sections on this page describe how to setup your own dashboard.

Adding a Widget

There are two ways a user can add a widget in a Perspective Session: by clicking on a single grid cell, or by dragging a grid cell over multiple grid cells that opens an add widget overlay as shown in the image below. Both ways result in displaying the add widget modal which provides a searchable list of all of the available widgets a user may add to their dashboard.

Dragging a grid cell creates an add widget overlay that specifies the desired dimensions of the widget to add. If the desired widget position overlaps other widgets, the overlapped widgets will be moved to any available space on the dashboard. Widgets do not overlap when being added, resized, and moved unless there happens to be no space for a widget so that it is placed within the grid.

Widget's Minimum Dimensions
When adding a widget, if the desired dimensions are less than the configured minimum dimensions, the desired dimensions will get overridden by the minimum dimensions. If a single grid cell is clicked, the configured default dimensions will be applied, if and only if, the default meets the required minimum dimensions, otherwise the minimum dimensions are applied. By default, the minimum and default dimensions for a widget are 1x1.

On mobile devices, activating a grid cell requires a long-press of about a second. Once a grid cell is activated, you can then drag to create the add widget overlay. The image below shows Widget 1 dropped over the multiple selected grid cells in the dashboard. You'll notice the active widget has a dashed blue border.

Removing a Widget

Click the **Edit** icon, select the widget, and you'll notice the widget has a dashed blue border indicating the widget is active, then click the **Delete** '**X**' icon in the top right corner to remove the widget from the dashboard. You will then be prompted with a confirmation modal to delete the widget. Click **Remove**.

Moving a Widget

Put the dashboard in Edit mode, select the widget so that it becomes active (dashed blue border). Drag the widget to the desired position. As you move the widget, any overlapped widgets will be repositioned into the first available space.

Resizing a Widget

To resize a widget, put the dashboard in Edit mode, then simply select the widget you'd like to resize and drag one of the resize handles. If, while resizing, the widget overlaps other widgets, the overlapped widgets will be repositioned into the first available space.

Configuring a Widget

The dashboard allows your users to configure a widget in a Perspective Session. To do this, you need make a few changes to your view and Dashboard component configuration.

Setting a Widget as Configurable in the Designer

To make a view allow configuration, you need to set the **isConfigurable** property for each widget that needs to be configured. This will set a param value on your view (in the runtime) that you can use to create a configuration display in your view. The purpose of this parameter is to avoid having to make a separate widgets for each possible variation of the same view.

1. Select the Dashboard component.
2. Expand the **availableWidgets** parameter, and expand the array object for the widget that you want to make configurable.
3. Set the **isConfigurable** property to 'true' for this widget.

Creating a Configurable View in the Designer

To make a view configurable, you need to do a bit of work to alter what is in the view. This is possible because the Dashboard component was created to use a parameter named **`configuring`** that is set to **'true'** when the widget is in put into configuration mode. The idea here is to have a second 'mode' or 'display' version of the view that has controls on it to effect the primary display. The best way to do this is to create two containers in your view; one for configuration, and one for display. You can then bind the visibility on each container so only one is shown at a time.

To learn more about using parameters to pass properties, refer to the Perspective Component Properties page. You will not need to pass any value into the param though, it is done automatically for you if you get the param name correct.

1. Create a new Coordinate view. For the example, we named our Configurable_View.

2. **Note:** If you use a Flex container, some of the settings will be different further down in the example.

   In the Property Editor, add a view param. Name the param **'`configuring`'** and set the value to 'true'. Note the spelling and (lack of) capitalization.

3. Drag a Coordinate Container component inside your view. Give the container a good name like 'Runtime.'
4. Add any display components you want.

   Note: If you started from an existing view, move all existing components into the new container then make the container fill your entire view.

   a. Deep Select the Configure container.
   b. Drag a Cylindrical Tank component into it.
   c. Bind the value property to a Tag.
   d. Drag a Temperature Gauge component into it.
   e. Bind the value property to a Tag.

5. We need to create a second new container in the view for your configuration. **Duplicate** the Runtime container. Give the container a good name like 'Configure.' This container will be a sibling to the Runtime container, not inside the Runtime container.

6. Add a Checkbox to toggle the temperature component visibility.

> **Note:** If you started with an existing view, this step is completely up to you. You will decide what should be configurable and create controls for that in your configuration container. For example, you could create a list of Tags for the user to select between and display only the selected Tags on a chart.

   a. Deep Select the Configure container.
   b. Drag a **Checkbox** component into it.
   c. Set the Name and Text to "Show Temp".

7. Now we need to alter the Runtime container components to listen to our new controls.

   a. Deep Select the Runtime Container.
   b. Select the Temperature Gauge component.
   c. In the Property Editor under META, bind the **Visible** property to the Selected value of the Show Temp Checkbox component.

8. Now we just need to show one container at at time.

> **Note:** If you used a Flex container at the start of this example, then use the 'display' property instead of the 'visible' property in the following steps.

   a. Bind the `'visible'` property for the Runtime container using an expression. It should be true when the **`configuring`** param is false.

b. Bind the '`visible`' property for the Configure container using an expression. It should be true when the **`configuring`** param is true.

9. Save your project and then put the Designer into Preview mode. When you click on the Temp Show button, you'll see the Temperature component appear or disappear.

10. Now you can use this view in your dashboard.

Configuring in the Dashboard Component in the Runtime (Perspective Session)

This section is for the people using the Dashboard in a Perspective Session.

1. To use a configurable widget in the runtime, put the dashboard in Edit mode and select the widget you'd like to configure.
2. Click the edit icon (pencil) in the top right corner of the widget. The widget's border will change colors from blue to orange (shown in the image below).
3. The view changes to show the 'configuring' mode you set up previously for the view, allowing users to configure the widget.

Saving Perspective Session Edits and Populating Widgets

Edits that an end user makes in their dashboard in a Perspective Session are not automatically saved and do not persist when the end user's session restarts. A session can be refreshed within the same session. One possible solution for populating widgets for the next editing session is to add a property change script on the '`widgets`' prop to listen for changes and then write that value back to a database along with any user information derived from the active session. The value of the '`widgets`' prop will be an array of QualifiedValues, which you'll need to handle accordingly.In similar fashion, consider adding an '`onStartup`' event action that will query the database and then populate the '`widgets`' prop with the users last saved configuration and optionally populate the '`availableWidgets`' prop (possibly for varying user roles).

Related Topics ...

- Perspective Dashboard Component
- Passing Parameters
- Views and Containers

Perspective - Icon

General

Component Palette Icon:

Icon

INDUCTIVE UNIVERSITY

Icons

Watch the Video

Description

The Icon is an image path commonly used to augment a label or text property of a component by placing an image next to it.

The materials icon library is a primary source for icons, see https://fonts.google.com/icons?selected=Material+Icons. You can also add your own custom repository of icons. For more information, see Images, SVGs, and Icons in Perspective.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
path	Shorthand path to icon source, in format: library/IconName (i.e., material/3d_rotation). The materials icon library is a great source for icons, see https://fonts.google.com/icons?selected=Material+Icons.	value: string
color	Color of the icon. See Color Selector.	color
style	Sets a style for this component. Full menu of style options is available. You can also specify a style class.	object

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1

Property	Value
props.color	#00AC00

Example 2

Property	Value
props.path	material/record_voice_over
props.color	#000088
props.style.borderstyle	ridge
props.style.borderColor	#FFAC47
props.style.borderWidth	6px

Perspective - Image

General

Component Palette Icon:

INDUCTIVE UNIVERSITY

Images

Watch the Video

Description

The Image component displays either vector or raster format images, such as a jpeg, gif, png, or svg. For examples see Images, SVGs, and Icons in Perspective.

When attempting to show images from the Image Management Tool on this component, you'll need to prefix `/system/images/` to the path. For example:

```
/system/images/Builtin/icons/48/about.png
```

External Images

The Image component can also be used to show external images stored relative to the local file system on the client. The file path is similar to having your browser view a local document:

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
source	The image source URL. It could be a URL to an image on the internet or Gateway, or even an embedded image. You can find images in the Image Management tool and copy their path for this source property. Format: **/system/images/<imagepath>** For example, the Builtin/icons/16/about.png image path would be **/system/images/Builtin/icons/16/about.png** In addition, source can be set to a Base64 encoded image	value: string
altText	An alternate text for the image if the image cannot be displayed because of a slow connection, an error in the source attribute, if the user uses a screen reader, or some other reason.	value: string
fit	Whether or not the image will size to fit. When in percent mode, the parameters are used to fit based on the percentage of the width and height. When in absolute mode, the image will fit the width and height sizes in pixels.	object

Name	Description	Property Type
mode	Can be one of the following modes: none, fill, contain, cover, percent, or absolute.	value: string
width	Width of the image in pixels.	value: numeric
height	Height of the image in pixels.	value: numeric
scroll	If false, scrolling is not enabled.	value: boolean

tint	Enables you to tint the entire image a color.	object

Name	Description	Property Type
enabled	Turn tint on (true) and off (false).	value: boolean
color	If the tint filter is on, this is the color of the tint. See Color Selector.	color

style	Sets a style for this component. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous. Classes are predefined styles in a project.	object

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1

Property	Value
props.source	/system/images/Builtin/Flow/Flow 7.png
props.fit.mode	contain
props.tint.enabled	true
props.tint.color	#FFF00
position.width	150
position.height	115

Example 2

Property	Value	Style Category
props.source	https://inductiveautomation.com/static/images/logos/inductive-automation-logo.png	N/A
props.style.borderStyle	solid	border
props.style.borderWidth	1px	border

Perspective - Inline Frame

General

Component Palette Icon:

Description

The Inline Frame component enables you to display a webpage within the component, allowing another HTML page to be embedded in the view.

Note that many websites will not support rendering if they're inside a frame, such as this component. Website choose to opt in to this via the x-frame-options HTTP header, which all browsers support. The x-frame-options header is designed to help prevent a class of web security attacks called Clickjacking.

Thus, if a given page returns a DENY or SAMEORIGIN value for x-frame-options, then a web browser will refuse to render the content in the iframe on your Perspective page.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type

src	The source URL of the webpage you want to embed in this frame.	value: string
allowFullScreen	Whether or not to allow embedded webpage to display full screen. Default is false.	value: boolean
referrerPolicy	Referrer Policy is used to determine what information is sent along with the requests. The referral value is stripped when going from a page using HTTPS to a page using the HTTP protocol. This is because this is the default setting for the Referrer Policy if nothing is specified. Technically, this is “no-referrer-when-downgrade,” which means it will strip the referral when downgrading to an insecure request like switching from HTTPS to HTTP. You don’t have to use the default setting, though. Options as follows: • no-referrer • no-referrer-when-downgrade • origin • origin-when-cross-origin • unsafe-url	value: string
style	Sets a style for this component. 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
border	Specifies how the border property is set on the webpage inside the Inline Frame. The default is unset. To set a border, use the borderStyle property in Style on this component. Using the borderStyle property on the component overrides the border property.	value: string

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

Property	Value
props.src	https://www.youtube.com/embed/hYXUZeLw5ek
props.allowFullScreen	true
props.referrerPolicy	origin
props.style.borderStyle	ridge
props.style.borderWidth	6px
props.style.borderColor	#FF8C00

Perspective - Label

General

Component Palette Icon:

Description

The Label component displays text and can be customized with a full menu of style options for the appearance of text, background, border, color, etc. You can use bindings to display additional information on the Label component.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
text	Text to display.	value: string
alignVertical	Vertical alignment of the text within the component (top, center, or bottom), based on the dimensions of the component. Default is top.	value: string
style	Sets a style for this component. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

Property	Value	Style Category
props.text	(Expression Binding) `"The time is : "+dateformat(now(),"h:mm a")`	N/A

props.style.borderStyle	groove	border
props.style.padding	12px	margin and padding

Perspective - LED Display

General

Component Palette Icon:

Description

The LED Display is a stylized numeric and/or alphanumeric label. It has two visual styles: 7-segment and 14-segment and supports nine common number format patterns. Use the value property to enter numeric and/or alphanumeric characters.

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

TLED Display component has two pre-configured variants:

- 14 Segment - Appearance is that of an LED with 14 light segments.
- 7 Segment - Appearance is that of an LED with 7 light segments.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
value	Value to be displayed.	value: numeric
segmentFormat	Style of each character/digit and the number of segments that compose the character. There are two different visual styles: 7 segment and 14 segment. Default is 14 segment.	value: string
numberFormat	Format of display for numeric characters, including commas, decimal places, percent, etc. There are nine options available from a dropdown list. Default is #,##0.00.	value: string
backgroundColor	Background color of the LED display. Default is #3A3A3A. See Color Selector.	color
diodeOnColor	Color of LED segments when switched on. Default is #00FF00. See Color Selector.	color
diodeOffColor	Color of LED segments when switched off. Generally different from display background color to preserve analog look. See Color Selector.	color
locale	Localization code that determines rules for commas, decimals, etc. Default is en-US.	value: string
style	Sets a style for this component. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component

events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1

Property	Value
props.value	12/34
props.segmentFormat	7 segment
props.backgroundColor	#D5D5D5
props.diodeOnColor	#0062FF
props.OffColor	#0062FF1A

Example 2

Property	Value	Style Category
props.value	52,345.9	N/A
props.numberFormat	#,##0.00	N/A
props.backgroundColor	#000000	N/A
props.diodeOnColor	#00FF00	N/A
prop.diodeOffColor	#000000	N/A
props.style.borderStyle	groove	border
props.style.padding	2px	margin and padding

Perspective - Linear Scale

General

Component Palette Icon:

Description

The Linear Scale component displays a series of tick marks and labels that represent a range between a minimum value and a maximum value. It also displays indicators that represent a value or range of values, correctly positioned on the linear scale.

Linear Scale component allows floating point tick marks such as 0.25, 0.5, 0.75, 1.25, etc.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
minValue	The minimum value displayed on the scale.	value: numeric
maxValue	The maximum value displayed on the scale.	value: numeric
majorTicks	Settings for the major tick marks on the scale.	object

Name	Description	Property Type
span	Distance between each tick mark of this type, in pixels. Default is 20.	value: numeric
length	Length of each tick mark, in pixels. Default is 20.	value: numeric
color	Color for the major ticks. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	color

	stroke	Width of each tick mark, in pixels. Default is 1.	value: numeric

minorTicks	Settings for the minor tick marks on the scale.	object

Name	Description	Property Type
span	Distance between each tick mark of this type, in pixels. Default is 20.	value: numeric
length	Length of each tick mark, in pixels. Default is 20.	value: numeric
color	Color for the major ticks. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	color
stroke	Width of each tick mark, in pixels. Default is 1.	value: numeric

fineTicks	Settings for the fine tick marks on the scale.	object

Name	Description	Property Type
span	Distance between each tick mark of this type, in pixels. Default is 1.	value: numeric
length	Length of each tick mark, in pixels. Default is 5.	value: numeric
color	Color for the major ticks. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	color
stroke	Width of each tick mark, in pixels. Default is 1.	value: numeric

labels	Displays of the numeric values of major tick marks. Options are as follows:	object

Name	Description	Property Type
angle	Rotation of the numeric labels. Default is 0.	value: numeric
style	Sets a style for the label. 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

indicators	Markers of special significance that can be placed along the scale.	object

Name	Description	Property Type
value	Numeric value along the scale where the indicator is placed or started.	value: numeric
label	Text to display with the indicator. Default is "High".	value: string
labelColor	Color of the indicator label. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	color
labelAngle	Rotational angle of the indicator label.	value: numeric
color	Color of the indicator or the area making up the indicator. See Color Selector.	color
stroke	If indicatorStyle is set to line or wedge, stroke is the width (in pixels) of the indicator.	value: numeric
opacity	Opacity of the indicator. 0 is fully transparent, 1 is fully opaque.	value: numeric
length	Length of the indicator as measured by its x value within the scale. Default is 25.	value: numeric

	indicatorStyle	Indicator style can be set to line, wedge, or range. Line is similar to a tick mark. Wedge displays a triangular shape. Range displays a rectangular range along the scale as measured by the property extent. Default is range.	value: string dropdown	
	extent	If indicatorStyle is set to range, this is the extent along the scale that the indicator is placed. Default is 15.	value: numeric	
mirror	Aligns the scale to the opposite side of the component. Default is false.			value: boolean
reverse	Inverts the order of the scale values so min to max is ordered in reverse. Default is false (minimum to max).			value: boolean
style	Sets a style for this scale. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

Property	Value
props.maxValue	120
props.minorTicks.span	10
props.fineTicks.span	5
props.indicators.0.value	95
props.indicators.0.color	#D90000
props.indicators.0.extent	25
props.indicators.1.value	00
props.indicators.1.indicatorStyle	range
props.indicators.1.color	#D9D900

props.indicators.1.extent	20
props.indicators.2.value	74
props.indicators.2.indicatorStyle	wedge
props.indicators.2.color	#0000D9

Perspective - Map

General

Component Palette Icon:

INDUCTIVE UNIVERSIT

Map

Watch the Video

Description

The Map component provides a powerful, mobile-friendly, and interactive map. Settings can be customized to control the initial view, zoom, mouse interaction, keyboard interaction and more.

The Map component is based on the Leaflet open-source JavaScript library for interactive maps. For more information on Leaflet, see https://leafletjs.com/reference-1.6.0.html.

User Interaction

The Map component properties have impact on the way a user can interact with a table in the runtime.

Interaction	Description
Zoom	Depending on the property settings, users can zoom the Map component in several ways: • Shift and drag the mouse to a rectangular shape. • Double click to zoom in and Shift double-click to zoom out. • Roll the scroll wheel up to zoom in and down to zoom out. • Press the + (plus) key to zoom in and the - (minus) key to zoom out.
Pan	Depending on the property settings, users can pan across the Map component in multiple ways: • Use the keyboard arrow keys to pan left, right, up, down. • Use the mouse to click and drag the map.
Popups	Depending on property settings, users may see the following popup actions.

	• Popups close when they click on the map. • Popups close when they use the escape key. • Popups open as they scroll past certain areas of the map.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description
init	Map initial state when loaded. Options as follows: **Name** / **Description** / **Property Type** center — Sets the latitude and longitude for the initial state of the map. — object **Name** / **Description** / **Property Type** lat — Latitude value for the map. — value: numeric lng — Longitude value for the map. — value: numeric zoom — Initial map zoom level. Percentage value from 1 to 100. Default is 13. — value: numeric
location	Map location. **Name** / **Description** / **Property Type** enabled — Enables geolocation. Default is false (disabled). — value: boolean showHeadingIndicator — Enables the heading indicator. Default is true. — value: boolean
zoom	Zoom properties for the map. **Name** / **Description** controls — Whether zoom controls are added to the map. Default is true. delta — Controls how much the map's zoom level will change after a zoomIn() or zoomOut() or the user presses + or - on the k smaller than 1 allow for greater granularity. onBoxZoom — Enables the map to be zoomed to a rectangular area defined by pressing the shift key while dragging the mouse. Defa onDoubleClick — Enables the map to be zoomed in by double-clicking on it and zoomed out by pressing the shift key while double clicki onScrollWheel — Enables the map to be zoomed in and out using the mouse scroll wheel. Default is true. snap — Forces the map's zoom level to be a multiple of this value. Applicable after fitBounds() or a pinch-zoom. max — Maximum zoom level of the map. Default is null. min — Minimum zoom level of the map. Default is null. animation — Animation settings for the map. **Name** / **Description** / **Property Type**

enabled	Whether the map zoom animation is enabled.	value: boolean
threshold	Won't animate zoom if the zoom difference exceeds this value. Default is 4.	value: numeric

attribution	Enables an attribution control on the map. Default is true.
closePopupsOnClick	When set to true, popups will close when a user clicks anywhere else on the map. Default is true.
trackResize	Enables the map to automatically handle browser window resizing. Default is false.
keyboardNav	Enables navigation of the map with keyboard arrow key and with the + (plus) and - (minus) keys. The + key zooms in; the - key zooms ou
keyboardPanDelta	The number of pixels to pan when keyboard a arrow key is pressed. Default is 80.
dragging	Enables the map to be dragged with a mouse/touch. Default is true.
fadeAnimation	Enables the fade animation. Default is true.
layers	Settings for the map UI, vector, and raster layers that can be displayed on top of the map.

Name	Description
raster	Map raster layers.

Name	Description
tile	Settings enable you to load and display tile layers on the map. The tile provider can be customized to change See https://leaflet-extras.github.io/leaflet-providers/preview/ for some possibilities.

Name	Description
url	URL can be either a URL template if using a standard tile service or a Web Map Services U tps://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png.
urlTemplateParams	An object of params to use with a url template string, if the url prop is a template string (i.e, {y}.
WMS	Web Map Service tile layer provider configurations. Used only if the url property is not a tem See: https://en.wikipedia.org/wiki/Web_Map_Service.
options	Options for standard tile layer creation.

Name	Description	
layers	Comma-separated list of WMS layers to display. (Required)	
styles	Comma-separated list of WMS styles.	
format	WMS image format (use 'image/png' for layers with transparency).	
transparent	If enabled, the WMS service will return images with transparency. Default is false.	
version	Version of the WMS service to use.	
uppercase	If enabled, WMS request parameters keys will be uppercase. Default is false.	

Name	Description
attribution	This tile layers attribution.
opacity	Opacity of tiles.
zIndex	The z-index of tiles in the grid.
tileSize	Width and height of tiles in the grid.

Name	Description	Property Type
width	Width of tiles in the grid.	value: numeric
height	Height of tiles in the grid.	value: numeric

update — Tile update options.

Name	Description	Property Type
whenZooming	By default, a smooth animation will update grid layers at every integer zoom level. Setting this to false will update the grid layer only when the smooth animation ends. Default is true.	value: boolean
whenIdle	Load new tiles only when panning ends. True by default on mobile browsers, in order to avoid too many requests and keep smooth navigation. Default is false otherwise, in order to display new tiles during panning.	value: boolean
interval	Tiles will not update more than once every update interval in miliseconds when panning.	value: numeric

zoom — Zoom options.

Name	Description	Property Type
max	Maximum zoom level up to which this layer will be displayed (inclusive). Default is 18.	value: numeric
min	Minimum zoom level down to which this layer will be displed (inclusive). Default is 0.	value: numeric
offset	Zoom number used in the tile URLs will be offset with this value.	value numeric
reverse	If set to true, the zoom number used in the tile URLs will be reversed (maxZoom - zoom instead of zoom to maxZoom). Default is false.	value: boolean
native	Native zoom levels.	object

Name	Description	Property Type
max	Maximum zoom number the tile source has available. If specified, the tiles on all zoom levels higher than maxNativeZoom will be	Value: string

	loaded from maxNativeZoom level and auto-scaled. Default is null.	
min	Minimum zoom number the tile source has available. If specified, the tiles on all zoom levels lower than minNativeZoom will be loaded from minNativeZoom level and auto-scaled. Default is null.	Value: string

subdomains	Subdomains of the tile service. Passed in the form of an array of strings (where each string is a subdomain name). For example, ['a','b','c'].

Name	Description	Property Type
errorTileUrl	URL to the tile image to show in place of the tile that failed to load.	value: string
tms	If true, inverses Y axis numbering for tiles (turn this on for Tile Map Service services). Default is false.	value: boolean
detectRetina	If true and user is on a retina display, it will request four tiles of half the specified size and a bigger zoom level in place of one to utilize the high resolution. Default is false.	value: boolean
crossOrigin	Enables the crossOrigin attribute to be added to the tiles. If a string is provided, all tiles wil have their crossOrigin attribute set to the String provided. Default is false.	value: boolean

image	Settings for images displayed over specific bounds of the map.

Name	Description
source	Image url.
bounds	Rectangle bounds.

Name	Description	Property Type
corner1	Settings for lat and lng bounds for corner1.	object
corner2	Settings for lat and lng bounds for corner2.	object

Name	Description	Property Type
lat	Latitudinal coordinate	value: numeric
lng	Longitudinal coordinate	value: numeric

Name	Description	Property Type
lat	Latitudinal coordinate	value: numeric
lng	Longitudinal coordinate	value: numeric

options — Options for the image overlay.

Name	Description
opacity	The opacity of the image overlay.
alt	Text for the alt attribute of the image (useful for accessibility).
crossOrigin	Whether the crossOrigin attribute will be added to the image. If a string is provided, the image will have its crossOrigin attribute set to the String provided.
errorOverlayUrl	URL to the overlay image to show in place of the overlay that failed to load.
zIndex	The explicit zindex of the image layer.

vector — Map vector layers. You can set an array of the following layer shapes.

Name	Description
polygon	An array of polygon layers.

polygon — An array of polygon layers.

Name	Description
polygons	An array of polygons each consisting of an array of points that create a single polygon.
smoothFactor	How much to simplify each vector on each zoom level. Higher the number means better per smoother look, and the lower the number means more accurate representation.
noClip	Disables polyline clipping.
stroke	Stroke settings for the individual polygon layer.

polygons — An array of polygons each consisting of an array of points that create a single polygon.

Name	Description	Property Type
lat	Latitudinal coordinate	value: numeric
lng	Longitudinal coordinate	value: numeric

stroke — Stroke settings for the individual polygon layer.

Name	Description
enabled	Whether to draw stroke along the path. Set it to false to disable borders.
color	Stroke color.
weight	Stroke weight in pixels.
opacity	Stroke opacity (0-1).
dashArray	Stroke dash array
dashOff	Defines the distance in the dash patter to start the dash.

set	
lineCap	A string that defines shape to be used at the end of the stroke. Options are round, butt, or square.
lineJoin	A string that defines shape to be used at the corners of the stroke. Options are round, arc, bevel, miter, or miter-clip.

fill — Fill settings for the individual polygon layer.

Name	Description
enabled	Whether to fill the pattern with color.
color	Fill color.
opacity	Fill opacity (0-1).
rule	Presentation attribute defining the algorithm to use to determine the inside part of the shape. Options are nonzero or evenodd.

polyline — An array of polyline layers.

Name	Description
polylines	An array of polylines each consisting of an array of points that create a single polyline.
smoothFactor	How much to simplify each polyline on the zoom level.
noClip	Disables polyline clipping.
stroke	Stroke settings for the individual polyline layer.
fill	Fill settings for the individual polyline layer.

polylines — An array of polylines each consisting of an array of points that create a single polyline.

Name	Description	Property Type
lat	Latitudinal coordinate	value: numeric
lng	Longitudinal coordinate	value: numeric

stroke — Stroke settings for the individual polyline layer.

Name	Description
enabled	Whether to draw stroke along the path. Set it to false to disable borders.
color	Stroke color.
weight	Stroke weight.
opacity	Stroke opacity.
dashArray	Stroke dash array
dashOffset	Defines the distance in the dash patter to start the dash.
lineCap	Shape to be used at the end of the stroke. Options are round, butt, or square.
lineJoin	Shape to be used at the corners of the stroke. Options are round, arc, bevel, miter, or miter-clip.

fill — Fill settings for the individual polyline layer.

Name	Description

enabled	Whether to fill the pattern with color.
color	Fill color.
opacity	Fill opacity.
rule	Presentation attribute defining the algorithm to use to determine the inside part of the shape. Options are nonzero or evenodd.

rectangle — An array of rectangle layers.

Name	Description
rectangles	An array of rectangle bounds each consisting of two corners that create a single rectangle.
smoothFactor	How much to simplify each rectangle on the zoom level.
noClip	Disables rectangle clipping.
stroke	Stroke settings.

Name	Description
enabled	Whether to draw stroke along the path. Set it to false to disable borders.
color	Stroke color.
weight	Stroke width in pixels.
opacity	Stroke opacity (0-1).
dashArray	Stroke dash array.
dashOffset	Defines the distance in the dash patter to start the dash.
lineCap	A string that defines the shape to be used at the end of the stroke. Options are round, butt, or square.
lineJoin	A string that defines the shape to be used at the corners of the stroke. Options are round, arc, bevel, miter, or miter-clip.

fill — Fill settings for the individual rectangle layer.

Name	Description
enabled	Whether to fill the pattern with color.
color	Fill color.
opacity	Fill opacity (0-1).
rule	Presentation attribute defining the algorithm to use to determine the inside part of the shape. Options are nonzero or evenodd.

circle	An array of circle layers.

Name	Description
circles	An array of circles each consisting of center point corners that creates a single circle.
radius	Radius of the circle marker, in pixels.
stroke	Stroke settings for individual circle layer.

Name	Description
enabled	Whether to draw stroke along the path. Set it to false to disable borders.
color	Stroke color.
weight	Stroke width in pixels.
opacity	Stroke opacity.
dashArray	Stroke dash array
dashOffset	Defines the distance in the dash patter to start the dash.
lineCap	Shape to be used at the end of the stroke. Options are round, butt, or square.
lineJoin	Shape to be used at the corners of the stroke. Options are round, arcs, bevel, miter, or miter-clip.

Name	Description
fill	Fill settings for individual circle layer.

Name	Description
enabled	Whether to fill the pattern with color.
color	Fill color.
opacity	Fill opacity.
rule	Presentation attribute defining the algorithm to use to determine the inside part of the shape. Options are nonzero or evenodd.

ui	Map user interface (UI) layers. An array of popup layer configurations for this map. Includes popup location, width, height

Name	Description
marker	Map marker layers.

Name	Description
name	A unique name that is used to distinguish this marker from another.
lat	Latitudinal coordinate.
lng	Longitudinal coordinate.
opacity	Marker opacity.

icon	Marker icon configuration.

Name	Description
path	Icon path. Otherwise uses default icon.
color	Fill color for the icon.
size	Size settings.

Name	Description	Property Type
width	Width in pixels.	value: numeric
height	Height in pixels.	value: numeric

Name	Description
style	Style for the icon. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a style class.

tooltip	This marker's tooltip configuration, if tooltip is enabled.

Name	Description
content	The tooltip content to display.

Name	Description	Property Type
text	Text to display.	value: string.
view	The view to display as popup content. If configured, this overrides the text property.	object

Name	Description	Property Type
path	Path of view to display.	value: string
params	Parameters to be passed to the view. Names in this object must match input parameters defined on the view.	object

Name	Description
direction	Direction where to open the tooltip. Possible values are: right, left, top, bottom, center, auto. Auto will dynamically switch between right and left according to the tooltip position on the map.
permanent	Whether to open the tooltip permanently or on a mouseover.
sticky	If true, the tooltip will follow the mouse instead of being fixed at the feature center.
opacity	Tooltip opacity.

popup	Popup configuration for this marker.

Name	Description

enabled	Enable marker popup.
content	The popup content to display.

Name	Description	Property Type
text	Text to display.	value: string.
view	The view to display as popup content. If configured, this overrides the text property.	object

Name	Description	Property Type
path	Path of view to display.	value: string
params	Parameters to be passed to the view. Names in this object must match input parameters defined on the view.	object

width

Name	Description	Property Type
max	Maximum popup width in pixels.	value: numeric
min	Minimum popup width in pixels.	value: numeric

height	Maximum popup height in pixels.
pan	Popup pan configuration.

Name	Description	Property Type
auto	Set it to false if you don't want the map to do panning animation to fit the opened popup.	value: boolean

closeButton	Controls the presence of a close button in the popup.
autoClose	Set to false if you want to override the default behavior of the popup closing when another popup is opened.
closeOnEscapeKey	Set to false if you want to override the default behavior of the escape key for closing the popup.
closeOnClick	Set if you want to override the default behavior of the popup closing when user clicks on the map.

popup	Array of popup layer configurations for this map.

Name	Description
enabled	Show popup.
lat	Latitudinal coordinate.
lng	Longitudinal coordinate.

content	The popup content to display.

Name	Description
text	Text to display.
view	The view to display as popup content. If configured, this overrides the text property.

Name	Description	Property Type
path	Path of view to display.	value: string
params	Parameters to be passed to the view. Names in this object must match input parameters defined on the view.	object

width	Width settings for the popup.

Name	Description	Property Type
max	Maximum popup width in pixels.	value: numeric
min	Minimum popup width in pixels.	value: numeric

height	Maximum popup height in pixels.
pan	Popup pan configuration.

Name	Description
auto	Set it to false if you don't want the map to do panning animation to fit the opened popup.

closeButton	Controls the presence of a close button in the popup.
autoClose	Set to false if you want to override the default behavior of the popup closing when an opened.
closeOnEscapeKey	Set to false if you want to override the default behavior of the escape key for closing t
closeOnClick	Set if you want to override the default behavior of the popup closing when user clicks

view	Array of view layer configurations for this map.

Name	Description
path	Path of view to display.
params	Parameters to be passed to the view. Names in this object must match input defined on the view.
lat	Latitudinal coordinate.
lng	Longitudinal coordinate.
transparentBackground	If enabled, disables the default background color.
shadow	If enabled, displays a box shadow around the view.

		other	Other map layers.

Name	Description
geoJSON	GeoJSON objects to include as a feature layer. Hint: Try an HTTP binding. For more information, see http//

style	Sets a style for this component. Full menu of style options is available for text, background, margin and padding, border, shape and misce

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Map Parameters

Because this component uses the Leaflet library, there are several common objects you'll want to be aware of when using the the callable methods on this component. Each of these "objects" is just a Python dictionary with specific keys. However, we'll name and document these objects below.

An object representing a point at a certain latitude and longitude. For the purposes of interacting with the map component, this object is typically a Python dictionary.

```
# Example
{ lat:50, lng:30 }
```

Option	Type	Default	Description
`lat`	`Number`	`None`	Numerical value representing a latitude value.
`lng`	`Number`	`None`	Numerical value representing a longitude value.

A Python dictionary containing keys that can modify the panning behavior on the Map component. The contents of this dictionary are listed below, but the original explanation can be found in Leaflet's documentation.

Option	Type	Default	Description
`animate`	`Boolean`	`false`	If `true`, panning will always be animated if possible. If `false`, it will not animate panning, either resetting the map view if panning more than a screen away, or just setting a new offset for the map pane (except for `panBy` which always does the latter).
`duration`	`Number`	`0.25`	Duration of animated panning, in seconds.
`easeLinearity`	`Number`	`0.25`	The curvature factor of panning animation easing (third parameter of the Cubic Bezier curve). 1.0 means linear animation, and the smaller this number, the more bowed the curve.
`noMoveStart`	`Boolean`	`false`	If `true`, panning won't fire `movestart` event on start (used internally for panning inertia).

A Python dictionary containing keys that can modify the zooming behavior on the Map. The contents of this dictionary are listed below, but the original explanation can be found in Leaflet's documentation.

Option	Type	Default	Description
`animate`	`Boolean`	`false`	If `true`, panning will always be animated if possible. If `false`, it will not animate panning, either resetting the map view if panning more than a screen away, or just setting a new offset for the map pane (except for `panBy` which always does the latter).

A Python dictionary containing keys that can modify the zooming behavior on the Map. The contents of this dictionary are listed below, but the original explanation can be found in Leaflet's documentation.

Option	Type	Default	Description
`paddingTopLeft`	`Point`	`[0, 0]`	Sets the amount of padding in the top left corner of a map container that shouldn't be accounted for when setting the view to fit bounds. Useful if you have some control overlays on the map like a sidebar and you don't want them to obscure objects you're zooming to.
`paddingBottomRight`	`Point`	`[0, 0]`	The same for the bottom right corner of the map.
`padding`	`Point`	`[0, 0]`	Equivalent of setting both top left and bottom right padding to the same value.
`maxZoom`	`Number`	`null`	The maximum possible zoom to use.

Scripting Functions

- Description

  Returns the geographical center of the man in latitude and longitude.

- Parameters

  Nothing

- Return

  LatLng Returns the geographical center of the map view as`{ lat: number, lng: number }`.

- Scope

  Session

- Description

  Returns the current zoom level of the map view as a number.

- Parameters

  Nothing

- Return

  Numeric Returns the current zoom level of the map view as a number.

- Scope

  Session

- Description

  Returns the geographical bound of the map as a dictionary.

- Parameters

  Nothing

- Return

  Dictonary A dictionary containing the following keys:

```
north: number
northEast:
LatLngeast: number
southEast: LatLng
south: number
southWest: LatLng
west: number
northWest: LatLng
```

- Scope

  Session

- Description

Returns a string with bounding box coordinates in a '*South West longitude, South West latitude , North East longitude, North East latitude*' format.

- Parameters

  Nothing

- Return

  String Returns the bounding box of the map as a string.

- Scope

  Session

- Description

  Increases the zoom of the map by delta.

- Parameters

  Numeric delta - The numerical value to increase the zoom by. If omitted, uses the value of props.zoom.delta. [optional]

  Dictionary options - A dictionary of parameters to use during the zoom, typically containing a single key, animate. See Perspective - Map#ZoomOptions. [optional]

- Return

  Nothing

- Scope

  Session

- Description

  Decreases the zoom of the map by delta.

- Parameters

  Numeric delta - The numerical value to increase the zoom by. If omitted, uses the value of props.zoom.delta. [optional]

  Dictionary options - A dictionary of parameters to use during the zoom, typically containing a single key, animate. See Perspective - Map#ZoomOptions. [optional]

- Return

  Nothing

- Scope

  Session

- Description

  Zooms the map while keeping a specified geographical point on the map stationary (e.g. used internally for scroll zoom and double-click zoom)

- Parameters

  Dictionary point - The geographic point that the map will zoom around. See Perspective - Map#LatLng. [required]

  Numeric zoom- The numerical value to increase the zoom by. If omitted, uses the value of props.zoom.delta. [optional]

  Dictionary options - A dictionary of parameters to use during the zoom, typically containing a single key, animate. See Perspective - Map#ZoomOptions. [optional]

- Return

  Nothing

- Scope

  Session

- Description

  Sets a map view that contains the given geographical bounds with the maximum zoom level possible

- Parameters

Dictionary latLngBounds - A dictionary consisting of two LatLng objects. The LatLng objects combined represent the geographical bounds the map view should be set to. [required]

Dictionary options - A dictionary of parameters used to manipulate the FitBound settings. See Perspective - Map#FitBounds. [optional]

- Return

  Nothing

- Scope

  Session

- Description

  Sets a map view that mostly contains the whole world with the maximum zoom level possible.

- Parameters

  Dictionary options - A dictionary of parameters used to manipulate the FitBound settings. See Perspective - Map#FitBounds. [optional]

- Return

  Nothing

- Scope

  Session

- Description

  Pans the map to a given center.

- Parameters

  Dictionary latLng - The geographic point to pan to. [required]

  Dictionary options - A dictionary of parameters used to modify the panning behavior. See Perspective - Map#PanOptions. [optional]

- Return

  Nothing

- Scope

  Session

- Description

  Pans the map by a given number of pixels (animated).

- Parameters

  Dictionary point - The geographic point to pan to. The dictionary should contain an 'x' and 'y' key, both with numeric values. [required]

  Dictionary options - A dictionary of parameters used to modify the panning behavior. See Perspective - Map#PanOptions. [optional]

- Return

  Nothing

- Scope

  Session

- Description

  Sets the view of the map (geographical center and zoom) performing a smooth pan-zoom animation.

- Parameters

  Dictionary latLng - A Python Dictionary representing the coordinates to fly to. [required]

  Numeric zoom - Sets the zoom level to transition to during the flight. If omitted, uses the value on props.zoom.delta. [optional]

  Dictionary options - A dictionary of panning options to use. See Perspective - Map#PanOptions. [optional]

- Return

  Nothing

- Scope

  Session

- Description

  Sets the view of the map with a smooth animation like flyTo, but takes a bounds parameter like fitBounds.

- Parameters

  Dictionary latLngBounds - A dictionary consisting of two LatLng objects. The LatLng objects combined represent the geographical bounds the map view should be set to. [required]

  Dictionary options - A dictionary of panning options to use. See Perspective - Map#PanOptions. [optional]

- Return

  Nothing

- Scope

  Session

- Description

  Pans the map to the closest view that would lie inside the given bounds (if it's not already), controlling the animation using the options specific, if any.

- Parameters

  Dictionary latLngBounds - A dictionary consisting of two LatLng objects. The LatLng objects combined represent the geographical bounds the map view should be set to. [required]

  Dictionary options - A dictionary of panning options to use. See Perspective - Map#PanOptions. [optional]

- Return

  Nothing

- Scope

  Session

- Description

  Pans the map the minimum amount to make the latLng visible.

- Parameters

  Dictionary latLng - A Python dictionary representing the coordinates to pan to. [required]

  Dictionary options - A dictionary of panning options to use. See Perspective - Map#PanOptions. [optional]

- Return

  Nothing

- Scope

  Session

- Description

  Returns height and width of the Map component.

- Parameters

  Nothing

- Return

  JSON Object Returns a Python dictionary. Contains two items: `height` and `width` .

- Scope

  Session

Scripting Example

```
def doMapStuff(self):
        map = self.getSibling("Map")
        coordinateBounds = {'corner1': {'lat': 39.086798, 'lng': -120.069014}, 'corner2': { 'lat':
38.815319, 'lng': -119.787519 }}
        latLngTahoe = {'lat': 39.086798, 'lng': -120.069014 }
        latLngInductive = {'lat': 38.652511, 'lng': -121.189438 }
        zoomPanOptions = { 'animate': True, 'duration': 3, 'easeLinearity': 0.25, 'noMoveStart': False }
        fitBoundsOptions = { 'padding': { 'x': 100, 'y': 100 }, 'animate': True, 'duration': 3 }
        panPixels = { 'x': 200, 'y': 200 }


        print map.getCenter()
        # Returns the geographical bounds visible in the current map view as latLngBounds

        map.flyTo(latLng = latLngTahoe, options = zoomPanOptions)
        # Sets the view of the map with a smooth animation like flyTo, but takes a bounds parameter like
fitBounds
```

Component Methods

Interaction event. Fired when the a marker is clicked. Returns the unique name of the marker.

- Object Path

  event.name

- Type

  String

- Description

  The name of the marker.

Interaction event. Fired when the map is clicked. Returns the lat and lng of the mouse click as it translates on the map.

- Object Path

  event.lat

- Type

  Numeric

- Description

  The latitude of where on the map the user clicked.

- Object Path

  event.lng

- Type

  Numeric

- Description

  The longitude of where on the map the user clicked.

Interaction event. Fires as the mouse moves over the map. Returns lat and lng of mouse as it translates on the map.

- Object Path

  event.lat

- Type

  Numeric

- Description

  The latitude of where the mouse moved.

- Object Path

  event.lng

- Type

  Numeric

- Description

  The longitude of where the mouse moved.

Map state event. Fired when the map zoom is about to change (e.g. before zoom animation). Returns the zoom level.

- Object Path

  event.zoom

- Type

  Numeric

- Description

  What the zoom level on the map was *before* zooming.

Map state event. Fired repeatedly during any change in zoom level, included zoom and fly animations. Returns the zoom level.

- Object Path

  event.zoom

- Type

  Numeric

- Description

  What the zoom level was changed to.

Map state event. Fired when the map has changed, after any animations. Returns the zoom level.

- Object Path

  event.zoom

- Type

  Numeric

- Description

  What the zoom level on the map was *after* zooming.

Map state event. Fired when the view of the map starts changing (e.g., user starts dragging the map). Returns the map center as lat and lng.

- Object Path

  event.lat

- Type

  Numeric

- Description

  The latitude of the center of the map *before* moving.

- Object Path

  event.lng

- Type

  Numeric

- Description

  The longitude value for the center of the map *before* moving.

Map state event. Fires repeatedly during any movement on the map, include pan and fly animations. Returns the map center as lat and lng.

- Object Path

  event.lat

- Type

  Numeric

- Description

  The latitude value for the center of the map *during* moving.

- Object Path

  event.lng

- Type

  Numeric

- Description

  The longitude value for the center of the map *during* moving.

Map state event. Fired when the center of the map stops changing (e.g. user stopped dragging the map). Returns the new map center as lat and lng.

- Object Path

  event.lat

- Type

  Numeric

- Description

  The latitude value for the center of the map *after* moving.

- Object Path

  event.lng

- Type

  Numeric

- Description

  The longitude value for the center of the map *after* moving.

Map state event. Fires when the map size has changed. Returns the map size as oldSize and newSize.

- Object Path

  event.oldSize

- Type

  PyJsonObjectAdapter Returns an object that's functionally similar to a Python dictionary. Contains two two values: height and width.

- Description

  The starting size of the map before the resize.

- Object Path

  event.newSize

- Type

  PyJsonObjectAdapter Returns an object that's functionally similar to a Python dictionary. Contains two two values: height and width.

- Description

  The new size of the map.

Example

Property	Value
props.init.center.lat	38.65256
props.init.center.lng	-121.189028
props.init.center	18
props.zoom.controls	false
props.layers.ui.marker.0.lat	38.65256
props.layers.ui.marker.0.lng	-121.189028
props.layers.ui.marker.0.tooltip.enables	true
props.layers.ui.marker.0.tooltip.content	Welcome to Inductive Automation HQ

Perspective - Markdown

General

Component Palette Icon:

Description

The Markdown component allows users to format any type of text so it is publishable as plain text without looking like it's been marked up with tags or formatting instructions. The Markdown component provides is a lightweight formatting language which is easy to write and easy to read. Markdown's formatting syntax only addresses issues that can be conveyed in plain text. For any marked up content that is not covered by Markdown's syntax, you can use HTML. You can even change the color of text in the component with HTML tags using the 'source' prop in the Property Editor as shown in Example 2.

To learn more about how to use Markdown component for publishing plain text, refer to the following articles: Markdown Basics and Markdown Node Types.

Manually Adding Line Breaks

If you would like to manually add line breaks, the most direct approach would be to disable the escapeHTML property, and add your own paragraph and line break elements. Once disabled, you can add line breaks as seen below:

Value of "source"

```
<p>
one <br>
two <br>
three <br>
four
five
</p>
```

Because there isn't a line break between "four" and "five", they're on the same line.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
source	Text annotated with markdown syntax to display.	value: string
section Spacing	Number of pixels of vertical space between each section or header.	value: numeric
style	Sets a style for this component. 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
markdown	Lightweight formatting language that is easy to write and easy to read.	object

Name	Description	Property Type
sourcePos	If true, will keep track and log source positioning for debugging purposes. Default is false.	value: boolean
escapeHtml	Setting to false will cause HTML to be rendered. Be aware that setting this to false might cause security issues if the input is user-generated. Use at your own risk. Default is true.	value: boolean
skipHtml	Setting to true will skip inlined and blocks of HTML. Default is false.	value: boolean
disallowedTypes	Defines which types of types of nodes should be disallowed (not rendered).	array
unwrapDisallowed	Setting to true will try to extract/unwrap the children of disallowed nodes. For instance, if disallowing Strong, the default behavior is to simply skip the text within the strong altogether, while the behavior some might want is to simply have the text returned without the strong wrapping it. Default is false.	value: boolean
renderers	Renders in a browser as plain text.	object

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1

Property	Value
props.source	MARKDOWN source: Markdown **[basic syntax](https://www.markdownguide.org/basic-syntax/)** link. From <https://www.markdownguide.org>. Text formatting examples: # Heading 1 ## Heading 2 ### Heading 3 ** Bold text** *Italic text*

	Bold and Italic text Line of plain text Many other formatting options are listed online.

Example 2

The following code was used in the example below and pasted into the 'source' property of the Markdown component.

```
<p style="color:#AC00AC;"><B> Adding Color to your Text in the <BR>Markdown component </p
```

Perspective - Moving Analog Indicator

General

Component Palette Icon:

Description

The Moving Analog Indicator displays an analog value in context with other information about that value so that you can visually and quickly see if the value is in the normal range or not. The current value is shown as an arrow pointing at a bar with segments showing the desired operating range, low and high alarm ranges, and interlock ranges.

The Moving Analog Indicator component allows for extremely fast information delivery. At a glance, it is obvious to an operator whether or not the value is where it should be, or if it needs attention. If the value is in one of its alarm ranges, then that range can be set to change color to get attention.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
processValue	Current value of the process.	value: numeric
setpointValue	Current value of the setpoint.	value: numeric
minValue	The minimum value shown on the indicator. Default is 0.	value: numeric
maxValue	The maximum value shown on the indicator. Default is 100.	value: numeric
desiredHigh	The upper limit of the desired range. Default is 65.	value: numeric

desiredLow	The lower limit of the desired range. Default is 40.	value: numeric
highAlarm	Value above indicating a high alarm. Default is 90.	value: numeric
highHighAlarm	Value above indicating a high-high alarm. Default is null.	value: numeric
highInterlock	Value above when an interlock will be activated. Default is null.	value: numeric
lowAlarm	Value below indicating a low alarm. Default is 10.	value: numeric
lowLowAlarm	Value below indicating a low-low alarm. Default is null.	integer
lowInterlock	Value below when an interlock will be activated. Default is null.	integer
desiredRangeColor	Color for the area in the desired range. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	color
defaultRangeColor	Color for the area not defined as a range. See Color Selector.	color
inactiveAlarmColor	Color for the inactive alarm range. See Color Selector.	color
level2AlarmColor	Color for the active level 2 alarm (high or low). See Color Selector.	color
level1AlarmColor	Color for the active level 1 alarm (high-high or low-low). See Color Selector.	color
interlockColor	Color for the interlock range. See Color Selector.	color
indicatorColor	Color for the process indicator value. See Color Selector.	color
setpointColor	Color for setpoint value marker. See Color Selector.	color
label	Numeric value displayed as text next to the indicator. **Name** / **Description** / **Property Type** visible — Whether to display the label. Default is false. — value: boolen format — Format of numeric value in label, including commas, decimal places, etc. Options as follows: • #,##0 • #,##0.0 • #,##0.00 • 0 • 0.0 • 0.00 • #,##0% — value: string dropdown style — Sets a style for the label property. 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	object
selectOutline	Border settings for the outline surrounding each range area. Options as follows: **Name** / **Description** / **Property Type** color — Sets the color for the outline surrounding the range borders. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector. — color	object

	width	Sets the width of the outline surrounding the range borders in pixels.	value: numeric
reverseI ndicator	Displays the process value indicator on the opposite side of the scale. Default is false.		value: boolean
style	Sets a style for this component. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1

The alignment of the component is based on the height and width of the component.

Height > Width

Width > Height

Example 2

Property	Value
props.processValue	96
props.highHighAlarm	96
props.reverseIndicator	true
props.label.visible	true

Perspective - PDF Viewer

General

Component Palette Icon:

INDUCTIVE UNIVERSITY

PDF Viewer

Watch the Video

Description

The PDF Viewer component displays a PDF that's hosted on a web server by providing a URL to the source property. A simple approach is to create either a File Resource or Mounted Folder within Web Dev, and set the source on the component to the resource's endpoint.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
source	Path to the .pdf file to be displayed. Expects a URL to a PDF hosted on a web server.	value: string
page	The current page being displayed.	value: numeric
pageCount	The number of pages the pdf contains. Read only.	value: numeric
showPage Number	If true, the current page number and page count will be shown at the bottom of the component.	value: bool ean
style	Sets a style for this component. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Scripting Functions

- Description

  This function will reload the PDF in the PDF Viewer component.

- Parameters

  string name - The name of the PDF.

- Return

  Nothing

- Scope

  Session

Perspective - Progress

General

Component Palette Icon:

Description

The Progress bar visually indicates the progress of a task. It is used to display any value that has an upper and lower bound. Custom settings are available for the track and the bar.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
min	The minimum value of the progress indicator. If the value reaches the max, the progress indicator will be completely filled. Must be less than the max. Default is 0.	value: numeric
max	The maximum value of the progress indicator. If the value reaches the max, the progress indicator will be completely filled. Must be greater than 0.0. Default is 100.	value: numeric
value	The current value representing the current progress. Must be greater than 0.0 and less than the value set in max. Default is 50.	value: numeric
mode	Determines if the component should show a determinate state, or an indeterminate loading state. When set to determinate, shows the progress of the value relative to the min and max properties.	value: string
bar	Settings for the bar.	object

Name	Description	Property Type
color	A convenience property for setting the base color of the bar. This can also be accomplished by using the prop.bar.style to set the background color.	object
style	Sets a style for the bar. Full menu of style options is available. You can also specify a style class.	object

Name	Description	Property Name
color	A convenience property for setting the base color of the bar. This can also be accomplished by using the prop.bar.style to set the background color.	object
style	Sets a style for the bar. Full menu of style options is available. You can also specify a style class.	object
determi	Determinate bar configuration.	

			nate				
				Name	**Description**	**Propert Type**	
				color	A convenience property for setting the color of the bar when mode is determinate. This can also be accomplished by using the prop.bar.determinate.style to set the background color.		
				style	Sets a style for the bar when mode is determinate. Full menu of style options is available. You can also specify a style class.	object	
			indeterminate	Indeterminate bar configuration.			
				Name	**Description**	**Property Type**	
				color	A convenience property for setting the color of the bar when mode is determinate. This can also be accomplished by using the prop.bar.determinate.style to set the background color.		
				style	Sets a style for the bar when mode is determinate. Full menu of style options is available. You can also specify a style class.	object	
track	Settings for the track.						object
	Name	**Description**				**Property Type**	
	color	A convenience property for setting the base color of the track. This can also be accomplished by using the prop.track.style to set the background color.				object	
	style	Sets a style for the track. Full menu of style options is available. You can also specify a style class.				object	
	determinate	Determinate track configuration.					
		Name	**Description**			**Property Table**	
		color	A convenience property for setting the color of the track when mode is determinate. This can also be accomplished by using the prop.bar.determinate.style to set the background color.				
		style	Sets a style for the track when mode is determinate. Full menu of style options is available. You can also specify a style class.			object	
	indeterminate	Indeterminate track configuration.					
		Name	**Description**			**Property Type**	
		color	A convenience property for setting the color of the track when mode is indeterminate. This can also be accomplished by using the prop.track.indeterminate.style to set the background color.				
		style	Sets a style for the track when mode is indeterminate. Full menu of style options is available.			object	
valueDisplay	Value display configuration. Renders and styles a value overlay above the progress bar.						object
	Name	**Description**				**Property Type**	

	enabled	If true, will show the value display.	value: boolean
	format	Format to apply to value, which is then used in the value display.	value: string
	justify	Horizontal alignment of the displayed value.	value: string
	style	Sets a style for the track. Full menu of style options is available. You can also specify a style class.	object
style	Sets a style for this component. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1

Property	Value
max	100
value	80

Example 2

Property	Value
max	100
value	56
mode	determinate
bar.color	#FFAC47
track.color	#FFE8CC
track.borderStyle	solid
track.borderWidth	2
track.borderBottomLeftRadius	15
track.borderBottomRightRadius	15
track.borderTopRightRadius	15
track.borderTopLeftRadius	15
track.borderColor	#A45324
valueDisplay.enabled	true
valueDisplay.format	percent

valueDisplay.justify	center
valueDisplay.fontFamily	Merriweather
valueDisplay.fontWeight	bold

Perspective - Sparkline

General

Component Palette Icon:

Description

The Sparkline chart is a minimalistic chart component that displays a line-chart history for a single datapoint. Sparklines were invented by Edward Tufte as a way to show a great deal of contextual information in a very small amount of space. Sparklines are typically used to display the recent history (up to current time) of a datapoint so that the viewer can quickly discern the recent trend of a datapoint: is it rising? falling? oscillating? etc..

To use a sparkline, bind its Data property either to a Tag Historian realtime query, or to a database query. There should be two columns in this dataset: the first one a date column, the second a number. Each row will become a datapoint on the chart, and the dataset must be sorted by time in ascending order.

Instead of using axes to convey scale, the Sparkline can display a band of color across the back of the chart which indicates the desired operating range of the datapoint. In this way, it is instantly obvious when a value is in its expected range, above that range, or below. The sparkline automatically configures its internal axes based on the data given to it. To display a desired range, fill in the **props.desired.high** and **props.desired.low** properties.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
points	Data points to plot. Accepts the following: May be a dataset, or an array of values or of objects containing X or Y coordinates. Also may be a string formatted with X and Y values separated by a comma. • An array of numbers containing X or Y coordinates. • A string of space delimited points where x and y are separated by a comma, i.e. '0,20 1,35 2,15' • An array of objects, where each object contains an x and a y property, and where each property's value is a number. • A dataset of a single column of number type. • A dataset of two columns, the first representing the x value and the second column representing the y value. The first column can be either of type number or type date. Dates and Timestamps are converted to unix timestamps which is used as the x value.	array or dataset
color	Color of the line. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	color
width	Thickness of the line, in pixels.	value: numeric
opacity	The opacity of the line ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque.	value:

		numeric
dashArray	The pattern of dashes and gaps used to paint the stroke. It's a list of comma separated lengths (in pixels) and percentages (percentage of the total stroke length) that specify the lengths of alternating dashes and gaps. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. Thus, "5,3,2,5,3,2" is equivalent to "5,3,2,5,3,2".	array
range	Settings for the upper and lower edge of the chart. **Name** / **Description** / **Property Type** high — A fixed value for the upper edge of the chart as a number. — value: numeric low — A fixed value for the lower edge of the chart as a number. — value: numeric	object
desired	The desired operating range. Settings for the desired properties operating range. **Name** / **Description** / **Property Type** high — The high value of the desired operating range. — value: numeric low — The low value of the desired operating range. — value: numeric stroke — Settings for the stroke. Options as follows: — object **Name** / **Description** / **Property Type** color — Color of the line. See Color Selector. — color width — Width of the line in pixels. — value: numeric opacity — Opacity of the line ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque. — value: numeric dashArray — The pattern of dashes and gaps used to paint the stroke. It's a list of space separated lengths (in pixels) and percentages (percentage of the stroke length for the desired operating range) that specify the lengths of alternating dashes and gaps. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. Thus, "5 3 2" is equivalent to "5 3 2 5 3 2". — array fill — Settings for the fill for the desired range. Options as follows: — object **Name** / **Description** / **Property Type** color — Fill color of the desired range. See Color Selector. — color opacity — Opacity of the fill ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque. — value: numeric	
marker	Settings for the first, last, high, and low markers on the chart. **Name** / **Description** / **Property Type** first — Settings for the first marker on the chart. — object **Name** / **Description** / **Property Type** shape — Shape of the marker used to indicate the first point. Options are circle, triangle, or square. Default is circle. — value: string dropdown size — Size of the marker, in pixels. — value: numeric	object

stroke	Settings for the stroke for the first marker. Options as follows: **Name** / **Description** / **Property Type** color / Color of the line. See Color Selector. / color width / Width of the line in pixels. . / value: numeric opacity / Opacity of the line ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque. / value: numeric dashArray / The pattern of dashes and gaps used to paint the stroke. It's a list of space separated lengths (in pixels) and percentages (percentage of the stroke length for the desired operating range) that specify the lengths of alternating dashes and gaps. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. Thus, "5 3 2" is equivalent to "5 3 2 5 3 2". / array	object
fill	Settings for the fill for the first marker. Options as follows: **Name** / **Description** / **Property Type** color / Color of the line. See Color Selector. / color opacity / Cursor line opacity. 0 is fully transparent, 1 is fully opaque. / value: numeric	object
style	Sets a style for this component. 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

last	Settings for the last marker on the chart. **Name** / **Description** / **Property Type** shape / Shape of the last marker. Options are circle, triangle, or square. Default is circle. / value: string dropdown size / Size of the last marker, in pixels. / value: numeric stroke / Settings for the stroke for the last marker. Options as follows: **Name** / **Description** / **Property Type** color / Color of the line. See Color Selector. / color width / Width of the line in pixels. / value: numeric opacity / Opacity of the line ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque. / value: numeric dashArray / The pattern of dashes and gaps used to paint the stroke. It's a list of space separated lengths (in	object

	pixels) and percentages (percentage of the stroke length for the desired operating range) that specify the lengths of alternating dashes and gaps. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. Thus, "5 3 2" is equivalent to "5 3 2 5 3 2".	
fill	Settings for the fill for the last marker. Options as follows: **Name** / **Description** / **Property Type** color — Color of the line. See Color Selector. — color opacity — Opacity of the line ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque. — value: numeric	object
style	Sets a style for this component. 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

high	Settings for the high marker on the chart.	object

Name	Description	Property Type
shape	Shape of the high marker. Options are circle, triangle, or square. Default is square.	value: string dropdown
size	Size of the high marker, in pixels.	value: numeric
stroke	Settings for the stroke for the high marker. Options as follows:	object

Name	Description	Property Type
color	Color of the line. See Color Selector.	color
width	Width of the line in pixels.	value: numeric
opacity	Opacity of the line ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque.	value: numeric
dashArray	The pattern of dashes and gaps used to paint the stroke. It's a list of comma separated lengths (in pixels) and percentages (percentage of the stroke length for the desired operating range) that specify the lengths of alternating dashes and gaps. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. Thus, "5,3,2" is equivalent to "5,3,2,5,3,2".	array

fill	Settings for the fill for the high marker. Options as follows:	object

Name	Description	Property Type

color	Color of the line. See Color Selector.	color
opacity	Opacity of the line ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque.	value: numeric

style	Sets a style for this component. 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

low	Settings for the low marker on the chart.	object

Name	Description	Property Type
shape	Shape of the low marker. Options are circle, triangle, or square. Default is square.	value: string dropdown
size	Size of the marker, in pixels.	value: numeric
stroke	Settings for the stroke for the low marker. Options as follows:	object

Name	Description	Property Type
color	Color of the line. See Color Selector.	color
width	Width of the line in pixels.	value: numeric
opacity	Opacity of the line ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque.	value: numeric
dashArray	The pattern of dashes and gaps used to paint the stroke. It's a list of space separated lengths (in pixels) and percentages (percentage of the stroke length for the desired operating range) that specify the lengths of alternating dashes and gaps. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. Thus, "5 3 2" is equivalent to "5 3 2 5 3 2".	array

fill	Settings for the fill for the low marker. Options as follows:	object

Name	Description	Property Type
color	Color of the line. See Color Selector.	color
opacity	Opacity of the line ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque.	value: numeric

style	Sets a style for this component. 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

style	Sets a style for this component. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

This example demonstrates what changes to the various marker properties can do. In addition, the desired range has been enabled.

Property	Value
props.width	1
props.dashArray	6 2 2 2
props.desired.high	60
props.desired.low	25
props.desired.stroke.color	#AAAAAA
props.desired.stroke.width	3
props.desired.stroke.opacity	3
props.desired.fill.color	#CCFFF
props.desired.fill.opacity	4
props.marker.first.shape	circle
props.marker.first.size	10
props.marker.first.stroke.color	#000000
props.marker.first.fill.color	#00AC00
props.marker.last.shape	triangle
props.marker.last.size	20
props.marker.last.stroke.color	#000000
props.marker.last.fill.color	#EF4D4D
props.marker.low.shape	square
props.marker.low.size	7
props.marker.low.stroke.color	#000000
props.marker.low.fill.color	#0D9D9
props.marker.high.shape	square
props.marker.high.size	7

props.marker.high.stroke.color	#000000
props.marker.high.fill.color	#FF47FF
props.style.backgroundColor	#E2F1Fa
props.style.borderStyle	outset
props.style.borderWidth	7px
props.style.borderColor	#9BBFBF

Perspective - Table

General

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

1 2

Component Palette Icon:

INDUCTIVE UNIVERSIT

Table

Watch the Video

Description

The Table component provides a powerful and flexible means of displaying your database data in tabular form. Properties enable you to customize the data content, style, navigation, and user interaction of your table.

User Interaction

The Table component properties have impact on the way a user can interact with a table in the runtime.

Interaction	Description
Column Resizing	When configured through the designer via the corresponding column config, a column can be resized during runtime. The resize handle exists in a 36px swath centered on the end of the header cell. Hovering over this area will change the mouse cursor to column resizing. Dragging the resize handle will display a resize guide effectively providing a visual for the new column position as the user drags. These changes in width will not persist, and are merely for the convenience of the user.
Sorting	When sorting is enabled on a column and the table head is enabled, a sort indicator will display to the right of the header cell content. The sort indicator will display the sort direction. • Single Sort - Enabled by double clicking on a header cell. • Multi Sort - Enabled by holding down Shift then double-clicking on multiple header cells.
Selection	When selection is enabled, a user may select table data based upon the table's selection configuration. In the browser, selection is indicated by a light blue overlay rendered on cells. The root selection, or most recently selected cell has a light blue border. The root selection corresponds with the selected column and selected row propertiess of the table component's selection configuration. • Single - Single mouse click enabled. • Single Interval - Shift and single mouse click enabled. • Multiple Interval - Command/Ctrl + shift and single mouse click enabled.
Editing	

	When editing is enabled on an individual cell, a user can edit a cell by performing the interaction specified by the 'allow edit on' property of the table component. When in edit mode, an editing cell with the corresponding cells content will be presented for edit. To commit this edit, the user must press the return or enter key. To exit the edit, the user may either press the escape key or select another table cell. When an edit is committed, the edited data is sent to the cell edit extension function of the table component
Paging	When paging is enabled, a user may use the provided buttons to navigate between available pages and also jump to a specific page within range.
Filtering	When filtering is enabled, a user may filter all of the data, not just the data being displayed when pagers are enabled, of the table component. If paging also happens to be enabled, the table will automatically page jump if it becomes necessary so that it does not display an empty page.
Freeze a Column	If the table head property is enabled, a user can freeze a column by holding down Alt and double clicking the column header. This action "freezes" the column within the bounds of the table so that the user may scroll to perform data comparisons. To unfreeze a column, hold down Alt and double click on the column header of the frozen column or of the source column. A frozen column can be dragged horizontally within the bounds of the table by selecting and dragging with the mouse. It is possible to freeze as many columns as a user may like. The user is not confined to freezing columns that are only visible when at scroll start position.
Freeze a Row	A user may freeze an individual row by holding down 'alt' and double clicking on the desired row within the table body. This will fix the table row within the bounds of the table. To unfreeze, perform the same operation on either the frozen row or the source row. A frozen row can be dragged vertically within the bounds of the table by selecting and dragging with the mouse. The user is not confined to freezing rows that are only visible when at scroll start position.
Coloring /Look	The table is made up of various subareas (rows, cells, etc). To aid with styling the component, these subareas have dedicated style objects that can be used to change the look. Furthermore, some parts of the table's property model allow for more fine tuned control of the look. For example, changing the color of all the rows on table can be accomplished by setting a background color on the `rows.style` object. However, if you wanted to alternate colors on each row, you could instead look towards the `rows.striped.color` object, which allows you to pick colors for even and odd rows separately.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type

data	Can be a dataset, an array of arrays, or an array of objects. The preferred (recommended) data is either a dataset or an array of objects. Individual data items can be a string, a number, or an object with reserved keys.
	Object data items must have a **value** property. Optionally they can also have properties to indicate the style for the object and whether it is editable.
	
	`city: {`
	`  value: 'Folsom',`
	`  editable: true,`
	`  style: {`
	`  backgroundColor: 'grey',`
	`  classes: [] | ''`
	`  }`
	`}`
		array

virtualiz		value:

ed	Enables virtualization of table rows, which is an optimization method that only shows a portion of the underlying data on the chart at a time. While enabled, the table will only be populated with a smaller subset of data: just the visible rows, and a few rows above and below. The idea being the component will be populated with new records as the user scrolls down the listing, assuming there are enough records to necessitate a scrollbar. Enabling virtualization generally results in a performance gain in the session, in cases where the data property is populated with a large amount of content, as the table will only have to "load" a small subset of content. The trade off is that the table will need to load records as the user scrolls, so scrolling quickly may not feel as "smooth" when compared to disabling virtualization.	boolean
selection	When Selection is configured, a user will be able to select table data based upon the table's selection configuration. Si milar to Vision module, you can select single, single interval, and multiple interval selection modes. The current selection and selection data is written back to the table components property tree. With the exception of the selection data property, the selection properties are bidirectional, meaning that if you were to change the value of the selected column property, it should be reflected in the table component. **Name** / **Description** / **Property Type** mode — This option determines if only one row, cell, or column can be selected at once. Options are single, single interval, and multiple interval. — value: string enableRowSelection — This option is used in conjunction with the Column Selection allowed flag in order to determine whether whole rows, whole columns, or both (single cells) are selectable. Can be set to true or false. Default is true. — value: boolean enableColumnSelection — This option is used in conjunction with the Row Selection allowed flag in order to determine whether whole rows, whole columns, or both (single cells) are selectable . Can be set to true or false. Default is false. — value: boolean selectedColumn — The index of the first selected column, or null if none. — value: numeric selectedRow — The index of the first selected row, or null if none. — value: numeric data — An array of objects representing the current selection. — array	object
filter	Where Table filtering is configured, as well as the filtered data. **Name** / **Description** / **Property Type** enabled — Enables filtering. Default is false. — value: boolean text — Contains the text you want to filter on. — value: string results — The filtered data. — object **Name** / **Description** / **Property Type** enabled — Enables the filter results to be written back to the props. Doing so may cause performance decline. Default is false. — value: boolean data — An array of objects representing the current filtered data if filtering is enabled. Each object represents a row of the table. — array	object
enableHeader	When enabled, the table header is displayed including the main table header along with the Header Groups. Default is true.	value: boolean
enableFooter	When selected, this enables the table footer, including the main table footer along with the Footer Groups. Default is false.	value: boolean
enableHeader	Enable table header groups if available. Default is false.	value: boolean

Groups		
enableFooterGroups	Enable table footer groups if available. Default is false.	value: boolean
headerGroups	Header Groups are additional headers that are displayed above the main table header. Each header group equates to a single row with individual cells containing title text. **Name** / **Description** / **Property Type** title — Text displayed in the header. — value: string span — You can use the span property to instruct individual cells to span multiple columns of the table. However, header group cells cannot span more than the available columns. I f you specify more columns in the span property than are actually available in the table, the cell will only span the remaining space. — value: numeric justify — Justify content horizontally. Options are left, right, and center. — value: string align — Aligns the content vertically. Options are top, center, or bottom. — value: string style — Sets a style that applies to header. 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	object
footerGroups	Footer Groups are additional footers that display above the main table footer. Each footer group equates to a single row which consists of individual cells containing title text. **Name** / **Description** / **Property Type** title — Text displayed in the footer group. — value: string span — You can use the **span** property to instruct individual cells to span multiple columns of the table. However, footer group cells cannot span more than the available columns. If you specify more columns in the span property than are actually available in the table, the cell will only span the remaining space. — value: numeric justify — Justify content horizontally. Options are left, right, and center. — value: string align — Aligns the content vertically. Options are top, center, or bottom. — value: string style — Sets a style that applies to footer. 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	object
columns	The Columns property allows for granular column-by-column configurations, indicating how each column should be displayed in the table. When a Column Config option is present, the table reflects that custom configuration. When empty, the table component displays all available columns. Column configs enable the you to customize the table component's display and how users will be able to interact with the table in the runtime. A single cell of a Table component can be a whole view instead of just a simple value (like a string). **Name** / **Description** / **Property Type** field — A string that matches this column config with a table column. This string must correspond to the default column name of the column. — value: string visible — Toggles column visibility. Allows table columns to be invisible to users, but data will be available to view params and selection. — value: boolean	array

editable	Enables editing of all cells within this column. This can be overridden if the Editable property is set to false on an individual cell.	value: boolean
render	The default render setting is auto. Can be auto, number, date, boolean, string, or view.	value: string
justify	Sets the justification for the content of the column. Options are left, center, right, or auto. The default setting is auto.	value: string dropdown
align	Sets the alignment for the content of the column. Options are top, center, or bottom. The default alignment is center.	value: string
resizable	Enables columns to be resized. When enabled, users can resize columns in the runtime by hovering over the edge of the column header then dragging the cursor.	value: boolean
sortable	Enables the column to be sorted. When enabled, users can double click on the column header in the run time to sort by ascending or descending order.	value: boolean
viewPath	When render mode is set to View, the table will display the view found at the view path within each cell of this column.	value: string
viewParams	Parameters to feed the configured view. Will be added to implicit parameters as follows: {row:number;rowIndex:number;value:PlainObject;...viewParams}	object
boolean	When render mode is set to Boolean, you can then specify how the boolean is represented in the runtime, for example, as a checkbox, toggle switch, value, and so forth. See Toggle Switch below.	value: string
number	Type of component to render for boolean. Options are number or progress. When render mode is set to Number, you can then specify whether the number is represented in the runtime as value or as progress.	value: string
progressBar	A progress bar configuration that is used when Number property is set to progress bar. You can specify the maximum value of the progress bar, as well as configure the following:	object

Name	Description	Property Type
max	Progress bar maximum value.	value: numeric
bar	Settings for the bar.	object

Name	Description	Property Type
color	Color of the bar and the track. See Color Selector.	color
style	Sets a style for the progress bar. 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

Name	Description	Property Type
track	Settings for the track.	object

Name	Description	Property Type
color	Color of the bar and the track. See Color Selector.	color
style	Sets a style for the progress bar. 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

	value	Settings for the value on the Progress Bar. **Name** / **Description** / **Property Type** enabled — Whether or not to show the value. — value: boolean format — Format to apply to the value. — value: string justify — Horizontal alignment of the value. — value: string style — Sets a style for the value. 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	object	
toggleSwitch	Toggle switch configuration used when boolean is set to display as a toggle switch. Can specify selected and unselected color. **Name** / **Description** / **Property Type** color.selected — Color when the toggle switch is selected. See Color Selector. — color color.unselected — Color when the toggle switch is not selected. See Color Selector. — color		value: boolean
numberFormat	A number format string when render mode is set to number. Options are none, number [1,000.12], integer [1,200], four decimal precision [1.1200], percent [10.12%], scientific [1.01E+03], accounting [$(1,000.12)], financial [(1,000.12)], currency [$1,000.12], currency (rounded) [$1,012], duration [24:01:00], abbreviation [1.2k], or ordinal [100th].		value: string
dateFormat	Date format string used when render mode is set to date. Options are none, date [10/15/1018], time [3:59:00 PM], or date time [10/15/2018 15:59:00]		value: string
width	The width of this column. If resize is enabled, specifies the column width on initial load. User can override this in the runtime if the Resizable option is enabled.		value: numeric
strictWidth	If enabled, the width of the column becomes fixed.		value: boolean
style	Sets a style for this individual column. 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
header	Header cell configuration. **Name** / **Description** / **Property Type** title — Text for title of the column. — value: string justify — Setting for justification of the title. Options are right, left, and center. — value: string dropdown align — Setting for alignment of the title. Options are top, center, and bottom. — value: string dropdown style — Sets a style for this header. 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		value: object

footer	Footer cell configuration. **Name** / **Description** / **Property Type** title — Text for title of the footer. — value: string justify — Setting for justification of the title. Options are right, left, and center. — value: string dropdown align — Setting for alignment of the title. Options are top, center, and bottom. — value: string dropdown style — Sets a style for this footer. 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	value: string
Implicit Properties		
rowData	Used to map parameters on a view cell to an entire row of data. The view must have a rowData object input parameter, with sub values that match the names of the columns. Then add the new view to the props.columns.0.viewPath property, and the input parameter as the props.columns.0.field property.	value: string

sortOrder	The default weighted order in which columns and their contents are sorted relative to other columns and their contents. Used when the component loads.	array
rows	Configures all rows in the table component. Includes settings for expanding rows into subviews.	object

Name	Description	Property Type
subviewExpansionMode	Specifies how many subviews can be expanded at any given time. Options are multiple or single. Default is multiple.	value: string
subview	When enabled, each table row can be expanded into a subview. The Expandable Arrow opens the subview. Content of the subview is determined by the View Path property. **Name** / **Description** / **Property Type** enabled — Enable each row to allow toggling of the specified view. — value: boolean viewPath — A viewpath used to display a view as an expanded row — value: string viewParams — Parameters to feed the configured view. Will be added to implicit parameters as follows: {row:number;rowIndex: number;value:PlainObject;...viewParams} — object	object
striped	Settings for setting the striping (alternating background color) to the rows of the table. **Name** / **Description** / **Property Type** enabled — When enabled (true), the table will be displayed with alternating background color to the rows of the table. — value: boolean color — Color settings — object **Name** / **Description** / **Property Type** even — Background color for the even rows. See Color — color	object

	Selector.	
odd	Background color for the odd rows. See Color Selector.	color

highlight	Highlight settings. **Name** / **Description** / **Property Type** enabled — When enabled (true), this feature will highlight the row that is currently selected or moused over. — value: boolean color — Highlight color for the row. See Color Selector. — color	object
style	Sets a style that applies to every row in the table. 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

cells — object

Configures all cells in the table component.

Name	Description	Property Type
allowEditOn	Enables the table cells to be edited on a single click, double click, or long press.	value: string
style	Sets a style that applies to every cell in the table. 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

pager

Enables table pagination. Pagination improves performance and appearance on large tables, over 1000 rows.

Name	Description	Property Type
options	Rows to show per pager option.	array
initialOption	Initial option to use when the table first loads. Must exist as an available option.	value: numeric
top	Enables top pager. The pager is a menu that displays the current page and Previous < and Next > icons for navigation.	value: boolean
bottom	Enables bottom pager. The pager is a menu that displays the current page and Previous < and Next > icons for navigation.	value: boolean
activePage	Represents the current active page and corresponds to the value of the page jump input field.	value: numeric

resizeMode — value: boolean

Specifies whether the table resize mode is either Fill or Fixed. In Fill resized mode, the total width of all the columns cannot be less than the width of the table. In Fixed resized mode, the total width of all the columns can be less than the width of the table.

style — object

Sets a style that applies to the component. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a style class.

emptyMessage

Empty message configuration.

noData — object

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

Empty message configuration for when there is either no data source or the data source is empty.

Name	Description	Property Type

text	Text to display when there is no data source or the data source is empty.	value: string
textStyle	Sets a style that applies to the text. Full menu of style options is available. You can also specify a style class.	object
icon	Settings for the icon to be displayed when there is no data source or the data source is empty. path: Shorthand path to the icon source, in format: library /iconName. (value: string) color: Color of the icon. Alternatively, you can use fill settings in the style property. (value: string) style: Sets a style that applies to the icon. Full menu of style options is available. You can also specify a style class. (object)	object
bottom	Enables bottom pager. The pager is a menu that displays the current page and Previous < and Next > icons for navigation.	value: boolean
activePage	Represents the current active page and corresponds to the value of the page jump input field.	value: numeric

noFilterResults (object)

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

Empty message configuration for when a filter returns no results.

Name	Description	Property Type
text	Text to display when a filter returns no results.	value: string
textStyle	Sets a style that applies to the text. Full menu of style options is available. You can also specify a style class.	object
icon	Settings for the icon to be displayed when a filter returns no results. path: Shorthand path to the icon source, in format: library /iconName. (value: string) color: Color of the icon. Alternatively, you can use fill settings in the style property. (value: string) style: Sets a style that applies to the icon. Full menu of style options is available. You can also specify a style class. (object)	object

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Component Events

This onEditCellCommit event is used with a runAction script on a table to take user entry and store it in the table or a database.

Provides a chance do something once a user has typed something into a cell. The user must commit the new value (generally achieved by pressing the Enter/Return key) before the event will trigger.

Additionally, the cell must first be editable (*props.data.[rowNumber].[columnName].editable* is set to true). The first cell in the default dataset on a newly created instance of the component demonstrates where the *editable* property must be positioned.

Note: This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

- Object Path

  event.column

- Type

  String

- Description

  The name of the column under which the cell was edited.

- Object Path

  event.row

- Type

  Number

- Description

  The row index number (zero based) of the cell that was edited.

- Object Path

  event.value

- Type

  Any

- Description

  The value that was typed into the cell.

Example - Change the value in a cell

```
    # This example will set the value of a cell, based on what the user typed into it.

        # Get the value that was typed into the cell
        valueToSet = event.value

        # We need to set a value in a particular cell. The event object contains row and column
properties
        #  that report the position of the cell that was edited.
        self.props.data[event.row][event.column] = valueToSet
```

This onSelectionChange event will trigger when the selection in the chart changes.

Note:

The onSelectionChange event will fire on startup or mount if props do not equal the table components default selection config.

This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

- Object Path

  event.selectedColumn

- Type

String or null

- Description

  The name of the column that the selected cell is located under.

- Object Path

  event.selectedRow

- Type

  Number or null

- Description

  The row index of the selected row (zero based)

- Object Path

  event.data

- Type

  Array

- Description

  Represents the currently selected entries. The contents of the array is based on the enabledRowSelection and enableColumnSelection properties as represented on the table below. The actual resulting value may include additional values if the selection mode on the table is set to "single interval" or "multiple interval".

enabledRowSelection	enableColumnSelection	Resulting return type	Example Output
True	False	An array containing a number of JSON objects that each represent a single row. Each JSON object contains one key-value pair for each column on the table.	`[{"city":"Folsom"," country":"United States", "population": 77271}]`
False	True	An array of JSON objects, where each object represents a separate row in the selected column. Each object contains a single key-value pair, where the key is the column name and the value is the value of the cell.	`[{"city":"Folsom"}, {"city":"Helsinki"}, {"city":"Jakarta"},]`
True	True	An array containing a single JSON object, which can be treated like a Python dictionary.	`[{"city":"Folsom"}]`

This onEditCellStart event fires when the user starts editing a cell. For onEditCellStart , the value is the initial value before any edits.

Note: This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

- Object Path

  event.column

- Type

  String or number

- Description

  The column the editing cell is positioned under.

- Object Path

  event.row

- Type

Number

- Description

  The row index of the selected row (zero based).

- Object Path

  event.rowIndex

- Type

  Number

- Description

  The row index as it is represented in the current visible data. Useful in cases where some of the rows are hidden, such as when filtering.

- Object Path

  event.value

- Type

  Any

- Description

  The value of the cell before editing began.

This onEditCellCancel event is fired when the user has canceled a cell edit and has exited editing mode by effectively pressing the escape key.

Note: This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

- Object Path

  event.column

- Type

  String

- Description

  The column name of the cell being edited.

- Object Path

  event.row

- Type

  Number

- Description

  The row index of the selected row (zero based).

- Object Path

  event.rowIndex

- Type

  Number

- Description

  The row index as it is represented in the current visible data. Useful in cases where some of the rows are hidden, such as when filtering.

- Object Path

  event.value

- Type

  Any

- Description

  The value of the cell before editing began.

This onRowClick event is fired when a row in the table is clicked.

Note: This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

- Object Path

  event.row

- Type

  Number

- Description

  The row index of the selected row (zero based).

- Object Path

  event.rowIndex

- Type

  Number

- Description

  The row index as it is represented in the current visible data. Useful in cases where some of the rows are hidden, such as when filtering.

- Object Path

  event.value

- Type

  PlainObject

- Description

  The rows value as a JSON object.

This onRowDoubleClick event is triggered when a row in the table is double clicked.

Note: This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

- Object Path

  event.row

- Type

  Number

- Description

  The row index of the selected row (zero based).

- Object Path

  event.rowIndex

- Type

  Number

- Description

The row index as it is represented in the current visible data. Useful in cases where some of the rows are hidden, such as when filtering.

- Object Path

  event.value

- Type

  PlainObject

- Description

  The rows value as a JSON object.

This onSubviewExpand event is triggered when a row subview is expanded.

Note: This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

- Object Path

  event.row

- Type

  Number

- Description

  The row index of the selected row (zero based).

- Object Path

  event.rowIndex

- Type

  Number

- Description

  The row index as it is represented in the current visible data. Useful in cases where some of the rows are hidden, such as when filtering.

- Object Path

  event.value

- Type

  PlainObject

- Description

  The rows value as a JSON object.

This onSubviewCollapse event is triggered when a row subview is collapsed.

Note: This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

- Object Path

  event.row

- Type

  Number

- Description

  The row index of the selected row (zero based).

- Object Path

  event.rowIndex

- Type

  Number

- Description

  The row index as it is represented in the current visible data. Useful in cases where some of the rows are hidden, such as when filtering.

- Object Path

  event.value

- Type

  PlainObject

- Description

  The rows value as a JSON object.

Example

For examples of the Table component, please see the following pages:

- Displaying a SubView in a Table
- Table Column Configurations

Displaying a SubView in a Table

In a Perspective Table component, you have the option to enable subviews. When a subview is set up, you can click on the Expand icon in the table and have another view be displayed without closing the first view. This example sets up a table with several cities and statistics. When the Expand icon is selected for a city, a Map component will be displayed showing the location of the city on the map.

This example focuses on using a map component in a subview on the table, but the larger implication here is that subviews in table rows can receive values from each row in the table, and utlizie them with property bindings, allowing each subview to contain data unique to from the row. The image below shows what our finished view will look like.

On this page ...

- Create a View for the Table Data
- Create a View for Displaying the Map
- Use the Maps View as a Subview for the Table

Create a View for the Table Data

We'll start by creating a view that will contain the table.

1. Right click on Views to create a view. In the example, we named ours **CityStats**. Set it as a Coordinate layout and check the Page URL option if you want to create a page for this view (you can always add a page later if you want to).

2. Drag a Table component onto the view.

3. The table needs to have Latitude and Longitude data for the map to show that location. Highlight and copy the following data:


```
[
  {
    "city": "Folsom",
    "country": "United States",
    "population": 77271,
    "lat": 38.678287,
    "lng": -121.177318
  },
  {
    "city": "Jakarta",
    "country": "Indonesia",
    "population": 10187595,
    "lat": -6.208404,
    "lng": 106.849087
  },
  {
    "city": "Madrid",
    "country": "Spain",
    "population": 3233527,
    "lat": 40.41498,
    "lng": -3.702002
  },
  {
    "city": "Prague",
    "country": "Czech Republic",
    "population": 1241664,
    "lat": 50.073453,
    "lng": 14.450091
  },
  {
    "city": "San Diego",
    "country": "United States",
    "population": 1406630,
    "lat": 32.713832,
    "lng": -117.158616
  },
  {
    "city": "San Francisco",
    "country": "United States",
    "population": 884363,
    "lat": 37.776379,
    "lng": -122.423501
  },
  {
    "city": "Shanghai",
    "country": "China",
    "population": 24153000,
    "lat": 31.227167,
    "lng": 121.498839
  },
  {
    "city": "Tokyo",
    "country": "Japan",
    "population": 13617000,
    "lat": 35.69042,
    "lng": 139.746457
  },
  {
    "city": "Washington, DC",
    "country": "United States",
    "population": 658893,
    "lat": 38.90598,
    "lng": -77.04882
  },
  {
    "city": "Wellington",
    "country": "New Zealand",
    "population": 405000,
    "lat": -41.284336,
```

```
    "lng": 174.770488
  }
]
```

4. Right click on the data property of the Table component and select **Paste**.

5. Your table will now display the data for 10 rows and 5 columns.

city	country	population	lat	lng
Folsom	United States	77,271	38.68	-121.18
Jakarta	Indonesia	10,187,595	-6.21	106.85
Madrid	Spain	3,233,527	40.41	-3.70
Prague	Czech Republic	1,241,664	50.07	14.45
San Diego	United States	1,406,630	37.71	-117.14
San Francisco	United States	884,363	37.78	-122.42
Shanghai	China	24,153,000	31.23	121.50
Tokyo	Japan	13,617,000	35.69	139.75
Washington, DC	United States	658,893	38.91	-77.05
Wellington	New Zealand	405,000	-41.28	174.77

6. Next, enable the **subview** property under props.rows.

7. The table now has **Expand** ▶ icons for each row.

	city	country	population	lat	lng
▶	Folsom	United States	77,271	38.68	-121.18
▶	Jakarta	Indonesia	10,187,595	-6.21	106.85
▶	Madrid	Spain	3,233,527	40.41	-3.70
▶	Prague	Czech Republic	1,241,664	50.07	14.45
▶	San Diego	United States	1,406,630	37.71	-117.14

Create a View for Displaying the Map

Next we'll make the view that will be display a map of the cities in our table.

1. In the Project Browser, right click on Views to create a view. Name the new view **CityMaps**. Set it as a **Flex** layout, so the map easily takes up all available space. Lastly, do not check the Page URL option, as we don't need a corresponding page.

2. Drag a Map component onto the **CityMaps** View.
3. Set the Map's **Grow** property to **"1"** so it resizes to take up the entire view.
4. Click on the **CityMaps** view in the Project Browser. In the Property Editor, click the **Add View Parameter** link under **Params** and choose **Object**.
5. Double click on key, and enter **value** as the object name.
6. Next, we'll add two parameters to that value object.
    a. Click **Add Object Member** link under Params and choose **Value.**

    b. Double click on key, and enter **"lat"**. This matches the lat (latitude) column from the Table on the CityStats view. This name must *ex actly* match the column name in the table.

c. Click the **Add Object Member** icon next to value and choose **Value**.
d. Double click on key, and enter "**lng**". This matches the lng (longitude) column from the Table on the CityStats view. This name must *exactly* match the column name in the table.

7. Next select the Map component. We need to set the map's initial geographic center to the view parameters. In the Property Editor, expand the **init.center** property.
   a. Click on the **Binding** icon next to the **lat** property.
   b. On the Edit Binding screen, select **Property** as the binding type.
   c. Click the **Browse Properties** icon. Navigate to view, the params, the value, and then the **lat** property.
   d. Click **OK**, then click **OK** again to save the binding.

   > **Note:** At this point, the init.center.lat property is bound to view.params.value.lat where view.params.value.lat's value is "value" instead of a valid latitude number. This will cause a Component Error which is expected.

   e. Click on the **binding** icon next to the **lng** property.
   f. On the Edit Binding screen, select **Property** as the binding type.
   g. Click the **Browse Properties** icon. Navigate to view, the params, the value, and then the **lng** property.
   h. Click **OK**, then click **OK** again to save the binding.

   > **Note:** At this point, the init.center.lat property is bound to view.params.value.lat where view.params.value.lat's value is "value" instead of a valid latitude number. This will cause a Component Error which is expected.

Use the Maps View as a Subview for the Table

Lastly, we need to tell the CityStats View to use CityMaps as its subview.

1. On the CityStats View, select the **Table** component.
2. In the Property Editor, scroll down to the **rows.subview.enabled** property.
3. Next, enable the **"enabled"** property.

As a result, you'll notice the table now has **Expand** ▶ icons for each row.

	city	country	population	lat	lng
▶	Folsom	United States	77,271	38.68	-121.18
▶	Jakarta	Indonesia	10,187,595	-6.21	106.85
▶	Madrid	Spain	3,233,527	40.41	-3.70
▶	Prague	Czech Republic	1,241,664	50.07	14.45
▶	San Diego	United States	1,406,630	37.71	-117.14

4. Next, find **rows.subview.viewPath**, and click the dropdown to see the a list of possible views. Choose **CityMaps** from the list and click **OK**.

5. **Save** your project.
6. Put the Designer into **Preview** mode. Click on the **Expand** ▶ icon next to one of the cities. You'll see a map of the city appear underneath the table row for that city. To close the map, click the **Collapse** ▲ icon.

city	country	population	lat	lng
Folsom	United States	77,271	38.68	-121.18
Jakarta	Indonesia	10,187,595	-6.21	106.85
Madrid	Spain	3,233,527	40.41	-3.70

Prague	Czech Republic	1,241,664	50.07	14.45
San Diego	United States	1,406,630	37.71	-117.14
San Francisco	United States	884,363	37.78	-122.42
Shanghai	China	24,153,000	31.23	121.50
Tokyo	Japan	13,617,000	35.69	139.75
Washington, DC	United States	658,893	38.91	-77.05

1

Table Column Configurations

In a Perspective Table component, you have the ability to replace a cell with something other than just text or a number. Instead, you can have the column render other objects altogether such as progress bars or a view. This page contains several examples of changing how a column renders.

On this page ...

- Replacing a Value in a Cell with a Progress Bar
- Embedding a View in a Table Cell

Replacing a Value in a Cell with a Progress Bar

1. In the Project Browser, right click on Views to create a view. In this example, the view will be named **table**. Set it to have a **Coordinate** Root Container Type.

2. Drag a Perspective Table component onto your table view. We'll use the default population information that the component is initially configured with.
3. Add three array elements to the columns property of your table as shown in the Property Editor.

4. There is a **field** property inside each of the three columns array elements. Set the **"field"** property values to match each of the column names in your table. This is how we associate a column in the table (which is really just a key in the underlying data on the table) with one of these custom column configurations.

    a. Enter a **field** property value of **"city"** for the **"0"** column's element.
    b. Enter a **field** property value of **"country"** for the **"1"** column's element.
    c. Enter a **field** property value of **"population"** for the **"2"** column's element.

5. Let's display a progress bar on the table to show the population value. To do this, go to the columns array element for "population" and set its **render** property to **"number"** and set its **number** property to **"progress"** as shown in the image below.

6. Set the **progressBar.max** value to 30,000,000 to account for cities with a large population.

7. The table will look like this after the above configurations are applied.

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
Delhi	India	11,034,555
Dhaka	Bangladesh	14,399,000
Lagos	Nigeria	16,060,303
Karachi	Pakistan	14,910,352
Istanbul	Turkey	14,025,000
Cairo	Egypt	10,230,350
Mexico City	Mexico	8,974,724
London	United Kingdom	8,825,001
New York City	United States	8,622,698
Tehran	Iran	8,154,051

Embedding a View in a Table Cell

In a Perspective Table, it is possible to embed a view inside a table cell. In this example, instead of using the table's built-in progress bar, we'll embed a view that contains a custom progress bar, using the Progress component.

1. Let's create the cell view first. In the Project Browser, right click on Views to create a view. In this example, the view will be named **cell**. Set it as a **Flex** layout.

2. Drag and drop a Progress Bar component onto the cell view.
3. Select the Progress Bar component and set the **position.grow** property to **"1"** so that the bar takes up as much space in the container as possible.
4. In the Project Browser, select **"cell"** view, and resize the view so it is closer in size to the cell in the table on our **"table"** view.
5. Create a view parameter named **"value"** by clicking on the **"Add View Parameter"** option while your view is selected in the Project Browser as shown below.

6. On the Progress Bar component, set its **max** property value to 30,000,000 to account for large values.
7. Bind the Progress Bar's **value** property to the view parameter created in Step 3 as shown below.

8. Now let's create our table view. Right click on Views to create a view. In this example, name the view **table**. Set it to have a **Coordinate** Root Container Type.

9. Drag a Perspective Table onto the table view. The default population information that comes on the factory configured table component will be used.
10. Add three array elements to the columns property of the table like as shown in Property Editor as shown in the image below.

11. There is a **field** property inside each of the three column array elements. Set the field values to match each of the column names in your table.
    a. Enter a **field** property value of **"city"** for the **"0"** column's element.
    b. Enter a **field** property value of **"country"** for the **"1"** column's element.
    c. Enter a **field** property value of **"population"** for the **"2"** column's element.

12. The **cell view** is going to be embedded into the **Population cell** values. Go to the column array element with the **"field"** value of **"population"** and set its **"render"** value to **"view"** and its **"viewPath"** to **"cell"** as shown below.

13. After the population column is pointed to the cell view, the population number from the table cell will be passed to the cell view. Since the cell view's Progress Bar has its value property bound to the cell view's input parameter, the population value will then be displayed on the table by the Progress Bar in the cell view. If you wanted to resize the progress bar, simply change the **"height"** and **"width"** properties under the **d efaultSize** property on the **"cell"** view.

city
country
population
San Francisco
United States
Shanghai
China
Tokyo
Japan
Washington, DC
United States
Wellington
New Zealand
Delhi
India
Dhaka
Bangladesh
1
2

Perspective - Thermometer

General

Component Palette Icon:

Description

The Thermometer component displays a temperature value depicted as a level in a mercury thermometer. Three temperature intervals can optionally be defined with their own colors so that the mercury color changes based on the temperature range that it is in. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a style class.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
thermometerColor	Color of the outline of the thermometer. Default is black.	color
mercuryColor	Color of the mercury. See Color Selector.	color
axisLabelColor	The color of the thermometer's y-axis label. Default is black. See Color Selector.	color
strokeWidth	Width of the lines used to draw the thermometer in pixels.	value: numeric
highBound	The high boundary value for the whole thermometer.	value: numeric
lowBound	The lower boundary value for the whole thermometer.	value: numeric
value	The value to display in the thermometer. The mercury level and value label will change to reflect this.	value: numeric
unit	A string to describe the units for the current value label. Options are "F" for Fahrenheit or "C" for Celsius.	value: string

		dropdown
valueFont color	The color of the current value. See Color Selector.	color
valueFont	The font to use for the current value label. **Name** / **Description** / **Property Type** fontSize / Size of the font for the current value. / value: numeric	object
intervals	Defines the upper and lower bounds for each interval. **Name** / **Description** / **Property Type** color / Color of the mercury in the thermometer. See Color Selector. / color high / High bound value for the interval. / value: numeric low / Low bound value for the interval. / value: numeric	object
style	Sets a style for this component. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

Property	Value
props.unit	F
props.mercuryColor	#8AFF8A
props.intervals.1.high	45
props.intervals.0.color	#0000FF
props.intervals.1.color	#CCCCFF
props.intervals.2.color	#FF0000
props.intervals.2.low	85
props.value	90

Perspective - Tree

General

Component Palette Icon:

Description

The Tree component displays a tree hierarchy based on an array of objects. Icons can be chosen for the nodes of the tree, and different icons can be used when an node is expanded or collapsed.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
items	An array of objects, each of which represents a node on the tree. **Name** / **Description** / **Property Type** label / Label text for the list item. / value: string expanded / Whether or not the tree appears with all levels expanded. / value: boolean data / String data for list item. / value: string items / An array of objects, each of which represents a child node on the tree. / array	array
interactable	If set to false, the tree is displayed but the user can't interact with it in the runtime. Default is true.	value: boolean
selection	Holds the item index path of the current selection.	value: string
selectionData	Array of objects containing the data and index path for all currently selected items. **Name** / **Description** / **Property Type** itemPath / Index path. / value: numeric value / I am string data for the item. / value: string.	array

appearance	Settings for the appearance of the tree.	object

Name	Description	Property Type
textOverflow	Setting indicating whether overflowing text should cause the entire tree to scroll horizontally or whether the text should be truncated with an ellipsis. Default is scroll.	value: string dropdown
expandIcons	Settings for the expand icons . Options as follows:	object

Name	Description	Property Type
collapsed	Icon appearance when path is collapsed.	object

Name	Description	Property Type
path	Path to the icon source, in format: library/IconName. The materials icon library is a the primary source for icons, see https://fonts.google.com/icons?selected=Material+Icons.	value: string
Color	Fill color to apply to the icon.	string
style	Sets a style for the icon. 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

Name	Description	Property Type
expanded	Icon appearance when path is expanded.	object

Name	Description	Property Type
path	Path to the icon source, in format: library/IconName. The materials icon library is a the primary source for icons, see https://fonts.google.com/icons?selected=Material+Icons.	value: string
Color	Fill color to apply to the icon.	string
style	Sets a style for the icon. Full menu of style options is available. You can also specify a style class.	object

Name	Description	Property Type
empty	Icon appearance when path is empty.	object

Name	Description	Property Type
path	Path to the icon source, in format: library/IconName. The materials icon library is a the primary source for icons, see https://fonts.google.com/icons?selected=Material+Icons.	value: string
Color	Fill color to apply to the icon.	string
style	Sets a style for the icon. Full menu of style options is	object

		available. You can also specify a style class.	

	ht	numeric	
style	Sets a style for this component. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

Property	Value
appearance.defaultNodeIcons.collapsed.path	material/Play_arrow
appearance.defaultNodeIcons.collapsed.color	#D9D900
appearance.defaultNodeIcons.expanded.path	material/subdirectory_arrow_right
appearance.defaultNodeIcons.expanded.color	#FFAC47
appearance.defaultNodeIcons.empty.path	material/panorama_fish_eye
appearance.defaultNodeIcons.empty.color	#000000

Perspective - Video Player

General

Component Palette Icon:

Description

The Video Player component enables you to embed video or a live feed in Perspective views. In a view, the component displays either a live feed from an IP camera or a web hosted video file that is accessible from your gateway. The component is wrapped in a skin that gives you control over the style of the video controls and a uniform experience across browsers. Video controls can also be hidden (available on hover) to allow for a simple, clean video display.

The component requires a URL to a video or live feed. This also includes files placed on a WebDev mounted folder or file resource, which can be used to serve video files.

Designer Playback

The Designer contains an instance of JxBrowser to display your views as you build them. There are a few codecs that JxBrowser does not support. Because of this, you may find that some videos do not play or display correctly while in the Designer. This is *only* a limitation of the codecs available to the Designer. The video will work as expected in a client session assuming it supports the required codec.

> **Note:** This component plays embedded media files, which is not supported by the Safari 14 web browser. As a result, Sessions running in Safari 14 will not be able to utilize video playback on this component.

Mobile Platform Restrictions

Due to security restrictions on some mobile platforms (and in certain use cases), there are some special behaviors to be aware of when using this component.

iOS

All iOS devices require user interaction (touch, click, etc) to play the video. For this reason, the `controls.play` parameter will not play or pause the video. That must be done by the user clicking the play button. Because of this restriction, this platform also will only use the native look of the player (as determined by the web browser), rather than the custom look that is provided by the Perspective module.

iOS and Android Tablet:

On these platforms, security restrictions surround the use of the `controls.autoplay` property. Video content can only be automatically played if there is no audio. Because of this restriction, the `controls.mute` property must also be set to true.

Fullscreen Mode (all desktop and mobile platforms):

When in fullscreen mode, the native look of the player (determined by the web browser) is used as opposed to the custom look provided by the Perspective module. Because of this, the `controls.play` parameter will not play or pause the video. That must be done via user action (click, touch, etc).

User Interaction

The Video Player component properties have impact on the way a user can interact with it in the runtime.

Interaction	Description
Viewing on a Mobile Device with Android	On Android, you'll get the same experience as the desktop display with one exception; when going into fullscreen mode, you'll be presented with the native video control for a cleaner fullscreen experience on that platform.
Viewing on a Mobile Device with iOS	On iOS, you'll get the native video control for standard and fullscreen mode.

.

Controls

The following controls are available to the user in a session.

Icon	Definition	Description
	Play	Starts the video play.
	Pause	Pauses the video play.
	Playback Speed	Sets the speed of the playback. Options are .25, .5, Normal, 1.25, 1.5, 2, 5, and 10 (for example, .5 is half speed, 2 is double speed, etc.).
	Sound On	Sound is turned on for the video. Clicking on this icon brings up a sliding bar with which you can adjust the volume.
	Sound Off	Sound is turned off for the video.
0:17 / 0:52	Time elapsed/Time remaining	Displays the time elapsed in the video and the time remaining.
	Full Screen	Expands the video to full screen.
	Original size	Returns the video to original screen size. You can also press the Esc key to return to original size.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
source	The path to the source of the video or live feed.	value: string

liveFeed	Used to toggle the component to display a live feed. If set to true, the poster, autohideControls, controls, and status properties will be hidden as they pertain only to a video file.	value: boolean
poster	The path to an image that will display as the background image of the video file when the video has not yet loaded. (Hidden if props.liveFeed is set to true.)	value: string
autohideControls	Used to toggle the visible state of the control bar when displaying a video file. If set to `true`, the control bar will be displayed only when the mouse is hovered over the video. (Hidden if props.liveFeed is set to true.)	value: boolean
controls	Properties that are used to provide settings and interaction points with a video file. (Hidden if props.liveFeed is set to true.)	object

Name	Description	Property Type
autoplay	If set to true, the video will begin playing when the client session loads. While autoplay is enabled, the video will be muted while playing initially.	value: boolean
loop	If enabled, the video file will play over indefinitely once it has completed.	value: boolean
mute	If enabled, the video will show as being in a muted state and will have no volume.	value: boolean
play	If enabled, the video will play. If disabled, the video will be paused.	value: boolean
seek	The time (in seconds) from which the video should start playing.	value: numeric
volume	A number (percentage value) representing the current volume of the video file.	value: numeric
playRate	The speed at which the video will be played (where 1 is normal speed, .5 is half speed, 2 is double speed, etc.).	value: numeric

status	This property holds several sub-properties that are used to provide status updates while the video file goes through the playback process. These sub-properties should not be set as they are constantly overwritten during the playback process. (Hidden if props.liveFeed is set to true.)	object

Name	Description	Property Type
loadedData	True when the current playback position of the media has finished loading; often the first frame.	value: boolean
playing	True when playback is ready to start after having been paused or delayed due to lack of data.	value: boolean
paused	True when playback has been paused.	value: boolean
progress	A number representing the time (in seconds) where playback has occurred.	value: numeric
rateChanged	A number representing the current playback rate (1 being normal speed).	value: numeric
seeking	True when a seek operation is in progress.	value: boolean
seeked	A number representing the time (in seconds) where the seek operation was completed.	value: numeric
waiting	True when playback has stopped because of temporary lack of data.	value: boolean
ended	True when playback has stopped because the end of the media was reached.	value: boolean

controlStyle	Sets a style for the controls on this component: the control bar, all controls, error messaging, context menus, and control popups. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous. Classes are predefined styles in a project.	object
style	Sets a style for the background display of the component. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Perspective - Embedding Palette

The following components function a bit differently, but what they all have in common is that each component can be embedded in multiple views of a project.

The Carousel component allows you to display a selection of rotating views. An Embedded View is an instance of a view that is used as a component within another view. The Flex Repeater component lets you easily create multiple instances of components for display in another view, each having the same look, feel, and functionality of the original components.

Here is a complete list of the embedding components, and a link pointing to a page containing the component's description, properties, and an example of how to configure it.

In This Section ...

Perspective - Accordion

General

Component Palette Icon:

Description

The Accordion Component allows the embedding of expandable/collapsible views which can be toggled with a click or a tap of their headers. The headers may contain text or a view.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description
expansionMode	Determines how many items can be expanded at a given time. Options are 'single' and 'multiple'. When using 'single' any item that's curre using 'multiple', items that are expanded will remained open until they are collapsed.
items	An array of items in the accordion. Each item has a separate header and body configurations.

Name	Description

expanded	Determines if the the accordion body expanded. Set to true to expand, false to collapse.
header	An object containing configuration options for the toggle icon.

Name	Description
toggle	An object containing configuration options for the toggle icon.

Name	Description
enabled	Enables the collapse and expand toggle.
expandedIcon	An object containing configuration options for the header icon while the item body is expand

Name	Description	
path	Shorthand path to the icon source, in format: library/iconName.	va st
color	Color of the icon. May instead 'fill' in the styles prop.	va st
style	Sets a style for the expandedIcon. Full menu of style options is availab le for text, background, margin and padding, border, shape and miscellaneous. You can also specify a style class.	ol

Name	Description
collapsedIcon	An object containing configuration options for the header icon while the item body is collaps

Name	Description	
path	Shorthand path to icon source, in format: library/iconName	va st
color	Color of the icon. If deleted, the Shape "fill" property in the adjacent style object will determine the color of the icon.	va st
style	Sets a style for the collapsedIcon. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a style class.	ol

Name	Description
content	An object containing configuration options for the content.

Name	Description
type	Whether text or a view will be rendered in this accordion header. Set this property to either 't 'view'.
text	Text to display for this accordion header.
useDefaultViewWidth	Use of view's default width instead of adjusting based on the content's width.
useDefaultViewHeight	Use of view's default height instead of adjusting based on the content's height.
viewPath	Path to view to render in this according header.
viewParams	Params to pass to this view rendered in this accordion header.

	This feature is new in Ignition version **8.1.4** Click here to check out the other new features As of 8.1.4 a dropdown list of parameters is available when the user clicks the **Add Object I** icon. This makes it easy to add parameters from the rendered view.
style	Sets a style for the content. Full menu of style options is available for text, background, marg padding, border, shape and miscellaneous. You can also specify a style class.

height	The height of the header.
reverse	Reverses the order of the toggle and header content, (i.e., from left side to the right side).
style	Sets a style for the header. Full menu of style options is available for text, background, margin and padding, miscellaneous. You can also specify a style class.

body	An object containing configuration options for the body.

Name	Description
viewPath	Path of the view to display.
viewParams	Parameters to be passed to the view. Names in this object must match input parameters defined on t This feature is new in Ignition version **8.1.4** Click here to check out the other new features As of 8.1.4 a dropdown list of parameters is available when the user clicks the **Add Object Member** easy to add parameters from the rendered view.
useDefaultViewWidth	Use of view's default width instead of adjusting based on the content's width.
useDefaultViewHeight	Use of view's default height instead of adjusting based on the content's height.
height	The height of the body.
style	Sets a style for the body. Full menu of style options is available for text, background, margin and pad miscellaneous. You can also specify a style class.

unusedSpaceStyle	Sets a style for the empty area at the bottom of the accordion. Full menu of style options is available. You can also specify a style class.
style	Sets a style for this component. Full menu of style options is available. You can also specify a style class.

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

This Accordion example has three multiple embedded expandable and collapsible views. Each view can be expanded or collapsed by clicking on their headers. The Tank Farm and the Production Alarms views are both expanded while the Report view is collapsed.

The three views used in the accordion example are working views and the component was configured to use these existing views. Configuring the accordion component is just a matter of how you want to present the information on the component, and then to configure its properties.

Preview Mode

Property Settings

Property	Value
expansionMode	multiple
props.items.0.header.toggle.expandedIcon.path	material/assignment
props.items.0.header.toggle.collapsedIcon.path	material/expand_more
props.items.0.header.content.type	text
props.items.0.header.content.text	Tank Farm
props.items.0.body.viewPath	Tank Farm
props.items.0.body.style.marginTop	3px
props.items.0.body.style.margin	10px
props.items.1.header.toggle.expandedIcon.path	material/alarm
props.items.1.header.toggle.collapsedIcon.path	material/expand_more
props.items.1.header.content.type	text
props.items.1.header.content.text	Production Alarms
props.items.1.body.viewPath	Production Alarms
props.items.1.body.style.marginTop	3px
props.items.1.body.style.margin	10px
props.items.2.header.toggle.expandedIcon.path	material/info
props.items.2.header.toggle.collapsedIcon.path	material/expand_more
props.items.2.header.content.type	text
props.items.2.header.content.text	Report
props.items.2.body.viewPath	Report4
props.items.2.body.style.marginTop	3px
props.items.2.body.style.margin	10px

Perspective - Carousel

General

Component Palette Icon:

INDUCTIVE UNIVERSITY

Carousel

Watch the Video

Description

The Carousel component allows you to display a selection of rotating views at a defined rate with a link to the view on a page in your project. The Carousel component can automatically cycle through the views or a user can click through the views on demand, either way, still providing a link to the view on a page.

This version updates how the component handles drag transition ("swiping" across embedded views). Only common rotational angles are supported (90, 180, 270, 360) for drag transitions. If the Carousel's rotation doesn't match one of those angles, then drag transition is disabled.

Here are a few best practices when working with the Carousel component.

- Components such as the Video Player and Map are performance intensive components and should not be embedded in the Carousel since they can hurt session performance.
- Avoid embedding views containing carousels in a carousel. This can become confusing for users.
- Avoid embedding views that contain iFrame components. It's easy for content embedded in an iFrame to steal focus from other components. Also, depending on the content in the iFrame, it may impact performance.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description
views	Visible area of a page. Can have multiple views in the carousel component.

Name	Description	Property Type
viewPath	The path of the view to render in this carousel	value: string

	viewParams	Parameters to provide to this rendered view. This feature is new in Ignition version **8.1.4** Click here to check out the other new features As of 8.1.4 a dropdown list of parameters is available when the user clicks the **Add Object Member** icon. This makes it easy to add parameters from the rendered view. 	object
	direction	Direction of the child layout. Options are row or column.	value: string
	justify	Adjusts placement of view along the main axis. Options are flex-start, flex-end, or center.	value: string
	alignItems	Adjusts placement of view along the cross axis. Options are flex-start, flex-end, or center.	value: string
activePane	Active pane being displayed.		
lazyLoad	Load views on demand or progressively.		
autoplay	Settings controlling the rotation of views in the carousel.		

Name	Description	Property Type
enabled	If true, the carousel will automatically rotate the views according to the transitionDelay.	value: boolean
transitionDelay	Delay (in ms) at which slides scroll through the carousel when autoplay is true.	value: numeric
pauseOnHover	Pauses autoplay when user hovers the mouse over the view.	value: boolean
pauseOnFocus	Pauses autoplay on focus.	value: boolean
pauseOnDotHover	Pauses autoplay when user hovers the mouse over the dot for the view.	value: boolean

behavior — Behavior and interaction related carousel options.

Name	Description	Property Type
transitionSpeed	The speed (in ms) at which the carousel transitions between slides.	value: numeric
fade	Enables slides to fade in and out of view on transition	value: boolean
mobileSwipeable	Enables swiping on mobile devices to change slides.	value: boolean
desktopDraggable	Enables scrolling via drag the desktop.	value: boolean

appearance — Appearance related carousel options.

Name	Description	Property Type
dots	Carousel dots configuration. **Name** / **Description** / **Property Type** enabled — Enables dots at the bottom of the carousel component. — value: boolean iconPath — Path to the icon that will be used. — value: string styles — Configure active and inactive styles for the dot icon. — object **Name** / **Description** / **Property Type** active — Sets a style for the dot icon when active. Full menu of style options is available. You can also specify a style class. — object inactive — Sets a style for the dot icon when inactive. Full menu of style options is available. You can also specify a style class. — object	object
arrows	Carousel arrows configuration. **Name** / **Description** / **Property Type** enabled — Enables arrows at the sides of the carousel component. — value: boolean next — Next arrow icon configuration. — object **Name** / **Description** / **Property Type** iconPath — Path to the icon that will be used for the next arrow, if provided. — value: string fillColor — Fill color to apply to the icon. — string style — Sets a style for the next arrow. Full menu of style options is available. You can also specify a style class. — object previous — Previous arrow icon configuration. — object **Name** / **Description** / **Property Type** iconPath — Path to the icon that will be used for the next arrow, if provided. — value: string fillColor — Fill color to apply to the icon. — string style — Sets a style for the next arrow. Full menu of style options is available. You can also specify a style class. — object	object
useDefaultViewWidth	Enables the use of the view's default width instead of dynamically adjusting based on the available width.	value: boolean
useDefaultViewHeight	Enables the use of the view's default height instead of dynamically adjusting based on the available height.	value: boolean
slidesToShow	Number of views to show on each carousel page.	value: numeric
slidePadding	Applies padding between slides.	value: numeric

		reverse	Reverses the slide order.	value: boolean
style	Sets a style for this component. Full menu of style options is available. You can also specify a style class.			

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

In addition to the examples below, please see the Carousel Component Example page for a more thorough example of using the Carousel component.

Example 1

In this example we made two view containers: one is a coordinate container and the other is flex container.

The containers are named "generic_coordinate" and "generic_flex" accordingly. Each has a Label component and a Cylindrical Tank component. The label's text Property shows the name of the view it is in. The tank's Value property shows a level above 20. The Flex view is set to display as a row. We did not adjust any other settings on these two generic screens.

Once the two view containers were set up, we created the Carousel with the following properties and values:

Property	Value	Style Category
props.views.0.viewPath	generic_coordinate	N/A
props.view.1.viewPath	generic_flex	N/A
props.autoplay.enabled	true	N/A
props.autoplay.pauseOnHover	true	N/A
props.appearance.useDefaultViewWidth	false	N/A
props.appearance.useDefaultViewHeight	false	N/A
props.appearance.dots.iconPath	/system/images/material/crop_square	N/A
props.appearance.arrows.enabled	true	N/A

props.style.borderStyle	solid	border
props.style.borderWidth	1px	border
props.style.backgroundColor	#EFEFEF	background

Example 2

In this example we have set up a Carousel component that enables users to quickly move between four views showing valves and multistate buttons. The views must be created before the carousel but it doesn't matter what the contents of the views are. You could use any combination of views including duplicates. They are named as follows in the "Views - Test" folder:

- Views - Test/CarouselView
- Views - Test/FirstView
- Views - Test/SecondView
- Views - Test/ThirdView
- Views - Test/FourthView

Properties used on the Carousel component are as follows:

Property	Value	Style Category
props.views.0.viewPath	Views - Test/FirstView	N/A
props.views.1.viewPath	Views - Test/SecondView	N/A
props.views.2.viewPath	Views - Test/ThirdView	N/A
props.views.3.viewPath	Views - Test/FourthView	N/A
props.autoplay.enabled	false	N/A
props.appearance.useDefaultViewWidth	true	N/A

props.appearance.useDefaultViewHeight	true	N/A
props.appearance.dots.enabled	true	N/A
props.appearance.dots.iconPath	material/adjust	N/A
props.appearance.dots.styles.active.fillColor	#0000D9	N/A
props.appearance.arrows.enabled	true	N/A
props.appearance.arrows.next.iconPath	material/arrow_right	N/A
props.appearance.arrows.next.fillColor	#0000D9	N/A
props.appearance.arrows.previous.iconPath	material/arrow_left	N/A
props.appearance.arrows.previous.fillColor	#0000D9	N/A
props.style.borderStyle	ridge	border
props.style.borderWidth	5px	border
props.style.borderRadius	20	border
props.style.borderColor	#808080	border

Carousel Component Example

Perspective offers strategies for dragging (or swiping) left and right as a way of navigating between views. The **Carousel** component is specifically for this strategy, which is perfect when working with several instances of the same view (including a dynamic number of them). Maybe you're looking for a way of collapsing a lot of content onto a small screen, and need a way of scrolling through it.

On this page ...

- Initial Project Setup
- Set Up the Carousel View

Initial Project Setup

Configuring side scrolling through a carousel is pretty straightforward. First we'll create three views for the carousel to scroll through.

1. In the Project Browser, right click on Views and select the **NewFolder** option. We named our folder "Carousel Example".
2. Right click on the Carousel Example folder and select the **NewView** option.
   Name: **Motor**
   Layout: **Coordinate**
   Page URL: unchecked

3. Click **Create View**.
4. Next we'll create a view parameter that we can use for the motor number. In the Property Editor under PARAMS, click on the **Add View Parameter** icon.
   a. Select **value**.
   b. Change "key" to "Motor_Num".
   c. Change "value" to "1".
   d. Make sure the arrow icon is facing to the left, as an input parameter.

5. Next we'll make a title that will change depending on the Motor that's being displayed. Drag a Label component onto the view.
6. Bind the text property to the view parameter as follows:
    a. Click the **binding** icon next to the text property.
    b. Select **Expression** as the binding type.
    c. Enter the following: "Motor " +
    d. Click the **Property** icon then scroll down and select the Motor_Num view parameter.

e. Click **OK**. You'll now see the binding preview shows "Motor 1."

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

7. Next we'll setup a Gauge component to display motor Amps. Drag a Gauge component onto the view.
8. Align the Motor label so that it is approximately centered abouve the Gauge component.
9. In the Property Editor, expand the outerAxis properties. Change the properties as follows:
   minValue: **-50**
   maxValue: **100**

10. Now we'll set up an indirect Tag binding using one of the motor Tags in the Dairy simulator. (For more information, see Programmable Device Simulator.) Select the Gauge component.
    a. In the Property Editor, click the **binding** icon next to the value property. On the Configure Tag Binding page set the following:
    b. Choose **Tag** as the binding type.
    c. Select the **Indirect** option.
    d. Next to the Tag Path field, click on the **Tag** icon and navigate to the the Motor 1/Amps Tag in the Dairy simulator program.
    e. Click **OK**.

f. In the Tag Path field, replace the 1 with {1}.

g. In the References list, under Property, click on the **Functions** *fx* icon.

h. Click the **Properties** icon.

i. Scroll to the view parameters and select Motor_Num. Click **OK**.

j. Click **OK** again. You'll see the binding in the preview area.

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

11. The Gauge now displays the value of the AMPS Tag.

12. Drag a Label component onto your view. Place it under the Gauge component and change the text property to "**AMPS**".
13. Drag another Label component onto your view and place it next to the AMPS label. We'll set up a similar indirect Tag binding on this label.
    a. Select this Label and click the **binding** icon next to the value property.
    b. Select **Tag** as the binding type and click the **Indirect** radio button.
    c. Next to the Tag Path field, click on the **Tag** icon and navigate to the the Motor 1/Amps Tag in the Dairy simulator program.
    d. Click **OK**.
    e. In the Tag Path field, replace the 1 with {1}.
    f. In the References list, under Property, click on the **Functions** icon.
    g. Click the **Properties** icon.
    h. Scroll to the view parameters and select Motor_Num. Click **OK**.
    i. Click **Apply**. You'll see the binding in the preview area.
    j. For this Label component we don't want the full Tag value displayed. So we'll add a Transform to limit it to two decimal points. Click on **Add Transform.**
    k. Select **Format**, then click **Add Transform**.

l. Select **Numeric** as the Format type. The displayed value will now be shortened to just two decimal points. You can see the format in the Binding Preview.
m. Click **OK** to save the binding.

14. Your view should look something like this now:

Set Up the Carousel View

Now we'll set up a view that holds the Carousel component.

1. Right click on the Carousel Example folder and select the **NewView** option
2. Name the View Carousel Overview. Check the Page URL option.
3. Click **Create View**.

4. Drag a Carousel component onto the view.
5. In the Property Editor, expand the props.views.0.viewPath property.
6. Click the **Expand** icon in the viewPath property and select the **Motor** view.

7. Under viewParams, click the **Add** icon then choose value.
8. Replace "key" with "Motor_Num" and replace "value" with "1".
9. Click **OK**. The Motor view now appears in the Carousel component. You may need to expand your Carousel component slightly to fit the Motor view in without scrollbars.

10. Next we'll add two more views. Click on the **Add** icon two times.

11. Under the props.views.1.viewPath property, choose the **Motor** view.
    a. Under viewParams, click the **Add** icon then choose value.
    b. Replace "key" with "Motor_Num" and replace "value" with "2". This will point this instance to the Motor 2/AMPS Tag.
    c. Click **OK**.
12. Under the props.views.2.viewPath property, choose the **Motor** view.
    a. Under viewParams, click the **Add** icon then choose value.
    b. Replace "key" with "Motor_Num" and replace "value" with "3". This will point this instance to the Motor 3/AMPS Tag.
    c. Click **OK**.
13. Save your project.
14. Put the Designer into Preview mode. Click the left right arrows or the dots to scroll between the three Motor views.

15. We decided to change a few properties on the Carousel to update the appearance. Here are the settings we used:

Property	Value
props.appearance.dots.enabled	false
props.appearance.arrows.next.iconPath	material/navigate_next
props.appearance.arrows.next.fillColor	#AC00AC
props.appearance.arrows.previous.iconPath	material/navigate_before
props.appearance.arrows.previous.fillColor	#AC00AC
props.style.borderStyle	outset
props.style.borderColor	#AC00AC
props.style.borderWidth	7
props.style.borderTopLeftRadius	15
props.style.borderTopRightRadius	15
props.style.borderBottomLeftRadius	15
props.style.borderBottomRightRadius	15

16. Put the Designer into Preview mode. Click the next or previous arrow to scroll between the three Motor views.

Perspective - Embedded View

General

Component Palette Icon:

INDUCTIVE UNIVERSITY

Embedded View

Watch the Video

Description

The Embedded view component allows you to include an entire view inside another. Using this component allows you to select a view to display, and to pass parameters into the view. Because of this, views can easily act as templates for information.

For example, you could create a tank view and embed several into another, larger view that shows an overview of the facility.

The embedded view is different than a container because you cannot alter the contents of a view using the Embedded View. A new container would allow you to create a new grouped set of components.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
path	Path of the view to load in wrapper.	value: string
params	Parameters for the view. If passing parameters into the embedded view, the names here must match the parameters on that view. This feature is new in Ignition version **8.1.4** Click here to check out the other new features As of 8.1.4 a dropdown list of parameters is available when the user clicks the **Add Object Member** icon. This makes it easy to add parameters from the embedded view. See also Embedded Views.	object

useDefaultViewWidth	Use of view's default width instead of adjusting based on the content's width.	value: boolean
useDefaultViewHeight	Use of view's default height instead of adjusting based on the content's width.	value: boolean
style	Sets a style for this component. Full menu of style options is available. You can also specify a style class.	object

Example

In this example, there is a Map component and a Carousel component on the top of the page. Underneath them, we've placed an Embedded View component. The idea was to embed this overview to give users a quick visual reference to bigger picture trends for the site. This example assumes you have a view already created named "Pressure Trend Overview". Here are the properties for just the Embedded View:

Property	Value	Style Category
props.path	Pressure Trend Overview	N/A
props.style.backgroundColor	#FFE8CC	background
props.style.borderStyle	groove	border
props.style.borderWidth	6px	border

Perspective - Flex Repeater

General

Component Palette Icon:

INDUCTIVE UNIVERSITY

Flex Repeater

Watch the Video

Description

The Flex Repeater component lets you easily create multiple instances of views for display in another view.

When first dropped on a view, the Flex Repeater looks like any other empty container. Set the **'path'** to the component that you want to create multiple instances of, and then under **'instances'** add an object for each instance that you want to create. The **'object'** will usually contain one or more parameters that will be passed into that particular instance.

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

The Flex Repeater component has two pre-configured variants:

- Row - Repeated views will be arranged in a row.
- Column - Repeated views will be arranged in a column.

Properties

> Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
> This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
path	Path to the desired view to display.	value: string
instances	Number of instances of the view that you want to display in the container. Each instance can contain an instanceStyle and instancePosition property. Changing these properties will override any styles and position settings applied by elementStyle and elementPosition. This is where a value property can be added that matches up with a parameter in the view to pass in a value. **Name** / **Description** / **Property Type**	array

instanceStyle	Sets a style for this instance of a 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
instancePosition	Position properties such as grow, shrink, or basis that would apply to all instances. See Flex Containers.	object

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

As of 8.1.4 a dropdown list of parameters is available when the view being displayed has view parameters. This makes it easy to add, delete, or synchronize parameters from that view.

direction	Direction of layout of repeated views. Options are row, row-reverse, column and column-reverse.	value: string
wrap	Whether the container should allow instances to wrap to the next line if space has run out. Options are nowrap, wrap, wrap-reverse.	value: string
justify	Adjusts placement of instances along the main axis when there is extra space, which may be used to fill areas before, after, or in-between: flex-start, flex-end, center, space-between, space-around, space-evenly.	value: string
alignItems	Adjusts placement of instances along the cross axis when there is extra space: flex-start, flex-end, center, baseline, stretch.	value: string
alignContent	Adjusts alignment of wrapped content when there is free space in the cross axis: flex-start, flex-end, center, space-between, space-around, stretch.	value: string
useDefaultViewWidth	Use view's default width instead of adjusting based on the content's width.	value: boolean
useDefaultViewHeight	Use view's default height instead of adjusting based on the content's height .	value: boolean
elementStyle	Sets a style for this element. 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
elementPosition	Sets a position for this element. Element position properties that apply to all instances, unless overridden by instancePosition. See Flex Containers.	object

Name	Description	Property Type
grow	**grow** and **shrink** control the way that a component responds to changes in the flex container's width or height. For columns **row** controls what happens when additional space is available. See Flex Containers.	value: numeric
shrink	**shrink** controls what happens when the component does not have enough space to fulfill its basis. See Flex Containers.	value: numeric
basis	Controls the default size of a component along the flex repeater'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.	value: numeric

style	Sets a style for the Flex Repeater. 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

Name	Description	Property Type
classes	Sets a style class for the Flex Repeater.	object
overflow	Options are auto, visible, scroll, or hidden.	value: string

Property User Interface

When a Flex Repeater is deep selected, there is a Graphical User Interface (GUI) at the top of the Perspective Property Editor that enables you to set the container's properties. Functionality is similar to that of the properties in the Props Tree, but you may find the visual interface easier or quicker to use.

Direction

This sets the direction for the child layout. Options are **Row** or **Column**. When the **Reverse** icon is selected, the contents of this container are displayed in reverse order.

Direction: Row

The following table shows the icons and properties they represent when **Direction: Row** is selected. The icon that's displayed if Reverse order is selected is also shown.

Left/right/top/bottom notes in the descriptions refer to non-reversed directions. The phrase "when there is extra space" means when no components have are stretching to fill the space (i.e., when no components have "grow" greater than 0).

Items			
Row Icon	**Row Reversed Icon**	**Property**	**Description**

		Flex Start	Child items are placed along the start (top) of the container when there is extra space.
		Center	Child items are placed along the center of the container when there is extra space.
		Flex End	Child items are placed along the end (bottom) of the container when there is extra space.
		Stretch	Child items are stretched from top to bottom of the container.
		Baseline	Child items are placed so the baseline of the text matches for all of them when there is extra space.
Justify			
Row Icon	**Row Reversed Icon**	**Property**	**Description**
		Flex start	Adjusts placement of children to the start (left) of the container when there is extra space. If reversed, children are placed along the right.
		Center	Adjusts placement of children along the center of the container when there is extra space.
		Flex End	Adjusts placement of children along the end (right) of the container when there is extra space. If reversed, children are placed along the left.
		Space Between	Adjusts placement of children with space in between them reaching to the edges of the container when there is extra space.
		Space Around	Adjusts placement of children with even spacing in between them with some space along the edges when there is extra space.
		Space Evenly	Adjusts placement of children with even spacing in between them and the edges of the container when there is extra space.
Children			
Icon		**Property**	**Description**
N/A		Don't Wrap	If there are more components than the width allows, shrink them.
N/A		Wrap	If there are more components than the width allows, wrap onto the next line.
		Reverse Wrap	Toggle to reverse the direction of wrap from top-down to bottom-up
Content (Only applicable when Children:Wrap is selected.)			
Row Icon	**Row Reversed Icon**	**Property**	**Description**
		Flex start	Adjusts placement of wrapped content to the start (top) of the container when there is free space.
		Center	Adjusts placement of wrapped content to the middle of the container when there is free space.
		Flex End	Adjusts placement of wrapped content to the end (bottom) of the container when there is free space.
		Space Between	Adjusts placement of wrapped content evenly with space in between each wrapped line, reaching to the edges (top and bottom) of the container when there is extra space.
		Space Around	Adjusts placement of wrapped content evenly with space in between each wrapped line and the edges (top and bottom) of the container when there is extra space.
		Stretch	Adjusts placement of wrapped content evenly with space in between each wrapped line and after the last line (bottom) of the container when there is extra space.

Direction: Column

The following table shows the icons and properties they represent when **Direction: Column** is selected. The icon that's displayed if Reverse order is selected is also shown.

Items			
Column Icon	**Column Reversed Icon**	**Property**	**Description**
		Flex start	Child items are placed along the start (left) of the container when there is extra space.
		Center	Child items are placed along the center of the container when there is extra space.
		Flex End	Child items are placed along the end (right) of the container when there is extra space.
		Stretch	Child items are stretched from left to right of the container.
		Baseline	Child items are placed so the baseline of the text matches for all of them when there is extra space.
Justify			
Column Icon	**Column Reversed Icon**	**Property**	**Description**
		Flex start	Adjusts placement of children to the start (top) of the container when there is extra space. If reversed, children are placed along the bottom.
		Center	Adjusts placement of children along the center of the container when there is extra space.
		Flex End	Adjusts placement of children along the end (bottom) of the container when there is extra space. If reversed, children are placed along the top.
		Space Between	Adjusts placement of children with space in between them reaching to the edges of the container when there is extra space.
		Space Around	Adjusts placement of children with even spacing in between them with some space along the edges when there is extra space.
		Space Evenly	Adjusts placement of children with even spacing in between them and the edges of the container when there is extra space.
Children			
Icon		**Description**	
N/A		Don't Wrap	If there are more components than the width allows, shrink them.
N/A		Wrap	If there are more components than the width allows, wrap onto the next line.
		Reverse	Toggle to reverse the direction of wrap from top-down to bottom-up

Content (Only applicable when Children:Wrap is selected.)			
Column Icon	**Column Reversed Icon**	**Property**	**Description**
		Flex start	Adjusts placement of wrapped content to the start (left) of the container when there is free space.
		Center	Adjusts placement of wrapped content to the middle of the container when there is free space.
		Flex End	Adjusts placement of wrapped content to the end (bottom) of the container when there is free space.
		Space Between	Adjusts placement of wrapped content evenly with space in between each wrapped line, reaching to the edges (left and right) of the container when there is extra space.
		Space Around	Adjusts placement of wrapped content evenly with space in between each wrapped line and the edges (left and right) of the container when there is extra space.
		Stretch	Adjusts placement of wrapped content evenly with space in between each wrapped line and after the last line of the container when there is extra space.

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

1. In order to use the Flex Repeater, you need a view that will be shown inside the repeater. To do this, we made a new View with a Coordinate layout called "RepeatedView".

    a. In the Project Browser, select the new View
    b. In the Property Editor. Under **PARAMS**, select **Add View Parameter**, select a property type of **Value**.
    c. Rename the parameter from "key" to "labelText".
    d. Select the **root** container for the view, and set the **mode** property to "percent".

2. Add a Label component to the view, and configure a property binding on its **text** property to the **labelText** parameter we just created. We set the component's **alignVertical** property to "center", and we stretched the label to fill the entire view. We also configured some styling on the Label:

Style Category	Value

borderStyle	solid
fontSize	30px
textAlign	center

3. Now we can configure our Flex Repeater. Drag a Flex Repeater component onto the view, then set the following properties:

Property Name	Value
path	RepeatedView
direction	Column
useDefaultViewWidth	false
useDefaultViewHeight	false

4. Finally, create three **object** members in the **instances** array. Add a labelText property to each object of type **value**, and replace the value strings to the desired strings to show. Here's how the property editor looks for our Flex Repeater:

5. You should now see the Flex repeater populated as shown in the image above.

Perspective - View Canvas

General

Component Palette Icon:

INDUCTIVE UNIVERSITY

View Canvas

Watch the Video

Description

The View Canvas component can display multiple Perspective views, each positioned on a coordinate based system. The component offers smooth transition animations when views are relocated. Familiarity with CSS is helpful in taking full advantage of the this component.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description
instances	Array of views to display in the canvas.

Name	Description	Property Type
position	Mode that defines how the element is positioned within the canvas. For position absolute, the view is placed within the canvas based on its top, left, bottom, right positions. Views that are positioned absolute do not participate in the flow of the document. For position relative, the view is placed within the canvas placed in the normal document flow and then offset by its top, left values. This is the same with left and right.Options are relative or absolute. Default is absolute.	value: string
top	The top position of the view.	value: numeric
left	The left position of the view.	value: numeric
bottom	The bottom position of the view. Note: If both top and bottom are set, bottom is respected only if position is set to absolute and height is unspecified.	value: numeric
right	The right position of the view. Note: If both left and right are set, left is respected only if position is set to absolute and height is unspecified.	value: numeric
zIndex	The z-order position of the view.	value: numeric
width	The width of the view.	value:

		numeric
height	The height of the view.	value: numeric
viewPath	Path to the view you want to display.	value: string
viewParams	The parameters of the view. This feature is new in Ignition version **8.1.4** Click here to check out the other new features As of 8.1.4 a dropdown list of parameters is available when the user clicks the **AddObject Member** icon. This makes it easy to add parameters from the rendered view.	object
style	Sets a style for this view. Full menu of style options is available. You can also specify a style class.	object

transitionSettings

Transition settings on each view. The properties affected by transition settings are top, left, bottom, right, and zIndex.

Name	Description	Property Type
duration	Duration of the transition. Units are seconds or milliseconds.	value: numeric
timingFunction	Mathematical function that defines how fast one-dimensional values change during the transition. The transition can be described as a cubic Bezier or steps function. The presets for cubic Bezier functions are linear, ease, ease-in, ease-in-out, and ease-out. The presets for steps functions are step-start and step-end.	value: string

enableTransitions	Determines whether transitions should play when transitions are defined.
defaultStyle	Sets a style for all views. Full menu of style options is available. You can also specify a style class.
style	Sets a style for the canvas. Full menu of style options is available. You can also specify a style class.

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Component Events

Event is fired when a view instance is clicked.

- Object Path

  event.index

- Type

  Number

- Description

  The index of the view instance.

- Object Path

  event.params

- Type

Dictionary

- Description

  The position of the view instance in relation to the canvas.

- Object Path

  event.path

- Type

  String

- Description

  The path of the view instance.

- Object Path

  event.position

- Type

  JSON Object

- Description

  A JSON Object representing the current position values.

- Object Path

  event.position.top

- Type

  Number

- Description

  The top position of the view instance.

- Object Path

  event.position.left

- Type

  Number

- Description

  The left position of the view instance.

- Object Path

  event.position.bottom

- Type

  Number

- Description

  The bottom position of the view instance.

- Object Path

  event.position.right

- Type

  Number

- Description

  The right position of the view instance.

- Object Path

  event.size

- Type

  JSON Object

- Description

  A JSON Object representing the current size.

- Object Path

  event.size.width

- Type

  Number

- Description

  The width of the view instance.

- Object Path

  event.size.height

- Type

  Number

- Description

  The height of the view instance.

Perspective - Input Palette

Input Components

Perspective provides a host of Input components that allow users to enter or select data, and even control a device.

Here is a complete list of Input components, and a link pointing to a page containing the component's description, properties, and an example of how to configure it.

Input Components

Watch the Video

In This Section ...

Perspective - Barcode Scanner Input

General

Component Palette Icon:

Description

The Barcode Scanner Input component awaits for input from a barcode scanner. The component was designed for keyboard wedge scanners, as the component provides dedicated prefix and suffix properties to define scanner input. As such, it can be useful to think of the Barcode Scanner Input component as a specialized text field that does not require focus, and uses characters to decide when to accept and reject text input.

The scanner component is continuously listening, waiting for the prefix and suffix characters to be entered. Once triggered, the component will load the scanned barcode string (excluding the prefix and suffix) into the data property for processing. The regex property can be used to extract specific fields from a scan, or validate data from the scan.

The component has a grey, dashed border as a default, and accepts a variety of style settings, including border, background color, etc.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
prefix	String that marks the start of the barcode scan capture.	value: string
suffix	String that marks the end of the barcode scan capture.	value: boolean
regex	Regex describing format of scans. The first capture will be used as barcode.	object
window	Length of buffer to monitor for regex match.	value: numberic
data	Barcode scans returned from scanner.	array
dataStyle	Sets a style for data returned to this component. 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
style	Sets a style for this component. Full menu of style options is available text, background, margin and padding, border, shape and miscellaneous. You can also specify a style class. Styles can can be set on the component before a value is scanned.	object

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Perspective - Button

General

Component Palette Icon:

Description

The Button component is a versatile component, that is used to initiate some sort of action in response to being pressed. It can be used for showing status, as well. For example, you can configure buttons to be active or inactive, change color, text or any other property, and you can alter these configurations in response to conditions in your project. Button components support icons as well. For an example, see Example 2 below.

To get buttons to do things, you configure one or more Actions that occur following an Event. For instance, you might call a Script action on the onActionPerformed component event, which triggers when the button is pressed.

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

The Button component has three pre-configured variants: Primary and Secondary.

Properties

> Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
> This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
text	Text to display on a button.	value: string
textStyle	Style properties that are directly applied to the text within the Button component. 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
primary	Toggle between default primary and secondary button style. Default is true.	value: boolean
enabled	Enables button interaction. Default is true.	value: boolean
image	An optional image embedded in the button. **Name** / **Description** / **Property Type** source — The image source URL. It could be a URL to an image on the internet or Gateway, or even an embedded image. For an image already in the Image Management console, use /system/images/{path to your image}. For example: `/system/images/Builtin/icons/24/lightbulb_on.png` — value: string	object

icon	An image path used to augment the writingState of the component by placing an image next to it. **Name** / **Description** / **Property Type** path: Shorthand path to icon source, in format: library /IconName. The materials icon library is a the primary source for icons, see https://fonts.google.com/icons?selected=Material+Icons. — value: string color: Color of the icon. Can also assign color in "fill" of the style property. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector. — color	object
width	Width of the button image in pixels.	value: numeric
height	Height of the button image in pixels.	value: numeric
position	Horizontal position of the image within the button relative to the text: left, center, right, top, or bottom.	value: string
style	Sets a style for the image. 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

align	Aligns the text and image (if present) along the cross axis. Vertical if imagePosition is top or bottom, otherwise it's horizontal. Options are start, center, end, and stretch. Default is center.	value: string
justify	Justifies the text and image (if present) along the main axis. Horizontal if the imagePosition is top or bottom, otherwise it's vertical. Options are start, center, end, space-around, space-between, and space-evenly. Default is center.	value: string
style	Sets a style for this component. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1

Property	Value
props.text	Complete
props.image.source	/system/images/Builtin/icons/48/check2.png
props.image.position	right
props.textStyle.color	#000000
props.justify	space-evenly
props.style.backgroundColor	#D5D5D5
props.style.borderStyle	solid

props.style.borderColor	#000000
props.style.borderWidth	2px

Example 2

Property	Value
props.text	Save
props.textStyle.color	#00AC00
props.image.source	/system/images/Builtin/icons/48/disk_green.png
props.image.position	top
props.image.width	40
props.image.height	40
props.align	end
props.justify	space-between
props.style.backgroundColor	#FFFFFF
props.style.borderStyle	inset
props.style.borderWidth	5px
props.style.borderColor	#00AC00

Perspective - Checkbox

General

Component Palette Icon:

Description

A Checkbox is a familiar component that represents a bit - it is either on (selected) or off (not selected). In addition, the 'triState' property can be enabled, adding a third state to represent an indeterminate value. It is functionally equivalent to the Toggle Switch component.

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

The Checkbox component has two pre-configured variants: Text Right and Text left.

Properties

> Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
> This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
selected	Output value for checkbox.	value: boolean
text	Label for the checkbox.	value: string
textPosition	Where to place the label text in relation to the checkbox: top, right, bottom, or left.	value: string
enabled	Whether the user can currently interact with the checkbox. **Note:** If the component is disabled, scripts can still run on the component. For example, if you add a script action to the onClick event, the script will fire when the user clicks on the Checkbox.	value: boolean
triState	Whether the checkbox supports a third state of "indeterminate" - effectively 'null' or 'no choice'.	value: boolean
checkedIcon	Settings for the appearance of the check box's icon when it is selected (checked). **Name** / **Description** / **Property Type** path — Path to the icon source, in format: library/IconName. For more information on icons, see the Images, SVGs, and Icons in Perspective page. — value: string color — Settings for the fill color for the checked icon. — object **Name** / **Description** / **Property Type**	object

		enabled	Color of the icon when enabled. Can be a named color.	value: string	
		disabled	Color of the icon when disabled. Can be a named color.	value: string	
	style	Sets a style for the icon. Full menu of style options is available. You can also specify a style class.			object

uncheckedIcon	Settings for the appearance of the check box's icon when it is not selected.
	
	Name / **Description** / **Property Type**
	path — Path to the icon source, in format: library/IconName. For more information on icons, see the Images, SVGs, and Icons in Perspective page. — value: string
	color — Settings for the fill color for the unchecked icon. — object
	  Name / **Description** / **Property Type**
	  enabled — Color of the icon when enabled. Can be a named color. — value: string
	  disabled — Color of the icon when disabled. Can be a named color. — value: string
	style — Sets a style for the icon. Full menu of style options is available. You can also specify a style class. — object
	
	Sets a style for this component. 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

indeterminateIcon	Settings for the appearance of the check box's icon when it is in the indeterminate state. Only applies if props.triState is set to true.
	
	Name / **Description** / **Property Type**
	path — Path to the icon source, in format: library/IconName. For more information on icons, see the Images, SVGs, and Icons in Perspective page. — value: string
	color — Settings for the fill color for the icon. — object
	  Name / **Description** / **Property Type**
	  enabled — Color of the icon when enabled. Can be a named color. — value: string
	  disabled — Color of the icon when disabled. Can be a named color. — value: string
	style — Sets a style for the icon. Full menu of style options is available. You can also specify a style class. — object
		object

style	Sets a style for this component. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1

Property	Value
props.text	Checkbox 1
props.textPosition	left
props.triState	true

Example 2

Property	Value
props.text	Checkbox 2
props.textPosition	left
props.selected	null

Example 3

Property	Value
props.text	Checkbox 3
props.textPosition	top
props.enabled	false
props.selected	false

Perspective - DateTime Input

General

02/25/2018 10:55 pm

Component Palette Icon:

DateTime Input

Description

A DateTime Input is an easy way to select a date from a popup calendar. Similar to the DateTime Picker component, it takes up much less real estate on the screen. Configure the date and time format in the Property Editor using the 'formattedValue' property.

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

The DateTime Input component has two pre-configured variants:

- Date and Time - Opens a calendar from which users can select a date and time.
- Time - Enables users to set a time using the up and down arrows on the component.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
value	Current date/time as a Date object or timestamp in milliseconds.	value: string
formattedValue	Date and time in configured format.	value: string
inputProps	Sets a style for this component. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous.	object
modalStyle	Style applied to the Date picker modal (popup). 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
pickerType	Whether to display and enable picker for date only, time only, or both date and time.	value: string
minDate	Minimum date/time as a Date object or timestamp in milliseconds. If null, the minimum date is 10 years in the past from today.	value: string
maxDate	Maximum date/time as a Date object or timestamp in milliseconds. If null, the minimum date is 10 years in the future from today	value: string
format	Template for formatting date display - must be valid moment.js format, e.g., 'MM/DD/YYYY h:mm a'.	value: string
enabled	'False' will disable any interaction with the calendar. **Note:** If the component is disabled, scripts can still run on the component. For example, if you add a script action script action to the onClick event, the script will fire when the user clicks on the calendar..	value: boolean

placeholder	Text for input field to display when no date/time is selected.	value: boolean
locale	Code for localization of language and formatting. Use the dropdown to select language.	value: string
dismissOnSelect	Determines if the date picker should be dismissed when a date is selected.	value: boolean
style	Sets a style for this component. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

Property	Value	Style Category
props.pickerType	date	N/A
props.format	MM/DD/YYYY	N/A
props.style.borderStyle	solid	border
props.style.borderColor	#00AC00	border
props.style.borderWidth	2px	border

Perspective - DateTime Picker

General

Component Palette Icon:

Description

The DateTime Picker component uses the calendar to select the date and time. You can choose the "pickerType" to set both date and time, or just date. Configure the date and time format in the Property Editor using the formattedValue property. To use the DateTime Picker, select the month, date, and time on the component.

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

The DateTime Picker component has two pre-configured variants:

- Date and Time - Enables users can select a date and time on a calendar.
- Date - Enables users can select a date on a calendar.

Properties

> Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
> This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
value	Current date/time as a Date object or timestamp in milliseconds.	value:

		dropdown
formattedValue	Date and time in configured format.	value: string
pickerType	Whether to display and enable picker for date only or for both date and time.	value: string
minDate	Minimum date/time as a Date object or timestamp in milliseconds. If null, the minimum date is 10 years in the past from today.	value: string
maxDate	Maximum date/time as a Date object or timestamp in milliseconds. If null, the minimum date is 10 years in the future from today.	value: strinng
format	Template for formatting date display - must be valid moment.js format, e.g., 'MM/DD/YYYY h:mm a'.	value: string
locale	Code for localization of language and formatting. Use the dropdown to select language.	value: string
enabled	'False' will disable any interaction with the calendar. **Note:** If the component is disabled, scripts can still run on the component. For example, if you add a script action to the onClick event, the script will fire when the user clicks on the DateTime Picker.	value: boolean
style	Sets a style for this component. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

Property	Value	Style Category

props.format	MM/DD/YY hh:mm:ss a	N/A
props.style.borderStyle	solid	border
props.style.borderWidth	1px	border
props.style.borderColor	#D90000	border

Perspective - Dropdown

General

Component Palette Icon:

Description

The Dropdown component is a great way to display a list of choices in a limited amount of space. The current selection is shown, and the choices are presented when the user clicks on the dropdown button. There is also the capability to search for an element by typing the name of that element in the dropdown field. If the element is present, it will appear as you are typing and you can select it. If the element doesn't exist, you can define text to display that the text is not found by configuring the **noResultsText** property.

The choices that are displayed in the Dropdown depend on what Elements are defined in the **options** property of the Property Editor. The **placehol der** property defines what text is shown before any of the choices are selected. For example, you can have the word 'Select...' to inform the user to select any of the Elements from the dropdown list.

There is also a **multiSelect** property which allows the user to select multiple elements from the dropdown. Selected elements can be deleted by clicking the 'x' icon.

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

The Dropdown component has two pre-configured variants:

- Single Selection - Default layout that displays a list of choices of which the user can select one.
- Multi-Selection - Layout with a list of choices of which the user can select more than one.

Properties

> Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
> This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
value	The result of current selections (input) after any processing.	variable, based on which item in props. options property is selected.
options	And array of objects for each dropdown option. **Name** / **Description** / **Property Type** value — Actual value to be matched by the input or selection. The type of this property is initially a Value-type, but it can be converted to an Object-type or Array-type. Doing so will populate the PROPS.value property with the entire object/array, allowing a single selection on the dropdown to return multiple values — variable	array

	label	Text to display in the menu representing this option.	value: string
	isDisabled	Whether this option is currently disabled from selection. If set to true, option will not be selectable, and will use a grey font (assuming another text color isn't being applied	value: boolean

multiSelect	Enable multiple selected values. Default is false.	value: boolean
placeholder	Settings for the text displayed when value is empty.	object

Name	Description	Property Type
text	Prompt text to display when no options are selected.	value: string
color	Color of placeholder text. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	color
icon	Settings for an icon used as a placeholder.	object

Name	Description	Property Type
path	Shorthand path to the icon source. Format is library/iconName. The materials icon library is a the primary source for icons, see https://fonts.google.com/icons?selected=Material+Icons.	value: string
color	Color of the placeholder icon, if it exists. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	color
style	Sets a style for this component. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous. Three additional available settings are as follows:	object

Name	Description	Property Type
margin	Margin around the icon, in pixels. Default is 0px 8px -3px 0px	value: numeric
width	Width of the icon in pixels. Default is 16px.	value: numeric
height	Height of the icon in pixels. Default is 16px.	value: numeric

enabled	If set to false, component is disabled. Field will not focus and dropdown is hidden. **Note:** If the component is disabled, scripts can still run on the component. For example, if you add a script action to the onClick event, the script will fire when the user clicks on the Dropdown.	value: boolean

search	Enter text to start search. **Name** / **Description** / **Property Type** enabled — Whether options are searchable by typing text into the field. Default is true. — value: boolean matching — Whether search string must match from the start or may match any position of an option: start or any. — value: string noResultsText — Text to display in dropdown when no option matches the search. Default is "No results found." — value: string searchParam — The text being searched for. — value: string	object
showClearIcon	Whether to display a button that the user can use to clear the selection. Default is false.	value: boolean
allowCustomOptions	Whether a user may enter a custom value to be submitted. Default is false.	value: boolean
style	Sets a style for this component. 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
dropdownOptionStyle	Sets a style for the dropdown options. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1

Property	Value
props.options.0.value	[default]_Dairy_/Bldg25/valve1
props.options.0.label	Valve 1
props.options.1.value	[default]_Dairy_/Bldg25/valve2
props.options.1.label	Valve 2
props.options.2.value	[default]_Dairy_/Bldg25/valve3

props.options.2.label	Valve 3
props.options.3.value	[default]_Dairy_/Bldg25/valve4
props.options.3.label	Valve 4
props.placeholder.text	Select a Valve...
props.placeholder.color	#0000AC
props.placeholder.icon.path	material/grade
props.placeholder.icon.color	#008000
props.style.borderStyle	solid
props.style.color	#0000D9
props.style.fontFamily	garamound
props.style.fontSize	16px
props.style.fontWeight	bold
props.style.borderWidth	2px
props.style.borderColor	#008000
props.style.borderRadius	8px
props.dropdownOptionStyle.color	#008000
props.dropdownOptionStyle.fontSize	14px
props.dropdownOptionStyle.fontWeight	bold
props.dropdownOptionStyle.textAlign	right

Example 2

In this example, we have a dropdown list with an expression binding on the options property. There is also a label on the view with the word "Email" as its text.

A default email address of j_smith@companyname.com is set as the starting value for the component.

As the user starts entering characters for an email address, the dropdown list provides typeahead options of the entered text plus three possible email options, @'cn.com', '@companyname.com', or '@gmail.com'.

Property	Value
value	j_smith@companyname.com
options	(Bound to an expression. See example below.)
props.search.enabled	true

transform code

```
# suggest auto-completed options for an email address
options = []
# skip if blank
if value:
        # check for @ symbol and suggest email address if not present
        if "@" not in value:
                options.append({ "value":value+"@cn.com", "label":value+"@cn.com"})
                options.append({ "value":value+"@gmail.com", "label":value+"@gmail.com"})
                options.append({ "value":value+"@companyname.com", "label":value+"@companyname.com"})
        # check for extension (.com) and suggest extensions if not present
        elif ".com" not in value and ".net" not in value and ".org" not in value:
                options.append({ "value":value+".com", "label":value+".com"})
                options.append({ "value":value+".net", "label":value+".net"})
                options.append({ "value":value+".org", "label":value+".org"})
# return a list of suggested options
return options
```

Edit Binding: Dropdown.props.options

Binding Type

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

Remove Binding

Configure Expression Binding

```
{this.props.search.searchParam}
```

Options

Enabled Overlay Opt-Out

Configure Transform(s)

Script

```
# suggest auto-completed options for an email address
    options = []
    # skip if blank
    if value:
        # check for @ symbol and suggest email address if not present
        if "@" not in value:
            options.append({ "value":value+"@cn.com", "label":value+"@cn.com"})
            options.append({ "value":value+"@gmail.com", "label":value+"@gmail.com"})
            options.append({ "value":value+"@companyname.com", "label":value+"@companyname.com"})
        # check for extension (.com) and suggest extensions if not present
        elif ".com" not in value and ".net" not in value and ".org" not in value:
            options.append({ "value":value+".com", "label":value+".com"})
            options.append({ "value":value+".net", "label":value+".net"})
            options.append({ "value":value+".org", "label":value+".org"})
    # return a list of suggested options
    return options
```

Add Transform +

Binding Preview

Expression → Script

[]

OK Cancel Ap

Perspective - File Upload

General

Component Palette Icon:

Description

The File Upload component allows users to upload files to the Gateway from a Perspective session. These files can be handled and saved via a script action on the onFileReceived component event.

The component has three different appearances based on its width: at its smallest, the component appears as a simple "Cloud" icon, and converts to a "Browse" button at larger widths.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
maxUploads	The maximum number of concurrent (simultaneous) uploads to allow. Default is 5.	value: integer
supportedFileTypes	An array of string values, indicating what file types are allowed to be uploaded. Example values are "pdf" or "txt".	array
fileSizeLimit	Specifies the maximum size of each uploaded file, in megabytes (MB). Default is 10 MB.	value: integer
style	Sets a style for this component. 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
fileUploadIcon	Determines the icon used when the File Upload component is small.	object

Name	Description	Property Type
path	Shorthand path to icon source, in format: library/iconName (i.e., material /arrow_right). The materials icon library is the default source for iconsin Ignition. See https://fonts.google.com/icons?selected=Material+Icons.	value: string
color	Color of the icon. Here for convenience, may instead assign 'fill' in the styles property. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	color

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Component Events

Provides a chance to handle file data uploaded to the component.

Note: This component event is designed to be used in tandem with a script action. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

- Object Path

  event.file.name

- Type

  String

- Description

  The name of the uploaded file.

- Object Path

  event.file.size

- Type

  Integer

- Description

  The size of the uploaded file in bytes.

- Object Path

  event.file.copyTo()

- Description

  Saves the uploaded file at a location accessible to the Gateway.

- Parameters

  String filePath - The path to where the file should be saved on the Gateway.

- Return

  none

- Object Path

  event.file.getBytes()

- Description

  Fetches the incoming file data. Suitable for further data processing.

- Parameters

  none

- Return

  byteArray - The raw data of the incoming file.

- Object Path

  event.file.getString()

- Description

Fetches the incoming file data and attempts to parse it as a string via UTF-8 (Eight-bit UCS Transformation Format) encoding . Defaults to UTF-8 (super common), but can use other character sets. Passed as a string, for example getString ("UTF_16BE).

- Parameters

  none

- Return

  byteArray - The raw data of the incoming file.

This event is fired when the user has cleared all uploads, but not while uploads are still in progress.

Note: This component event is designed to be used in tandem with a script action. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

- Object Path

  event.file.name

- Type

  String

- Description

  The name of the uploaded file.

- Object Path

  event.file.size

- Type

  Integer

- Description

  The size (in bytes) of the uploaded file.

Example
For an example, see Download and Upload Files page.

Download and Upload Files

Downloading and uploading files from a Perspective session typically involves storing and retrieving files from a database. A table will store all of the available files, and each row of the table represents a new file. This allows for long term storage that is accessible from any project.

The examples on this page show suggested methods of uploading files from a session, as well as how to download them.

On this page ...

- Query Examples

Uploading a File
Downloading a File

Query Examples

Before following along with the examples on this page, you'll need to create a table in the database that will hold the files. This process can vary by database, along with the column datatypes.

For the sake of brevity, the example assumes the files will be stored and retrieved from a SQL Server database. You may need to modify the query examples on this page if using a different database. The "files" database table used by these examples contains the following columns:

- **id** - integer, primary key, identity
- **filename** - varchar (255)
- **filedata** - varbinary (MAX)

Uploading a File

To upload a file in Perspective, we will want to use the File Upload component. This allows us an easy way to manage the upload.

1. Add the File Upload component to a view. The File Upload component has everything we need to upload a file into the database.
2. Right click on the File Upload component and select **Configure Events**.

3. Select the onFileReceived event and click the **Add** icon to add a script action to it.

4. Add the following script to the script action:

```
# Grab the file name and data
filename = event.file.name
filedata = event.file.getBytes()

# Use a query to insert the file
query = "INSERT INTO files (filename, filedata) VALUES(?, CONVERT(varbinary(MAX), ?))"
args = [filename, filedata]
system.db.runPrepUpdate(query, args)
```

As mentioned above, the query will also vary based on the database used.

5. Click **OK**. You can test out the upload functionality by dragging a file onto the File Upload component, either from a session, or the designer while it's in preview mode.

Downloading a File

To download a file that is stored in the database in Perspective, we will want to use the system.perspective.download function. This will allow us to download the file data that we receive from the database.

This example will show you how to do several things:

- Create a named query, that will return the contents of our file database table
- Create a table component, that shows a listing of potential files to download, using the named query above in conjunction with a Named Query Binding.
- Add a button component, that will allow users to download a file, assuming one of the rows in the table component are selected.

1. Create a Named Query that we will use to pull a list of files out of the database table. We're using a named query here since a named query binding is the easiest way to run a query when the view loads.
   The query should pull out the id of the row which we can use to later query the data, as well as the filename which the user can use to identify the file.

```
SELECT id, filename FROM Files
```

2. On a view, add a Table component. This will display a list of all files we currently have in the database table.
3. On the Table's **data** property, set up a binding. The binding should be a Query type, and it should use the query that we just made. We want to return the data in a JSON format, and you can enable polling so that it automatically updates if new files get uploaded.

4. On the Table's columns property, add an array element. Set **columns.0.field** to the name of the column that holds the filename. This will display only the filename column, as the id column does not need to be visible.

5. Add a Button to the view. This button will be used to download the file after the user has made a selection. However, we also want to make sure the user can't press the button unless a row in the table is selected.

6. On the Button's enabled property, configure a binding. The binding type should be an expression. The expression should check to see if the Table's selected row is null, and invert it.

```
!isNull({../Table.props.selection.selectedRow})
```

This will disable the component if no row is selected, to prevent the user from trying to download without making a selection.

7. Right click on the Button and go to **Configure Events**.
8. Select the onActionPerformed event, click the **Add** icon to add a script action to it.
9. Add the following script to the script action:

```
# Grab the selected row
selectedRow = self.getSibling("Table").props.selection.selectedRow

# Use the selected row to grab the id of the file at that row
id = self.getSibling("Table").props.data[selectedRow].id

# Use the id to grab the file data out of the database, along with its corresponding name.
query = "SELECT filename, filedata FROM Files WHERE id = ?"
args = [id]
data = system.db.runPrepQuery(query, args)

# Pull out the file name and data
filename = data[0][0]
filedata = data[0][1]

# Download the file
system.perspective.download(filename, filedata)
```

10. Test the script by selecting a row in the table and clicking on the button while in Preview mode.

Perspective - Multi-State Button

General

Component Palette Icon:

Description

The Multi-State button is really a series of two or more buttons, arranged in a column or row. Each button represents an integer-valued state. Each state defines two styles for a button: the selected style, and the unselected style. Each button is automatically displayed with the correct style based on the current state (the value of Indicator Value). When a button is pressed, its state's value is written to the Control Value.

When the Multi-State Button is dragged to a container, it is pre-configured with 'defaultSelectedStyle' and 'defaultUnselectedStyle properties'. These styles can be changed or deleted.

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

The Multi-State Button component has two pre-configured variants:

- Column - Default layout with options in a column format.
- Row - Layout with options in a row format.

Properties

> Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
> This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
controlValue	Bind this to the Tag that controls the state. (Typically, this is bound to the same location as the indicatorValue property.)	value: numeric
indicatorValue	Bind this to the Tag that indicates the current state. (Typically, this is bound to the same location as the controlValue property).	value: numeric
states	The value that will be written to **controlValue** when any of the buttons are clicked. Shows a list of the possible states for the component. You can add, remove, and the change the order of each state listed. Each state has two default visual styles applied for each button: Selected Style and Unselected Style. The Multi-State Button has default visual styles defined for both the selectedStyle and unselectedStyle. (Refer to 'defaultSelectedStyle' and 'defaultUnselectedStyle' properties in this table).	object

Name	Description	Property

		Type
text	Text displayed on the button.	string
value	Value assigned to the state.	value: numeric
selectedStyle	Style settings for the button when it is selected. 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
unselectedStyle	Style settings for the button when it is not selected. 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

orientation	Physical position of the button: Column or Row.	boolean
defaultSelectedStyle	Default selected style. 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
defaultUnselectedStyle	Default styles for unselectedStyles when any of the buttons are *not* selected. 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
primary	Toggles between the default primary and secondary button style.	value: boolean
enabled	If true, the user is able to interact with the buttons. **Note:** If the component is disabled, scripts can still run on the component. For example, if you add a script action to the onClick event, the script will fire when the user clicks on the buttons.	value: boolean
buttonGap	Space, in pixels, between each button in a group.	value: numeric
endButtonCornerRadius	Amount to round the end of the corners of the first and last button.	value: numeric
style	Sets a style for this component. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

Property	Value
props.indicatorValue	(property binding)
props.states.0.text	Open
props.states.0.value	2
props.states.0.selectedStyle.backgroundColor	#FFF809
props.states.1.text	Close

props.states.1.value	0
props.states.1.selectedStyle.backgroundColor	#FF8C00
props.states.2.text	Auto
props.states.2.value	1
props.states.2.selectedStyle.backgroundColor	#62ED2A
props.states.3.text	Bypass
props.states.3.value	4
props.states.3.selectedStyle.backgroundColor	#FF0000
props.orientation	row
props.buttonGap	8
props.endButtonCornerRadius	2

Perspective - Numeric Entry Field

General

Component Palette Icon:

Description

The Numeric Entry Field is similar to the standard Text Field, except that it is specialized for use with numbers. When the 'enabled' property is set to true, it allows users to alter the value on the component. There are three different modes for how users can edit the value in the component: direct, protected or by clicking an edit button. To change the value, click once in the field for 'direct' mode, double click for 'protected' mode, and click on the Edit icon for the 'button' mode. When done, press enter or leave the field, and the field becomes editable again. When the 'enabled' property is false, the field is not editable even when it receives input focus.

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

The Numeric Entry Field component has three pre-configured variants:

- Direct - Default design of the field.
- Protect - Requires a double-click or long-press to enter edit mode.
- Button - Clicking the button brings up a popup window, allowing the user to edit the value from the popup, or cancel the edit.

Properties

> Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
> This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
value	Value as number or numeric string to display.	value: numeric
format	The formatting string to be applied to the input value. Options are currency, number, integer, four decimal precision, percent, scientific, accounting, financial, currency, currency (rounded), duration, abbreviation, or ordinal. A list of format specifiers can be found here. This feature is new in Ignition version **8.1.2** Click here to check out the other new features The Numeric Entry Field supports locale-specific formatting, allowing changes to the session's locale to update how numbers are formatted on the Numeric Entry Field. Note that the localization conversion occurs automatically *after* the initial format specifiers are applied.	value: string
mode	Determines how users will edit the value in the component. The following modes are available: **Mode** — **Description** Direct — Requires no special action to enter edit mode simply click in the field.	value: string

	Protected: Requires a double-click or long-press to enter edit mode. Button: Places an **Edit** icon next to the Numeric Entry Field. Clicking the button brings up a popup window, allowing the user to edit the value from the popup, or cancel the edit. The value in the component may only be edited via the popup.	
align	Aligns the input value right or left.	value: boolean
inputBounds	Max and min bounds configuration. **Name** / **Description** / **Property Type** maximum: The max allowable value. — value: numeric minimum: The min allowable value. — value: numeric invalidStyle: Sets an invalid style when the min or max values are out of bounds for this component. Modify the invalidStyle using the style properties. Full menu of style options is available. You can also specify a style class as an invalid style. — object	object
placeholder	Text to be displayed when value is empty.	value: string
tooltipText	Mousing over this button will display a tooltip with this text, if present.	value: string
enabled	Indicates if user should be allowed to alter the value. **Note:** If the component is disabled, scripts can still run on the component. For example, if you add a script action to the onClick event, the script will fire when the user clicks on the Numeric Entry Field.	value: boolean
containerStyle	This feature is new in Ignition version **8.1.2** Click here to check out the other new features Sets a style for the outer area of the component. Ideal for adding padding between the outer area and the numeric display/input.	object
style	Sets a style for this component. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1

This example shows the component set to "button" mode, providing a button to click on when entering a new value.

Property	Value
props.value	45.678
props.format	#0.0
props.mode	button
props.align	center
props.placeholder	setpoint

Example 2

setpoint

This examples demonstrates the placeholder property, showing a default entry in cases where the value is null.

Property	Value
props.value	null
props.format	#0.0
props.mode	button
props.align	center
props.placeholder	setpoint

Example 3

23.45%

This example demonstrates the format property, allowing custom formatting to be applied to the value in the component.

Property	Value
props.value	0.2345
props.format	0.0#%
props.mode	direct
props.align	right
props.placeholder	setpoint

Perspective - One-Shot Button

General

Component Palette Icon:

Description

The One-Shot Button is designed to send of a write, and wait for a response, disabling the button until something resets the 'value' property on the component.

When the 'value' property and the 'setValue' property are equal, the component transitions to the writing state. Once 'value' and 'setValue' are no longer equal, the button returns the the a ready state.

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

The One-Shot Button component has three pre-configured variants:

- Primary - Default design of the button.
- Secondary - A secondary design for the button.
- Require Confirm - Default design of the button but requires confirmation from user before action is submitted.

Properties

> Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
> This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
value	The current value displayed on the component. Should be bound bi-directionally to a Tag. Default is 0. This feature is new in Ignition version **8.1.4** Click here to check out the other new features As of 8.1.4 the property type for value can be numeric, boolean, string, or null.	value: numeric, boolean, string, or null.
setValue	The value to set when the button is pushed. Default is 1. This feature is new in Ignition version **8.1.4** Click here to check out the other new features As of 8.1.4 the property type for setValue can be numeric, boolean, string, or null.	value: numeric, boolean, string, or null.
primary	Toggle between the default primary and secondary button style. Default is true.	value: boolean
enabled	Whether the user can interact with the One-Shot Button. If disabled, the component cannot be used. Default is true. **Note:** If the component is disabled, scripts can still run on the component. For example, if you add a script action to the onClick event, the script will fire when the user clicks on the One-Shot Button.	value: boolean

<table>
<tr><td>readyState</td><td>Displays the readyState value on the component.
<table>
<tr><th>Name</th><th>Description</th><th>Property Type</th></tr>
<tr><td>text</td><td>The text of the button while it's value is not being written.</td><td>value: string</td></tr>
<tr><td>style</td><td>Modify readyState style using the style properties. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a style class.</td><td>object</td></tr>
<tr><td>icon</td><td>The Icon is an image path used to augment the readyState of the component by placing an image next to it.
<table>
<tr><th>Name</th><th>Description</th><th>Property Type</th></tr>
<tr><td>path</td><td>Shorthand path to icon source, in format: library /IconName. The materials icon library is a the primary source for icons, see https://fonts.google.com/icons?selected=Material+Icons.</td><td>value: string</td></tr>
<tr><td>color</td><td>Color of the icon. Can also assign color in "fill" of the style property. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.</td><td>color</td></tr>
<tr><td>style</td><td>Modify the readyState style using the style properties. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a style class.</td><td>object</td></tr>
</table></td><td>object</td></tr>
</table></td><td>object</td></tr>
<tr><td>writingState</td><td>Displays the writingState value on the component.
<table>
<tr><th>Name</th><th>Description</th><th>Property Type</th></tr>
<tr><td>text</td><td>The text of the button while its value is being written. Default is "Writing...".</td><td>value: string</td></tr>
<tr><td>style</td><td>Modify the writingState style using the style properties. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a style class.</td><td>object</td></tr>
<tr><td>icon</td><td>An image path used to augment the writingState of the component by placing an image next to it.
<table>
<tr><th>Name</th><th>Description</th><th>Property Type</th></tr>
<tr><td>path</td><td>Shorthand path to icon source, in format: library /IconName. The materials icon library is a the primary source for icons, see https://fonts.google.com/icons?selected=Material+Icons.</td><td>value: string</td></tr>
<tr><td>color</td><td>Color of the icon. Can also assign color in "fill" of the style property. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.</td><td>color</td></tr>
<tr><td>style</td><td>Modify icon style using the style properties. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a style class.</td><td>object</td></tr>
</table></td><td>object</td></tr>
</table></td><td>object</td></tr>
<tr><td>confirm</td><td>If enabled, a message that asks the user to approve the requested operation.
<table>
<tr><th>Name</th><th>Description</th><th>Property Type</th></tr>
</table></td><td>object</td></tr>
</table>

	enabled	If true, a confirmation box will be shown. Default is false.	value: boolean
	text	Message to show user if confirmation is enabled. Default is "Are you sure?"	value: string

style	Sets a style for this component. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

Property	Value	Style Category
props.readyState.text	Start Process	N/A
props.readyState.icon.path	material/trending_flat	N/A
props.readyState.icon.color	#008000	N/A
props.writingState.text	Starting	N/A
props.writingState.style.backgroundColor	#8AFF8A	background

Perspective - Password Field

General

Component Palette Icon:

Description

The Password Field component is similar to a Text Field component. It allows users to enter their password text. When the Password field is empty, you can create a placeholder that informs user to "Login". You can also enable the "allowReveal" property to allow users to view their password entry.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
text	Password text.	value: string
placeholder	Text displayed when password text is empty.	value: string
enabled	Whether the user can alter the password text. Default is true. **Note:** If the component is disabled, scripts can still run on the component. For example, if you add a script action to the onClick event, the script will fire when the user clicks on the Password Field.	value: boolean
allowReveal	Whether the user can temporarily remove the password mask, revealing the password.	value: boolean
style	Sets a style for this component. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

Property	Value	Style Category
props.placeholder	Enter Password	N/A
props.borderStyle	solid	border
props.borderColor	#D90000	border
props.borderWidth	3px	border
props.fontSize	14px	text
propsfontWeight	bold	text

Perspective - Radio Group

General

Component Palette Icon:

Description

The Radio Group allows you to create multiple radio buttons in a single container. The number of radio buttons in the group is determined by the number of elements in the "radios" object. Only one radio button in a group may be selected at a time. Radio groups are a good way to let the user choose just one of a number of options. If multiple selections are expected, the Checkbox or Dropdown components can be used.

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

The Radio Group component has three pre-configured variants:

- Text Right - Default layout with text on the right of the radio button.
- Text Left - Layout with text on the left of the radio button.
- Multiple - Layout with multiple radio buttons and text on the right.

Properties

> Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
> This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
value	The value of the selected radio.	string, integer, boolean, or null
radios	List of radios that make up this group. **Name** / **Description** / **Property Type** text / Text to pair with this radio. / value: string selected / If 'true,' this radio is selected. / value: boolean value / The value of the radio to be evaluated when selected. / value: numeric	array
orientation	Placement of the Radio Button: row or column. Default is row.	value: string
align	Align radios along the cross axis. Vertical if orientation is set to row; horizontal if orientation is set to column.	value: string
justify	Justify radios along the main axis. Horizontal if orientation is set to row; vertical if orientation is set to column.	value: string
textPosition	Where to place the label text in relation to the Radio Button: top, right, bottom, or left. Default is right.	value: string

enabled	If true, user is allowed to select a radio. Default is true. **Note:** If the component is disabled, scripts can still run on the component. For example, if you add a script action to the onClick event, the script will fire when the user clicks on the Radio Button.	value: boolean

selectedIcon	Settings for the appearance of the radio's icon when it is selected.
	
	Name / **Description** / **Property Type**
	path — Path to the icon source, in format: library/IconName. For more information on icons, see the Images, SVGs, and Icons in Perspective page. — value: string
	color — Fill color settings to apply to the icon. — object
	  Name / **Description** / **Property Type**
	  enabled — Color of the icon when enabled. Can be a named color. — value: string
	  disabled — Color of the icon when disabled. Can be a named color. — value: string
	style — Sets a style for the icon. Full menu of style options is available. You can also specify a style class. — object
		object

unselectedIcon	Settings for the appearance of the radio's icon when it is not selected.
	
	Name / **Description** / **Property Type**
	path — Path to the icon source, in format: library/IconName. For more information on icons, see the Images, SVGs, and Icons in Perspective page. — value: string
	color — Fill color settings to apply to the icon. — object
	  Name / **Description** / **Property Type**
	  enabled — Color of the icon when enabled. Can be a named color. — value: string
	  disabled — Color of the icon when disabled. Can be a named color. — value: string
	syle — Sets a style for the icon. 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
		object

radioStyle	This feature is new in Ignition version **8.1.4** Click here to check out the other new features Sets a style for the radio buttons. Full menu of style options is available including margin and padding, border, shape and miscellaneous.	object
style	Sets a style for this component. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

Property	Value	Style Category
props.radios.0.text	System 1	N/A
props.radios.0.value	5	N/A
props.radios.1.text	System 2	N/A
props.radios.1.value	10	N/A
props.radios.2.text	Storage	N/A
props.radios.2.value	15	N/A
props.orientation	column	N/A
props.textPosition	left	N/A
props.style.borderStyle	solid	border
props.style.borderWidth	1px	border
props.style.borderColor	#000000	border

Perspective - Signature Pad

General

Component Palette Icon:

Description

The Signature Pad component enables users to draw a signature and “submit” it. Submitting a signature triggers a component event, enabling Ignition to do something with the signature.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
enabled	Enables the canvas, clear button, and submit button. When enabled, component scripting functions for clearSignature and submitSignature are also enabled.	value: boolean
pad	Settings for the pad.	value: object

Name	Description	Property Type
pen	Settings for the pen.	value: object

Name	Description	Property Type
color	Color used to draw the lines with the pen. You can set the color with a HEX, RGB, or HSL value. See also Color Selector Reference.	value: color
width	Width (in pixels) of the line drawn by the pen.	value: numeric

Name	Description	Property Type
canvas	Settings for the canvas.	value: object

Name	Description	Property Type
clearColor	Color used to paint over the signature pad when cleared. Default is transparent. You can set the color with a HEX, RGB, or HSL value. See also Color Selector Reference.	value: color
style	Sets a style for this property. Full menu of style options is	object

			available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a style class.

actionBar	Settings for the actionBar. **Name** / **Description** / **Property Type** position: Action bar position relative to the canvas. Options are top, bottom, left, or right. Default is bottom. — value: sting submitButton: Settings for the submit button. — object clearButton: Settings for the clear button. — object style: Sets a style for this property. 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	value: numeric
status	Settings for the status of the component. **Name** / **Description** / **Property Type** touched: True when the signature pad contains a signature. — value: boolean	object
style	Sets a style for this component. 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

submitButton

Name	Description	Property Type
text	Text to display on the button. Default is submit.	value: string
enabled	Enables submit button interaction. This does not affect the submitSignature component scripting function.	value: boolean
primary	Toggle between the default primary and secondary button style.	value: boolean
style	Sets a style for this property. 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

clearButton

Name	Description	Property Type
text	Text to display on the button. Default is clear.	value: string
enabled	Enables clear button interaction. This does not affect the submitSignature component scripting function.	value: boolean
primary	Toggle between the default primary and secondary button style.	value: boolean
style	Sets a style for this property. 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

status

Name	Description	Property Type
touched	True when the signature pad contains a signature.	value: boolean

Scripting Functions

- Description

  Clears the current signature on the component

- Parameters

  None

- Return

  Nothing

- Scope

  Session

- Description

  Submits the signature, triggering the onSignatureSubmitted component event.

- Parameters

  None

- Return

  Nothing

- Scope

  Session

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Component Events

Event is fired after the Gateway has received a submitted signature.

Note: This component event is designed to be used in tandem with a run action script. Within the script action, special properties and methods are available on the event object, which is passed to the script action as a parameter.

- Object Path

  event.signature

- Type

  String

- Description

  Base64-encoded PNG **DataURL** of the submitted signature.

- Object Path

  event.signatureFile.name

- Type

  String

- Description

  A name for the signature file.

- Object Path

  event.signatureFile.size

- Type

  Integer

- Description

The size of the signature image file in bytes.

- Object Path

  event.signatureFile.copyTo()

- Description

  Saves the uploaded signature file at a location accessible to the Gateway.

- Parameters

  String filePath - The path to where the file should be saved on the Gateway.

- Return

  none

- Object Path

  event.signatureFile.getBytes()

- Description

  Returns a bytearray of the image, allowing the signature file to be saved from the session (with system.perspective.download()).

- Parameters

  none

- Return

  byteArray - The raw data of the incoming file.

- Object Path

  event.file.getString()

- Description

  Fetches the incoming file data and attempts to parse it as a string via UTF-8 (Eight-bit UCS Transformation Format) encoding. Defaults to UTF-8 (super common), but can use other character sets. Passed as a string, for example getString ("UTF_16BE).

- Parameters

  none

- Return

  byteArray - The raw data of the incoming signature file.

This event is fired when the Gateway has received a signal that the signature has been cleared.

Note: This component event is designed to be used in tandem with a run action script. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

- Object Path

  event

- Type

  Null

- Description

  An empty event object.

Example

In this example we set a few properties to customize the look of the Signature Pad. The buttons are on the left. The blue background is set with the prop.canvas.clearColor property, which enables the color to show up in our project but not get saved as part of the signature. Lastly, we put two Label components above and below the Signature Pad with signing instructions.

Property	Value
props.enabled	true
props.canvas.clearColor	#DAF9FF
props.actionBar.position	left
props.actionBar.submitButton.text	Submit
props.actionBar.submitButton.enabled	true
props.actionBar.submitButton.primary	true
props.actionBar.submitButton.style.fontFamily	Merriweather
props.actionBar.clearButton.text	Clear
props.actionBar.clearButton.enabled	true
props.actionBar.clearButton.primary	true
props.actionBar.clearButton.style.fontFamily	Merriweather
props.style.borderStyle	dashed
props.style.borderwidth	1pt

Example 2

The following example downloads the signature image when a user clicks the Submit button on the component.

To set this up, do the following:

1. Drag a Signature component onto a Perspective view. Make sure it's a view that has a URL.
2. Right click on the component and select **Configure Events**.

3. Select the **OnSinatureSubmitted** event .
4. Click the **Add** icon and select **Script**.
5. In the Configure Script Action box, add the following script:

```
#Downloads a signature file.

image = event.signatureFile.getBytes()
system.perspective.download("Signature.PNG", image)
```

6. Click **OK** to save the script.
7. Save your project.
8. Open a Perspective Session with the view that has the Signature Pad component.
9. Sign the component and click Submit.

10. An image file is saved to your computer. In this example, we're running Ignition on Windows. The file Signature.PNG appears in our Downloads folder:

Perspective - Slider

General

Component Palette Icon:

Description

The Slider component lets the user drag an indicator along a scale to choose a value in a range. Enable the "show" and "interval" properties under "labels" to visually display the values within a range. The slider can be configured to orient horizontally or vertically with the "orientation" property.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
value	Value represented by slider handle. Current value of the slider.	value: numeric
min	The minimum value for the slider scale: all the way left or down.	value: numeric
max	The maximum value for the slider scale: all the way right or up.	value: numeric
orientation	Specifies whether the slider track is aligned vertically or horizontally.	value: boolean
step	Intervals along track at which a value may be set. Specifies the size of increments between values of the slider. **Note:** This does not force the value into that step size. Setting the slider value manually or through a binding will cause it to show the actual value.	value: numeric
labels	Label settings along the track. Name | Description | Property Type show | If true, displays labels at periodic values along track. | value: boolean interval | Interval at which to display periodic labels along track. | value: numeric	object
enabled	Whether slider interaction is currently active. **Note:** If the component is disabled, scripts can still run on the component. For example, if you add a script action to the onClick event, the script will fire when the user clicks on the Slider.	value: boolean
handleColor	Color of slider handle. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	color
trackCol	Color of slider track. See Color Selector.	color

or		
style	Sets a style for this component. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

Property	Value
props.value	65
props.orientation	vertical
props.step	5
props.labels.show	true
props.labels.interval	10
props.handleColor	#8AFF8A
props.trackColor	#CCFFFF

Component Test - WIP

Perspective - Text Area

General

Component Palette Icon:

Description

Suitable for multi-line text display and editing. Will scroll vertically on demand. Horizontal scroll is determined by the "wrap" property.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
text	Text to display.	value: string
placeholder	Text displayed when Text Area is empty.	value: string
enabled	If true, user is allowed to alter text. **Note:** If the component is disabled, scripts can still run on the component. For example, if you add a script action to the onClick event, the script will fire when the user clicks on the Text Area.	value: boolean
deferUpdates	When true, updates to props.text will be deferred until focus is lost or enter is pressed.	value: boolean
rejectUpdates WhileFocused	When true, props.text will not accept updates from external sources while focused.	value: boolean
resize	Sets whether text is resizable, and if so, in which direction: none, both, horizontal, or vertical.	value: string
wrap	Specifies how to wrap text: hard, soft, or off. (Soft wrap is the break resulting from a line wrap or word wrap.	value:

	Hard wrap is an intentional break, which moves text to the next line, or creates a new paragraph).	string
style	Sets a style for this component. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

This example shows the Text Area in two states: the top image is with text entered and the bottom image is without text entered.

Property	Value
props.text	Data Point 01: xy4, zBeta 3 Data Point 02: xy12, zBeta 45 Notes: Example of how a longer sentence can wrap (or not) around to the next line. Enter was pressed between each of these three lines.
props. placeholder	Lab Data Entry
props.wrap	off

Perspective - Text Field

General

This is a Text Field

Component Palette Icon:

Description

The Text Field component is used for input of any single-line text. This component will accept any alpha-numeric input.

If you need a field that accepts multiple lines of text, see the Perspective - Text Area component. If you're looking for a numeric field, see the Perspective - Numeric Entry Field component.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
text	Text to display.	value: string
placeholder	Text displayed when Text Field is empty.	value: string
enabled	If true, user is allowed to alter text. **Note:** If the component is disabled, scripts can still run on the component. For example, if you add a script action to the onClick event, the script will fire when the user clicks on the Text Field.	value: boolean
deferUpdates	When true, updates to props.text will be deferred until focus is lost or enter is pressed.	value: boolean
rejectUpdates WhileFocused	When true, props.text will not accept updates from external sources while focused.	value: boolean
styles	Sets a style for this component. 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

User Interaction

The Text Field component properties have impact on the way a user can interact with a table in the runtime.

Interaction	Description
Enabled	When the enabled property is set to true, a user can edit the text field in the runtime. The user must double click on the field or press enter in order to edit the field. When done, press enter or leave the field, and the text field becomes non-editable again.

	When the **enabled** property is set to false, it is not editable even when it receives input focus.
General	The Text Field also supports the reject updates during edit feature. This feature ignores updates coming from property bindings while the component is being edited by a user.

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

This example shows the Text Field in two states: the top image is with text entered and the bottom image is without text entered.

Property	Value
props.text	Lab Results from QA: Chem Analysis OK
props.placeholder	Single Line Entry

Perspective - Toggle Switch

General

Component Palette Icon:

Description

The Toggle Switch represents a bit: on (selected) or off (not selected). By default, when the switch is selected the color is blue. It is gray when it is not selected. Logically, this component is very similar to the Checkbox component.

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

The Toggle Switch component has three pre-configured variants:

- No Text - Default layout with no text.
- Text Right - Layout with text on the right of the Toggle Switch.
- Text Left - Layout with text on the left of the Toggle Switch.

Properties

> Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
> This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
selected	The selected state of the Toggle Switch.	value: boolean
label	Settings for the label for the Toggle Switch.	object

Name	Description	Property Type
text	Text for the Toggle Switch.	string
position	Text position relative to the Toggle Switch: right or left.	value: boolean
style	Modify text style using the style properties. 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

Name	Description	Property Type
color	Color settings for the Toggle Switch when it is selected and unselected.	object

Name	Description	Property Type
selected	Color of the Toggle Switch when selected (on). Can be chosen from color wheel,	color

	chosen from color palette, or entered as RGB or HSL value. See Color Selector.	
unselected	Color of the Toggle Switch when unselected (off). See Color Selector.	color

enabled	Whether the user should be allowed to alter the Toggle Switch's selected state. Default is true. **Note:** If the component is disabled, scripts can still run on the component. For example, if you add a script action to the onClick event, the script will fire when the user clicks on the Toggle Switch.	value: boolean
style	Sets a style for this component. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

Image above shows both selected and deselected positions for the following properties.

Property	Value
props.label.text	Off / On
props.label.position	left
props.color.selected	#FF0000
props.color.unselected	#8AFF8A

Perspective - Navigation Palette

One of the most important aspects to consider when developing a Perspective application is a solid navigation design so the user knows where they are, where they've been, and where they are going.

The following navigational components provide you with design strategy options to navigate within a Perspective Session, a link pointing to a page containing the component's description, properties, and an example of how to configure it.

In This Section ...

Perspective - Horizontal Menu

General

Component Palette Icon:

Description

Horizontal Menu component enables you to build a menu structure by setting up multiple links to different page URLs from the component. The Horizontal Menu occupies a large amount of horizontal space and a comparatively small amount of vertical space. The menu starts with a list of root-level menu items that make up the main display area of the component.

If you have more menu items than will fit the width of the component, arrow buttons appear to enable you to scroll left and right through all menu items. The control can also be disabled as a whole.

Each menu item can be configured with a target that will serve as either a link to a page that should be shown (e.g. *"/my-page"*), or a link to an external web page (e.g. *"http://www.inductiveautomation.com/"*). They can also be given a list of child menu items that will show temporarily in a popup as the user is interacting with them. Additionally, menu items can be labeled, disabled, and be given an icon to show to the left of their label.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective.
This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
items	Configure items representing the main menu items. **Name** / **Description** / **Property Type** enabled — Whether this option is currently enabled to perform its action or render its submenu. — value: boolean target — A url (external) or mounted path to a page. If "items" is empty (no subtree to this item), this will navigate to that location. Entering Page paths without the leading "/" will open pages in a new tab. If provided, the action to be taken when clicked. If a page should be shown, an example target would be *"/my-page"*. If an external web page should be shown, an example target would be *"http://www.google.com/"*. — value: string items — Configure items representing child menu items from this option. If defined, a submenu will branch from here with these options. — object enabled — Whether this option is currently enabled to perform its action or render its submenu. — value:	array

		boolean
target	A url (external) or mounted path to a page. If "items" is empty (no subtree to this item), this will navigate to that location.	value: string
items	Configure items representing child menu items from this option. If defined, a submenu will branch from here with these options.	array
icon	Icon image appended to the left of the menu item.	object

Name	Description	Property Type
path	Shorthand path to icon source, in format: library/IconName. The materials icon library is a the primary source for icons, see https://fonts.google.com/icons?selected=Material+Icons.	value: string
color	Color of the icon. Can also assign the "fill" property in styles. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	color

label	Text to display for this menu item label.	value: string
Style	Sets a style for this item. 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

enabled	If true, the component is enabled/active. Default is true.	value: boolean
itemStyle	Sets a style for all the items in the component. 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
style	Sets a style for the entire component. 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

Perspective Component Events
The Perspective Event Types Reference page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Component Methods

Fired when an item is selected.

- Object Path

  event.enabled

- Type

  Boolean

- Description

  Whether the item interacted with is enabled.

- Object Path

  event.label

- Type

  String

- Description

  Text to display for this option.

- Object Path

  event.label

- Type

  List

- Description

  A list containing the item indexes leading to the item that was clicked.

- Object Path

  event.target

- Type

  String

- Description

  A url (external) or a mounted path to a page.

Examples

In addition to the examples below, learn more about the Horizontal Menu component on the Navigating with the Horizontal Menu Component page.

Example 1

This example shows a Horizontal Menu with four items. The first item also has two subitems. Each item links to a webpage. Icons are taken from the Material Design icons that can be found here: https://material.io/tools/icons/.

Top Level Properties

Property	Value
props.items.0.enabled	true
props.items.0.target	https:// (link to Acme home)

props.items.0.icon.path	material/home
props.items.0.icon.color	#FF0000
props.items.0.label	Acme Inc.
props.items.1.enabled	true
props.items.1.target	https:// (link to products)
props.items.1.icon	material/flare
props.items.1.color	#D97700
props.items.1.label	Products
props.items.2.enabled	true
props.items.2.target	https:// (link to forum)
props.items.2.icon.path	material/forum
props.items.2.icon.color	#800080
props.items.2.label	User Forum
props.items.3.enabled	true
props.items.3.target	https:// (link to help)
props.items.3.icon.path	material/help
props.items.3.icon.color	#00AC00
props.items.3.label	Help
props.style.borderStyle	solid
props.style.color	#FF4747
props.style.fontSize	18px
props.style.fontWeight	bold
props.style.borderWidth	1pc
props.style.borderColor	#FFCCCC

Sub Level Properties of Item 0

Property	Value
props.items.0.items.0.target	https:// (link to about)
props.items.0.items.0.icon.path	material/info
props.items.0.items.0.icon.color	#FF0000
props.items.0.items.0.label	About Us
props.items.0.items.0.target	/screen_2
props.items.0.items.0.enabled	true
props.items.0.items.1.target	https:// (link to about)
props.items.0.items.1.icon.path	material/local_see
props.items.0.items.1.icon.color	#FF0000
props.items.0.items.1.label	Map
props.items.0.items.1.target	https:// (link to map)
props.items.0.items.1.enabled	true

Example 2