Appendix

The appendix is your reference. Once you become experienced in developing with Ignition you will most likely keep this page open on your desktop or on your second screen. This page provides you the fastest route to the information you are looking for.

Components

Expression Functions

Scripting Functions

Components

Components are what fill up your windows with useful content. Anyone familiar with computers should already understand the basic concept of a component - they are the widgets that you deal with every day - buttons, text areas, dropdowns, charts, and so on. The Vision module comes 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 will likely be the bulk of a designer's work when designing a Vision project. The basic workflow is to take a component from the palette and drop it into a container on 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 Property Editor panel to alter the component's properties, which changes the component's appearance and behavior.

Shapes are components too. Each shape may be individually selected, named, and has its own properties. Shapes have some additional capabilities that other components don't have, such as the ability to be rotated. Shapes are created using the shape tools, not dragged from the component palette.

To make the component do something useful, like display dynamic information or control a device register, you configure property bindings for the component. To make the component react to user interaction, you configure event handlers for it.

Component Scripting Overview

Common Component Functions

There are a number of functions that are common to all components in Ignition. These functions can called from the components event handlers.

requestFocusInWindow()

This function requests that the designated component be given input focus.

Suppose for example that your window has two buttons. "Button 1" performs some script and you would like the focus in the window to move directly to "Button 2" whereupon the user can simply press the enter button to run the script for "Button 2." The following would be the script that is called from the first button's event handler.

```
#Previous code for the button that runs before the window's focus changes to Button 2
event.source.parent.getComponent('Button 2').requestFocusInWindow()
```

setPropertyValue(name, value)

Sets the value attribute of the named property to the specified value.

The following two scripts to the same thing. In this example the property that is being set is the name property but it could be a different property. The first example is the preferred way because it is more in line with the python style. Whereas the second example is more like Java in it's style. Regardless, when you see this in a project be assured that they accomplish the same result.

```
#This set the value attribute of the components name property to the string "Ignition."
event.source.name = "Ignition"

#This also sets the value attribute of the component's name property but just in a different way.
event.source.setPropertyValue("name", "Ignition")
```

getPropertyValue(name)

Returns the value attribute of the property.

The following two scripts do the same thing. In this example the property that is being changed is the name property but it could be a different property. The first example is the preferred way because it is more in line with the python style. Whereas the second example is more like Java in it's style. Regardless, when you see this in a project be assured that they accomplish the same result.

```
#This returns the value attribute of the components name property.
name = event.source.name

#This also returns the value attribute of the component's name property
name = event.source.getPropertyValue("name")
```

Moving/Resizing Components and Windows

You can use scripting to move and resize a component at runtime. The functions system.gui.moveComponent, system.gui.reshapeComponent and system.gui.resizeComponent are used for this.

They simply take a component, and a new size, location, or both. Locations are always measured in pixels from the upper left point of the component's parent container.

Note that if you're moving a component that is set to relative layout, your coordinates will act as if they were coordinates to the sizes of the relevant containers last time they were saved in the Designer, not the current real coordinates of the runtime. This is very helpful for creating animations. In effect, what this means is that the coordinates fed into these functions "respect" the relative layout system automatically.
You can move and resize windows as well. If you have a reference to a window, you can set its size and location directly. For example, if you wanted to move the floating window Popup3 to certain
location, you could do so like this:

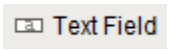
```
try:
        window = system.gui.getWindow("Popup3") window.setSize(250,600) window.setLocation(0,0)
except ValueError:
        # ignore error with a pass keyword
        pass
```

Input

Text Field

General

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're looking for a numeric field, see the Numeric Text Field.

This field features a protected mode. When you enable the protectedMode property, the field is not editable even when it receives input focus. The user must double click on the field or press enter in order to edit the field. When they are done (press enter again or leave the field), the field becomes non-editable again.

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.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

- Description

  Returns the currently selected or highlighted text in the text field.

- Parameters

  Nothing

- Return

  String - Returns the currently selected or highlighted text in the text field.

- Scope

  Client

Event Handlers

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it.

.source	The component that fired this event.
.oppositeComponent	The other component involved in this focus change. That is, the component that lost focus in order for this one to gain it, or vise versa.

This event occurs when a component that had the input focus lost it to another component.

.source	The component that fired this event
.oppositeComponent	The other component involved in this focus change. That is, the component that lost focus in order for this one to gain it, or vise versa.

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is.

.source	The component that fired this event.
.keyCode	The key code for this event. Used with the `keyPressed` and `keyReleased` events. See below for the key code constants.
.keyChar	The character that was typed. Used with the `keyTyped` event.
.keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants, the keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
.altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
.controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
.shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3.

.source	The component that fired this event.
.keyCode	The key code for this event. Used with the `keyPressed` and `keyReleased` events. See below for the key code constants.
.keyChar	The character that was typed. Used with the `keyTyped` event.
.keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation, the keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
.altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
.controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
.shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

Fires when a key is pressed and then released when source component has the input focus. Only works for characters that can be printed on the screen.

.source	The component that fired this event.
.keyCode	The key code for this event. Used with the `keyPressed` and `keyReleased` events. See below for the key code constants.
.keyChar	The character that was typed. Used with the `keyTyped` event.
.keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation, the keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
.altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
.controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
.shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires *after* the pressed and released events have fired.

.source	The component that fired this event.
.button	The code for the button that caused this event to fire.
.clickCount	The number of mouse clicks associated with this event.
.x	The x-coordinate (with respect to the source component) of this mouse event.
.y	The y-coordinate (with respect to the source component) of this mouse event.
.popupTriger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
.altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
.controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
.shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

This event fires when the mouse enters the space over the source component.

.source	The component that fired this event.
.button	The code for the button that caused this event to fire.
.clickCount	The number of mouse clicks associated with this event.
.x	The x-coordinate (with respect to the source component) of this mouse event.
.y	The y-coordinate (with respect to the source component) of this mouse event.
.popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
.altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
.controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
.shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

This event fires when the mouse leaves the space over the source component.

.source	The component that fired this event.
.button	The code for the button that caused this event to fire.
.clickCount	The number of mouse clicks associated with this event.
.x	The x-coordinate (with respect to the source component) of this mouse event.
.y	The y-coordinate (with respect to the source component) of this mouse event.
.popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
.altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
.controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
.shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

This event fires when a mouse button is pressed down on the source component.

.source	The component that fired this event.
.button	The code for the button that caused this event to fire.
.clickCount	The number of mouse clicks associated with this event.
.x	The x-coordinate (with respect to the source component) of this mouse event.
.y	The y-coordinate (with respect to the source component) of this mouse event.
.popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
.altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
.controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
.shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

This event fires when a mouse button is released, if that mouse button's press happened over this component.

.source	The component that fired this event.
.button	The code for the button that caused this event to fire.
.clickCount	The number of mouse clicks associated with this event.
.x	The x-coordinate (with respect to the source component) of this mouse event.
.y	The y-coordinate (with respect to the source component) of this mouse event.
.popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
.altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
.controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
.shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

Fires when the mouse moves over a component after a button has been pushed.

.source	The component that fired this event.
.button	The code for the button that caused this event to fire.
.clickCount	The number of mouse clicks associated with this event.
.x	The x-coordinate (with respect to the source component) of this mouse event.
.y	The y-coordinate (with respect to the source component) of this mouse event.
.popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
.altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
.controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
.shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

Fires when the mouse moves over a component, but no buttons are pushed.

.source	The component that fired this event.
.button	The code for the button that caused this event to fire.
.clickCount	The number of mouse clicks associated with this event.
.x	The x-coordinate (with respect to the source component) of this mouse event.
.y	The y-coordinate (with respect to the source component) of this mouse event.
.popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
.altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
.controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
.shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

Fires whenever a *bindable* property of the source component changes. This works for standard and custom (dynamic) properties.

.source	The component that fired this event.
.newValue	The new value that this property changed to.
.oldValue	The value that this property was before it changed.
.propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Examples

Code Snippet

```
#The following code will return the value of the text box's previous value into a variable.
#This code is fired on the property change scripting for this component.

oldValue = event.source.oldValue
```

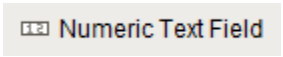
Vertical Slider with Border and Blue Text

Property Name	Value
Border	Tabbed Border with Title
Font	Dialog, Bold, 12
Horizontal Alignment	Center

Numeric Text Field

General

Component Palette Icon:



Description

The Numeric Text Field is similar to the standard Text Field, except that it is specialized for use with numbers. Instead of a "text" property, it has four numeric "value" properties. Which one you use depends on the mode of the text box.

Like the standard Text Field, this text field can operate in protected mode. When you enable the protected property, the field is not editable even when it receives input focus. The user must double click on the field or press enter in order to edit the field. When they are done (press enter again or leave the field), the field becomes non-editable again.

The Numeric 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.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

- Description

  Returns the currently selected or highlighted text in the text field.

- Parameters

  Nothing

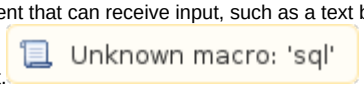
- Return

  String - Returns the currently selected or highlighted text in the text field.

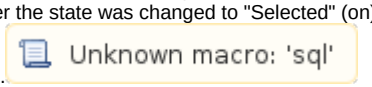
- Scope

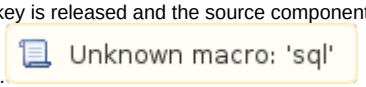
  Client

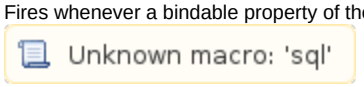
Event Handlers

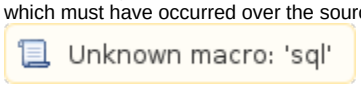
This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it. 

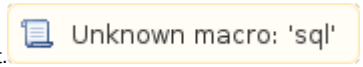
This event occurs when a component that had the input focus lost it to another component. 

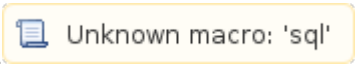
An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is. 

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3. 

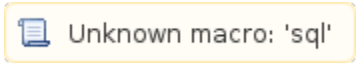
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties. 

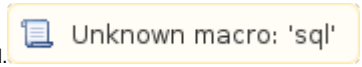
This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired. 

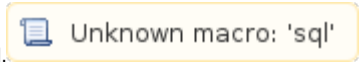
This event fires when the mouse enters the space over the source component. 

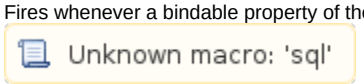
This event fires when the mouse leaves the space over the source component. 

This event fires when a mouse button is pressed down on the source component. 

This event fires when a mouse button is released, if that mouse button's press happened over this component. 

Fires when the mouse moves over a component after a button has been pushed. 

Fires when the mouse moves over a component, but no buttons are pushed. 

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties. 

Customizers

This component does not have any custom properties.

Examples

Code Snippet

```
#The following script can be executed on a mouse released event handler.
#This would write the selected text to a custom property called highlightedText.


event.source.highlightedText = event.source.getSelectedText()
```

Vertical Slider with Border and Blue Text

Property Name	Value
Border	Field Border
Number Type	Float
Font	Dialog, BoldItalic, 15
Decimal Format	#,##0.00

Spinner

General

Component Palette Icon:

Description

The spinner component represents a value that is part of a series of values, such as numbers and dates. It allows you to not only edit the value directly, but to 'spin' the value up or down, using the up and down buttons that are part of the component. When setting up property bindings, make sure you use the value property that corresponds to the spinner mode. For example, if you chose the Double spinner mode, you should bind the doubleValue property.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.Remove the panel below if there are no scripting functions.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Unknown macro: 'sql'

Customizers

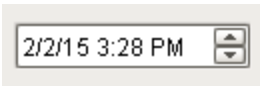
This component does not have any custom properties.

Examples

Code Snippet

```
#The following code will return the value of the slider's previous value into a variable.
#This code is fired on the property change scripting for this component.

oldValue = event.source.oldValue
```

Vertical Slider with Border and Blue Text



Property Name	Value
Spinner Mode	Date

Formatted Text Field

General

Component Palette Icon:

Description

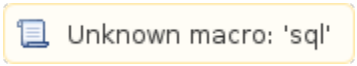
This specialized text field is used for alphanumeric text input that must match some specific pattern or needs to be formatted in a specific way. It operates in two modes:

Formatted Mask
In this mode, input is automatically formatted and restricted based on a format mask. For example, a format mask like: (###) ###-#### will allow the entry of a 10-digit US phone number. The formatting characters are automatically inserted if the user does not type them in. Any other characters are restricted. The following characters may be used in a formatted mask pattern:

Symbol	Description
#	Any valid number, Such as 0-9.
'	Escape character, used to escape any of the special formatting characters.
U	Any letter. All lowercase letters will be mapped to upper case automatically.
L	Any letter. All upper case letters will be mapped to lower case automatically.
A	Any letter or number.
?	Any letter, case is preserved.
*	Anything.
H	Any hex character (0-9, a-f or A-F).

Regular Expression
In this mode, input is validated against a regular expression. A regular expression is a special string that defines a set of allowed strings. Any input that matches the given regular expression is allowed, and input that doesn't match, is restricted. And yes, while powerful, regular expressions are decidedly difficult to decipher.

Properties



Scripting

Scripting Functions

This component does not have scripting functions associated with it.Remove the panel below if there are no scripting functions.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it.Unknown macro: 'sql'

This event occurs when a component that had the input focus lost it to another component.Unknown macro: 'sql'

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is.Unknown macro: 'sql'

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3.Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.Unknown macro: 'sql'

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component.Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component.Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component.Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed.Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed.Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

Formatted Mask

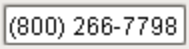
Example	Description
##U-####/UU	A product code with a specifc format, like 28E-8213/AR
0xHHHH	A hex digit, automatically prepends "0x" no the front. e.g. "0x82FF"
#UUU###	A California license plate, eg. 4ABC123

Regular Expression

Example	Description
\p{Upper}\p{Lower}*, \p{Upper}\p{Lower}*	A name, formatted such as Smith, John
\d{3}-\d{2}-\d{4}	A US social security number, like 123-45-6789
\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}	A network IPv4 address, like 67.82.120.116
^[a-f0-9A-F]{6}$	A six-digit hexadecimal number

Gallery

Horizontal Slider without Tickmarks



Property Name	Value
Validation Mode	Formatted Mask
Formatted Mask Pattern	(###) ###-####

Password Field

General

Component Palette Icon:

Description

A password field is like a text field that doesn't display the text that is being edited. You may alter the echo character (*) if you'd like.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.Remove the panel below if there are no scripting functions.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it.Unknown macro: 'sql'

This event occurs when a component that had the input focus lost it to another component.Unknown macro: 'sql'

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is.Unknown macro: 'sql'

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3.Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.Unknown macro: 'sql'

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component.Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component.Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component.Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed.Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed.Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

Code Snippet

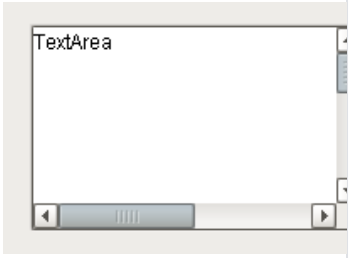
```
#The following code will return the value of the slider's previous value into a variable.
#This code is fired on the property change scripting for this component.

oldValue = event.source.oldValue
```

Vertical Slider with Border and Blue Text

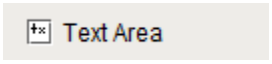
Property Name	Value
Forground Color	0,0,217
Echo Charactor	*
Text	This is my Password

Text Area

General



Component Palette Icon:



Description

Suitable for multi-line text display and editing. Will scroll vertically on demand. Will scroll horizontally if line wrap is off. Only supports plain-text, no HTML formatting or styled text.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it.Unknown macro: 'sql'

This event occurs when a component that had the input focus lost it to another component.Unknown macro: 'sql'

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is.Unknown macro: 'sql'

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3.Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.Unknown macro: 'sql'

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component.Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component.Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component.Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed.Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed.Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.Unknown macro: 'sql'

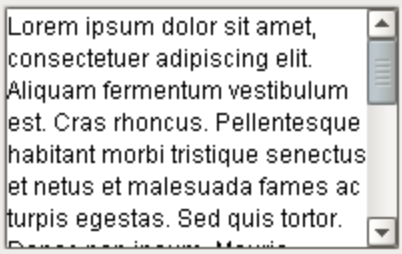
Customizers

This component does not have any custom properties.

Examples

Gallery

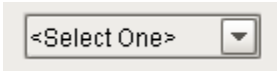
Horizontal Slider without Tickmarks



Property Name	Value
Line Wrap	True
Text	514 Characters

Dropdown List

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 only presented when the user clicks on the dropdown button. The choices that are shown depend on the data property. This is a dataset, which can be typed in manually in the Designer, or (more commonly) it can be populated dynamically from a property binding, often a SQL Query binding.

It is often the case that you want to display choices to the user that are 'dressed up' versions of the actual choices. For instance, suppose that you are selecting choices for a downtime tracking entry. The choices might be: "Operator Error", "Machine Malfunction", and "Other". But, you really want to map these choices to some numeric code which is how the choice is stored. So, for instance, when the user chooses "Other" you really want to get the number 3. The dropdown component is perfect for such a use. The data property can be set up in one of three fashions, which control how the "selected values" properties change.

The 3 ways to set up the data dataset and the corresponding behavior is as follows:

Scenario 1: One column with a set of string values

Column1
Apples
Oranges
Bananas

- Drop down displays values from the first column
- **Selected value** is undefined
- **Selected String Value** represents value from first column
- **Selected Label** represents value from first column

Scenario 2: Two column with an integer and string column

Column1	Column2
201	Apples
202	Oranges
203	Bananas

- Dropdown displays values from the second column
- **Selected Value** represents a value from the first column
- **Selected String Value** represents value from first column
- **Selected Label** represents value from first column

Scenario 3: Two column with two string columns

Column1	Column2
APL	Apples
ORN	Oranges
BAN	Bananas

- Dropdown displays values from second column
- **Selected Value** is undefined
- **Selected String Value** represents value from first column
- **Selected Label** represents value from second column

The dropdown component can operate in one of three Selection Modes. These modes affect how the dropdown's current selection (defined by the values of its Selected Value, Selected String Value, and Selected Label properties) behave when the selection properties are set to values not present in the choice list, or conversely, when the choice list is set to a new dataset that doesn't contain the current selection:

• **Strict**. Selected values must always correlate to an option in the list defined by the Data property. If an invalid selection is set (via a binding or a script), the selection will be set to the values defined by the No Selection properties. If the Data property is set to a list that does not contain the current selection, the current selection will be reset to the No Selection values.

• **Lenient**. (default) Selected values are independent of the list defined by the Data property. This mode is useful to avoid race conditions that can cause problems in Strict mode when both the Data and the Selected Value properties are bound. If the current selection is not present in the Data list, the read-only property Selected Index will be -1.

• **Editable**. The same selection rules as defined by Lenient mode, except that the dropdown itself becomes editable, allowing a user to type in their own arbitrary value. This value will be set as the dropdown's Selected Label.

Properties

The component's properties are populated from a sql query. The following properties are from the Alarm Status Table. Change this to the correct component. Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it. Unknown macro: 'sql'

This event occurs when a component that had the input focus lost it to another component. Unknown macro: 'sql'

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is. Unknown macro: 'sql'

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties. Unknown macro: 'sql'

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired. Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component. Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties. Unknown macro: 'sql'

Customizers

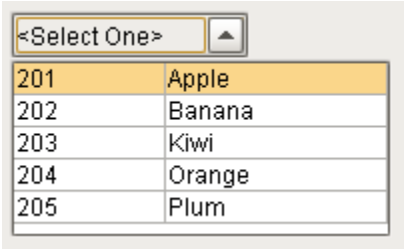
This component does not have any custom properties.

Examples

Code Snippet

```
#The following code will return the value of the slider's previous value into a variable.
#This code is fired on the property change scripting for this component.

oldValue = event.source.oldValue
```

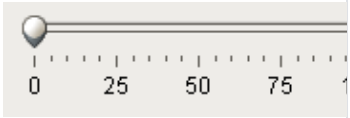
Vertical Slider with Border and Blue Text



Property Name	Value
Dropdown Display Mode	Table
Show Table Header	False

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. The slider can be configured to orient horizontally or vertically.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it.

.source	The component that fired this event.
. oppositeComponent	The other component involved in this focus change. That is, the component that lost focus in order for this one to gain it, or vise versa.

This event occurs when a component that had the input focus lost it to another component.

.source	The component that fired this event
. oppositeComponent	The other component involved in this focus change. That is, the component that lost focus in order for this one to gain it, or vise versa.

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is.

.source	The component that fired this event.
.keyCode	The key code for this event. Used with the `keyPressed` and `keyReleased` events. See below for the key code constants.
.keyChar	The character that was typed. Used with the keyTyped event.
.keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
.altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
.controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
.shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3.

.source	The component that fired this event.
.keyCode	The key code for this event. Used with the `keyPressed` and `keyReleased` events. See below for the key code constants.
.keyChar	The character that was typed. Used with the `keyTyped` event.
.keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
.altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
.controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
.shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

Fires when a key is pressed and then released when source component has the input focus. Only works for characters that can be printed on the screen.

.source	The component that fired this event.
.keyCode	The key code for this event. Used with the `keyPressed` and `keyReleased` events. See below for the key code constants.
.keyChar	The character that was typed. Used with the `keyTyped` event.
.keyLocation	Returns the location of the key that originated this key event. Some keys occur more than once on a keyboard, e.g. the left and right shift keys. Additionally, some keys occur on the numeric keypad. This provides a way of distinguishing such keys. See the KEY_LOCATION constants in the documentation. The keyTyped event always has a location of KEY_LOCATION_UNKNOWN.
.altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
.controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
.shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires *after* the pressed and released events have fired.

.source	The component that fired this event.
.button	The code for the button that caused this event to fire.
.clickCount	The number of mouse clicks associated with this event.
.x	The x-coordinate (with respect to the source component) of this mouse event.
.y	The y-coordinate (with respect to the source component) of this mouse event.
.popupTriger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
.altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
.controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
.shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

This event fires when the mouse enters the space over the source component.

.source	The component that fired this event.
.button	The code for the button that caused this event to fire.
.clickCount	The number of mouse clicks associated with this event.
.x	The x-coordinate (with respect to the source component) of this mouse event.
.y	The y-coordinate (with respect to the source component) of this mouse event.
.popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
.altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
.controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
.shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

This event fires when the mouse leaves the space over the source component.

.source	The component that fired this event.
.button	The code for the button that caused this event to fire.
.clickCount	The number of mouse clicks associated with this event.
.x	The x-coordinate (with respect to the source component) of this mouse event.
.y	The y-coordinate (with respect to the source component) of this mouse event.
.popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
.altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
.controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
.shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

This event fires when a mouse button is pressed down on the source component.

.source	The component that fired this event.
.button	The code for the button that caused this event to fire.
.clickCount	The number of mouse clicks associated with this event.
.x	The x-coordinate (with respect to the source component) of this mouse event.
.y	The y-coordinate (with respect to the source component) of this mouse event.
.popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
.altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
.controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
.shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

This event fires when a mouse button is released, if that mouse button's press happened over this component.

.source	The component that fired this event.
.button	The code for the button that caused this event to fire.
.clickCount	The number of mouse clicks associated with this event.
.x	The x-coordinate (with respect to the source component) of this mouse event.
.y	The y-coordinate (with respect to the source component) of this mouse event.
.popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
.altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
.controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
.shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

Fires when the mouse moves over a component after a button has been pushed.

.source	The component that fired this event.
.button	The code for the button that caused this event to fire.
.clickCount	The number of mouse clicks associated with this event.
.x	The x-coordinate (with respect to the source component) of this mouse event.
.y	The y-coordinate (with respect to the source component) of this mouse event.
.popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
.altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
.controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
.shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

Fires when the mouse moves over a component, but no buttons are pushed.

.source	The component that fired this event.
.button	The code for the button that caused this event to fire.
.clickCount	The number of mouse clicks associated with this event.
.x	The x-coordinate (with respect to the source component) of this mouse event.
.y	The y-coordinate (with respect to the source component) of this mouse event.
.popupTrigger	Returns True (1) if this mouse event is a popup trigger. What constitutes a popup trigger is operating system dependent, which is why this abstraction exists.
.altDown	True (1) if the Alt key was held down during this event, false (0) otherwise.
.controlDown	True (1) if the Control key was held down during this event, false (0) otherwise.
.shiftDown	True (1) if the Shift key was held down during this event, false (0) otherwise.

Fires whenever a *bindable* property of the source component changes. This works for standard and custom (dynamic) properties.

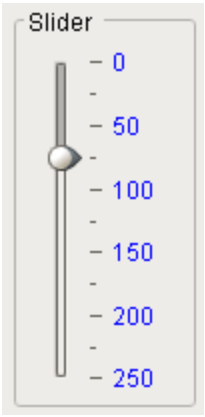
.source	The component that fired this event.
.newValue	The new value that this property changed to.
.oldValue	The value that this property was before it changed.
.propertyName	The name of the property that changed. NOTE: remember to always filter out these events for the property that you are looking for! Components often have many properties that change.

Examples

Code Snippet

```
#The following code will return the value of the slider's previous value into a variable.
#This code is fired on the property change scripting for this component.

oldValue = event.source.oldValue
```

Vertical Slider with Border and Blue Text



Property Name	Value
Maximum Value	250
Minor Tick Spacing	25
Foreground Color	0,0,255
Major Tick Spacing	50

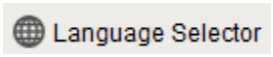
Horizontal Slider without Tickmarks

Property Name	Value
Paint Ticks?	False
Minor Tick Spacing	0
Major Tick Spacing	100

Language Selector

General

Component Palette Icon:



Description

The Language Selector component gives an easy way to set the user's locale to control display of dates, times, numbers, and the language used for translations.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.Remove the panel below if there are no scripting functions.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

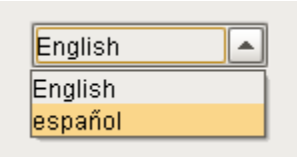
Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

Vertical Slider with Border and Blue Text



Property Name
No property changes made to this component for this example.

Buttons

Button

General

Component Palette Icon:

Description

The Button component is a versatile component, often used for things like opening/closing windows, writing to tags, and triggering any sort of scripting logic. It can be used for showing status, as well. For example, if you have three buttons, Hand, Off, and Auto, not only can they set those modes, but their background color can display the current mode, although you'd be better off using the Multi-State Button for this.

To get buttons to do things, you add an event handler to the *actionPerformed* event. Many new users to the 1.0.0 module will configure an event handler for the mouseClicked event instead. While this will work, it is better to use the actionPerformed event. Why? Buttons can also be activated by tabbing over to them and hitting the space key, or they could be activated by pressing Alt and the button's mnemonic character. So, to make sure that your button works in all of these cases, configure your event handler on the actionPerformed event, not the mouseClicked event.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.Remove the panel below if there are no scripting functions.

.doClick()

- Description

  Virtually "clicks" the button, meaning that its actionPerformed event handler will run.
- Parameters

  Nothing
- Return

  Nothing
- Scope

  Client

Event Handlers

Unknown macro: 'sql'

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it.
This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it. Unknown macro: 'sql'

This event occurs when a component that had the input focus lost it to another component. Unknown macro: 'sql'
An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is. Unknown macro: 'sql'
Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3. Unknown macro: 'sql'
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Unknown macro: 'sql'

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'
This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

Styled Button

Property Name	Value
Border	Etched
Font	Dialog, Bold, 18
Text	Press Me!
Image Path	Builtin/icons/48/check2.png

Styled Button

Property Name	Value
Border	No Border
Fill Area?	False
Border Painted?	False
Text	*None*
Image Path	Builtin/icons/48/stop.png

2 State Toggle

General

Component Palette Icon:

Description

This button is similar to the basic Toggle Button, but more finely tuned to work in realistic controls environments. Use this button any time you want to toggle a value between two states, such as On/Off, Stop/Run, etc. If you have more than two states (for example, Hand/Off/Auto, use the Multi-State Button).

If you have a tag whose value you want to toggle between 2 values (like zero and one), you can simply drag and drop the tag onto the button. This will bind both the Control Value and Indicator Valueproperties to that tag. Now set the State 1 Value and State 2 Value to your two states (they default to zero and one, respectively). Lastly, use the Styles Customizer to define the styles for your two states.

This button has four integer values that you use to set it up: the Control Value, the Indicator Value, and values that define the 2 different states: Sta te 1 Value and State 2 Value. Every time you press the button, one of the state values is written to the control value. The Indicator Value is used to determine which state you're in. For example, suppose that State 1 Value was zero and State 2 Value is one. If Indicator Value is zero and you press the button, it'll write a one to the Control Value. The Style of the component is typically driven by the read-only property Current State. Current State equals zero when Indicator Value=State 1 Value and one otherwise.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it. Unknown macro: 'sql'

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it. Unknown macro: 'sql'

This event occurs when a component that had the input focus lost it to another component. Unknown macro: 'sql'

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is. Unknown macro: 'sql'

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties. Unknown macro: 'sql'

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired. Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component. Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties. Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

Vertical Slider with Border and Blue Text

Property Name	Dataset

Styles

state	animationIndex	animationDurati...	text	buttonBG	foreground
0	0	50	OFF		
1	0	50	ON		
3	0	50	STARTING		

Multi-State Button

General

Component Palette Icon:

Description

This button is really a series of two or more buttons, arranged in a column, row, or grid. 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 then released, its state's value is written to the Control Value.

To configure a Multi-State Button, simply drag a tag that represents your state onto the Multi-State Button. This will bind both the Control Value and Indicator Value to that tag. Now open up the Multi-State Button customizer, and define your states: their order, values and styles. Lastly choose if you want the buttons to be a column, row, or grid by setting the Display Style property.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is.Unknown macro: 'sql'
Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3.Unknown macro: 'sql'
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'
This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component.Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component.Unknown macro: 'sql'

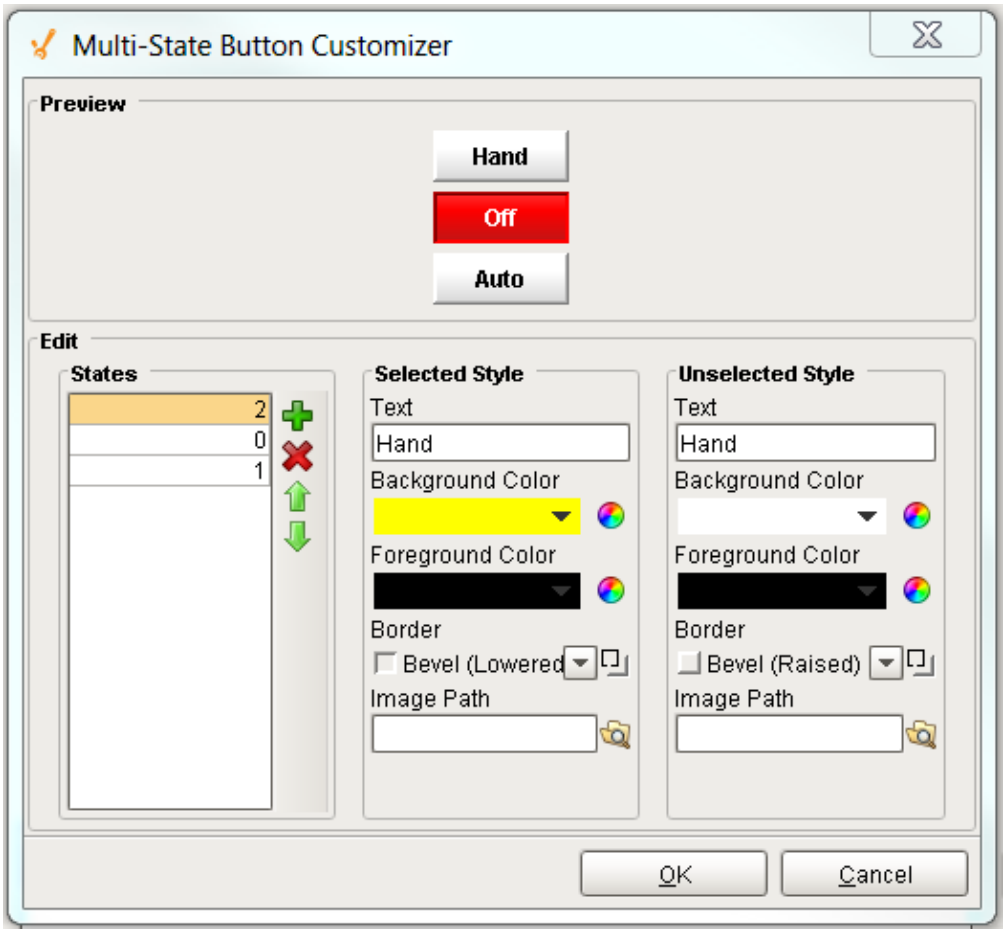
This event fires when a mouse button is pressed down on the source component.Unknown macro: 'sql'
This event fires when a mouse button is released, if that mouse button's press happened over this component.
Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed.Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed.Unknown macro: 'sql'
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

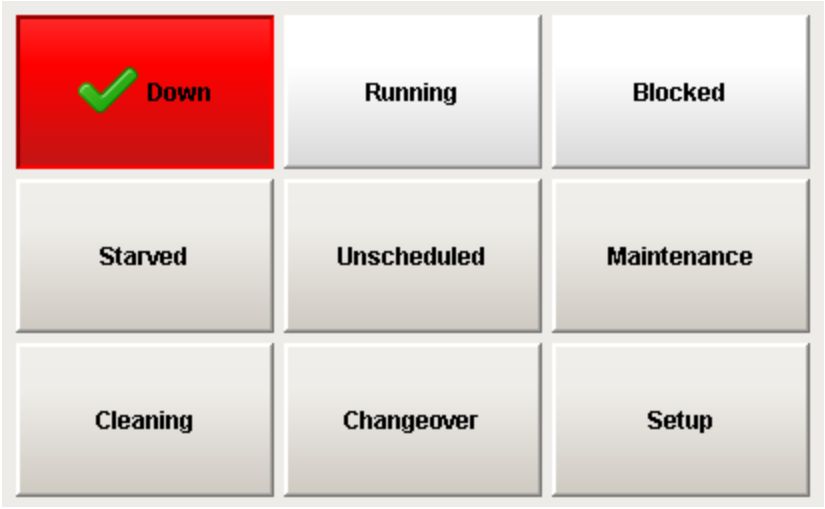
Customizers

The multi-state button customizer provides a way to edit the states dataset in the component's property.



Examples

Stylized Multi-State Button



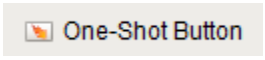
Property Name	Value
Display Style	Grid
Styles	

value	selectedText	unselectedText	selectedBackgr...	unselectedBack...	selectedForegro...	unselectedFore...	selectedBorder	unselectedBorder	selectedImage	unselectedImage
0	Down	Down					border(bevel;1)	border(bevel;0)	Builtin/icons/32/...	
1	Running	Running					border(bevel;1)	border(bevel;0)	Builtin/icons/32/...	
2	Blocked	Blocked					border(bevel;1)	border(bevel;0)	Builtin/icons/32/...	
3	Starved	Starved					border(bevel;1)	border(bevel;0)	Builtin/icons/32/...	
4	Unscheduled	Unscheduled					border(bevel;1)	border(bevel;0)	Builtin/icons/32/...	
5	Maintenance	Maintenance					border(bevel;1)	border(bevel;0)	Builtin/icons/32/...	
6	Cleaning	Cleaning					border(bevel;1)	border(bevel;0)	Builtin/icons/32/...	
7	Changeover	Changeover					border(bevel;1)	border(bevel;0)	Builtin/icons/32/...	
8	Setup	Setup					border(bevel;1)	border(bevel;0)	Builtin/icons/32/...	

One-Shot Button

General

Component Palette Icon:



Description

The One-Shot button is great for telling a PLC to do something. It simply writes a value, and then waits for it to be reset by the PLC before it is available again. Note that this is only applicable when the PLC is programmed to reset the value after reading it. If your PLC expects the HMI to reset the bit, use the Momentary Button. Also note that this component is considered safer than the momentary button, because it receives positive feedback from the PLC that the signal was received, avoiding the timing dangers associated with a Momentary Button.

To use the One-Shot button, bind an OPC tag bidirectionally to the button's Value property. When clicked, the button will write the value in its Set Value property to the Value property. Typically, Set Value is 1, and Value is 0 in a ready state, although the logic could be reversed or change simply by altering Set Value. The button can disable itself when it is writing, and will display different text. Note that the button considers itself to be writing whenever Value equals Set Value - you must make sure that the PLC resets this value, otherwise the button will remain in a writing state.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.Remove the panel below if there are no scripting functions.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it.Unknown macro: 'sql'

This event occurs when a component that had the input focus lost it to another component.Unknown macro: 'sql'

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is.Unknown macro: 'sql'

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3.Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.Unknown macro: 'sql'

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component.Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component.Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component.Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed.Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed.Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

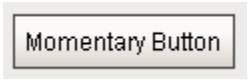
One Shot Button

A One-Shot button, waiting to be pressed

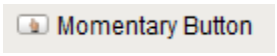
A One-Shot button, waiting for a PLC reset

Momentary Button

General



Component Palette Icon:



Description

Momentary buttons are used to set a value for either a fixed amount of time, or however long the button remains held down, whichever is longer. Once the button is released, or the minimum time expires, the value is reset.

The momentary button uses its Control Value property to affect the underlying data. Typically, this property uses a bidirectional tag binding to an OPC tag. When pressed, it will write its On Value toCont rol Value. When released, it will either write Off Value to Control Value immediately, or wait until On Time has elapsed (since the pressed event).

The button's Indicator Value, which is typically bound to the same OPC tag as Control Value, is used to draw an "active" indication border around the button. This gives the operator positive feedback that the value has written successfully. It also lets an operator at one terminal know if an operator at a different terminal is using the button currently.

Note that you may want to use the Oneshot Button inste ad of the Momentary Button if you simply need to send a signal to a PLC, and the PLC is able to reset the value. This lets the PLC reset the value, avoiding the potential for the bit to be left high. This is possible with the Momentary Button if, for example, the power to the client was cut while the button was held down.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it. Unknown macro: 'sql'

This event occurs when a component that had the input focus lost it to another component. Unknown macro: 'sql'

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is. Unknown macro: 'sql'

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties. Unknown macro: 'sql'

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired. Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component. Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties. Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

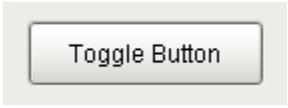
Vertical Slider with Border and Blue Text

Momentary Button waiting to be pressed

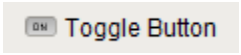
Activated Momentary Button

Toggle Button

General



Component Palette Icon:



Description

The toggle button represents a bit: on (selected) or off (not selected). Visually the button looks down or depressed when it is selected, and up when it is not selected. Logically, this component is very similar to the Check Box component. Note that for implementing a controls screen, the 2 State Toggle is usually more appropriate than this component.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.Remove the panel below if there are no scripting functions.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it.Unknown macro: 'sql'

This event occurs when a component that had the input focus lost it to another component.Unknown macro: 'sql'

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is.Unknown macro: 'sql'

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3.Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.Unknown macro: 'sql'

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component.Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component.Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component.Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed.Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed.Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.Unknown macro: 'sql'

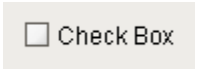
Customizers

This component does not have any custom properties.

Examples

Check Box

General



Component Palette Icon:

Description

A CheckBox is a familiar component that represents a bit - it is either on (selected) or off (not selected). It is functionally equivalent to the Toggle Button component.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it. Unknown macro: 'sql'

This event occurs when a component that had the input focus lost it to another component. Unknown macro: 'sql'

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is. Unknown macro: 'sql'

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties. Unknown macro: 'sql'

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired. Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component. Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

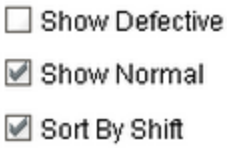
Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties. Unknown macro: 'sql'

Customizers

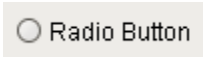
This component does not have any custom properties.

Examples

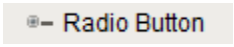


Radio Button

General



Component Palette Icon:



Description

The radio button is similar to the CheckBox component, except for one special property. All radio buttons in the same Container (including the Root Container) will automatically be mutually exclusive. This means that only one radio button can be selected at a time. Radio buttons are a good way to let the user choose just one of a number of options. Dropdown Lists are another good way to do this.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it. Unknown macro: 'sql'

This event occurs when a component that had the input focus lost it to another component. Unknown macro: 'sql'

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is. Unknown macro: 'sql'

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties. Unknown macro: 'sql'

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired. Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component. Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties. Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

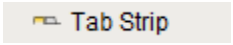
Vertical Slider with Border and Blue Text

Radio buttons inside a container will be exclusive therefore selecting one radio button will de-select the other radio buttons.

Tab Strip

General

Component Palette Icon:



Description

In general, a tab strip is just a single-selection multiple choice component. In practice it is used anywhere that a user needs to be able to select between multiple windows or screens. It is most commonly used in a docked window to provide automatic window navigation. To support this typical use-case, the tab strip has two navigation modes:

1. **Swap to Window.** (default) The tab strip will automatically call system.nav.swapTo() with the name of the selected tab. This facilitates very easy navigation for most common projects.
2. **Disabled.** The tab strip doesn't do anything when the tab selection changes. Users can implement their own via property bindings or by responding to the propertyChange scripting event.

The tab strips visual style is highly customizable. There are different rendering styles, and things such as fonts, colors, line thicknesses, hover colors, and gradients are customizable within each rendering style. Use the Tab Strip's customizer to come up with a style that suits your project, as well as to manage the tabs that are present. The tabs and their styles are all stored in a dataset property (called Tab Data), so they can be modified at runtime as well.

Properties

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

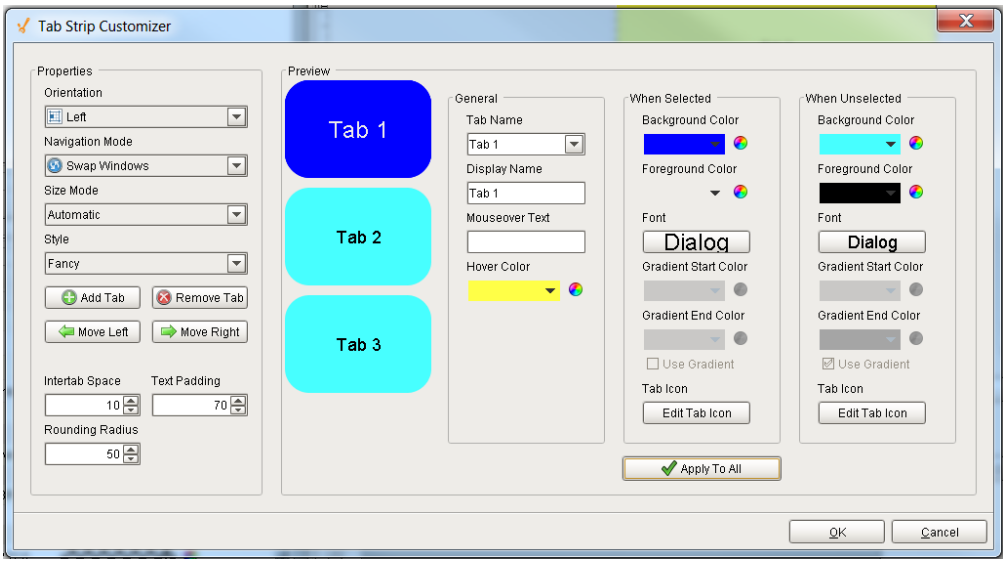
Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Unknown macro: 'sql'

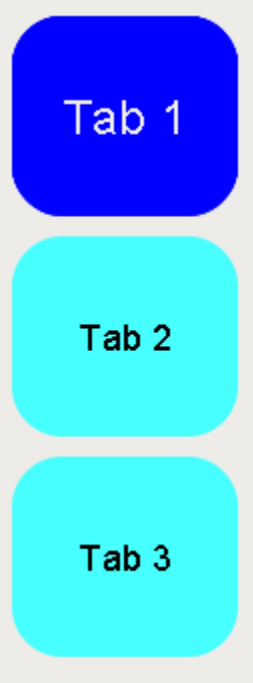
Customizers

This component has a customizer that allows you to change its appearance. The result of these changes are available in the Tab Data dataset property.



Examples

Vertical Slider with Border and Blue Text



Property Name	Value
Renderer	Fancy
Orientation	Left
Tab Data	Dataset customized with the tab strip customizer.

Display

Label

General

Label

Component Palette Icon:

Label

Description

The Label is one of the most versatile components. It can display text, images, or both. Its text can be HTML formatted (like most components). It can even be made to respond to user interaction through its events.

Labels are one of the most common components that you will want to add dynamic properties to. For instance, you can put an integer dynamic property "state" on a label, and then bind the text to be "On" when the state=1 and "Off" otherwise, using an expression binding. Bind the background color to be red when the state is 0, and green when the state is 1 using a property binding. Now you have a re-usable binary state indicator. While you could have used the Multi-State Indicator to achieve the same effect, the exercise is good practice for creating custom components. You can see how the flexibility of bindings and dynamic properties make the Label extremely versatile.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

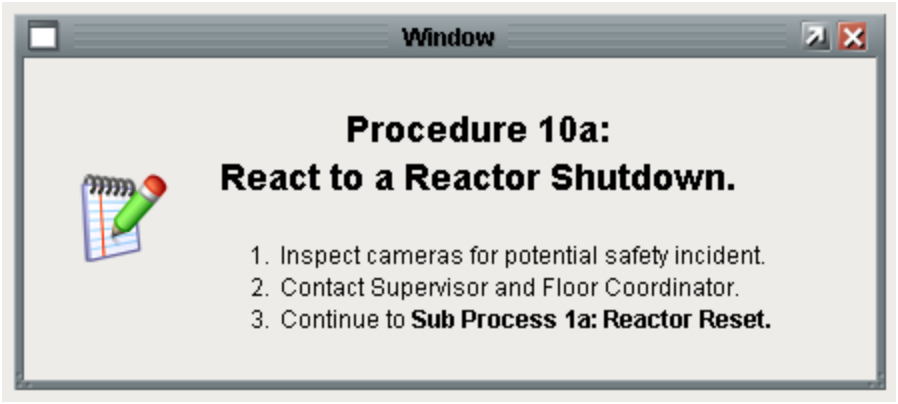
Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

Stylized Label Inside a Popup Window

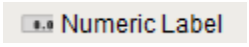


Property Name	Value
Image Path	Builtin/icons/48/edit.png
Text	<html> Procedure 10a: React to a Reactor Shutdown. - Inspect cameras for potential safety incident. - Contact Supervisor and Floor Coordinator. - Continue to Sub Process 1a: Reactor Reset. </html>

Numeric Label

General

Component Palette Icon:



Description

This component is a specialized label designed to display a number. It can include units, and has an integrated number format string. By default the number is displayed bold and the units are not. This can be customized, see the Prefix and Suffix expert properties. This label's text is constructed as follows:

Prefix + numberFormat(Value, Pattern) + Suffix + Units

It is important to note that you could customize the standard Label component using custom properties and bindings to mimic this component exactly. If this component doesn't do something that you need, you can make your own numeric label and use it everywhere in your project.

Properties

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

Numeric label with red background and percent sign

Property Name	Value
Units	%
Background Color	255,0,0

Multi-State Indicator

General

Component Palette Icon:

Description

This component is a specialized label used to display a discrete state. The state must be represented by an integer, but the values and number of different states is customizable. Use the component's styles customizer to configure the different states.

Properties

The component's properties are populated from a sql query. The following properties are from the Alarm Status Table. Change this to the correct component.

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.Remove the panel below if there are no scripting functions.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

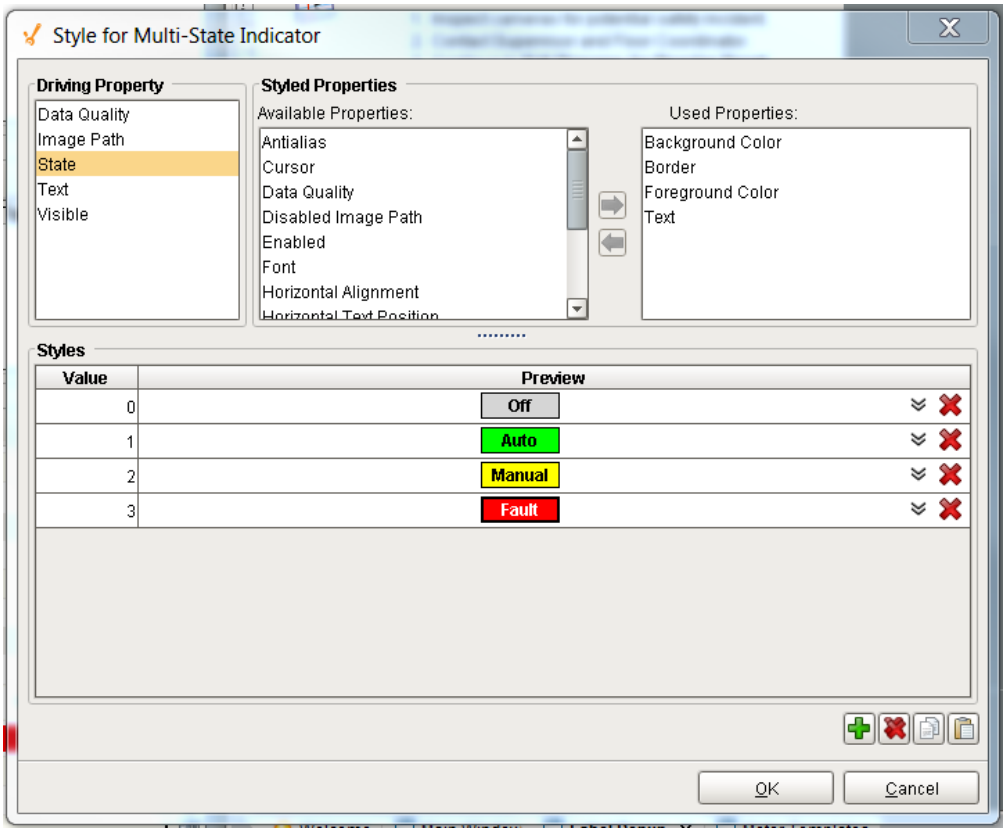
Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Unknown macro: 'sql'

Customizers

This component does not have a customizer however this component relies on custom styles. The example below has the styles defined here:



Examples

Vertical Slider with Border and Blue Text

Property Name	Value
Styles	As defined by the style customizer.

LED Display

General

Component Palette Icon:

Description

The LED display is a stylized numeric or alphanumeric label. It has three different visual styles which all correspond to a kind of physical display: 7-segment, 14-segment, and 5x7 matrix. By default this component is in numeric mode, which means you should use its Value property. If you need to display characters as well, switch the mode to alphanumeric, and use the Text property.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.Remove the panel below if there are no scripting functions.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

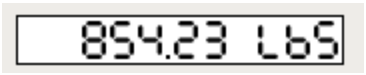
Custom LED Component

Property Name	Value
Mode	Alphanumeric
Text	ERR-28
Background Color	0,0,0
LED Lit	255,0,0
LED Unlit	0,0,0

Custom LED Component

Property Name	Value
Mode	Alphanumeric
Text	Hello World
Horizontal Alignment	Center

Custom LED Component



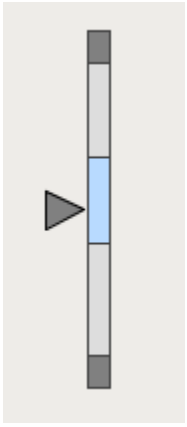
Property Name	Value
Border	Line Border
Mode	Alphanumeric
Text	852.23 lbs
Style	7 Segment
Background Color	255,255,255
LED Lit	0,0,0
LED Unlit	255,255,255

Custom LED Component

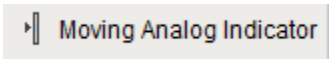
Property Name	Value
Style	5x7 Matrix
Background Color	64,64,64

Moving Analog Indicator.

General



Component Palette Icon:



Description

The moving analog indicator is a component that displays an analog value in context with other information about that value. 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.

This 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 changes color to get attention.

To switch this component between a horizontal vs vertical orientation, simply change the size so that it is either wide or tall, respectively. Typical setup of this component involves setting the ranges, and binding the Process Value property to a tag's value. Some properties may be cleared out (null value) in order to disable them. For example, you may indicate where the current setpoint is by setting the Setpoint Value property. If you don't want to display the setpoint, simply clear this value out.

Properties

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Unknown macro: 'sql'

Customizers

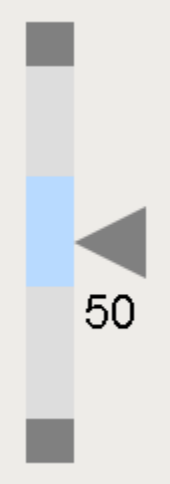
This component does not have any custom properties.

Examples

Moving Analog Indicator Expanded Horizontally

Property Name	Value
None	n/a

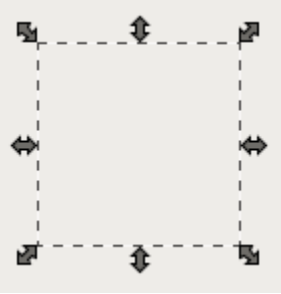
Stylized Moving Analog Indicator



Property Name	Value
Show Value	True
Reverse Indicator	True
Stroke Width	0.0

Image

General



Component Palette Icon:

Description

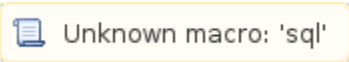
The image component is a deceptively powerful component. While you can use other components, like the Label, to display images as well, this component gives you much more flexibility. In particular, this component has 4 important features for displaying images:

1. Scaling
2. Rotation - Rotate to create spinning animations by binding to a timer component.
3. Color Tinting - Dynamically apply a color tint to an image to allow it to display real-time status
4. Color Swapping - Color swapping to change one specific color in an image to another in real time.

To choose an image, simply press the browse button ( ) next to this component's Image Path property. You can drag new images (*.png, *.gif, *.jpg, *.bmp) into the Image Management window to upload them.

Images are stored on the Gateway, not in your window or project. This means that you can alter an image globally, and it will affect all windows in all projects. It also means that you must be careful to migrate custom images if you do project backups (as opposed to Gateway backups, which will automatically include both projects and images)

Properties



Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

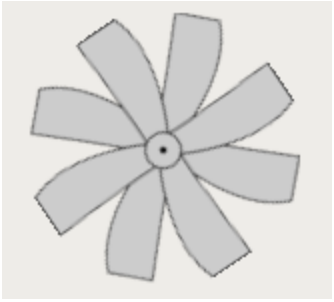
Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

Vertical Slider with Border and Blue Text

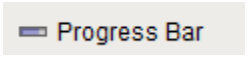


Property Name	Value
Image Path	Builtin/Air Flow/Air Flow 9.png

Progress Bar

General

Component Palette Icon:



Description

Visually indicates the progress of some task. Can be used to display any value that has an upper and lower bound.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

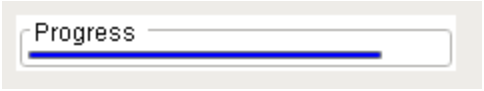
Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

Vertical Slider with Border and Blue Text



Property Name	Value
Border	Titled Border
Value	85
Foreground Color	0,0,255
Horizontal?	True

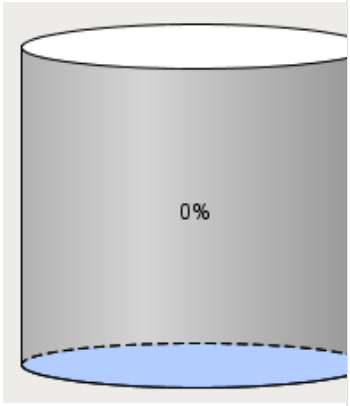
Gallery

Horizontal Slider without Tickmarks

Property Name	Value
Border	Titled Border
Value	85
Foreground Color	0,0,255
Horizontal?	False

Cylindrical Tank

General



Component Palette Icon:

Cylindrical Tank

Description

A component that looks like a 3D cylindrical tank, with some liquid inside. The liquid rises and falls as the Value property changes.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.Remove the panel below if there are no scripting functions.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

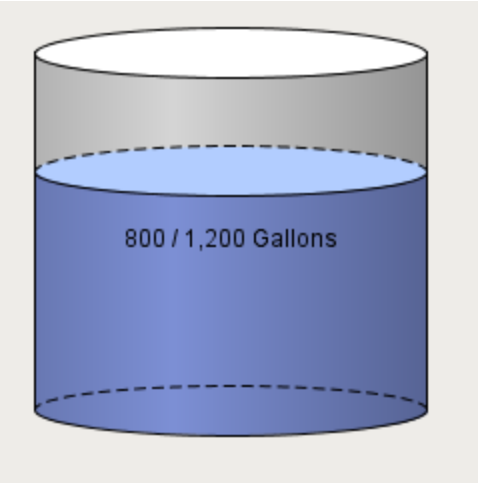
Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

Cylindrical Tank



Property Name	Value
Value	800
Show Value	True
Show Percentage	False

Level Indicator

General

Component Palette Icon:

Level Indicator

Description

A component that looks like a 3D cylindrical tank, with some liquid inside. The liquid rises and falls as the Value property changes.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

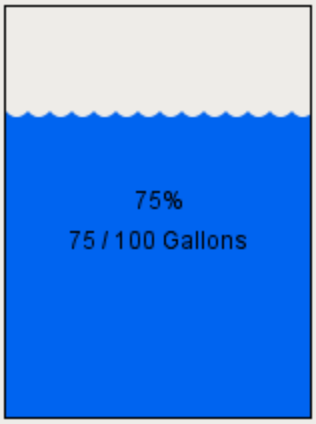
Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

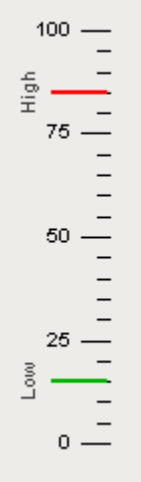
Level Indicator



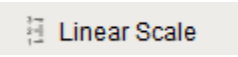
Property Name	Value
Border	Line Border
Value	75
Units	Gallons
Show Value	True
Gradient	False
Filled Color	0,100,240

Linear Scale

General



Component Palette Icon:



Description

The linear scale component has two main purposes. The first is to display a series of tick marks and labels that visually represent a linear range between a minimum value and a maximum value. The second purpose is to display indicators that represent a value or range of values, correctly positioned in on the linear scale. In the example above, two linear scales are used to flank a level indicator. The scale on the left has only tick marks, and no indicators. The scale on the right is used to display three indicators and no tick marks.

To configure the indicators, you use the Linea Scale's "Scale Indicators" customizer. To configure the tick marks, you use the scale's various properties that determine the minimum value, maximum value, and the various tick mark spans.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

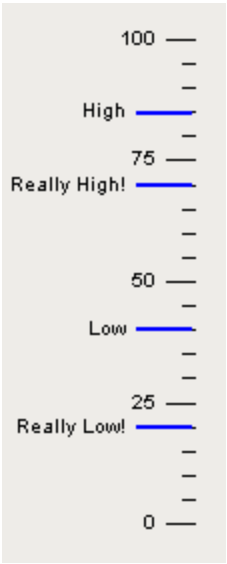
Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

Vertical Slider with Border and Blue Text



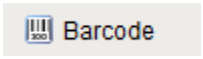
Property Name	Value
Indicators	

Style	Value	Extent	Length	Width	Color	Label	LabelColor	LabelAngle
Line	70	5	30	2		Really High!		0
Line	85	5	30	2		High		0
Line	40	0	30	2		Low		0
Line	20	0	30	2		Really Low!		0

Barcode

General

Component Palette Icon:



Description

The barcode component displays some text as a barcode. The supported formats are:

- Code 128
- Code 39
- Extended Code 39
- Codabar
- Interleaved Code 25
- MSI
- EAN-13
- EAN-8
- Aztec*
- Data Matrix*
- PDF-417*
- QR Code*
- UPC-A*

* Introduced in Ignition 7.8.0

Properties

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

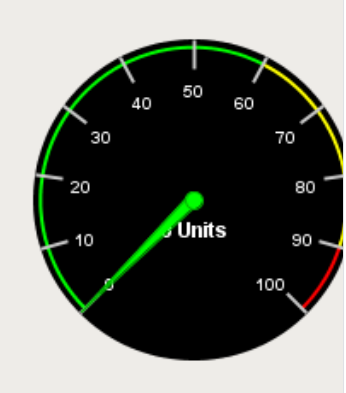
Examples

Barcode

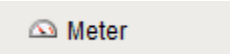
Property Name	Value
Code	123456789
Barcode Format	Extended Code 39 (narrow)
Show Text?	True

Meter

General



Component Palette Icon:



Description

A meter display shows a value on a needle-gauge. The gauge's range can be broken up into five intervals. The intervals can have their own edge and background colors.How the meter looks is affected by its appearance properties.

You can modify colors, thicknesses, start and extend angles, needle size, etc to get the meter that you want. For example, the meter on the far right of the example has a Meter Angle Extent of 90°, a Meter Angle of 45°, a reversed range, and 2 intervals.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.Remove the panel below if there are no scripting functions.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

Updated fonts

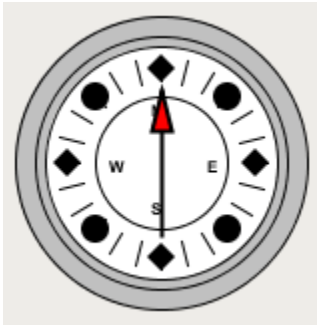
Property Name	Value
Value	35
Unit	m/s
Value Label Font	Vijaya, Bold, 15
Tick Label Font	Vijaya, Bold, 15

Chord Meter with modified value intervals

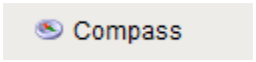
Property Name	Value
Value	35
Units	'None'
Arc Width	10
Meter Angle Extent	220
Dial Shape	Chord
Interval 1 High	40
Interval 2 Low	40
Interval 3 Low	70

Compass

General



Component Palette Icon:



Description

The compass is a component that displays up to three needles at once on a cardinal direction compass. This can be useful for plotting anything that has a cardinal direction, such as the wind direction.

Each needle can be one of 9 different styles. Use the "Disabled" style to turn off any needle.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.Remove the panel below if there are no scripting functions.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

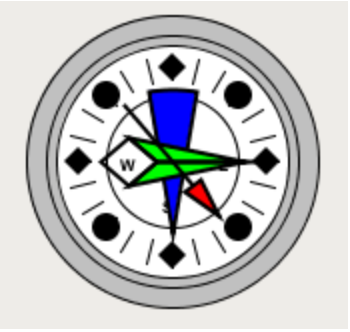
Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

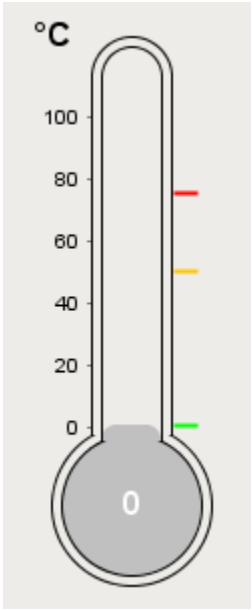
Vertical Slider with Border and Blue Text



Property Name	Value
Value 1	140
Value 1 Needle	Arrow
Value 2	90
Value 2 Needle	Long
Value 3	180
Value 3 Needle	Plum

Thermometer

General



Component Palette Icon:

Thermometer

Description

This component displays a temperature value depicted as a level in a mercury thermometer. Three temperature intervals can optionally be defined with their own colors. The mercury will change color based on the range that it is in.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

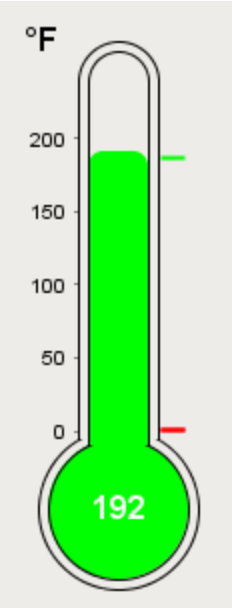
Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

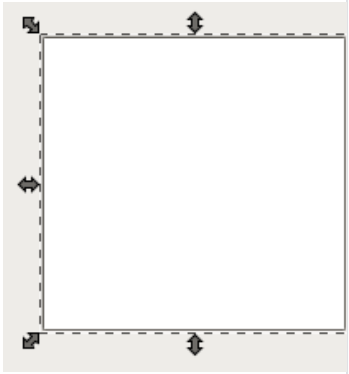
Vertical Slider with Border and Blue Text



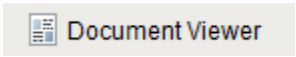
Property Name	Value
Units	Fahrenheit
Value	192
Use Range Color	True
Interval 1 Low	187
Interval 1 High	212
Interval 2 Low	0
Interval 2 High	187
Interval 3 Low	0
Interval 3 High	187

Document Viewer

General



Component Palette Icon:



Description

The document viewer is capable of loading and displaying a document that is available over the network at a URL. It is capable of displaying simple HTML and RTF documents. Although HTML links will be followed, it is not a fully functional interactive web browser. Its HTML support is rudimentary at best, and there is no JavaScript support. See the system.net.openURL function for a more robust solution for launching webpages, PDFs, etc.

This is component is useful for viewing machine manuals or operator protocol in HTML or RTF format. Note that in addition to HTML URLs (like "htt p://www.google.com"), you can load files as well using the URL format for files. Some examples:

- file://localhost/C:/myfolder/file.txt
- file://MyFileServer/resources/manuals/instructions.rtf

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it. Unknown macro: 'sql'

This event occurs when a component that had the input focus lost it to another component. Unknown macro: 'sql'

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is. Unknown macro: 'sql'

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is. Unknown macro: 'sql'

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties. Unknown macro: 'sql'

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired. Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component. Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

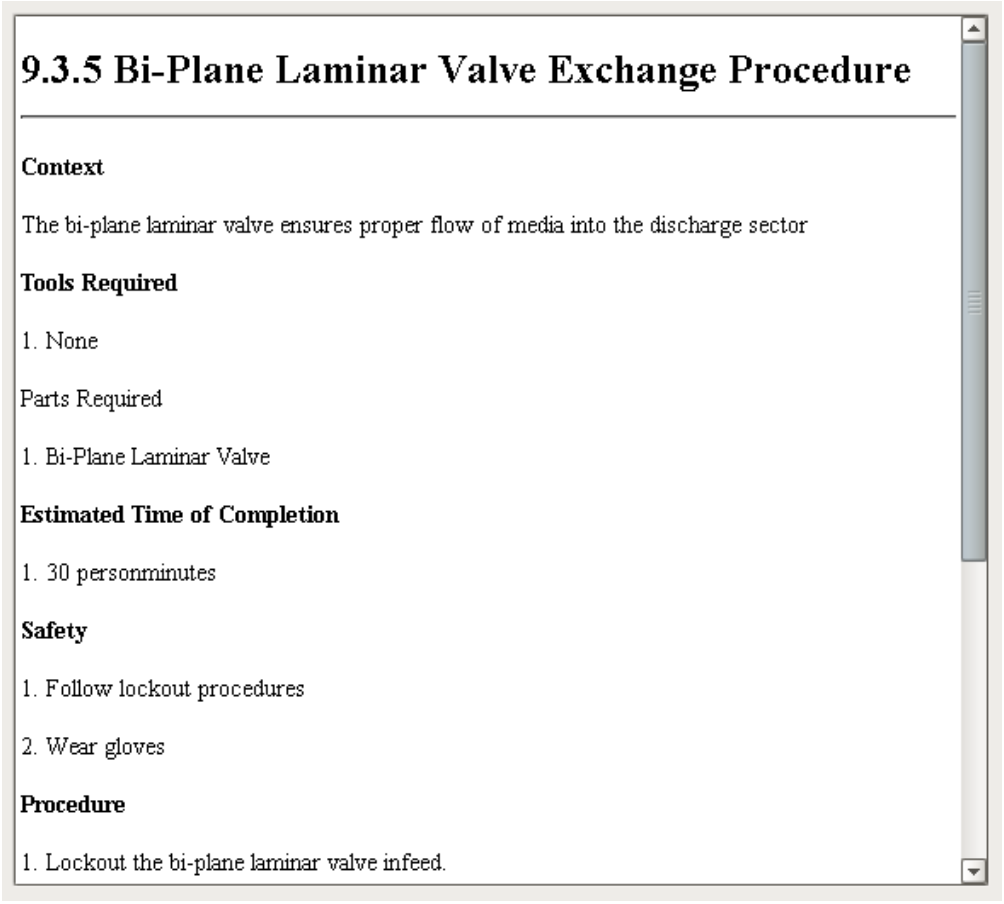
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties. Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

Vertical Slider with Border and Blue Text



9.3.5 Bi-Plane Laminar Valve Exchange Procedure

Context

The bi-plane laminar valve ensures proper flow of media into the discharge sector

Tools Required

1. None

Parts Required

1. Bi-Plane Laminar Valve

Estimated Time of Completion

1. 30 personminutes

Safety

1. Follow lockout procedures

2. Wear gloves

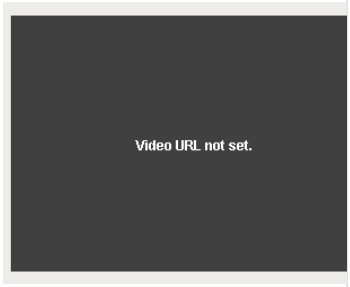
Procedure

1. Lockout the bi-plane laminar valve infeed.

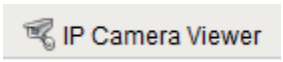
Property Name	Value
Page URL	http://localhost:8088/main/system/webdev/test/Procedures/example.html

IP Camera Viewer

General



Component Palette Icon:



Description

The IP camera viewing component displays a video stream from a network camera directly in one of your windows. This can be a very powerful tool for allowing operators to view remote or inaccessible locations. Cameras can provide positive feedback about the state and position of machinery, weather, and other factors.

This component is capable of displaying two types of video:

- MJPEG (a.k.a. Motion JPEG) is a streaming video protocol that compresses video frames using standard JPEG compression. Compression rates are quite good, requiring low network bandwidth utilization. Framerates depend greatly on the dimensions of the video, but typically range from 1-20 frames per second.
- JPEG stills is not a true video protocol, but is rather the practice of continually refreshing an image that a camera is constantly overwriting. Its simplicity means that many cameras support it (usually along with another protocol). Frame rates are typically lower than MJPEG because a new connection must be opened for each frame.

Most network cameras on the market support one, if not both of these protocols. Even better, if you have an existing CCTV camera system, video server devices are available that CCTV camera inputs and provide MJPEG streams the network.

Finding the URL for your network camera's video stream is usually the only challenge in connecting this component. Most, if not all, network cameras have an internal web server, allowing viewers to use web browsers to view their video stream. If you go to that webpage, and look at the HTML source of the page, you should be able to find the URL of the MJPEG or JPEG still stream.

> **High Resolution Streams**
>
> When viewing a feed from a High Resolution camera, the Camera Buffer Size property may need to be increased to contain all of the data from the stream.

Some examples:

Axis 2100 (MJPEG)

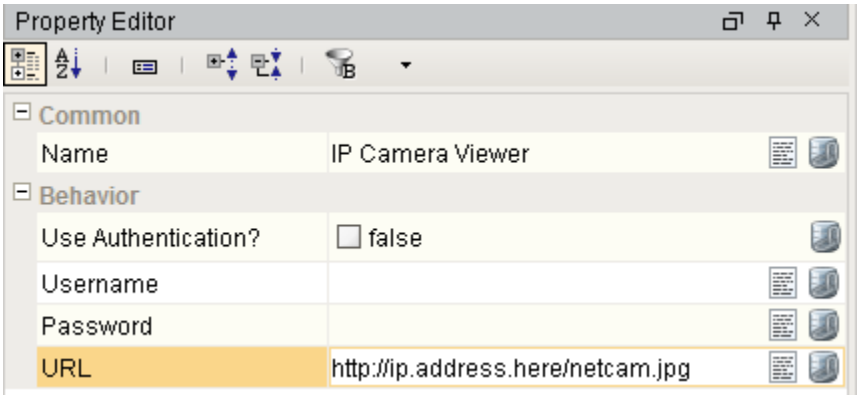
```
http://ip.address.here/axis-cgi/mjpg/video.cgi?resolution=640x480
```

Panasonic BL-C10A (MJPEG)

```
http://ip.address.here/nphMotionJpeg?Resolution=640x480&Quality=Standard
```

StarDot Netcam (JPEG stills)

```
http://ip.address.here/netcam.jpg
```



Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

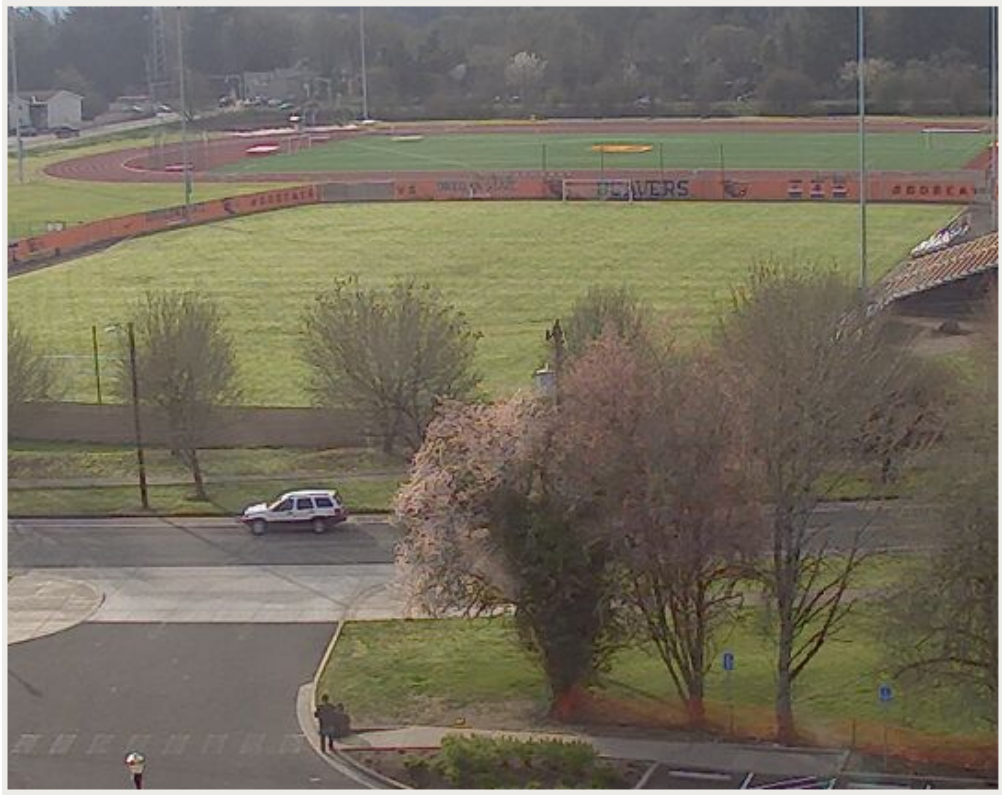
Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

IP Camera Viewer

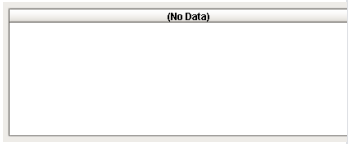


Property Name	Value
URL	http://trackfield.webcam.oregonstate.edu/mjpg/video.mjpg

Tables

Table

General



Component Palette Icon:

Description

The Table component is very powerful and easy to configure. It is very flexible, allowing you to easily display your tabular data in a variety of ways. Important features include:

- Column Sorting. Your users can easily sort the data by clicking on the column headers. The sorting is a 3-mode sort: Ascending, Descending, and "Natural", which uses the default order of the data.
- Mapped Row Coloring. Map the background color of each row to a particular column. This allows you to give powerful visual indication of different types of rows in you tables, such as differentiating between alarm states.
- Column Translation. Allow the table component to handle all code mapping, such as mapping 0 to "Off" and 1 to "On". No fancy SQL knowledge required.
- Images. Map values to images, allowing intuitive visual cues.
- Progress Bar Indication. Display numeric data as progress bars inside cells, providing fast visual reference for bounded amounts.
- Number and Date formatting. Format numbers and dates to your exact specification.
- Column Hiding. Hide columns from view that contain identifying data used by the row coloring or by other components.
- Printing. Print tables directly to multi-paged printouts.
- Editing. Columns can be made editable. Changes will be reflected in the underlying dataset, at which point they can be mapped back to a database.

Basic Usage

The basic usage of the Table is to use a SQL Query binding on its Data property to let the table display data from a database. Often this query will by dynamic or indirect. See the Property Binding section for more information.

Binding to Selected Data

It is common to want to bind other components to values in the selected row of the table. In order to do this safely, you need to write an expression binding that protects against the case when nothing is selected or there are no rows. An expression like this would bind a label to the selected row's value for a column named "ProductCode":

Expression Binding

```
if({Root Container.MyTable.selectedRow} = -1,
        "n/a", // this is the fail case
    {Root Container.MyTable.data}[{Root Container.MyTable.selectedRow},
     "ProductCode"]
)
```

If you're binding to an integer, date, or other non-String type value thats inside a dateset, you'll need to cast the value to the correct type to make the expression parser happy. This binding would cast the selected "Quantity" column to an integer:

Expression Binding

```
if({Root Container.MyTable.selectedRow} = -1,
    -1, // this is the fail case
    toInt({Root Container.MyTable.data}[{Root Container.MyTable.selectedRow},
     "Quantity"])
)
```

Changing the Column Widths

To change a table's column's widths, simply switch into preview mode and use your mouse to resize the columns, and then switch back to design mode. To insure that the changes to the column widths appear in the client, right-click on the table to open the table customizer and click OK without clicking anywhere else in the customizer. Clicking anywhere else in the customizer before clicking OK will reset the table column widths.

Editable Table

By setting any column to editable in the Table's customizer, the user will be able to double-click in the cell and edit the data. You can the respond to the resulting cellEdited event with an event handler and persist the data. See the Event Types section for more information.

Exporting to HTML

You can export the table to an HTML file that retain's the table's formatting. To do this, use a script like this: (more about the table's export HTML function is here.)

Python Scripting

```
# Get a reference to the table
table = event.source.parent.getComponent("Table")

# Prompt user to save the exported file
table.exportHTML("MyTable.html", "My Table Header", 500)
```

Exporting to CSV

You can export the table's raw data to a CSV file. To do this, use a script like this: (more about the fpmi.db.exportCSV function is here.)

Python Scripting

```
# Get a reference to the table
table = event.source.parent.getComponent("Table")
system.dataset.exportCSV("mydata.csv", 1, table.data)
```

Printing

Printing a table is a snap! Simply use the table's built in print function like this: table = event.source.parent.getComponent("Table") # Get a reference to the table table.print()

Python Scripting

```
table = event.source.parent.getComponent("Table") # Get a reference to the table table.print()
```

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

- Description

  Adds a new row to the end of the table's dataset

- Parameters

  PySequence newRow - A sequence containing the values for the new row. The length of the sequence must match the number of columns in the table, and each value must be coercible into the correct datatype of the corresponding column.

- Return

  Nothing

- Scope

  Client

- Description

  Deletes a row from the table's dataset.

- Parameters

  int rowIndex - The index of the row to delete.

- Return

  Nothing

- Scope

  Client

- Description

  Prompts the user to save the table's data as a CSV file.

- Parameters

  String filename - A suggested filename for the user. For example: "table_data.csv"

  boolean showHeaders - If true, include headers in CSV file.

- Return

  String - The path to the saved file, or null if the operation was cancelled.

- Scope

  Client

- Description

  Creates an HTML page as a string in memory. This can then be written to a file, a database, emailed, etc.

- Parameters

  String title - The title for the HTML page.

  int width - The width (in pixels) for the "table" element in the resulting html page.

- Return

  String - A string containing an HTML-formatted version of the table's data.

- Scope

  Client

- Description

  Returns a list of ints that represent the underlying dataset's rows as they appear in the current sort order that the user is viewing.
- Parameters

  none
- Return

  List of Integers
- Scope

  Client
- Description

  Returns the index of the currently selected column, or -1 if none is selected.
- Parameters

  none
- Return

  int
- Scope

  Client
- Description

  Returns the number of columns that are currently selected.
- Parameters

  none
- Return

  int
- Scope

  Client
- Description

  Returns the index of the currently selected row, or -1 if none is selected.
- Parameters

  none
- Return

  int
- Scope

  Client
- Description

  Returns a list of the indexes of the selected row, or none if none is selected.
- Parameters

  none
- Return

  List, None
- Scope

  Client

- Description

  Returns the number of rows that are currently selected.

- Parameters

  none

- Return

  int

- Scope

  Client

- Description

  Tests whether the cell at the given row and column is currently selected or not.

- Parameters

  int row

  int column

- Return

  boolean

- Scope

  Client

- Description

  Tests whether the given column is currently selected or not.

- Parameters

  int column

- Return

  boolean

- Scope

  Client

- Description

  Tests whether the given row is currently selected or not.

- Parameters

  int row

- Return

  boolean

- Scope

  Client

- Description

  This specialized print function will paginate the table onto multiple pages.This function accepts keyword-style invocation.

- Keyword Args

  fitWidth- If true, the table's width will be stretched to fit across one page's width. Rows will still paginate normally. If false, the table will paginate columns onto extra pages. (default = true) [optional]

  HeaderFormat- A string to use as the table's page header. The substring "{0}" will be replaced with the current page number. (default = None) [optional]

  footerFormat- A string to use as the table's page footer. The substring "{0}" will be replaced with the current page number. (default = "Page {0}") [optional]

  showDialog- Whether or not the print dialog should be shown to the user. Default is true. [optional]

  landscape- Used to specify portrait (0) or landscape (1) mode. Default is portrait (0). [optional]

- Return

  boolean- True if the print job was successful.

- Scope

  Client

- Description

  Used to set a column's header label to a new string at runtime.

- Parameters

  int column

  String label

- Return

  nothing

- Scope

  Client

- Description

  Sets the given range of columns to be selected. If index0==index1, it will select a single column.

- Parameters

  int index0

  int index1

- Return

  boolean - True if selection range is valid.

- Scope

  Client

- Description

  Used to set a column's width at runtime.

- Parameters

  int column

  int width

- Return

  nothing

- Scope

  Client

- Description

  Sets the given range of rows to be selected. If index0==index1, it will select a single row.

- Parameters

  int index0

  int index1

- Return

  boolean - True if selection range is valid.

- Scope

  Client

- Description

  Sets the given column to be the selected column.

- Parameters

  int column

- Return

  nothing

- Scope

  Client

- Description

  Sets the given row to be the selected row.

- Parameters

  int row

- Return

  nothing

- Scope

  Client

- Description

  Sets the value in the specified cell, altering the table's Data property. Will fire a propertyChange event for the "data" property, as well as a cellEdited event.

- Parameters

  int row - The index of the row to set the value at.

  int column - The index or name of the column to set a value at.

  PyObject value - The new value to use at the given row/column location.

- Return

  nothing

- Scope

  Client

- Description

  Instructs the table to sort the data by the named column.

- Parameters

  String columnName

  boolean asc - 1 means ascending, 0 means descending. (default = 1) [optional]

- Return

  nothing

- Scope

  Client

- Description

  Instructs the table to clear any custom sort columns and display the data as it is sorted in the underlying dataset.

- Parameters

  nothing

- Return

  nothing

- Scope

  Client

- Description

  Updates an entire row of the table's dataset.

- Parameters

  int rowIndex - The index of the row to update.

  PyDictionary changes - A sequence containing the updated values for the row. The length of the sequence must match the number of columns in the table, and each value must be coercible into the correct datatype of the corresponding column.

- Return

  nothing

- Scope

  Client

Extension Functions

- Description

  Called for each cell, returns the appropriate background color. Do not block, sleep, or execute any I/O; called on painting thread.

- Parameters

  Self- A reference to the component that is invoking this function.

  row-The row index of the cell.

  col-The column index of the cell.

  isSelected: A boolean representing if the cell is currently selected.

  value-The value in the table's dataset at index [row, col].

  defaultColor-The color the table would have chosen if this function was not implemented.

- Return

  Color

- Scope

  Client

- Description

  Called for each cell, returns the appropriate foreground (text) color. Do not block, sleep, or execute any I/O; called on painting thread.

- Parameters

  Self- A reference to the component that is invoking this function.

  row-The row index of the cell.

  col-The column index of the cell.

  isSelected: A boolean representing if the cell is currently selected.

  value-The value in the table's dataset at index [row, col].

  defaultColor-The color the table would have chosen if this function was not implemented.

- Return

  Color

- Scope

  Client

- Description

  Called for each cell, returns the appropriate foreground (text) color. Do not block, sleep, or execute any I/O; called on painting thread.

- Parameters

  Self- A reference to the component that is invoking this function.

  row-The row index of the cell.

  col-The column index of the cell.

  isSelected: A boolean representing if the cell is currently selected.

  value-The value in the table's dataset at index [row, col].

  defaultColor-The color the table would have chosen if this function was not implemented.

- Return

  Color

- Scope

  Client

Event Handlers

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it. Unknown macro: 'sql'

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it. Unknown macro: 'sql'

This event occurs when a component that had the input focus lost it to another component. Unknown macro: 'sql'

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is. Unknown macro: 'sql'

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties. Unknown macro: 'sql'

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired. Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component. Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties. Unknown macro: 'sql'

Customizers

This component has a customizer to manage the column configurations and the background color mapping.

Examples

Code Snippet

```
#The following would add a row to the table.
#Note that this function takes a list
#And that the property types of the list are the same as the table.

name = "Motor 1"
state = 2
amps = 35
list = [name, state, amps]
table = event.source.parent.getComponent('Table')
table.addRow(list)
```

Table Customizer

Column Configuration

Header - Provide a custom name to the column header.

Hide? - Hides the column

Editable - Allows the editing of the cell pertaining to the column.

Sortable - Allows the user to sort the table according to the selected column

Horiz Align - Aligns the contents of the column.

Vert Align - Aligns the contents of the column.

Hdr Horiz Align - Aligns the contents of the column.

Prefix - A custom text that proceeds the contents of each cell.

Suffix - A custom text that follows the contents of each cell.

Number Format - A format of the cell is the contents of the cell are number types.

Boolean? - Changes the contents of the cell to reflect a 'check box' look and feel.

Progress Bar? - A graphical bar is represented in the cell instead of a number.

Progress Bar Range - Sets the min and max range of the progress bar.

Hide Text Over P-Bar? - Makes the value and text that controls the progress bar visible or invisible.

P-Bar Color - The color of the progress bar.

P-Bar Background - The color of the cell that has a progress bar.

Translation List Column - This works in conjunction with the Translation List. The key is provided by a named column resulting in the cells being translated according to the list that contains the key pairs.

Translation List - Defines the key/Translation pairs and translates the contents of the cell accordingly.

Image Path Column - This works in conjunction with the Image Path List. The key is provided by a named column resulting in the cells being translated according to the list that contains the key pairs.

Image Path List - Defines the key/Translation pairs and translates the contents of the cell accordingly.

Background Color Column - This works in conjunction with the Background Color List. The key is provided by a named column resulting in the cells being translated according to the list that contains the key pairs.

Background Color List - Defines the key/Translation pairs and translates the contents of the cell accordingly.

Foreground Color Column - This works in conjunction with the Foreground Color List. The key is provided by a named column resulting in the cells being translated according to the list that contains the key pairs.

Foreground Color List - Defines the key/Translation pairs and translates the contents of the cell accordingly.

Font Map Column - This works in conjunction with the Foreground Color List. The key is provided by a named column resulting in the cells being translated according to the list that contains the key pairs.

Font Map - Defines the key/Translation pairs and translates the contents of the cell accordingly. An example of a font translation could look like this "Dialog, Bold, 12"

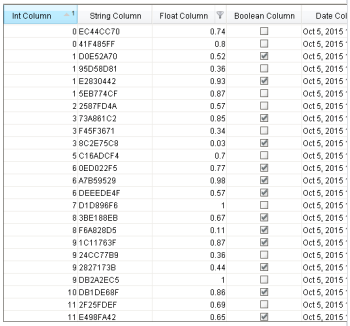
Background Color Mapping

By setting the table's Background property to 'Mapped', you can choose a column to govern the background color of each row. The column is specified in the Mapping Column dropdown selector. The column must be a numeric type.

The number to color translation works with the contents of the mapping column rows to format the cells in accordance with the selected color.

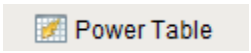
Power Table

General



Int Column	String Column	Float Column	Boolean Column	Date Col
0	EC44CC70	0.74	☐	Oct 5, 2015
0	41F485FF	0.8	☐	Oct 5, 2015
1	D0E52A70	0.52	☑	Oct 5, 2015
1	95D58D81	0.36	☐	Oct 5, 2015
1	E2630442	0.93	☑	Oct 5, 2015
1	5EB774CF	0.87	☐	Oct 5, 2015
2	2587FD4A	0.57	☐	Oct 5, 2015
3	73A861C2	0.85	☑	Oct 5, 2015
3	F45F3671	0.34	☐	Oct 5, 2015
3	8C2E75C8	0.03	☑	Oct 5, 2015
5	C16ADCF4	0.7	☐	Oct 5, 2015
6	0ED022F5	0.77	☑	Oct 5, 2015
6	A7B59529	0.98	☑	Oct 5, 2015
6	DEEEDE4F	0.57	☑	Oct 5, 2015
7	D1D896F6	1	☐	Oct 5, 2015
8	3BE188EB	0.67	☑	Oct 5, 2015
8	F6A828D5	0.11	☑	Oct 5, 2015
9	1C11763F	0.87	☑	Oct 5, 2015
9	24CC77B9	0.36	☐	Oct 5, 2015
9	2827173B	0.44	☑	Oct 5, 2015
9	DB2A2EC5	1	☐	Oct 5, 2015
10	DB1DE68F	0.86	☑	Oct 5, 2015
11	2F25FDEF	0.69	☐	Oct 5, 2015
11	E498FA42	0.65	☑	Oct 5, 2015

Component Palette Icon:



Description

The power table is a much more customizable version of the table component, and has many more features. The power table contains advanced features such as drag-and-drop rows, multi-column sorting, column filtering, and cell-spanning. Customization comes through extensive use of exte nsion functions, which are available to configure how each cell of the table looks, how the headers look, etc.

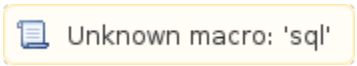
Basic Usage

The basics are just like the classic table - you simply bind the table's "data" property to your data, most often by using a SQL query binding. Note that many of the options built into the classic table have been moved to extension functions in the power table.

Power Table Features

- **Multi-column sorting.** To sort multiple columns, select the header of the first column, hold down the Control key, then select the header of the next column. Click on the header again to reverse the sort order, and click a third time to remove sorting on the column.
- **Column filtering.** Columns can be temporarily hidden from view using column filtering. Right-click on the header of the table, and uncheck columns that you would like to hide. You can disable this feature by disabling the Column Chooser Menu property on the table.
- **Column reordering.** You can switch the locations of columns on the table using column reordering. Drag the header of the column that you would like to move to a new location on the table. You can disable this feature by disabling the Columns Re-Orderable property on the table.
- **Cell spanning.** A cell can be spanned across multiple columns and rows. Keep in mind that you must explicitly define the locations of cells that must be spanned. This means that if you would like to use cell spanning, any other table features that change how the table is displayed will be disabled automatically (such as sorting, column filtering and column reordering). Click on the Cell Span Data dataset to configure spanning. Within the dataset, add a row for each new span. The "row" column controls the row in the table where the span will start. The "column" column controls the column where the span will start. The "width" column controls how many columns the span will cover. The "height" column controls how many rows the span will cover. Adding a row where "row=4, column=1, width=2, height=3" results in a span starting on the fifth row of the table and the second column (using 0-based indexing). The span will cover the second and third columns in the row and will also cover two rows below the fifth row, as shown below.
- **Drag and Drop.** This feature allows you to drag rows from one power table to another power table. In order to perform drag and drop, you must implement the onRowsDropped() extension function on the destination table. This is so that you can adapt the data from one table to the other within the function. You must also enable the Row Dragging Enabled property on both tables.

Properties



Scripting

Scripting Functions

- Description

  Returns a list of ints representing the currently selected columns.

- Parameters

  none

- Return

  List of Integers

- Scope

  Client

- Description

  Returns a list of ints representing the currently selected rows.

- Parameters

  none

- Return

  List of Integers

- Scope

  Client

- Description

  This specialized print function will paginate the table onto multiple pages.This function accepts keyword-style invocation.

- Keyword Args

  fitWidth- If true, the table's width will be stretched to fit across one page's width. Rows will still paginate normally. If false, the table will paginate columns onto extra pages. (default = true) [optional]

  HeaderFormat- A string to use as the table's page header. The substring "{0}" will be replaced with the current page number. (default = None) [optional]

  footerFormat- A string to use as the table's page footer. The substring "{0}" will be replaced with the current page number. (default = "Page {0}") [optional]

  showDialog- Used to determine if the print dialog should be shown to the user. Default is true. [optional]

  landscape- Used to specify portrait (0) or landscape (1) mode. Default is portrait (0). [optional]

- Return

  boolean- True if the print job was successful.

- Scope

  Client

- Description

  Used to set a column's width at runtime.

- Parameters

  int Column

  int width

- Return

  Nothing

- Scope

  Client

Extension Functions

- Description

  Provides a chance to configure the contents of each cell. Returns a dictionary of name-value pairs with the desired attributes. Available attributes (and their Java types) include: 'background' (color), 'border' (border), 'font' (font), 'foreground' (color), 'horizontalAlignment' (int), 'iconPath' (string), 'text' (string), 'toolTipText' (string), 'verticalAlignment' (int).

- Parameters

  self - A reference to the component that is invoking this function.

  value - The value in the dataset at this cell.

  selected - A boolean indicating whether this cell is currently selected.

  rowIndex - The index of the row in the underlying dataset

  colIndex - The index of the column in the underlying dataset

  colName - The name of the column in the underlying dataset

  rowView - The index of the row, as it appears in the table view (affected by sorting)

  colView - The index of the column, as it appears in the table view (affected by column re-arranging and hiding)

- Return

  Dictionary of Attributes

- Scope

  Client

- Description

  Provides a change to configure how each column is edited. Returns a dictionary of name-value pairs with desired editor attributes. Visual attributes to modify existing editors include: 'background', 'border', 'font', 'foreground', 'horizontalAlignment', 'toolTipText', and 'verticalAlignment'

  If the attribute 'options' is specified, it is expected to be a list of tuples representing (value, label). The editor in this case will become a dropdown list.

  If the attribute 'editor' is specified, it is expected to be an instance of javax.swing.table.TableCellEditor, and other attribute will be ignored.

- Parameters

  self - A reference to the component that is invoking this function

  colIndex - The index of the column in the underlying dataset

  colName - The name of the column in the underlying dataset

- Return

  Dictionary of name value pairs

- Scope

  Client

- Description

  Provides a chance to configure the style of each column header. Return a dictionary of name-value pairs with the designed attributes. Availible attributes include: 'background', 'border', 'font', 'foreground', 'horizontalAlignment', 'toolTipText', 'verticalAlignment'

- Parameters

  self - A reference to the component that is invoking this function

  colIndex - The index of the column in the underlying dataset

  colName - The name of the column in the underlying dataset

- Return

  Dictionary of name value pairs

- Scope

  Client

- Description

  Called when the window containing this table is opened, or the template containing it is loaded. Provides a change to initialize the table further, for example, selecting a specific row.

- Parameters

  self - A reference to the component that is invoking this function

- Return

  Nothing

- Scope

  Client

- Description

  Called when the user has edited a cell in the table. It is up to the implementation of this function to alter the underlying data that drives the table. This might mean altering the dataset directly, or running a SQL UPDATE query to update data in the database.

- Parameters

  self - A reference to the component that is invoking this function

  rowIndex - Index of the row that was edited, relative to the underlying dataset

  colIndex - Index of the column that was edited, relative to the underlying dataset

  colName - Name of the column in the underlying dataset

  oldValue - The old value at the location, before it was edited

  newValue - The new value value input by the user

- Return

  Nothing

- Scope

  Client

- Description

  Called when the user double-clicks on a table cell.

- Parameters

  self - A reference to the component that is invoking this function

  rowIndex - Index of the row, starting at 0, relative to the underlying dataset

  colIndex - Index of the column starting at 0, relative to the underlying dataset

  value - The value at the location clicked on

  event - The MouseEvent object that caused this double-click event

- Return

  Nothing

- Scope

  Client

- Description

  Called when the user right-clicks on a table cell. This would be the appropriate time to create and display a popup menu.

- Parameters

  self - A reference to the component that is invoking this function

  rowIndex - Index of the row, starting at 0, relative to the underlying dataset

  colIndex - Index of the column starting at 0, relative to the underlying dataset

  value - The value at the location clicked on

  event - The MouseEvent object that caused this double-click event

- Return

  Nothing

- Scope

  Client

- Description

  Called when the user has droppoed rows on this table. Note that the rows may have come from this table or another table. The source table must have dragging enabled.

- Parameters

  self - A reference to the component that is invoking this function

  sourceTable - A reference to the table that the rows were dragged and dropped in the same table.

  rows - An array of the rows indices that were dragged, in the order they were selected

  rowData - A dataset containing the rows that were dragged

  dropIndexLocation - Row index where the rows were dropped

- Return

  Nothing

- Scope

  Client

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Unknown macro: 'sql'

Customizers

This component has a table customizer that allows customization of the individual columns including hiding columns, enabling editing, changing format, etc. It is important to note that when editing cells directly in the Power Table, it doesn't modify the underlying Dataset. You can use the onCellEdited extension function and uncomment the sample code to make table edits change the underlying Dataset, or even the original source of data (ie: if using a SQL Query).

Examples

Code Snippet

```
#Example of an onRowsDropped() extension script for two power tables with identical columns:

def onRowsDropped(self, sourceTable, rows, rowData, dropIndexLocation):
    if self != sourceTable:
        destDataset = self.getData()
        pyRowData = system.dataset.toPyDataSet(rowData)
        # Loop thru all the rows that have been selected and dragged to the
        # destination table.
        for row in pyRowData:
            newRow = []
            for column in row:
                newRow.append(column)
            destDataset = system.dataset.addRow(destDataset, dropIndexLocation, newRow)
        # Adds the rows to the destination table.
        self.setData(destDataset)
        # Optional. Deletes the dragged rows from the source table.
        sourceDataset = system.dataset.deleteRows(sourceTable.getData(), rows)
        sourceTable.setData(sourceDataset)
    else:
        system.gui.messageBox("Dropping on to same table not supported")
        # To drop onto the same table, the new row indices must be calculated
        # for both the dropped and deleted rows, taking changes into account.
```

Power Table Customizer

Column Configuration

Header - Provide a custom name to the column header.

Hide - Hides the column

Editable - Allows the editing of the cell pertaining to the column.

Sortable - Allows the user to sort the table according to the selected column

Filterable - Allows the user to filter the table according to the selected column

Horiz Align - Aligns the contents of the column.

Vert Align - Aligns the contents of the column.

Wrap Text? - The text will wrap if its contents are longer than the width of the cell.

Prefix - A custom text that proceeds the contents of each cell.

Suffix - A custom text that follows the contents of each cell.

Number Format - A format of the cell is the contents of the cell are number types.

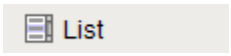
Date Format - A format of the cell is the contents of the cell are date types.

Boolean - Changes the contents of the cell to reflect a 'check box' look and feel.

List

General

Component Palette Icon:



Description

The List component displays a list of options, allowing freeform selection of the items. It is powered by a Dataset, from which it displays the first column.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

- Description

  Adds the options at indexes start through end (inclusive) to the selected options.

- Parameters

  int start - The first index (stating at 0) to add to the selection.

  int end - The last index (stating at 0) to add to the selection.

- Return

  Nothing

- Scope

  Client

- Description

  Clears the current selection, making nothing selected.
- Parameters

  Nothing
- Return

  Nothing
- Scope

  Client
- Description

  Returns a list of the selected indices in increasing order. Returns an empty list if nothing is selected.
- Parameters

  Nothing
- Return

  List of Integers
- Scope

  Client
- Description

  Returns the currently selected value, or None if the selection is empty.
- Parameters

  Nothing
- Return

  Object
- Scope

  Client
- Description

  Returns a list of the currently selected values. Returns an empty list if the selection is empty.
- Parameters

  Nothing
- Return

  Object[]
- Scope

  Client
- Description

  Checks whether or not the given index is currently selected.
- Parameters

  int index
- Return

  boolean
- Scope

  Client

- Description

  Checks to see if anything is selected in the list or not.
- Parameters

  Nothing
- Return

  boolean
- Scope

  Client
- Description

  Sets the currently selected value to the argument, if found in the list.
- Parameters

  Object value
- Return

  Nothing
- Scope

  Client

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it. Unknown macro: 'sql'

This event occurs when a component that had the input focus lost it to another component. Unknown macro: 'sql'

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is. Unknown macro: 'sql'

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties. Unknown macro: 'sql'

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired. Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component. Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties. Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

Code Snippet

```
#The following code will print the selected value to the console when called on the 'mouseClicked' event handler.
value = event.source.getSelectedValue()
print value
```

Tree View

General

Component Palette Icon:

Description

The Tree View component can display any tree hierarchy. It is configured by filling in a dataset. Each row in the dataset will become a node in the tree. Each node has a path, for example, "West Area/Process/Valve1" that determines its location in the tree. The Separation Character property (by default it is forward-slash), dictates how the paths are broken up. Any missing folder nodes needed by a leaf node are created implicitly.

The other columns in the dataset besides "Path" are used to configure the look for the node, both when it is selected and when it is not. This component recognizes the following column titles. The references to optional column titles means that a dataset does not need to have them present in the dataset for the tree to render and function.

- Path - the path determines the node's location. Broken up into a list by splitting on the separation character.
- Text - the text of the node while not selected.
- Icon - a path to an icon for the node. Use the value: "default" to use the tree automatic folder/leaf icons. [optional]
- Background - a string column that will be coerced into a color for the unselected background. e.g. "white" or "(255,255,255)" Use an empty string to use the default color.[optional]
- Foreground - a string representation of the unselected foreground color. [optional]
- Tooltip - if not empty, will be used as the tooltip for the node.[optional]
- Border - a string that will be coerced into a Border for the node while unselected. May be empty.[optional]
- SelectedText - the text of the node while selected.[optional]
- SelectedIcon - a path to an icon for the node while selected. Use the value: "default" to use the tree automatic folder/leaf icons.[optional]
- SelectedBackground - a string representation of the selected foreground color.[optional]
- SelectedForeground - a string representation of the selected foreground color.[optional]
- SelectedTooltip - if not empty, will be used as the tooltip for the node while selected.[optional]
- SelectedBorder - a string that will be coerced into a Border for the node while selected. May be empty.[optional]

Below is an example configuration of the treeview's items property. Notice how not all of the fields listed above are used, because there are certain properties that are not necessary to build our treeview. Notice how I have chosen a larger version of the same images for the Selected Icon, so that when an item gets selected, not only does the background color change, but the size of the image changes as well.

Path	Text	Icon	Background	Foreground	SelectedText	SelectedIcon	SelectedBackground	SelectedForeground
HMI Screens	Overview	Builtin /icons /16 /home. png	color (255,255,255,25 5)	color(0,0,0,255)	Overview	Builtin/icons/24 /home.png	color(250,214,138,255)	color(0,0,0,255)
Administ ration /Users	User Manage ment	Builtin /icons /16 /users3. png	color (255,255,255,25 5)	color(0,0,0,255)	User Management	Builtin/icons/24 /users3.png	color(250,214,138,255)	color(0,0,0,255)
Administ ration /Users	Schedul e Manage ment	Builtin /icons /16 /calend ar.png	color (255,255,255,25 5)	color(0,0,0,255)	Schedule Management	Builtin/icons/24 /calendar.png	color(250,214,138,255)	color(0,0,0,255)
Administ ration	Roster Manage ment	Builtin /icons /16 /clock. png	color (255,255,255,25 5)	color(0,0,0,255)	Roster Management	Builtin/icons/24 /clock.png	color(250,214,138,255)	color(0,0,0,255)

? Unknown Attachment

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

- Description

  Clears the current selection.

- Parameters

  Nothing

- Return

  Nothing

- Scope

  Client

- Description

  Collapses all nodes in the tree.

- Parameters

  Nothing

- Return

  Nothing

- Scope

  Client

- Description

  Expands all nodes in the tree.

- Parameters

  Nothing

- Return

  Nothing

- Scope

  Client

- Description

  Returns a list of the selected item's indexes. These are the row indexes that the selected tree nodes were found in the underlying dataset. Implicitly created folder nodes that have no index will not be included.

- Parameters

  Nothing

- Return

  List of Integers

- Scope

  Client

- Description

  Returns a list of the selected item's paths. A path to an item is the path to its parent plus its normal (non-selected) text.

- Parameters

  Nothing

- Return

  List of Strings

- Scope

  Client

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component.

Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component.

Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component.

Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed.

Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed.

Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Unknown macro: 'sql'

Customizers

The tree view customizer allows for easy custom manipulation of the tree view components underlying formatting.

Examples

Expression Snippet

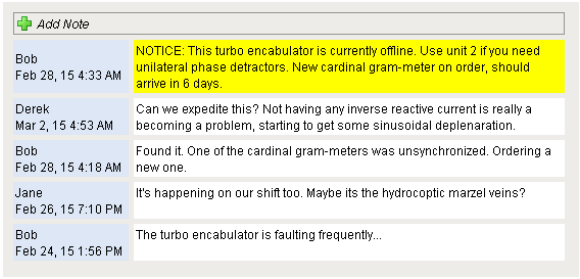
```
//The Selected Item property will be updated as the user selects different nodes in the tree.
//It represents the index in the Items dataset at which the node is defined. If the selected
//node was implicitly created, the Selected Item will be -1.
//You can use this index to get the path and name of the selected node with an expression binding like
this:
if ({Root Container.Tree View.selectedItem}<0,"n/a",{Root Container.Tree View.data}[{Root Container.Tree
View.selectedItem},"text"])
```

Script Snippet

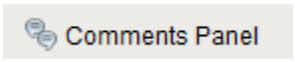
```
#This script will swap to the script that was double clicked on, if this code is placed in the
mouseClicked event handler for the treeview
#This script utilizes an extra column called windowPath that contains the full path to the window. You
can add an extra column to the Items dataset property
#as long as the column name doesn't match one of the reserved column titles listed above.
if event.clickCount == 2:
        row = event.source.selectedItem
        data = event.source.data
        if row != -1:
                # Grab the window path value out of the tree view's items dataset
                windowPath = data.getValueAt(row, "windowPath")
                system.nav.swapTo(windowPath)
```

Comments Panel

General



Component Palette Icon:



INDUCTIVE UNIVERSITY

Comments Panel

Watch the Video

The following feature is new in Ignition version **7.8.1**
Click here to check out the other new features

Additional information on the Comments Panel can be found on the Comments Panel Component page.

Looking for documentation on the legacy Comments Panel component? Please see the Legacy Comments Panel page.

Not sure which version you are looking at? The Legacy version of this component has several properties that the new one does not: "Insert Query 1", "Insert Query 2", "Delete Query", "Unstick Query", and "Download Attachment Query".

Description

The comments panel is used to power a blog-style comments system within your project. This can be useful for ad-hoc collaboration and communication between shifts, remote users, etc. This component is driven by a dataset that should be bound to a SQL query. Unlike most components, this component has built-in functionality to alter an external database. This is how the default Add Note functionality works. You have the opportunity to alter the queries that the components uses by enabling Extension Functions.

The schema that typically drives this component involves up to three tables:

- The first table (by default: **Notes**) stores all of the notes across the board.
- The second table (by default, **ItemNotes**) is used to associate notes with other things. This allows you to have different sets of notes for different screens/objects. Typically you'd bind the data to a query that joined these tables together restricting the second identifier in the ItemNotes table to the value appropriate for the window you're on.
- The third table (by default: **users**) is a user mapping table that assigns an ID to each user on the table. This is easiest to do if a database authentication profile is used as the _users table automatically creates the required columns, but non-database authentication profiles can be used as long as the table is manually created and maintained.

You can opt out of this three-table system by simply making use of the Extension Functions on the component. See below for more details.

By default, users can remove their own comments, and comments can have files attached. To allow attachments, make sure you have a BLOB field in your notes table.

Behavior Description: Three-Table Configuration vs Custom

The following is a list of the default behaviors when all Extension Functions are disabled

Behavior	Description
Adding a note	Runs a SELECT query against the users table to retrieve the ID value for the user that added the note. Then stores this ID into the Notes table along with the rest of the note data
Deleting a note	Deletes a row from the Notes table. Uses the ID value from the Notes table to determine which row should be deleted. Does not delete a row from the ItemNotes table
Unsticking a note	Updates the sticky column for the note on the Notes table. Sets the value to 0 based on the ID of the note
Download an attachment from a note	Returns the binary data from the Attachment column on the Notes table.
Ability for users to delete notes	Users may not delete notes from other users.

Enabling the Extension Functions on the component allow for custom functionality on the component. Some examples are:

- Store all note data on a single database table - simply modify each Extension Function to run queries against a single database table
- Save the attachment to a shared drive instead of a database column - modify insertNote to save the attachment to a hard drive.
- Allow users to delete all notes by role - check the role of the user in canDelete and return True if the user has a specific role

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

- Description

  Called when a note is added.

- Parameters

  component self - A reference to the component that is invoking this function

  string note - The text contents of the note

  string filename - The full filepath to the the attachment

  string sticky - A boolean indicating whether this note should be flagged as stickied

- Return

  Nothing

- Scope

  Client

- Description

  Called when a user clicks the 'delete' link on a note.

- Parameters

  component self - A reference to the component that is invoking this function

  integer id - The id of the note

- Return

  Nothing

- Scope

  Client

- Description

  Called when a user clicks the 'unstick' link on a note.

- Parameters

  component self - A reference to the component that is invoking this function

  integer id - The id of the note

- Return

  Nothing

- Scope

  Client

- Description

  Called when a user attempts to download an attachment from a note.

- Parameters

  component self - A reference to the component that is invoking this function

  integer id - The id of the note

- Return

  Nothing

- Scope

  Client

- Description

  Returns whether or not a note with the given id can be deleted. Notes that return True will show a 'delete' link.

- Parameters

  component self - A reference to the component that is invoking this function

  integer id - The id of the note

- Return

  boolean - Notes with a True return can be deleted by the user, False return can not be deleted.

- Scope

  Client

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Unknown macro: 'sql'

Examples

The following examples may need to be modifed to match the table and column names in your database.

insertNote: using default table configuration

```
# Inserts a note using the three default tables: notes, users, and itemNotes.
# Also stores only the file name in the database instead of the full path to the file
# Assumes a User ID is used in the notes table

# determine the ID for the logged in user
user = system.db.runScalarPrepQuery("SELECT id from users where username = ?", [system.security.
getUsername()])

# determine if a file is being attached
if filename is None:
        # a file was not attached, provide a blank for the bytes
        attachmentBytes = None
else:
        # get the bytes of the file at the path the user selects
        attachmentBytes = system.file.readFileAsBytes(filename)

        # splits the file name from the file path. This way we can show just the file name on the
component
        # Using '\' as a delimiter, but python requires 2 since it's an escape character
        pathAndFile = filename.rsplit('\\', 1)
        filename = pathAndFile[1]

# build the query and the arguments
query = "INSERT INTO Notes (note, whoid, tstamp, attachment, filename, sticky) VALUES (?, ?,
CURRENT_TIMESTAMP, ?, ?, ?)"
arguments = [note, user, attachmentBytes, filename, sticky]
# insert the note
insertId = system.db.runPrepUpdate(query, arguments)

# insert a row onto the itemNotes table
# replace 'MYID' with the proper code to fetch your id
myId = 'MYID'
system.db.runPrepUpdate("INSERT INTO ItemNotes (AccountId, NoteId) VALUES (?, ?)", [myId, insertId])
```

insertNote: using a single table

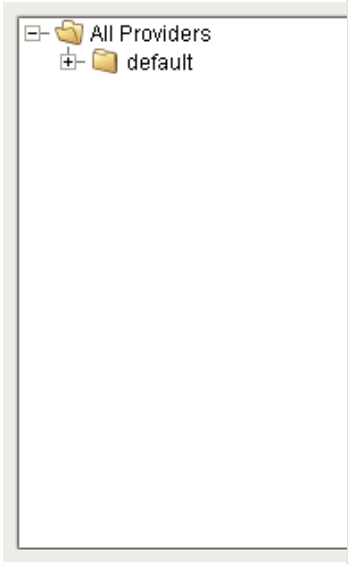
```
# Similar to the above example, but only a single database table is required.
# Assumes a User Name is used in the notes table

# determine the name for the logged in user
user = system.security.getUsername()

# determine if a file is being attached
if filename is None:
        # a file was not attached, provide a blank for the bytes
        attachmentBytes = None

else:
        # get the bytes of the file at the path the user selects
        attachmentBytes = system.file.readFileAsBytes(filename)

        # splits the file name from the file path. This way we can show just the file name on the
component
        # Using '\' as a delimiter, but python requires 2 since it's an escape character
        pathAndFile = filename.rsplit('\\', 1)
        filename = pathAndFile[1]

# insert the note
query = "INSERT INTO Notes (note, whoid, tstamp, attachment, filename, sticky) VALUES (?, ?,
CURRENT_TIMESTAMP, ?, ?, ?)"
arguments = [note, user, attachmentBytes, filename, sticky]
system.db.runPrepUpdate(query, arguments)
```

downloadAttachment

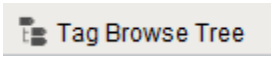
```
result = system.db.runPrepQuery("SELECT AttachmentName, Attachment FROM Notes WHERE Id=?", [id],
database="")
fileName, data = result[0][0], result[0][1]
# get the last part of the filename
fileName = system.file.saveFile(fileName.split('\\')[-1])
system.file.writeFile(fileName, data)
```

Tag Browse Tree

General



Component Palette Icon:



Description

The Tag Browse Tree component is similar to the Tag Browser in the Designer, allowing tags to be browsed in both the Designer and the Client, and dragged on to other components like the Easy Chart. Unlike the Tag Browser, tags can not be edited, tag properties can not be displayed, and UDT definitions can not be displayed. Tags in the component can be refreshed through scripting by calling refresh().

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

- Description

  Called for each tag loaded into tag browse tree. Return false to hide this tag from the tree.

- Parameters

  Self- A reference to the component that is invoking this function.

  Tag - The tag itself.

- Return

  Boolean

- Scope

  Client

- Description

  Returns a popup menu that will be displayed when the user triggers a popup menu (right click) on the tree. Use system.gui. createPopupMenu to create the popup menu.

- Parameters

  Self- A reference to the component that is invoking this function.

  clickedTag - The tag of the clicked on tree path.

  selectedTags - The tags of the selected paths of the tree.

- Return

  JPopupMenu

- Scope

  Client

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

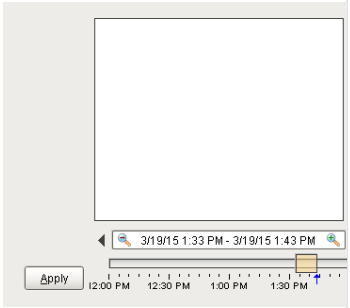
Code Snippet

```
# The following code shows a right-click popup menu.
# Add these lines after the """ """ section of the createPopupMenu extension function.
# Note how lines below are indented, the first def command should line up with the
# indentation of the """ """ section of the Extension Function.

        def showValue(self):
                value = str(clickedTag.value.value)
                system.gui.messageBox(value)


        def showLastChange(self):
                lastChange = str(clickedTag.value.timestamp)
                system.gui.messageBox(lastChange)

        itemsDict = {"Show Value": showValue, "Show Last Change":showLastChange}
        JPopupMenu = system.gui.createPopupMenu(itemsDict)
        return JPopupMenu
```

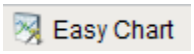
Charts

Easy Chart

General



Component Palette Icon:



Description

Description

This component is used to make powerful and runtime-configurable time-series charts. It is configured by defining a set of pens and axes. Each pen represents a series of data. Pens can be many different styles, such as line, area, bar, and shape. This chart automatically creates controls for picking the time range and for hiding or displaying pens.

Features

- Easy configuration
- User-selectable set of pens
- Automatic time-selection controls
- SQL Query and/or SQLTags Historian data sources
- Automatic SPC and calculated pen support
- Zoom, Pan, X-Trace modes
- Any number of Y-axes and subplots
- Realtime or Historical

Pens

There are three kinds of pens in the Easy Chart:

1. Tag Historian Pens. These pens pull their data from the Tag Historian system.
2. Database Pens. These pens will automatically create SQL SELECT queries to pull data from a database table. Typically, this is a table that is the target of a Historical Transaction Group.
3. Calculated Pens. These pens display a calculated dataset based off another pen, such as a moving average or an SPC function such as the UCL (Upper Control Limit).

Modes: Realtime vs Historical

The Easy Chart can operate in 3 different modes. These modes affect the range of data that is displayed, the controls the user is shown, and whether or not the chart polls for data.

1. Historical Mode. In this mode, the user is shown a Date Range component to pick the range of data to fetch and display. The initial values of this component are set through properties on the chart. In historical mode, the chart does not poll.
2. Realtime Mode. In this mode, the user is given the opportunity to pick the amount of time in the past to display. For example, the last 5 minutes or the last 2 hours. The chart will poll at a rate according to the Poll Rate parameter.
3. Manual Mode. In this mode, the chart will use the values if its Start Date and End Date parameters to govern what data is displayed. Polling is controlled by having the Poll Rate at zero (polling off) or greater than zero.

Basic Chart Configuration

The Easy Chart has many properties, like other components, that control its behavior. Things like its Mode, Polling Rate, etc are configured via the properties. All of the setup for adding pens, axes, subplots, etc is its Customizer. You can also drag and drop Historian-enabled tags onto the chart directly in the Designer to add those tags as chart pens.

Y-Axes

The easy chart supports any number of Y-axes. To add an axis, go to the Axes tab of the chart customizer. When adding an axis, you get a number of options such as the type (numeric or logarithmic), label, color, autorange vs fixed range, and auto-ticks vs fixed ticks. You can also modify the position of the axis, but note that by default the Chart's Auto Axis Positioning property is enabled, which means that the chart will balance the axes automatically between left and right depending on demand. As pens are turned on and off by the user, only the axes that are used by visible pens are shown.

After you add your axes, you edit any pens that you want to use your new axes. Simply choose the new axis in the axis dropdown of the pen editing window.

Subplots

The Subplots feature lets you break up the chart's plot area into multiple distinct subplots that share the X axis, but have their own Y axes. This is often useful for digital data, as shown in the screenshot above. By default the chart has 1 subplot (the main plot). To add a new subplot, simply hit the add button in the Subplots tab of the chart customizer.

Subplots have relatively few options. The Weight option determines how much room the subplot gets relative to the other subplots. For example, in the screenshot above subplot #1's weight is 5, and subplot #2's weight is 1, leading to a 5-to-1 distribution of space. Just like axes, once you add your subplots you should go back to your pens and modify you pens' subplot property for any pens you want to appear on the subplot.

Pen Groups

You can put your pens in groups to break up the pens into some logical separation. For instance, in the screenshot above there are three pen groups: C1, C2, and Valves. The group name is used as the titled border for the pens' grouping container. Groups also have another purpose, but it is more advanced and most people won't have to worry about it. For more, read the Dynamic Pens section below.

Advanced Configuration

Dynamic Pens

In is often the case that you'll want to make one chart window that services many similar pieces of equipment. For instance, if you have 30 tanks and they all have the same datapoints, you want to be able to use one window for all 30 of them and simply pass the tank number into the chart window as a parameter. There are actually a number of ways to accomplish this, each method suitable for different scenarios.

Database pens have 2 ways to be made dynamic. The first is the Chart's Where Clause property. This is a snippet of SQL where clause syntax, like "machine_num = 28" that will be included for alldatabase pens in their queries. The second is to use a dynamic group. Any group can be made a dynamic group in the customizer. For each dynamic group, the easy chart will get a special dynamic property associated with that group. That property is another snippet of SQL where clause that will be applied to all database pens in that group.

The other way to make your pens (and anything else about the chart) dynamic at runtime is to use dynamic configuration. Read on...

Dynamic Configuration

The Easy Chart is not just meant to be easy to configure, but also very powerful. In particular, there is an emphasis on the ability to make any configuration change dynamically in a client - not just statically in the Designer. While a bit of scripting or clever property binding may be required, the technique is very powerful. This is achieved by storing all of the settings that you alter in the customizer in a set of expert-level dataset properties. So altering the datasets alters the chart configuration. You can inspect these various datasets, which hold the pens, axes, and subplot information, to see their format. They all look up information by column name (case-insensitive). So, if you have pen configuration stored in a database, you can bind an indirect SQL Query binding to alter the chart's pen set at runtime.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

- Description

  This function save the chart's datasets as an Excel file. Returns a String of the complete file path chosen by the user, or None if the user canceled the save.

- Parameters

  String filename - The default file name for the Save dialog.

- Return

  String

- Scope

  Client

- Description

  This function will print the chart.

- Parameters

  Nothing

- Return

  Nothing

- Scope

  Client

Extension Functions

- Description

  Provides an opportunity uto perform further chart configuration via scripting. Doesn't return anything.

- Parameters

  self-A reference to the component that is invoking this function.

  chart-A JFreeChart object. Refer to the JFreeChart documentation for API details.

- Return

  Nothing

- Scope

  Client

- Description

  Provides an opportunity to configure the x-trace label. Return a string to override the default label.

- Parameters

  self-A reference to the component that is invoking this function.

  chart-A JFreeChart object. Refer to the JFreeChart documentation for API details.

  penName-The name of the pen the x-trace label applies to.

  yValue-The y-value of the pen at the x-trace location.

- Return

  Nothing

- Scope

  Client

- Description

  Called when the user has dropped rows from a power table on the chart. The source table must have dragging enabled.

- Parameters

  self-A reference to the component that is invoking this function.

  sourceTable-A reference to the table that the rows were dragged from.

  rows-An array of the row indicies that were dragged, in the order they were selected.

  rowData-A dataset containing the rows that were dragged.

- Return

  Nothing

- Scope

  Client

- Description

  Called when the user has dropped tags from the tag tree onto the chart. Normally, the chart will add pens automatically when tags are dropped, but this default behavior will be suppressed if this extension function is implemented.

- Parameters

  self-A reference to the component that is invoking this function.

  paths-A list

- Return

  Nothing

- Scope

  Client

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.
Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

Customizers

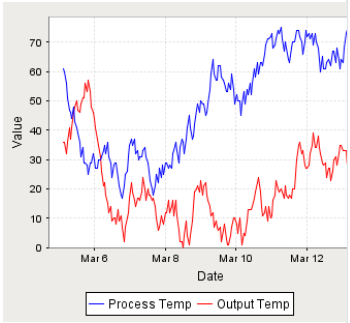
This component does not have any custom properties.

Examples

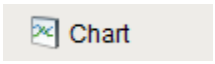
There are no examples associated with this component.

Chart

General



Component Palette Icon:



Description

The Chart component (also called the Classic Chart when contrasted with the Easy Chart) provides a flexible way to display either timeseries or X-Y charts that are powered by any number of datasets. Typically, these datasets are bound to SQL Query Bindings.

Features

- SQL Query and/or SQLTags Historian data sources
- Zoom, Pan, X-Trace modes
- Any number of Y-axes and subplots
- Realtime or Historical
- Many different rendering styles

Configuration

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

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

Datasets

Each dataset should define one or more "series" (a.k.a "pens"). The format for these datasets is quite simple. Each series in a dataset shares common X-values, defined by the first column. Each additional column are the Y-values for a series.

Binding Techniques

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

Chart Type: XY vs Category

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

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

- Description

  Provides an opportunity to perform further chart configuration via scripting.

- Parameters

  Self- A reference to the component that is invoking this function.

  Chart- A JFreeChart object. Refer to the JFreeChart documentation for API details.

- Return

  Nothing

- Scope

  Client

- Description

  Provides an opportunity to configure the x-trace label. Return a string to override the default label.

- Parameters

  Self- A reference to the component that is invoking this function.

  Chart - A JFreeChart object. Refer to the JFreeChart documentation for API details.

  penName - The name of the pen the x-trace label applies to.

  yValue - The y-value of the pen at the x-trace location

- Return

  Nothing

- Scope

  Client

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

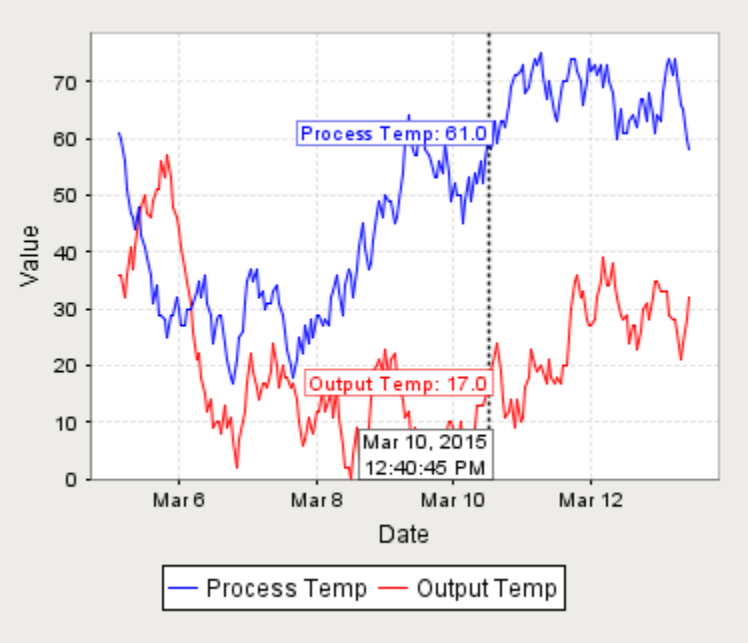
Unknown macro: 'sql'

Customizers

This component has a customizer.

Examples

Chart with Stacktrace



Sparkline Chart

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 give it a fixed range, simply fill in the Range Highand Rang e Low properties.

Properties

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Unknown macro: 'sql'

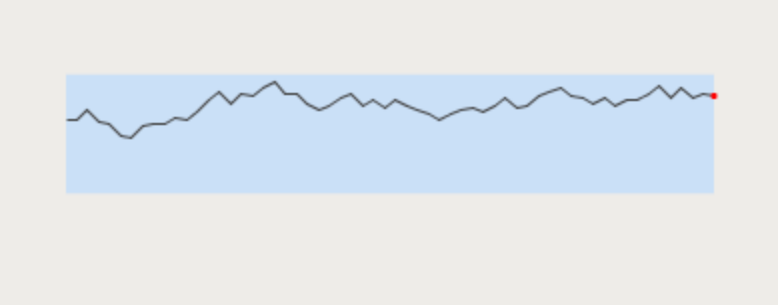
Customizers

This component does not have any custom properties.

Examples

Gallery

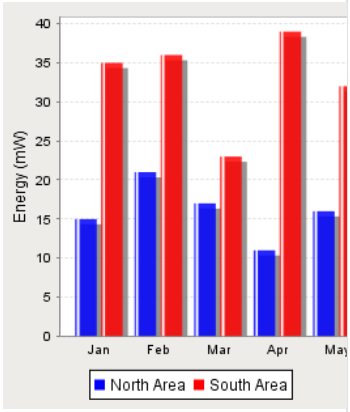
Sparkline Chart with Low and High Limits



Property Name	Value
Range High	100
Range Low	0
Desired High	75
Desired Low	40

Bar Chart

General



Component Palette Icon:

Bar Chart

Description

The Bar Chart is a very easy-to-use chart that provides a familiar bar representation of any numeric values. That is, the height of the bars is determined by some numeric value in the underlying dataset. It is often configured to display as a category chart. A category chart is a chart whose X-values are categories (strings, names, groupings, etc) rather than numeric values (numbers, dates).

Like most chart components (other than the Easy Chart), the Data property drives the chart. The first column in the Data dataset defines the names of the categories. The rest of the columns define the values for each of the series (if there is more than one series per category), and thus should be numeric. Note - if your data is 'turned on its side', meaning that the columns define the categories and rows define the series, then set the Extract Order to "By Column".

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

- Description

  Provides a chance to override the color of each bar. Can be used to have bar colors changed based upon bar value. Returning the value None will use the default bar color for the series.

- Parameters

  Component self - A reference to the component that is invoking this function.

  int series - The series index for this bar.

  int category - The category index for this bar.

  int value - The value (a number) of this bar.

  Color defaultColor - The color that the bar would be if this function wasn't invoked.

- Return

  Color

- Scope

  Client

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'
This event fires when a mouse button is released, if that mouse button's press happened over this component.
Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

Customizers

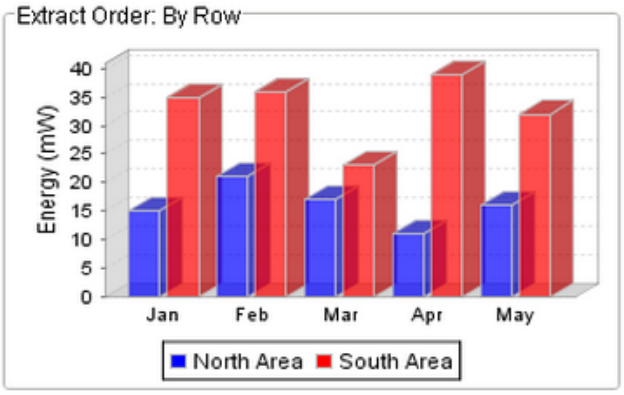
This component does not have any custom properties.

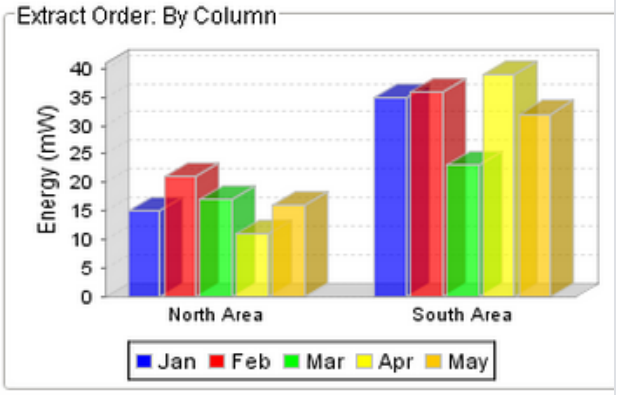
Examples

Extract Order Example

The following two charts demonstrate the effects of the extract order property on the given dataset

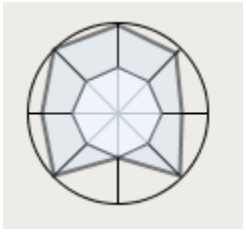
Label (String)	**North Area** (Integer)	**South Area** (integer)
Jan	15	35
Feb	21	36
Mar	17	23
Apr	11	39
May	16	32





Radar Chart

General



Component Palette Icon:

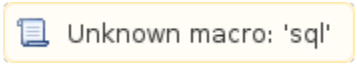
Description

Radar charts, also known as web charts, spider charts, spider plots, and a few other names, display a dataset as a two dimensional polygon. The plot is arranged as a set of spokes with equal angles between them. Each spoke represents a value axis for the variable it corresponds to. Each dataset is then drawn as a connected polygon, where the points of the polygon are arranged on the spokes according to their value. Each row of the dataset has a minimum and maximum column -- these values are used to determine the scale of the spoke for that variable, with the midpoint representing the desired value.

The intended use of radar plots is to display realtime information in such a way that outliers can be quickly identified. This can be an efficient way to convey if a process is running on-spec or off-spec at a glance.

The radar chart gets its data from a dataset. Each row in the dataset will become a single variable (spoke) on the chart. The dataset must have a columns labeled "Value", "Min", and "Max"; other columns will be ignored. To display realtime data on a radar chart, you can use a cell-update binding to bind individual values to tag values. You can also drop tags onto a radar chart, with the EngMin binding to min and EngMax binding to max. If there are no existing cell-update bindings, the tags will replace existing data, otherwise the tags will be added to the end of the dataset. Alternatively, you can have realtime information stored by a transaction group to a database table, and drive the radar chart's dataset with a query binding.

Properties



Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

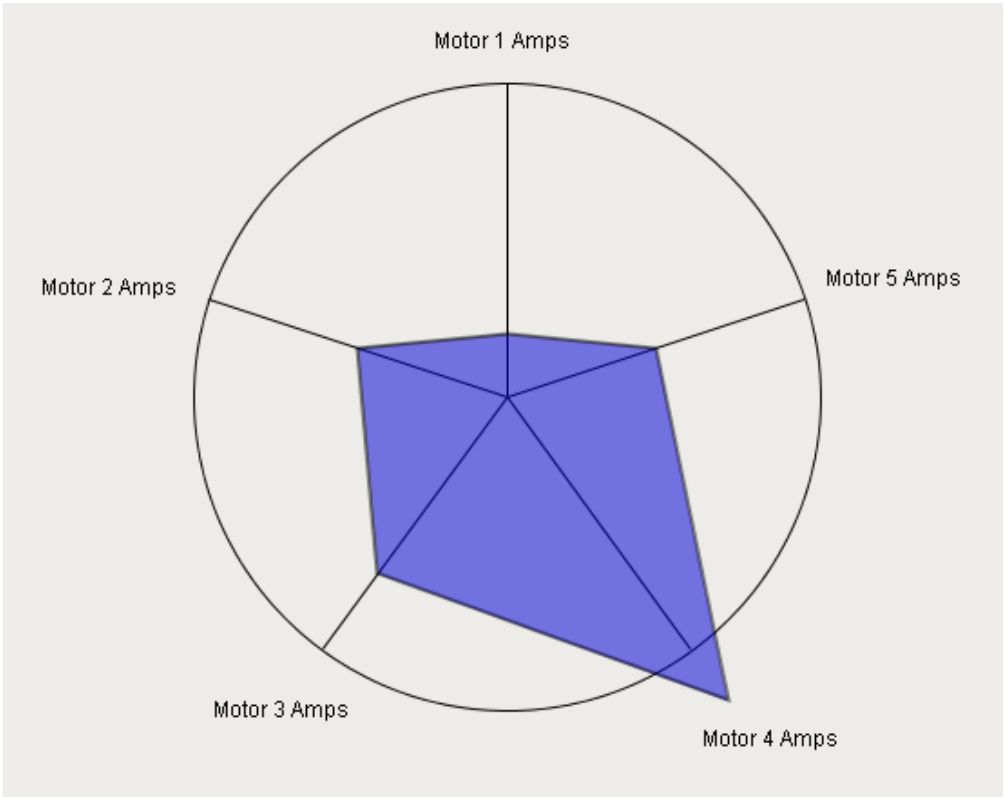
Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

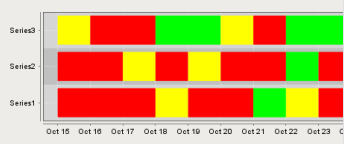
Examples

The following shows that the Motor 4 Amps are higher than the maximum allowed for Motor 4. In this example each motor has an independent min and max value for its amps. The radar chart allows an operator to quickly assess a group of independent variables to determine if anything is out of its specifications.

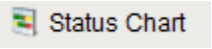


Status Chart

General



Component Palette Icon:



Description

The status chart component allows you to visualize the status of one or more discrete datapoints over a time range. The X-axis is always a timeseries axis, and the Y-axis is a category axis, with one entry per data series. The chart is populated with a single dataset, the first column of which must be a datetime column.

Wide vs Tall Datasets.

In Wide format, all of the columns but the first must be numeric. These "series" columns' headers will be used as the names on the y-axis. In Tall format, there should be exactly 3 columns. The first is the timestamp, the second is the series name, and the third is the value. For example:

Wide Format

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

Tall Format

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

Color Mapping

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

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

- Description

  Return a formatted tool tip String

- Parameters

  Self- A reference to the component that is invoking this function.

  seriesIndex-The series index corresponding to the column in the series dataset.

  selectedTimeStamp-The time stamp corresponding to the x value of the displayed tooltip. The time stamp is the numver of seconds since the epoch.

  timeDiff-The width of the current status interval measured in seconds since the epoch.

  seletedStatus-The status value corresponding to the x value of the displayed tooltip.

  data-The series dataset as a PyDataset.

  properties-The series properties dataset as a PyDataset.

  defaultString-The default tooltip string.

- Return

  String defaultString

- Scope

  Client

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

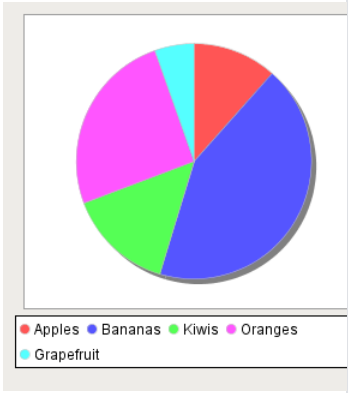
Unknown macro: 'sql'

Customizers
This component has a customizer to customize the colors of the each series.

Examples
There are no examples associated with this component.

Pie Chart

General



Component Palette Icon:

Description

The Pie Chart component displays a familiar-looking pie chart. 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. Typically, this dataset will be bound to a SQL Query Binding to pull dynamic data out of an external database.

Extract Order

Similar to other charts, the pie chart can actually accept data in two formats. You can tell the pie chart which format to use via its Extract Order pro perty. The two extract orders are By Column or By Row. The following table shows the two styles for the data that created the pie chart in the screenshot.

By Column

Label	Value
Grapefruit	7
Apples	15
Bananas	56
Kiwis	19

By Row

Grapefruit	Apples	Bananas	Kiwis
7	15	56	19

Labels

In addition to the color-coded legend, the pie chart can annotate each wedge with a label. The format of the label is controlled via the Label Format property.

For example, the format string used in the screenshot is "{0} = {2} ({3})"This is a pattern string that uses the following placeholders:

- {0} - the item label
- {1} - the item value
- {2} - the item percentage

Properties

The component's properties are populated from a sql query. The following properties are from the Alarm Status Table. Change this to the correct component. Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'
This event fires when a mouse button is released, if that mouse button's press happened over this component.
Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

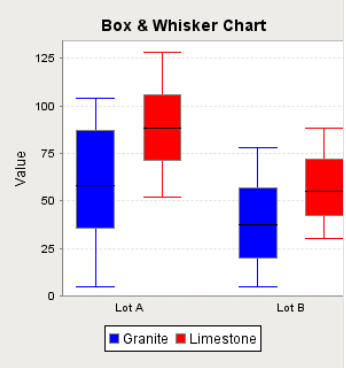
Examples

Code Snippet

```
#The following code will print namd and value of the selected wedge to the console.
#Alternatively this can be used to write to a custom property of a table that is used to create the
'Where' clause of a SQL query that populates a table.

selectedWedge = event.source.selectedData
print selectedWedge
```

Box and Whisker Chart

General



Component Palette Icon:

Box and Whisker Chart

Description

A Box and Whisker chart displays pertinent statistical information about sets of data. Each box represents a set of numbers. The upper and lower bounds of the box represent the 1st and 3rd quartiles. The line inside the box represents the median. The extends of the "whiskers" represent the max and min outliers. For a more detailed description, see http://mathworld.wolfram.com/Box-and-WhiskerPlot.html.

The configuration for setting up a box and whisker chart, like most charts, is populating the Data property. The dataset for a box and whisker chart contains sets of numbers. Each column defines a seriesof values, for which a "box" will be calculated. The column headers define the name for the box. You may also have an optional first column that is a String column, which can break up the series into categories.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'
This event fires when a mouse button is released, if that mouse button's press happened over this component.
Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

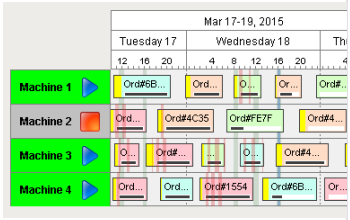
Customizers

This component does not have any custom properties.

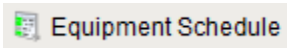
Examples

Equipment Schedule

General



Component Palette Icon:



Description

The equipment schedule view is a mix between the status chart, gantt chart, and a calendar view. It conveys a lot of information about equipment, including current status, production schedule, production status, scheduled and unexpected downtime.

The equipment schedule is powered by four datasets. Information is retrieved from the datasets by column name, case-insensitive. The order of the columns is not important. Optional columns may be omitted.

The "Items" Dataset

Describes the "items" or "cells" to display schedules for. Each entry in this dataset will become a row of the chart.

Name	Type	Optional	Description
ID	Any	N	The identifier for this item. May be any type, will referenced by each entry in the Scheduled Events dataset.
Label	String	N	The text to display in the header
Foreground	Color	Y	Text color
Background	Color	Y	Background color
StatusImagePath	String	Y	A path to an image to display to the right of the header label

The "Scheduled Items" Dataset

Lists the scheduled events for each item described in the "Items" dataset. Each scheduled event can have a colored lead, or change-over time, a label, a background color, and a progress.

Name	Type	Optional	Description
EventId	String	Y	An identifier for the event, used for event selection.
ItemId	Any	N	The ID of the item to correlate this event with. If no such item is found, the event won't be shown.
Label	String	N	The text ot display in the event's box
StartDate	Date	N	The start-time for the event
EndDate	Date	N	The end-time for the event
Foreground	Color	Y	The text color of the event
Background	Color	Y	The background color of the event
LeadTime	Integer	Y	Time, in seconds, to display as lead time.
LeadColor	Color	Y	The color for the lead time, if any.
PctDone	Number	Y	A value from 0 to 100 to be displayed as a progress bar, use -1 to hide progress bar.

The "Downtime" Dataset

Entries in this dataset will be displayed as simple colored overlays on top of the events, correlated against an item defined in the "Items" dataset.

Name	Type	Optional	Description
ItemId	Any	N	The ID of the item to correlate this downtime event with. If no such item is found, the downtime event won't be shown.
StartDate	Date	N	The start-time for the downtime event
EndDate	Date	N	The start-time for the downtime event
Color	Color	Y	The color to use, typically transparent.
Layer	Integer	Y	0 or 1, with 0 meaning that the rectangle gets painted below the events, and 1 means it will be painted above the events.

The "Breaks" Dataset

Entries in this dataset will be displayed as colored underlays beneath all events.

Name	Type	Optional	Description
StartDate	Date	N	The start-time for the break event
EndDate	Date	N	The start-time for the break event
Color	Color	Y	The color to use

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

- Description

  Called when the user drags a segment on the schedule background.

- Parameters

  Self- A reference to the component that is invoking this function.

  itemID-The ID of the equipment item of the row where the user dragged.

  startDate-The datetime corresponding to where the user started dragging.

  endDate-The datetime corresponding to where the user ended dragging.

  evnet-The mouse event.

- Return

  Nothing

- Scope

  Client

- Description

  Called when the user clicks on a scheduled event. Use event.clickCount to detect double clicks.

- Parameters

  Self- A reference to the component that is invoking this function.

  itemID-The ID of the equipment item of the event that was clicked on.

  eventId-The ID of the event that was clicked on.

  event-The mouse event.

- Return

  Nothing

- Scope

  Client

- Description

  Called when the user drags and drops a scheduled event. It is up to this script to actually alter the underlying data to reflect the schedule change.

- Parameters

  Self- A reference to the component that is invoking this function.

  eventId-The ID of the scheduled event that was moved.

  oldItemId-The ID of the item this event was originally correlated against.

  newItemId-The ID of the item whose schedule the event was dropped on.

  oldStartDate-The original starting datetime of the event.

  newStartDate-The new starting datetime of the event.

  newEndDate-The new ending datetime of the event.

- Return

  Nothing

- Scope

  Client

- Description

  Called when the user right-clicks on a scheduled event. This would be the appropriate time to create and display a popup menu.

- Parameters

  Self- A reference to the component that is invoking this function.

  itemId-The ID of the equipment item of the event that was right-clicked on.

  eventId-The ID of the event that was right-clicked on.

  event-The mouse event that caused the popup trigger.

- Return

  Nothing

- Scope

  Client

- Description

  Called when the user drags the edge of an event to resize its time span. It is up to this script to actually alter the underlying data to reflect the schedule change.

- Parameters

  Self- A reference to the component that is invoking this function.

  eventId-The ID of the scheduled event that was resized.

  itemId-The ID of the item this event is correlated against.

  oldStartDate-The original starting datetime of the event.

  oldEndData-The original ending datetime of the event.

  newStartDate-The new starting datetime of the event.

  newEndDate-The new ending datetime of the event.

- Return

  Nothing

- Scope

  Client

- Description

  Called when the user right-clicks outside of an event. This would be the appropriate time to create and display a popup menu.

- Parameters

  Self- A reference to the component that is invoking this function.

  itemId-The item ID of the equipment line that was clicked on (if any).

  event-The mouse event that caused the popup trigger.

- Return

  Nothing

- Scope

  Client

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Unknown macro: 'sql'

This event is deprecated. Please use the onEventDropped extension function.

Customizers

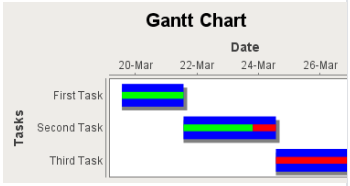
This component does not have any custom properties.

Examples

There are no examples associated with this component.

Gantt Chart

General



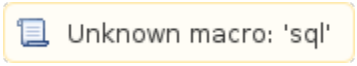
Component Palette Icon:

Description

A Gantt chart is used for task scheduling. It shows a list of named tasks, each of which have a start date, an end date, and a percentage complete. This allows an easy way to visualize tasks, workflows, and scheduling.

The Gantt chart is configured by populating its Data property. Each row of the dataset represents a task. There should be four columns: the task label, the start date, the end date, and the percentage (0-100) complete.

Properties



Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'
This event fires when a mouse button is released, if that mouse button's press happened over this component.
Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

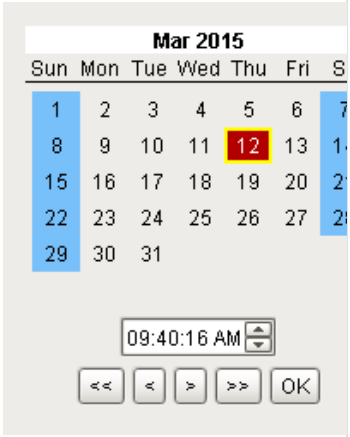
Examples

There are no examples associated with this component.

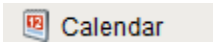
Calendars

Calendar

General



Component Palette Icon:



Description

Displays a calendar and time input directly embedded in your window. Most commonly used by including one of the two date properties (immediate or latched) from the calendar in dynamic SQL Query Binding.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'
This event fires when a mouse button is released, if that mouse button's press happened over this component.
Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

Customizers

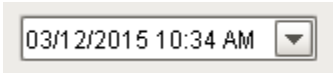
This component does not have any custom properties.

Examples

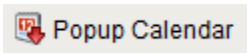
There are no examples associated with this component.

Popup Calendar

General



Component Palette Icon:



Description

The popup calendar is a popular way to provide date/time choosing controls on a window. Similar to the Calendar component, but takes up much less screen real estate. Most commonly used by including this component's Date property in dynamic SQL Query Binding.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'
This event fires when a mouse button is released, if that mouse button's press happened over this component.
Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

Customizers

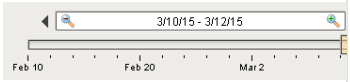
This component does not have any custom properties.

Examples

There are no examples associated with this component.

Date Range

General



Component Palette Icon:

Description

The date range component provides an intuitive, drag-and-drop way to select a contiguous range of time. The user is shown a timeline and can drag or stretch the selection box around on the timeline. The selected range is always a whole number of units, where the unit is determined by the current zoom level.

Note: The **Start/End** dates and **Outer Start/End** dates will be ignored when the window opens unless the Startup Mode property is set to "None."

Data Density Histogram

As an advanced optional feature, the date range can display a data density histogram inside the timeline. This is useful for historical data with gaps in it, so that the end user isn't hunting for data. (Tip: this is also great for demos, to make it easy to find historical data in a database that isn't being continuously updated).

To use this feature, bind the Data Density dataset to a query that returns just the timestamps of the target table. These timestamps will be used to fill in the histogram behind the timeline. You can use the Outer Range Start Date and Outer Range End Date properties in your query to limit the overall return size for the query. **Note**: timestamps must be ordered by date (ascending) to display correctly.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

- Since 7.8.1
- Description

  Sets the selected range. The outer range will move if needed. Note: the start and end times are determined based on the zoom level and may not move (or may move farther than intended) if the component is zoomed out too far for the amount of change attempted. IE: When days are showing, moving the start time 5 minutes forward will not effect the start, and moving the end time 5 minutes forward will add one day.
- Parameters

  Date start - The starting date for the new selection.

  Date end - The ending date for the new selection.
- Return

  Nothing
- Scope

  Client

Code Snippet

```
# This example moves the existing Start Date and End Date
# of a Date Range component ahead 8 hours
from java.util import Calendar

# Get the current start and end
dateRangeComponent = event.source.parent.getComponent('Date Range')
startDate = dateRangeComponent.startDate
endDate = dateRangeComponent.endDate

# Calculate the new start and end dates
cal = Calendar.getInstance();
cal.setTime(startDate);
cal.add(Calendar.HOUR, -8);
newStart = cal.getTime();

cal.setTime(endDate);
cal.add(Calendar.HOUR, -8);
newEnd = cal.getTime();

# Set the new range for the component. The outer range will
# automatically expand if needed.
dateRangeComponent.setRange(newStart, newEnd)
```

- Since 7.8.1
- Description

  Sets the outer range. The selected range will move if needed. Note: the start and end times are determined based on the zoom level and may not move (or may move farther than intended) if the component is zoomed out too far for the amount of change attempted. IE: When days are showing, moving the start time 5 minutes forward will not effect the start, and moving the end time 5 minutes forward will add one day.
- Parameters

  Date start - The starting date for the new outer range.

  Date end - The ending date for the new outer range.
- Return

  Nothing
- Scope

  Client

Code Snippet

```
# This example moves the existing Outer Date Range
# of a Date Range component back two days
from java.util import Calendar

# Get the current start and end of the outer range
dateRangeComponent = event.source.parent.getComponent('Date Range')
startDate = dateRangeComponent.outerRangeStartDate
endDate = dateRangeComponent.outerRangeEndDate

# Calculate the new start and end dates for the outer range
cal = Calendar.getInstance();
cal.setTime(startDate);
cal.add(Calendar.DAY_OF_MONTH, 2);
newStart = cal.getTime();

cal.setTime(endDate);
cal.add(Calendar.DAY_OF_MONTH, 2);
newEnd = cal.getTime();

# Set the new outer range for the component.
dateRangeComponent.setOuterRange(newStart, newEnd)
```

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

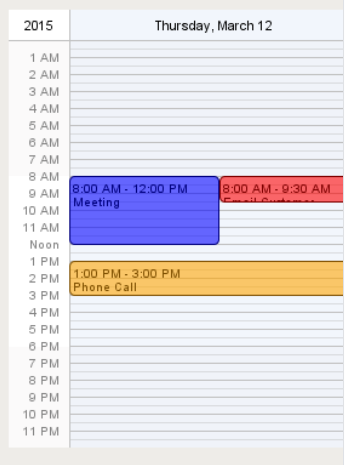
Examples

Code Snippet

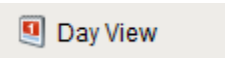
```
//A Query binding on another component on the same window might look like this:

SELECT Column1, Column2, Column3
FROM MyTable WHERE
  t_stamp >= {Root Container.Date Range.startDate} AND
  t_stamp <= {Root Container.Date Range.endDate}
```

Day View

General



Component Palette Icon:



Description

This component displays a timeline for a single day, similar to what you might find in a personal planner/organizer. By filling in the Calendar Events dataset property, the component will display events that occur on this day. Each event can have custom text and a custom display color associated with it.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'
This event fires when a mouse button is released, if that mouse button's press happened over this component.
Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

Customizers

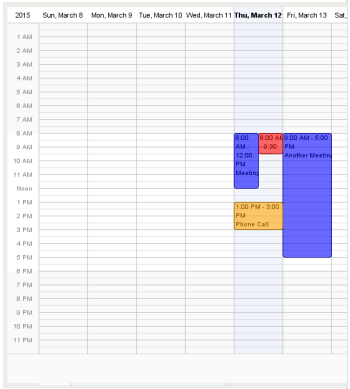
This component does not have any custom properties.

Examples

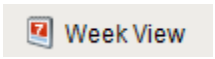
There are no examples associated with this component.

Week View

General



Component Palette Icon:



Description

Displays a full week's worth of events on a calendar. Configuration is achieved by populating the Calendar Events dataset. See the Day View for details.

Properties

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'
This event fires when a mouse button is released, if that mouse button's press happened over this component.
Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

Customizers

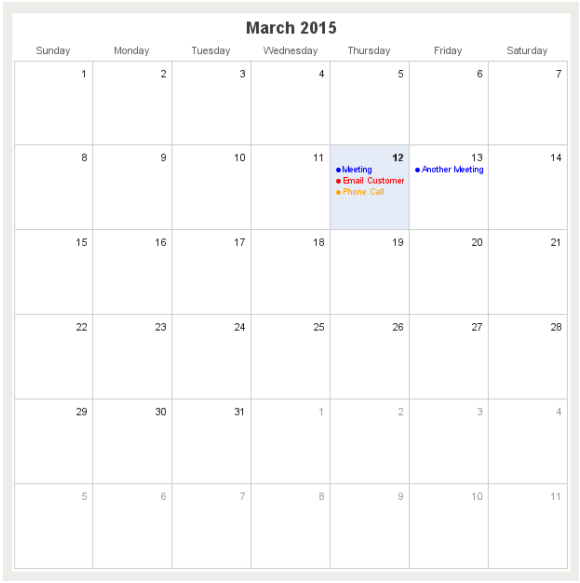
This component does not have any custom properties.

Examples

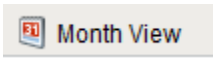
There are no examples associated with this component.

Month View

General



Component Palette Icon:



Description

This component displays events for an entire month. By filling in the Calendar Events dataset property, the component will display events that occur for each day of the month. Each event can have custom text and a custom display color associated with it.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'
This event fires when a mouse button is released, if that mouse button's press happened over this component.
Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

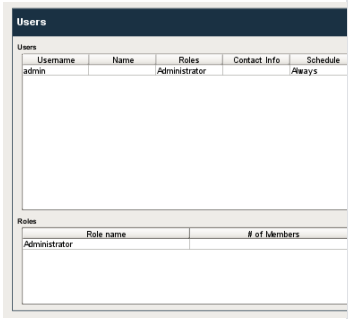
Examples

There are no examples associated with this component.

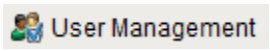
Admin

User Management

General



Component Palette Icon:



Description

The user management panel provides a built-in way to edit users and roles from a client. To use this component, you should be aware that it is only editing the users and roles from a single User Source.By default, the component will use the user source of the containing project. You can change this by typing in the name of another user source into the component's "User Source" property.

To make changes to the Gateway's system user source from the Designer or Client, Allow User Admin must be checked in Gateway Settings in the Gateway Configuration page.

This component can be run in one of three modes:

Manage Users Mode: In this mode, the component manages all of the users contained in the user source. Users and roles may be added, removed, and edited.

Edit Single Mode: In this mode, the component only edits a single user. Which user is being edited is controlled via the "User Source" and "Username" properties.

Edit Current Mode: In this mode, the user who is currently logged into the project can edit themselves. Obviously, the ability to assign roles is not available in this mode. This can be useful to allow users to alter their own password, adjust their contact information, and update their schedules.

Warning: Be careful to only expose this component to users who should have the privileges to alter other users. Access to this component in "Manage Users" mode will allow users to edit other users' passwords and roles.

Properties

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

- Description

  Called for each user loaded into the management table. Return false to hide this user from the management table. This code is executed in a background thread.

- Parameters

  Self- A reference to the component that is invoking this function.

  user-The user object itself. Call user.get('propertyName') to inpsect. Common properties: 'username',' schedule', 'language', user.getRoles() for a list of rolenames.

- Return

  Boolean

- Scope

  Client

- Description

  Called for each role loaded into the management table. Return false to hide this role from the management table. This code is executed in a background thread.

- Parameters

  Self- A reference to the component that is invoking this function.

- Return

  Boolean

- Scope

  Client

- Description

  Called for each schedule loaded into the schedule dropdown in the edit user panel. Return false to hide this schedule from the dropdown. This code is executed in a background thread.

- Parameters

  Self- A reference to the component that is invoking this function.

- Return

  Boolean

- Scope

  Client

- Description

  Called when the save button is pressed when adding or editing a role. This code is executed in a background thread.

- Parameters

  Self- A reference to the component that is invoking this function.

  saveContext-An object that can be used to reject the edit by calling saveContect.rejectSave('reason').

  oldName-The role name before editing. Will be None for a role being added.

  newName-The new name of the edited role.

- Return

  Nothing

- Scope

  Client

- Description

  Called when the save button is pressed when adding or editing a user. This code is executed in a background thread.

- Parameters

  Self- A reference to the component that is invoking this function.

  saveContect-An object that can be used to reject the edit by calleing saveContect.rejectSave('reason').

  user-The user that is trying to be saved. Call user.get('propertyName') to inspect. Common properties: 'username', 'schedule','language'. Call user.getRoles() for a list of rolenames.

- Return

  Nothing

- Scope

  Client

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'
This event fires when a mouse button is released, if that mouse button's press happened over this component.
Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

Customizers

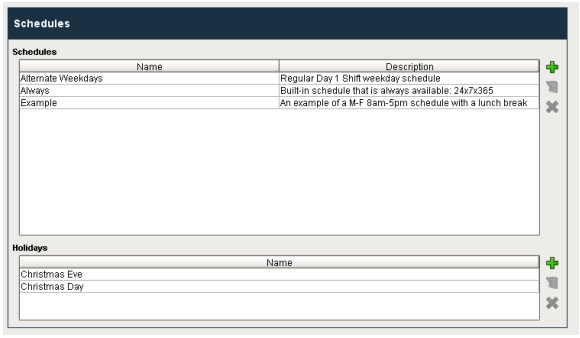
This component does not have any custom properties.

Examples

There are no examples associated with this component.

Schedule Management

General



Component Palette Icon:

Description

This component allows for management of schedules. Schedules can be defined by specifying which days of the week and which times of day they are active on. The times of day are defined using a string of time ranges, where the times are specified in 24-hr format with dashes between the beginning and the end. Multiple ranges can be specified by separating them with commas. Examples:

8:00-17:00	Valid from 8am to 5pm
6:00-12:00, 12:45-14:00	Valid from 6am to noon, and then again from 12:45pm to 2pm
0:00-24:00	Always valid.

Schedules that alternate weekly or daily can be specified by using the repetition settings. All repeating schedules need a starting day. For example, you could have a schedule that repeats on a weekly basis, with 1-week on and 1-week off. This schedule would be active for seven days starting on the starting day, and then inactive for the next seven days, then active for seven days, and so on. Note that the days of the week and time settings are evaluated in addition to the repetition settings. This means that both settings must be true for the schedule to be active. Also note that if you set "Repeat / Alternate" to a setting other than "Off" and you do not specify a starting day, the schedule will never be active.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

- Description

  Called for each holiday loaded into the management table. Return false to hide this holiday from the management table. This code is executed in a background thread.

- Parameters

  Self- A reference to the component that is invoking this function.

  Holiday-The holiday name.

- Return

  Boolean

- Scope

  Client

- Description

  Called for each schedule loaded into the management table. Return false to hide this schedule from the management table. This code is executed in a background thread.

- Parameters

  Self- A reference to the component that is invoking this function.

  Schedule-The schedule name

- Return

  Boolean

- Scope

  Client

- Description

  Called when the save button is pressed when adding or editing a holiday. This code is executed in a background thread.

- Parameters

  Self- A reference to the component that is invoking this function.

  saveContect-An object that can be used to reject the edit be calling saveContext.rejectSave('reason')

  oldName-The holiday name before editing. Will be None for a holiday being added.

  newName-The new name of the edited holiday.

- Return

  Nothing

- Scope

  Client

- Description

  Called when the save button is pressed when adding or editing a schedule. This code is executed in a background thread.

- Parameters

  Self- A reference to the component that is invoking this function.

  saveContext-An object that can be used to reject the edit by calling saveContect.rejectSave('reason').

  oldName-The schedule anme before editing. Will be None for a schedule being added.

  newName-The new name of the edited schedule.

- Return

  Nothing

- Scope

  Client

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.
Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

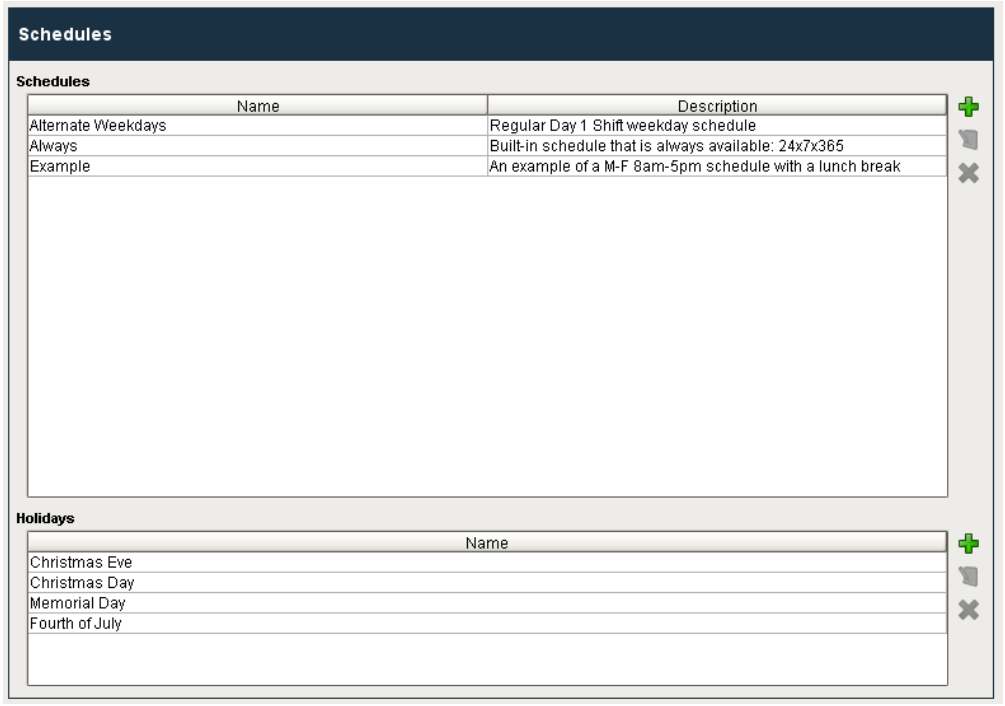
Customizers

This component does not have any custom properties.

Examples

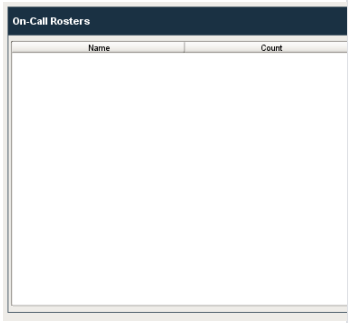
Here is an example of the schedule management component and its property table.

Property Name	Value
Name	Schedules
Enabled	True
Visible	True
Touchscreen Mode	Single-Click
Data Quality	-1



Roster Management

General



Component Palette Icon:

Description

The user management panel provides a built-in way to edit rosters from a client.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

- Description

  Called for each roster loaded into the management table. Return false to hide this roster from the management table. This code is executed in a background thread.

- Parameters

  Self- A reference to the component that is invoking this function.

  Roster-The name of the roster.

- Return

  Boolean

- Scope

  Client

- Description

  Called for each user in a user source to be shown as an available user for the roster currently being edited. Return false to hide this user so that it cannot be added to the roster. This code is executed in a background thread.

- Parameters

  Self- A reference to the component that is invoking this function.

  Roster-The name of the roster being edited.

  userSource-The name of the user source being used to populate the list of available users.

  user-The user object itself. Call user.get('propertyName') to inspect. Common properties: 'username','schedule','language'. Call user.getRoles() for a list of rolenames.

- Return

  Boolean

- Scope

  Client

- Description

  Called when the save button is pressed when editing a roster. This code is executed in a background thread.

- Parameters

  Self- A reference to the component that is invoking this function.

  saveContext-An object that can be used to reject the edit by calling saveContext.rejectSave('reason')

  rosterName-The same of the roster being edited.

- Return

  Nothing

- Scope

  Client

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.
Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

Customizers

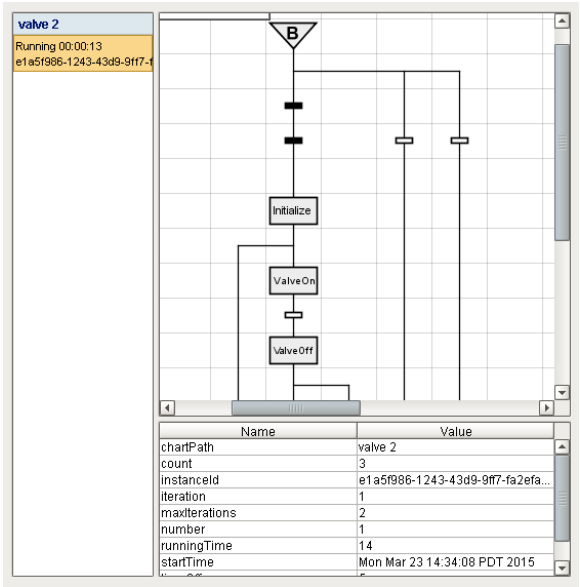
This component does not have any custom properties.

Examples

There are no examples associated with this component.

SFC Monitor

General



Component Palette Icon:

Description

A component to monitor SFC performance. In addition the component allows for the operator to control the chart instance through the charts instance 'id' property. The chart scoped variables are available through the scope dataset property.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'
This event fires when a mouse button is released, if that mouse button's press happened over this component.
Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

There are no examples associated with this component.

Alarming Components

Alarm Status Table

General

Component Palette Icon:

INDUCTIVE UNIVERSITY

Alarm Status Table

Watch the Video

Description

The alarm status table displays the current state of the alarm system. It can be configured to show active, unacknowledged, cleared, and acknowledged alarms. By default it shows all non-cleared/non-ack'ed alarms.

Acknowledgement is handled by selecting (checking) alarms and pressing the "Acknowledge" button. If any of the selected alarms require acknowledge notes, then a small text area will be presented in which the operator must add notes to the acknowledgement.

Shelving is supported by pressing the "Shelve" button when an alarm is selected. This will temporarily remove the alarm from the entire alarm system (not just the local client). When the time is up, if the alarm is still active, it will pop back into the alarm system. The times shown to the user are customizable by editing the values inside the "Shelving Times" dataset property. The alarms that have been shelved can be un-shelved by pushing the shelf management button in the lower right-hand side of the component.

If a more simplified alarm status table is needed, many of the features of the status table can be removed, for example, the header, footer, and multi-selection checkboxes. If a very short alarm status table is needed, turn on the "Marquee Mode" option, which will automatically scroll through any alarms if there is not enough vertical space to show all of them at once.

To change the columns that are displayed, the order of the columns, and/or the column width, put the Designer into preview mode. Then right-click on the table header to show/hide columns. Click and drag to re-order columns, and drag the margins of the columns to resize their width. No further action is necessary - the column configuration will remain in place after the window is saved.

For alarms that originate from tags that have tag history turned on, users can see an automatic ad-hoc chart for the value of the source tag by pressing the chart button.

An Example of configuring the Alarm Status Table can be found on the Alarm Status Component page.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

- Description

  This specialized print function will paginate the table onto multiple pages. This function accepts keyword-style invocation.

- Keyword Args

  fitWidth - If true, the table's width will be stretched to fit across one page's width. Rows will still paginate normally. If false, thetable will paginate columns onto extra pages. (default = true) [optional]

  headerFormat - A string to use as the table's page header. The substring "{0}" will be replaced with the current page number. (default = None) [optional]

  footerFormat - A string to use as the table's page footer. The substring "{0}" will be replaced with the current page number. (default = "Page {0}") [optional]

  showDialog - Whether or not the print dialog should be shown to the user. Default is true. [optional]

  landscape - Used to specify portrait (0) or landscape (1) mode. Default is portrait (0). [optional]

- Return

  Boolean- True if the print job was successful.

- Scope

  Client

Event Handlers

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Unknown macro: 'sql'

Extension Functions

- Description

  Returns a popup menu that will be displayed when the user triggers a popup menu (right click) in the table. Use *system.gui.PopupMenu()* to create the popup menu.

- Parameters

  Self- A reference to the component that is invoking this function.

  SelectedAlarmEvents - The alarm events selected on the Alarm Status Table. For an individual alarm Event, call *alarmEvent.get ('propertyName')* to inspect. Common properties: 'name', 'source', 'priority'.

- Return

  Object - the popup menu.

- Scope

  Client

- Description

  Called for each event loaded into the alarm status table. Return false to hide this event from the table. This code is executed in a background thread.

- Parameters

  Self- A reference to the component that is invoking this function.

  SelectedAlarmEvents - The alarm event itself. Call alarmEvent.get('propertyName') to inspect. Common properties: 'name', 'source','priority'.

- Return

  Boolean- Returns true or false for every alarm event in the table. True will show the alarm. False will not show the alarm.

- Scope

  Client

- Description

  Returns a boolean that represents whether the selected alarm can be acknowledged

- Parameters

  Self- A reference to the component that is invoking this function.

  SelectedAlarmEvents - The alarm events selected on the Alarm Status Table. For an individual alarmEvent, call alarmEvent.get ('propertyName') to inspect. Common properties: 'name','source','priority'.

- Return

  Boolean- Returns true or false for every alarm event in the table.

- Scope

  Client

- Description

  Returns a boolean that represents whether the selected alarm can be shelved.

- Parameters

  Self- A reference to the component that is invoking this function.

  SelectedAlarmEvents - The alarm events selected on the Alarm Status Table. For an individual alarmEvent, call alarmEvent.get ('propertyName') to inspect. Common properties: 'name','source','priority'.

- Return

  Boolean- Returns true or false for every alarm event in the table.

- Scope

  Client

- Description

  Called when an alarm is double-clicked on to provide custom functionality.

- Parameters

  Self- A reference to the component that is invoking this function.

  SelectedAlarmEvents - The alarm event that was double clicked. For an individual alarmEvent, call alarmEvent.get ('propertyName') to inspect. Common properties: 'name','source','priority'.

- Return

  Nothing

- Scope

  Client

Custom Methods

Custom methods allow you to create methods that are accessible to all of the event handlers thereby eliminating the need to copy your scripting from one handle to another.

Customizers

Alarm Row Styles

The Alarm Row Styles Customizer manages the way the Alarm Status Table renders each alarm.

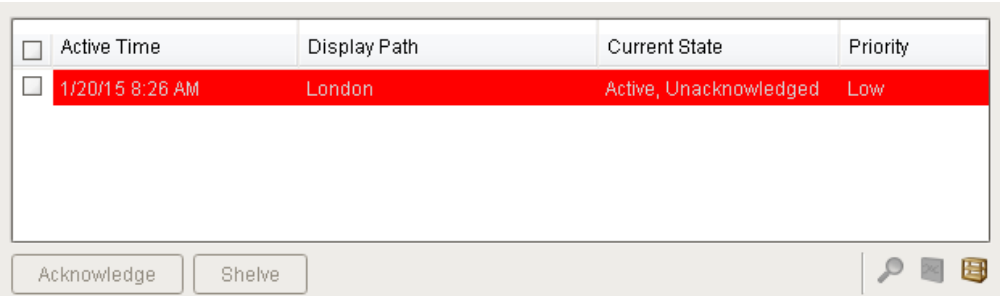
Examples

Code Snippet

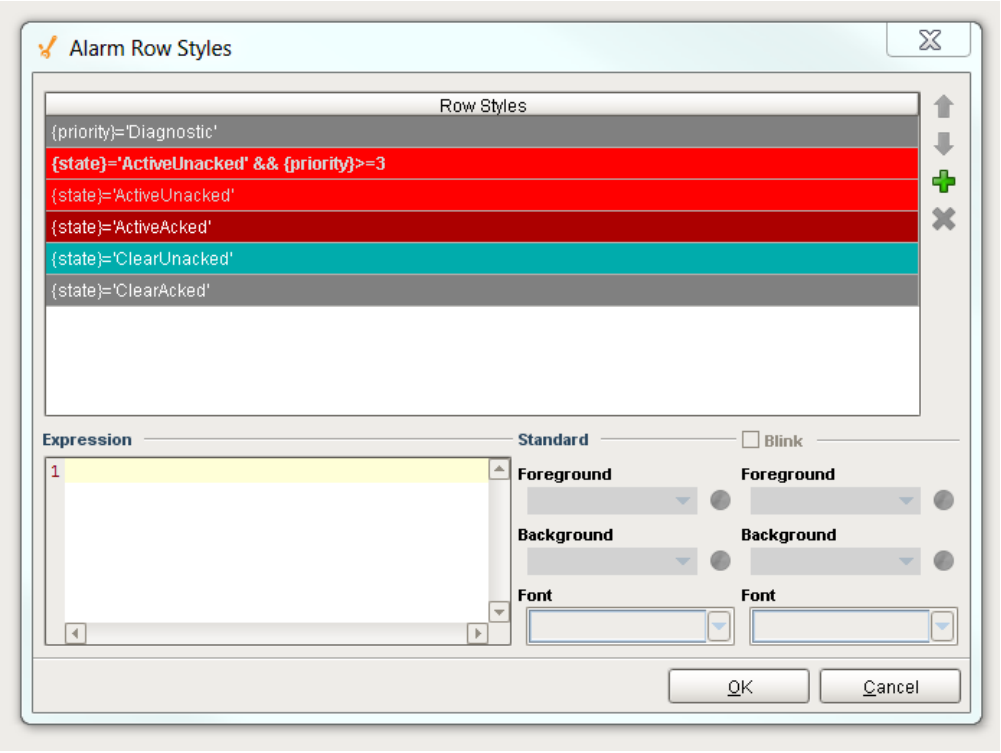
```
#The following code is an example of the filter alarm expression function.
#The function results in advanced filtering for the alarm table.
#In this example the alarm table will only show alarms with a name that matches the value of the
"AreaName" property located on the container the Alarm Status Table resides in.

name = self.parent.AreaName
if name == alarmEvent.get('name'):
        return True
else:
        return False
```

Gallery

Alarm Status Table with a Single Alarm



Alarm Row Style Customizer

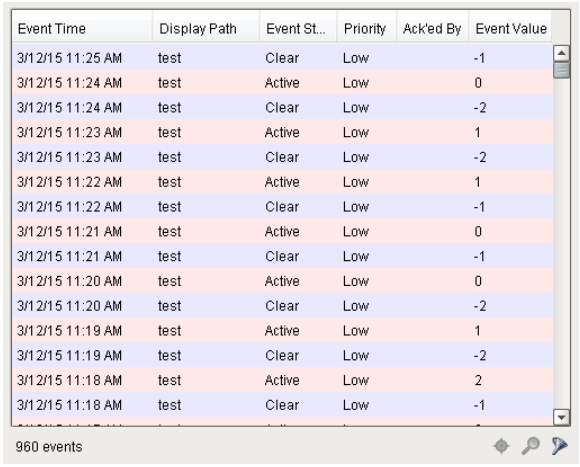


Description

The Alarm Row Styles Customizer manages the way the Alarm Status Table renders each alarm. The Alarm Row Styles Customizer allows you to change the styles of the alarms and the logic that governs each style. The Alarm Status Table evaluates each alarm and applies the the logic of the expression block to decide to implement a style. If the expression returns a logical "True" then the Alarm Row Style Customizer applies the color formatting options defined in the area to the right of the Expression block. If the expression returns a logical "False" then the Alarm Row Customizer evaluates the next expression associated with the next row style. The process continues until an expression returns a logical "True." There can be many rows with different logic and styles. You can add and remove rows by selecting the "plus" button or "delete" button.

Alarm Journal Table

General



Component Palette Icon:

Description

The alarm journal table provides a built-in view to explore alarm history that has been stored in an alarm journal. If you only have one alarm journal specified on your Gateway, then you do not need to specify the journal name. If you have more than one specified, then you need to provide the name of the journal you'd like to query.

The journal table shows the alarm history that is found between the Start Date and End Date properties. When you first put an alarm journal table on a window, these properties will be set to show the most recent few hours of journal history. Note that without further configuration, the journal table will always show the few hours before it was created. To properly configure an alarm journal table,please bind its start and end date properties to something what will update, such as the Date Range component or expressions involving the time now(). This way, you can configure it so that operators can choose the time to display, or have dates will be update automatically to have it poll.

To change the columns that are displayed, the order of the columns, and/or the column width, put the Designer into preview mode. Then right-click on the table header to show/hide columns. Click and drag to re-order columns, and drag the margins of the columns to resize their width. No further action is necessary - the column configuration will remain in place after the window is saved.

Additional examples of configuring the Alarm Journal Table can be found on the Alarm Journal Component page.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

- Description

  This specialized print function will paginate the table onto multiple pages.This function accepts keyword-style invocation.

- Keyword Args

  fitWidth - If true, the table's width will be stretched to fit across one page's width. Rows will still paginate normally. If false, the table will paginate columns onto extra pages. (default = true) [optional]

  headerFormat - A string to use as the table's page header. The substring "{0}" will be replaced with the current page number. (default = None) [optional]

  footerFormat - A string to use as the table's page footer. The substring "{0}" will be replaced with the current page number. (default = "Page {0}") [optional]

  showDialog - Whether or not the print dialog should be shown to the user. Default is true. [optional]

  landscape - Used to specify portrait (0) or landscape (1) mode. Default is portrait (0). [optional]

- Return

  boolean - True if the print job was successful.

- Scope

  Client

Extension Functions

- Description

  Returns a popup menu that will be displayed when the user triggers a popup menu (right click) in the table. Use system.gui.createPopupMenu to create the popup menu.

- Parameters

  self - A reference to the component that is invoking this function.

  selectedAlarmEvents - The alarm events selected on the Alarm Status Table. For an individual alarmEvent, call alarmEvent.get('propertyName') to inspect. Common properties: 'name','source','priority'.

- Return

  self - A reference to the component that is invoking this function.

  alarmEvent - The alarm event itself. Call alarmEvent.get('propertyName') to inspect. Common properties: 'name','source','priority'.

- Scope

  Client

- Description

  Called for each event loaded into the alarm status table. Return false to hide this event from the table. This code is executed in a background thread.

- Parameters

  self - A reference to the component that is invoking this function.

  alarmEvent - The alarm event itself. Call alarmEvent.get('propertyName') to inspect. Common properties: 'name', 'source', 'priority'.

- Return

  Boolean

- Scope

  Client

- Description

  Called when an alarm is double-clicked on to provide custom functionality. Does not return a value.

- Parameters

  self - A reference to the component that is invoking this function.

  alarmEvent - The alarm event itself. Call alarmEvent.get('propertyName') to inspect. Common properties: 'name', 'source', 'priority'.

- Return

  Nothing

- Scope

  Client

Event Handlers

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it. Unknown macro: 'sql'

This event occurs when a component that had the input focus lost it to another component. Unknown macro: 'sql'

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is. Unknown macro: 'sql'

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties. Unknown macro: 'sql'

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired. Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component. Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties. Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

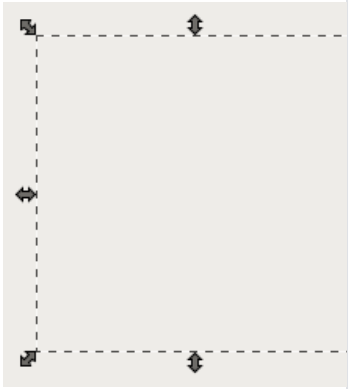
Examples

There are no examples associated with this component.

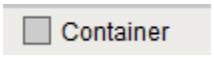
Containers

Container

General



Component Palette Icon:



Description

The container is a very important component. All components are always inside of a container, except for the special "Root Container" of each window (see Anatomy of a Window). A container is different than normal components in that it can contain other components, including other containers. Uses for containers include:

- **Organization.** Containers can be used to group components together. These components can then easily be moved, copied, or deleted as a group. Furthermore, they will all be organized inside of their parent container in the project navigation tree, which makes them easier to find.
- **Re-usability.** Containers allow a unique opportunity to create a complex component that is made up of multiple other components. The Container's ability to have dynamic properties aids this greatly. For instance, if you wanted to make your own custom HOA control, you can put three buttons inside of a container and configure them to all use a 'status' property that you add to their parent Container. Now you have built an HOA control that can be re-used and treated like its own component. The possibilities here are endless. Create a date range control that generates an SQL WHERE clause that can be used to control Charts and Tables. Create a label/button control that can be used to display datapoints, and pop up a parameterized window that displays meta-data (engineering units, physical location, notes, etc) about that datapoint. Creating re-usable controls with Containers containing multiple components is the key to rapid application development.
- **Layout.** Containers are a great way to improve window aesthetics through borders and layout options.

Properties

Scripting

Scripting Functions

This component does not have scripting functions associated with it.Remove the panel below if there are no scripting functions.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

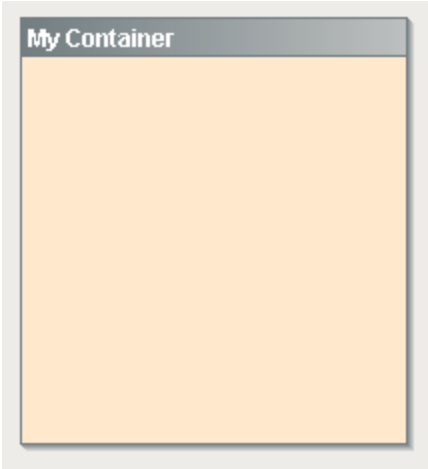
Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

Customized Container with Border

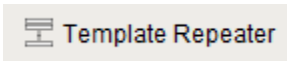


Property Name	Value
Border	Titled
Background Color	255,232,204

Template Repeater

General

Component Palette Icon:



INDUCTIVE UNIVERSITY

Template Repeater

Watch the Video

Description

The Template Repeater repeats instances of templates any number of times. It can arrange them vertically, horizontally, or in a "flow" layout, which can either be top-to-bottom or left-to-right. If there are too many to fit, a scrollbar will be shown. This makes it easy to quickly create screens that represent many similar pieces of equipment. It also can be used to create screens that are dynamic, and automatically configure themselves based on configuration stored in a database or tag structure. When first dropped on a window, the template repeater will look like any other empty container. To select the template to repeat, configure the repeater's Template Path property. There are two ways to set how many times the template should repeat:

- Count - The template will be repeated X times, where X is the value of "Repeat Count". The repeat count starts at zero and increments X amount of times. Each value for X will be inserted into the custom property of the template that will be repeated. Template repeater inserts the value of X into the custom property on the template with the same name as the template repeater's "Index Property Name." For example, if the template has a custom property of "index" and the template repeater's Index Property Name is also "index," then the template will be repeated X many time with the value of X being inserted into the template's custom property called "index."
- Dataset - The template will be repeated once for each row in the "Template Parameters" dataset. The template's custom properties with the same names as the dataset's column names will assume the values of each row of the dataset.

An Example of configuring the Template Repeater can be found on the Template Repeater Component page.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

- Description

  Returns a list of templates loaded into the Template Repeater. Properties on the components within each instance can be references by calling getComponent().

- Parameters

  None

- Return

  List of Templates

- Scope

  Client

Extension Functions

This component does not have any Extension Functions

Event Handlers

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

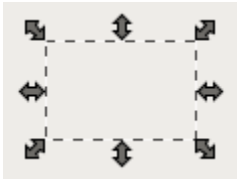
Examples

Code Snippet: getLoadedTemplates()

```
#This script will call getLoadedTemplates() on a Template Repeater, and
#then print the text property of a Label component in each instance

#Store a reference to the Template Repeater component in a variable
repeater = event.source.parent.getComponent('Template Repeater')

#Store the list of templates in another variable
templateList = repeater.getLoadedTemplates()

#Iterate through the list
for template in templateList:
        #find a component named "Label" in the instance,
        #and print the value of the text property
        print template.getComponent('Label').text
```

Template Canvas

General



Component Palette Icon:

INDUCTIVE UNIVERSITY

Template Canvas

Watch the Video

Description

The template canvas is similar to the template repeater but allows for more control of the templates than the template repeater.

The "Templates" property on the template canvas is a dataset. Each row in this dataset represents a manifestation of a template. It can be the same template or a different template on each row. This dataset allows for control over the size, position and layout of the template. There are two methods of controlling the layout of each template inside the template canvas:

- **Absolute Positioning:** The location of the template is explicitly managed through the "X" and "Y" columns of the "Templates" property's dataset. Consequently the columns labeled "width" and "height" control the size of the template.
- **Layout Positioning:** The template canvas uses "MiGLayout" to manage the location of the template. MigLayout is a common albeit complicated layout methodology. It supports layouts that wrap the templates automatically as well as docking the template to one side of the template canvas. You can learn more about MiG Layout at http://www.miglayout.com

In addition, control over data inside each template can be achieved by adding a column with the name "parameters" to the dataset and populating this column with dictionary style key works and definitions.

Additional templates can be added to the template canvas by inserting an additional row to the "Templates" property's dataset. The same applies to removing the templates but with removing the rows from the dataset.

Properties

Unknown macro: 'sql'

Example

The following example will create a form for users to input data.

Example

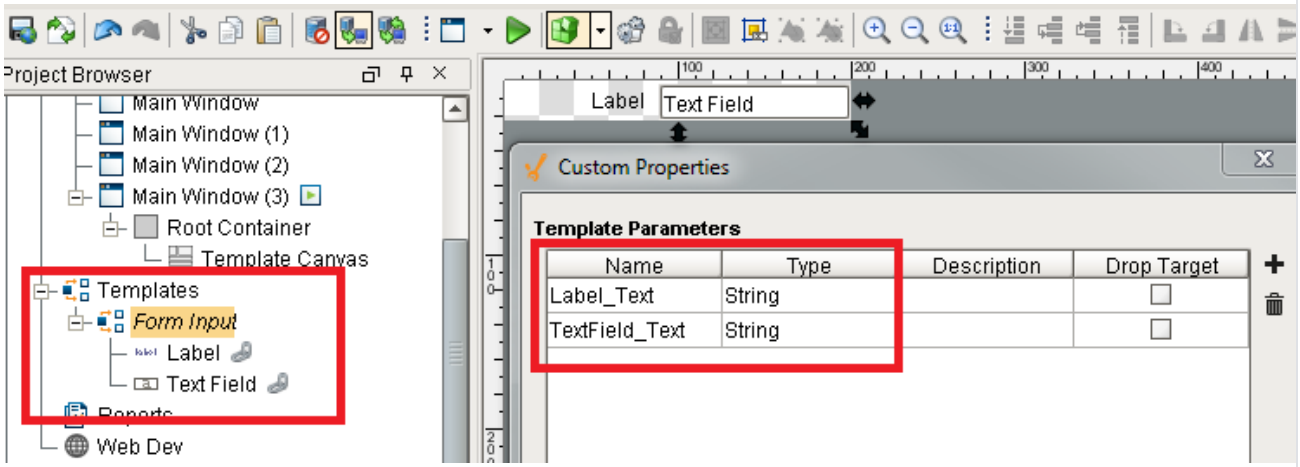
The following example will create a Form for users to input data in.

Absolute Positioning

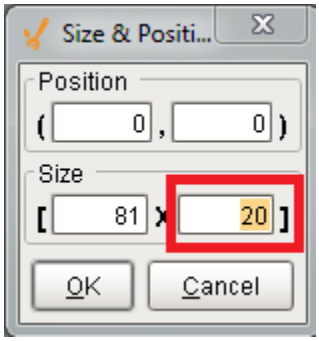
The example uses a template named Form Input. Form Input contains a Label component and Text Field component. Form Input also has two Template Parameteres: Label_Text and TextField_Text

- The Text property on the Label component is bound to the Label_Text parameter

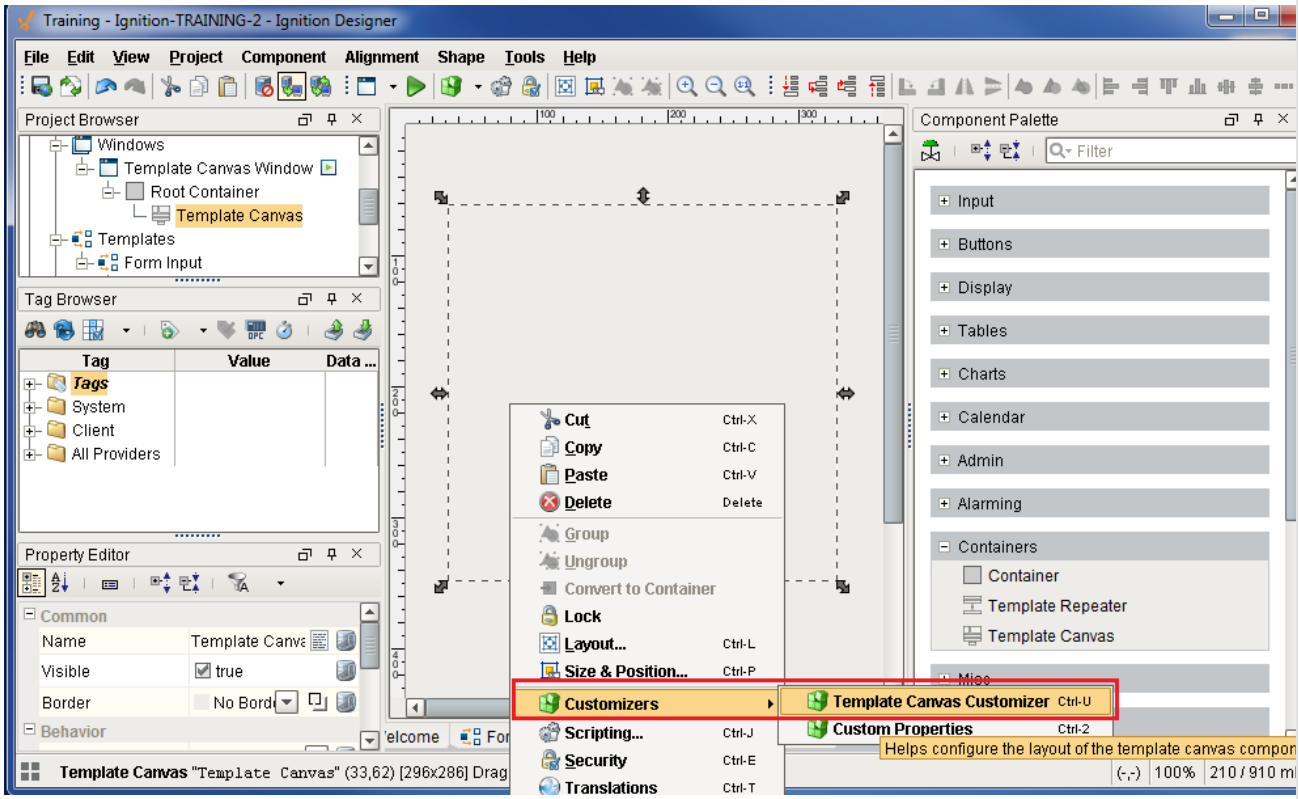
- The Text property on the Text Field component is bound to the TextField_Text parameter **bi-directionally**. This will allow the user to write to this property, and this makes reading the value later on easier.



Both Form Input and the components inside had their height set to 20 by right clicking on the components and selecting **Size and Position...** (Ctrl-P)

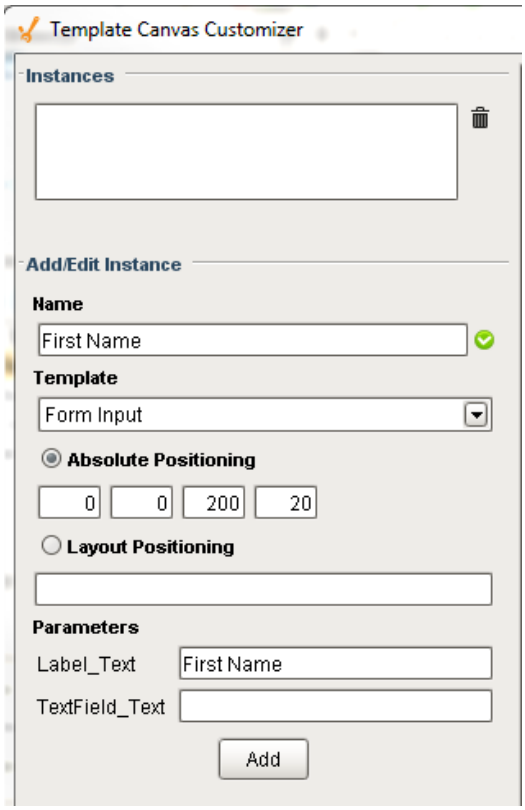


Once the template is in place, create a new window, place a Template Canvas Component down, and open the **Template Canvas Customizer** from either the right-click menu or Ctrl-U



Once open, the customizer allows us to specify the location of each template instance as well the value of any template parameters. Use the following values for the first instance of Form Input:

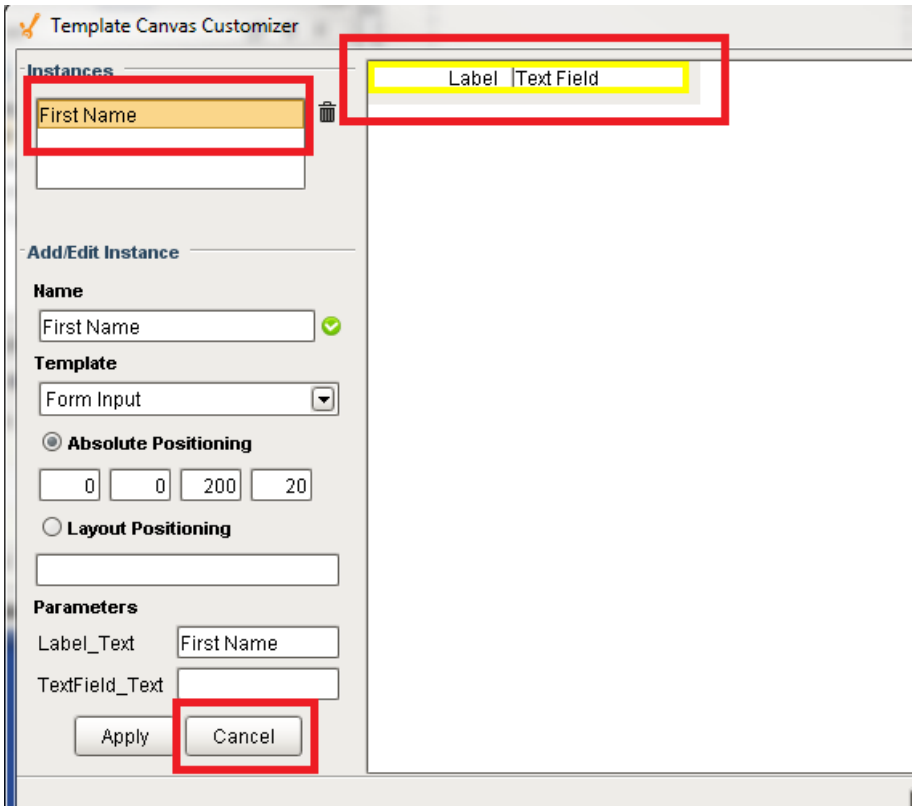
- Name: First Name
- Template: Form Input
- Position: Absolute, (0,0) [200x20]
- Parameters: Label_Text = First Name, TextField_Text = "" (blank)



Click the "Add" button to add an instance. Doing so will add an instance of Input Field. The instance will be visible in the preview section. Note that values for the components are still using the default values for the Label_Text and TextField_Text. This is intentional. The new values will appear once we close the customizer.

Take note of the yellow outline around the instance, and how First Name is highlighted at the top of the customizer. This means that the instance is selected, and the customizer is in edit mode. This allows use to make changes to the selected instance (First Name). To exit edit mode and add a new instance click on the "Cancel" button in the lower **left** of the window

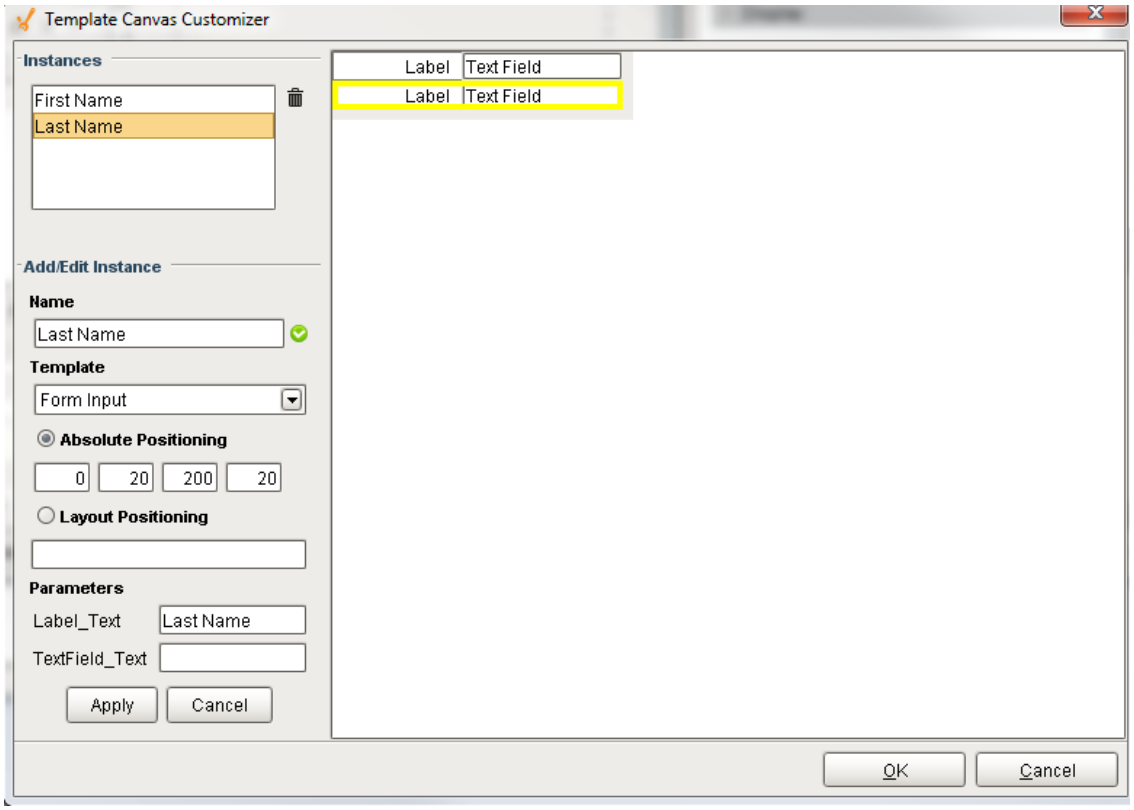
> ⚠ Clicking on the "Cancel" button in the lower **right** will cancel out of the customizer.



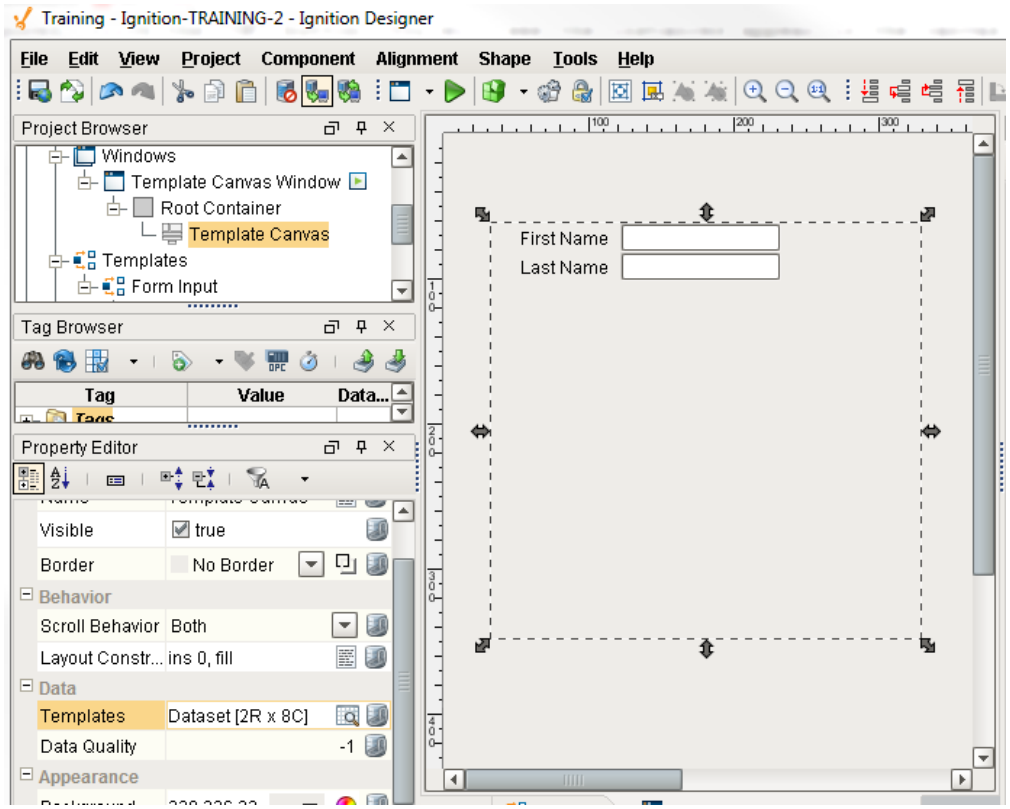
Once out of edit mode, start adding values for a new instance:

- Name: Last Name
- Template: Form Input
- Position: Absolute, (0,20) [200x20]
- Parameters: Label_Text = Last Name, TextField_Text = "" (blank)

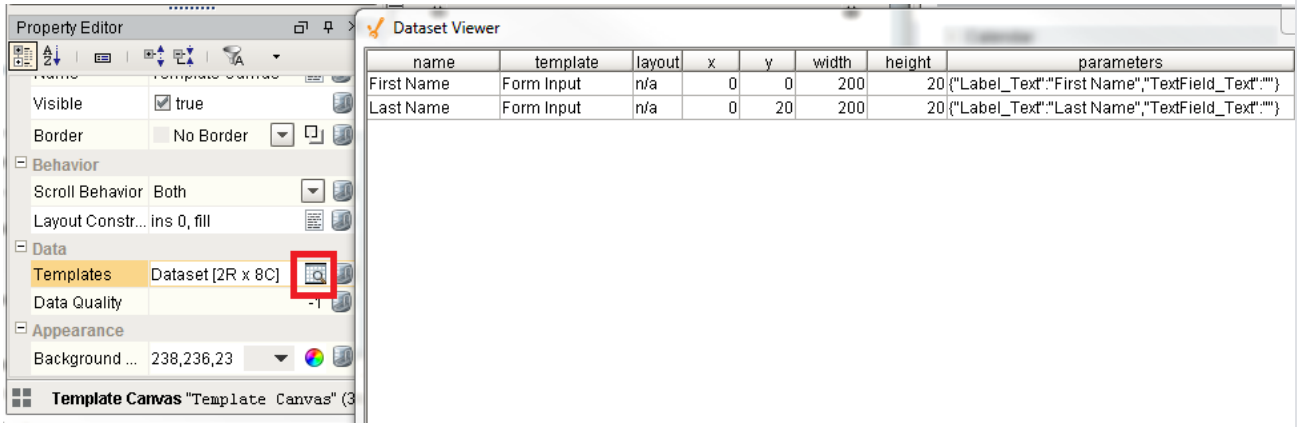
Once entered, click the add button again.



Once both instances have been configured, click the "OK" button. You will see the instances appear in the canvas.



If a new instance needs to be added, it can be added through the customizer. However, the Template Canvas also has a Templates property. This property stores all of the data that was entered into the customizer, so new instances can be configured directly on the Templates property. View the dataset by clicking the Dataset Viewer button next to the Templates property.



Furthermore, template instance configurations could be stored in a database table, and the template canvas could fetch the data with a SQL Query binding on the Templates property.

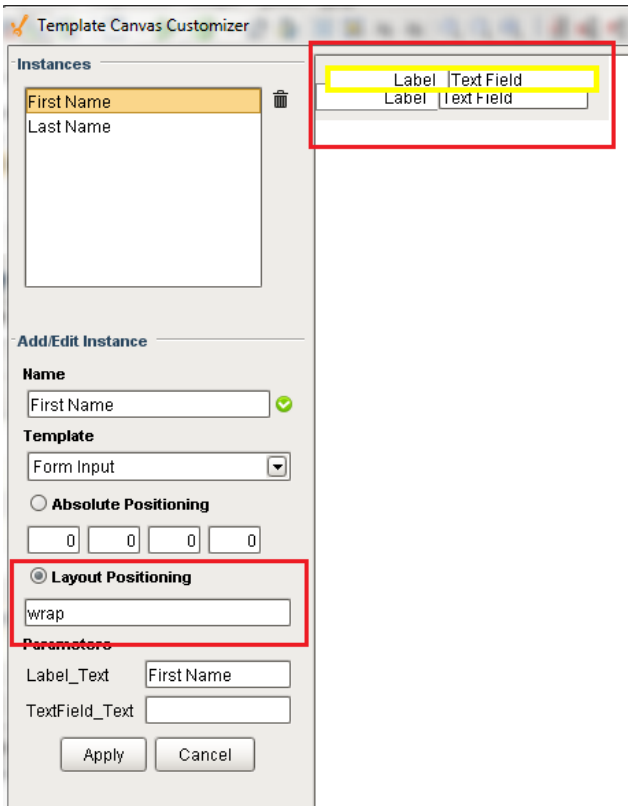
Layout Positioning

Instead of having to manually enter a size and position for each instance, we can make use of layout positioning to have the template canvas determine the best position for each instance, while also making suggestions as to where each instance is placed in relation to another. The layout positioning uses a grid-methodology to instance placement. Each instance, unless otherwise specified, is considered a single "cell" in the grid. We will tell the First Name instance to use the "wrap" parameter. This means the next cell in the grid should be placed on the next row.

Open the the Template Canvas Customizer again, and make the following modification to the First Name instance:

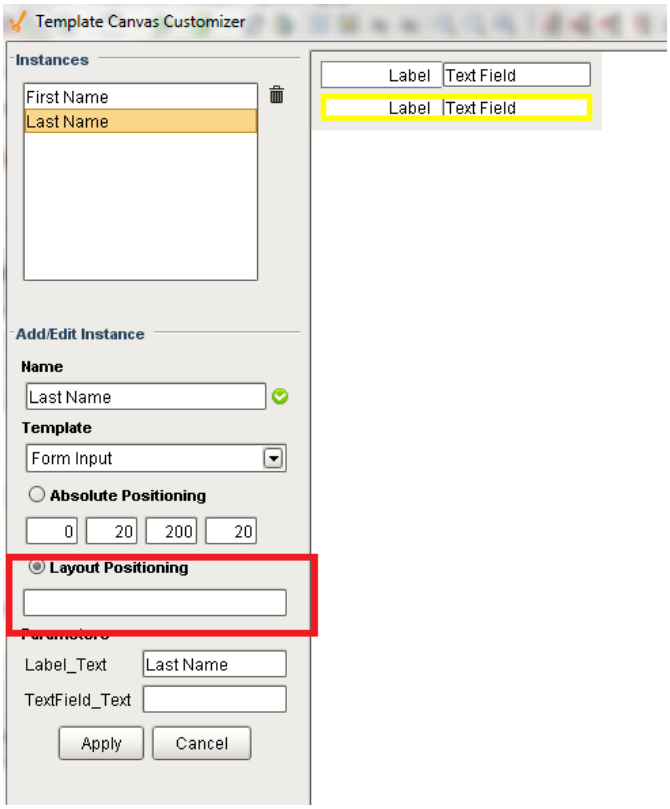
- Position: Layout Positioning
- Value: "wrap"

Click Apply, and the First Name instance will appear to overlap with the Last Name instance. This is because the grid only accounts for instances using the Layout Positioning.

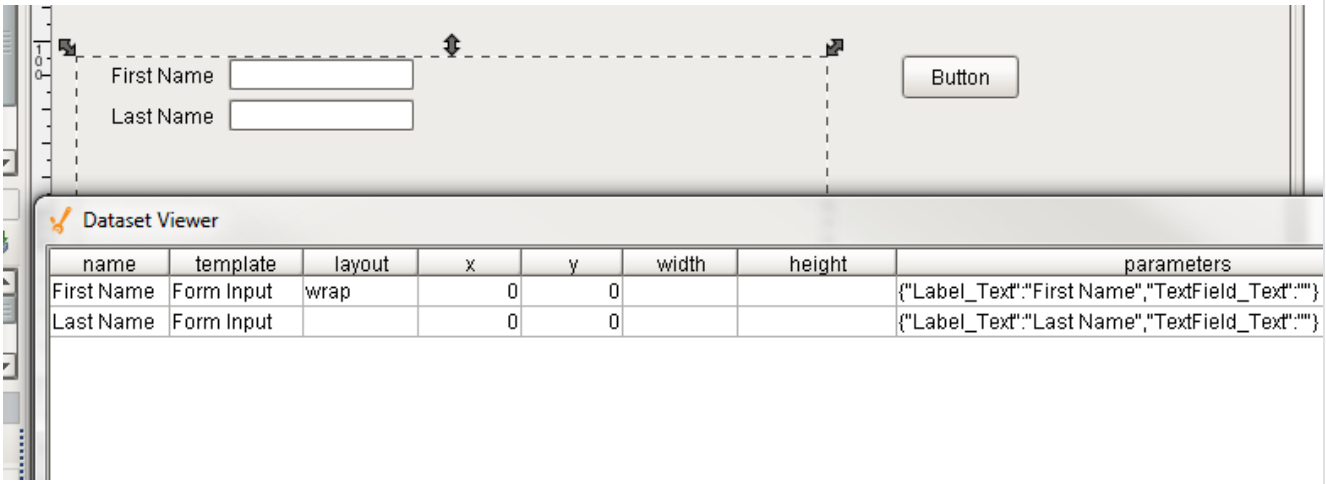


The easy way to fix this is to configure Last Name with Layout Positioning as well. Make the following changes to Last Name:

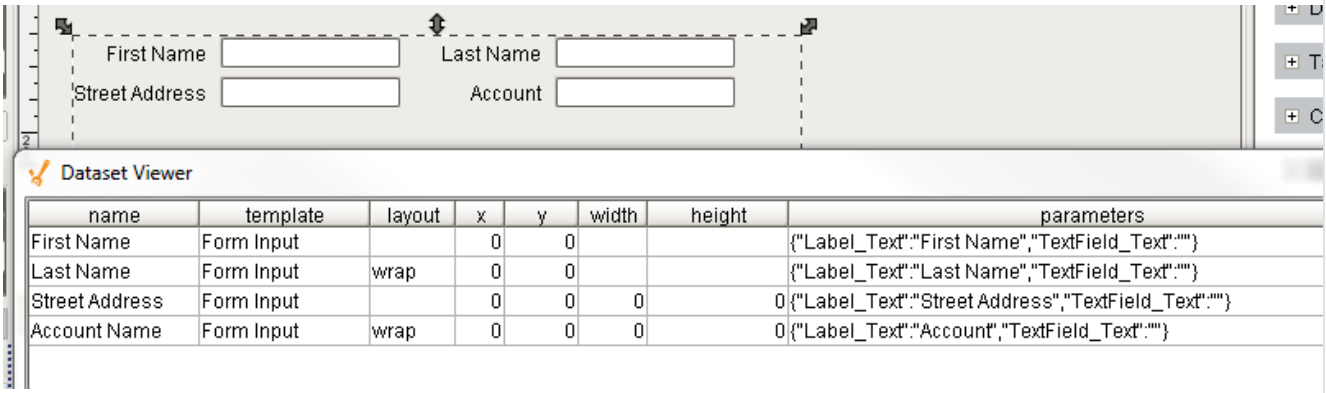
- Position: Layout
- Value: "" (blank)



Once the changes have been applied, Click "OK". Check the Templates property again. Notice that the x, y, width, and height columns are no longer used, but the layout column for First Name now has a value.

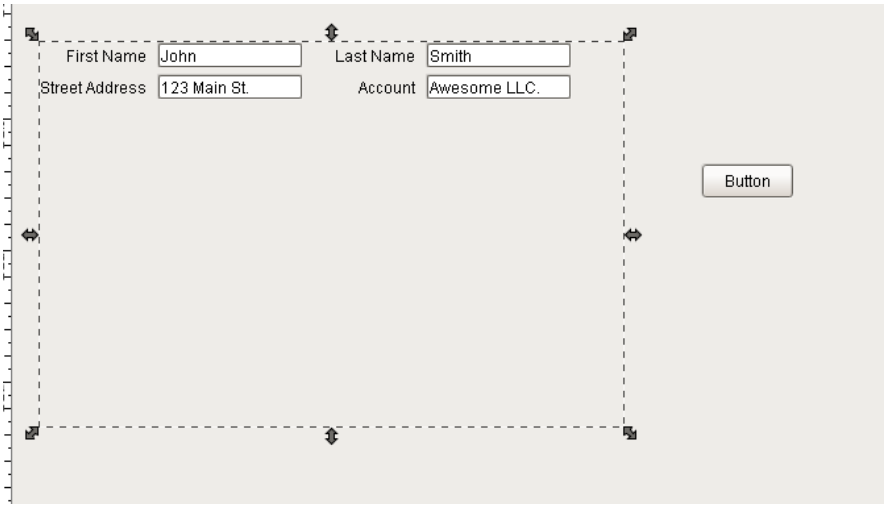


Like the previous example, new rows can be added directly to this dataset. Furthermore, the "wrap" value means the next template instance will begin on a new line. Add two new instances for Street Address and Account Name. Use either the customizer or simply add two new rows in the dataset viewer with the following values. Note that the layout value for First Name and Last Name have been changed since the previous image:

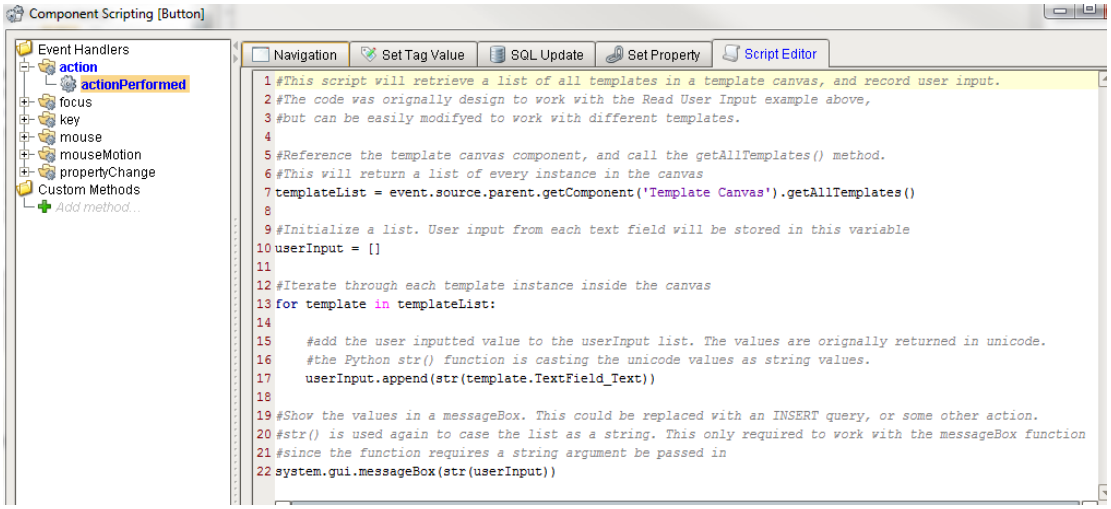


Read User Input

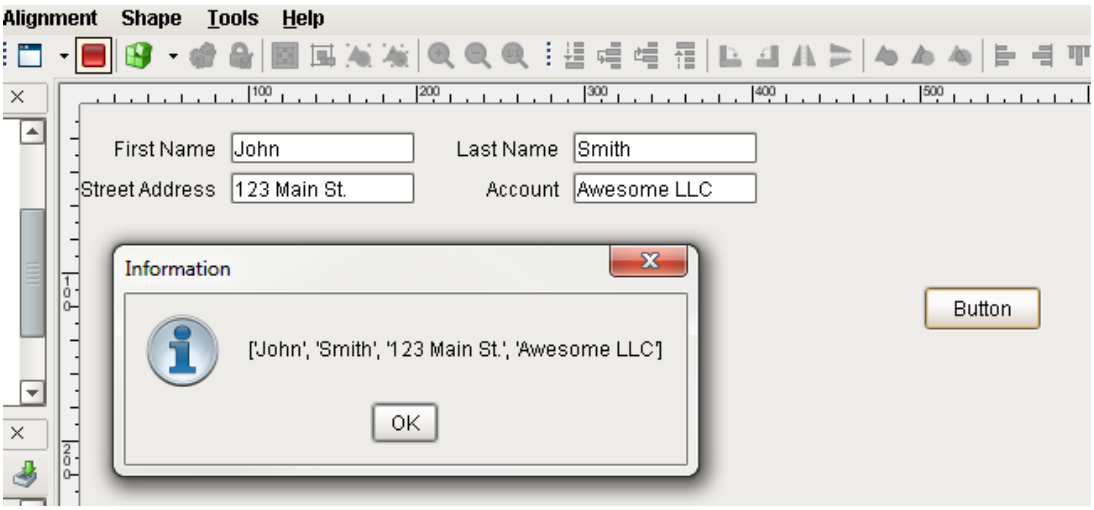
The last step is to read the user input. Put the designer into preview mode and add some values for each text field component. Once finished, Switch the Designer back to design mode, and add a button component to the window (not the template canvas)



Add a script to the button component using the Code Snippet - **Read User Input Example** below in the Examples section. Place the code on the actionPerformed event of your button.



Click "OK" to close the Component Scripting window. Save the Project, put the designer in preview mode, and click the button



Each value should appear in the message box. This example can easily be expanded to do something more meaningful with the input, like store to a database table.

Scripting

Scripting Functions

- Description

  Returns a list of the templates that comprise the template canvas.

- Parameters

  Nothing

- Return

  List - A list of VisionTemplate definitions. Each instance in the canvas will return it's definition's name. The names of each instance can be accessed with getInstanceName(). Individual components in each instance can accessed with getComponent().

- Scope

  Client

- Description

  Obtains the designated template object from the template canvas.

- Parameters

  name - The name of the template as defined by the "name" column of the dataset populating the template canvas.

- Return

  VisionTemplate - Returns the template instance. Properties on the instance can be access by calling .propertyName

- Scope

  Client

Extension Functions

- Description

  This will be called once per template that is loaded. This is a good chance to do any custom initialization or setting parameters on the template.

- Parameters

  Self- A reference to the component that is invoking this function.

  template - The template. The name of the template in the dataset will be available as template.instanceName

- Return

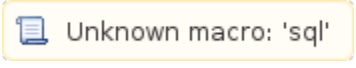
  Nothing

- Scope

  Client

Event Handlers

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.



Customizers

This component has a customizer.

Examples

Code Snippet

```
#This example demonstrates how to pull value information from templates that are inside the template
canvas.
#This example assumes that each template has a custom property called ContentValue

#Get all the template instances of the canvas.
templates = event.source.parent.getComponent('Template Canvas').getAllTemplates()

#The templates are a list therefore you can iterate through them.
for template in templates:

        #You can access the properties of the template. This example prints the ContentValue custom
property to the console.
        print template.ContentValue
```

Code Snippet - Seach by Name

```
#This example demonstrates how to iterate through each template in a template canvas
#looking for a named instance. Once found, print the value of a property on a component in
#that instance.

#This assumes that the canvas contains a template instance named "timerTemplate" and
#a Timer component (named Timer) is inside the instance.

#Create a reference to the Template Canvas
canvas = event.source.parent.getComponent('Template Canvas')

#Retrieve all template instances in the canvas
tempInstance = canvas.getAllTemplates()

#Iterate through each template instance
for template in tempInstance:

        #Compare the name of each instance.
        if template.getInstanceName() == "timerTemplate":

                #Print the Value property on the Timer component inside the template
                print template.getComponent("Timer").value
```

Code Snippet - Read User Input Example

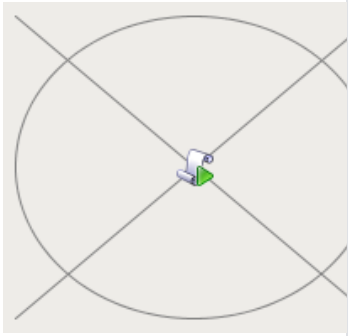
```
#This script will retrieve a list of all templates in a template canvas, and record user input.

#The code was originally design to work with the Read User Input example above,
#but can be easily modified to work with different templates.

#Reference the template canvas component, and call the getAllTemplates() method.
#This will return a list of every instance in the canvas
templateList = event.source.parent.getComponent('Template Canvas').getAllTemplates()

#Initialize a list. User input from each text field will be stored in this variable
userInput = []

#Iterate through each template instance inside the canvas
for template in templateList:

        #add the user inputted value to the userInput list. The values are originally returned in Unicode.
        #the Python str() function is casting the Unicode values as string values.
        userInput.append(str(template.TextField_Text))

#Show the values in a messageBox. This could be replaced with an INSERT query, or some other action.
#str() is used again to case the list as a string. This only required to work with the messageBox
function
#since the function requires a string argument be passed in
system.gui.messageBox(str(userInput))
```

Template Canvas Customizer

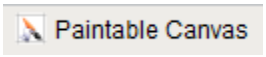
Misc

Paintable Canvas

General



Component Palette Icon:



Description

The Paintable Canvas component is a component that can be custom "painted" using Jython scripting. By responding to the component's repaint ev ent, a designer can use Java2D to draw anything within the component's bounds. Whenever any dynamic properties on the component change, the component is re-painted automatically, making it possible to create dynamic, vector-drawn components that can represent anything.

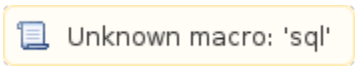
This component is an advanced component for those who are very comfortable using scripting. It is not user-friendly. The upside is that it is extraordinarily powerful, as your imagination is the only limit with what this component can be.

When you first drop a Paintable Canvas onto a window, you'll notice that it looks like a placeholder. If you switch the Designer into preview mode, you'll see an icon of a pump displayed. The pump is an example that comes pre-loaded into the Paintable Canvas. By editing the component's event scripts, you can dissect how the pump was drawn. You will notice that the script uses Java2D. You can read more about Java2D here. You will notice that as you resize the pump, it scales beautifully in preview mode. Java2D is a vector drawing library, enabling you to create components that scale very gracefully.

Tips:

- Don't forget that you can add dynamic properties to this component, and use the styles feature. Use the values of dynamic properties in your repaint code to create a dynamic component. The component will repaint automatically when these values change.
- You can create an interactive component by responding to mouse and keyboard events
- You can store your custom components on a custom palette and use them like standard components.

Properties



Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event occurs when a component that can receive input, such as a text box, receives the input focus. This usually occurs when a user clicks on the component or tabs over to it.Unknown macro: 'sql'

This event occurs when a component that had the input focus lost it to another component.Unknown macro: 'sql'

An integer that indicates whether the state was changed to "Selected" (on) or "Deselected" (off). Compare this to the event object's constants to determine what the new state is.Unknown macro: 'sql'

Fires when a key is released and the source component has the input focus. Works for all characters, including non-printable ones, such as SHIFT and F3.Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.Unknown macro: 'sql'

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component.Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component.Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component.Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed.Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed.Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.Unknown macro: 'sql'

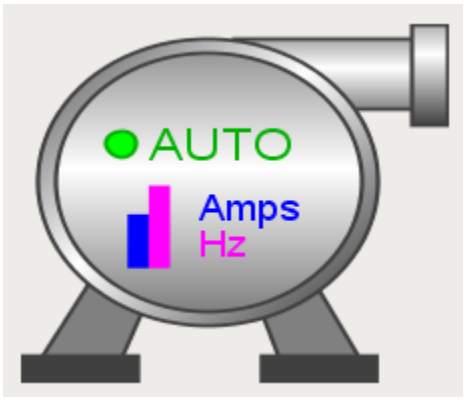
Customizers

This component does not have any custom properties.

Examples

There are no examples associated with this component. However examples are available in the component itself.

The component comes prescripted to render the following pump:



Line

General

Component Palette Icon:

Description

The line component displays a straight line. It can run north-south, east-west, or diagonally. You can add arrows to either side. The line can be dashed using any pattern you want. You can even draw the line like a sinusoidal wave!

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'
This event fires when a mouse button is released, if that mouse button's press happened over this component.
Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

This component does not have examples associated with it.

Pipe Segment

General

Component Palette Icon:

Description

The pipe segment component displays a quasi-3D pipe. In its basic form it looks very much like a rectangle with a round gradient. The difference comes in its advanced rendering of its edges and endcaps. You can configure each pipe segment's end to mate perfectly with another pipe segment butted up against it perpendicularly. The result looks like a pipe welded together in a 90° corner.

The control of the pipe's ends can be a bit confusing to a new user. It is done via 6 booleans - three per 'end'. End 1 is the top/left end, and End 2 is the bottom/right end. You turn off each boolean if there will be another pipe butted up against that side. The following diagram should make the naming conventions more clear:

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'
This event fires when a mouse button is released, if that mouse button's press happened over this component.
Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

Customizers

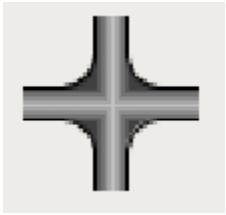
This component does not have any custom properties.

Examples

There are no examples associated with this component.

Pipe Joint

General



Component Palette Icon:

Description

The pipe joint displays a fancy joint component two join two pipe segments together. By turning off the cardinal directions, this will display a 2,3, or 4-pipe union. This component is optional, as pipes can butt up against each other and look joined.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.Remove the panel below if there are no scripting functions.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.

Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

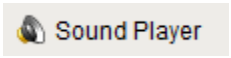
Examples

There are no examples associated with this component.

Sound Player

General

Component Palette Icon:



Description

The Sound Player component is an invisible component that facilitates audio playback in the client. Each Sound Player component has one sound clip associated with it, and will play that clip on demand. There is a built in triggering system, as well as facilities to loop the sound while the trigger is set. Note that the sound clip needs to be a *.wav file, and that the clip becomes embedded within the window that the sound player is on - clients do not need access to a shared *.wav file.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'
This event fires when a mouse button is released, if that mouse button's press happened over this component.
Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

There are no examples associated with this component.

Timer

General

Component Palette Icon:

Description

The timer button is an invisible button that can be used to create repeated events in a window. This is often used for animations or repetitive scripts within a window. When running, the timer's Valueproperty is incremented by the Step By value, until the value tis the Bound, at which point it repeats. It is often useful to bind other values to a timer's Value property.

For instance, if you set the timer's Bound property to 360, and bind an object's rotation to the Value property, the object will spin in a circle when the timer is running.

How fast the timer counts is up to the Delay property, which is the time between counts in milliseconds.

Want to run a script every time the timer counts? First, make sure you don't actually want to write a project Timer Script, which will run on some interval whenever the application is running. In contrast, a script that works via a Timer component will only run while the window that contains the Timer is open, and the Timer is running. The way to do this is to attach an event script to the actionPerformedevent.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

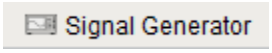
Expression Binding Example

```
//Suppose that you have images that make up frames of animation.
//Name your images: "Frame0.png", "Frame1.png", "Frame2.png". Set the timer's Bound to be 3, then bind
the image path of animate component to the following expression:
"Frame" + {Root Container.Timer.value} + ".png"
```

Signal Generator

General

Component Palette Icon:



Description

The signal generator is similar to the Timer component, but its value isn't simply a counter. Instead, you can choose from a variety of familiar 'signals'. You configure the frequency by setting the Periodproperty, which is in milliseconds. You configure the resolution by setting the Values /Period property.

For example, if you choose a sine wave signal with a period of 2000 milliseconds and 10 values/period, your sine wave will have a frequency of 0.5 Hz, and its value will change 10 times every 2 seconds.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed.
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.

Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

This component does not have any examples associated with it.

Reporting Components

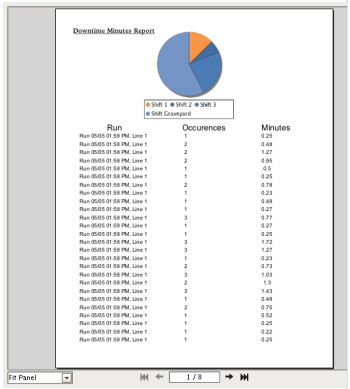
Report Viewer

The following feature is new in Ignition version **7.8.0**
Click here to check out the other new features

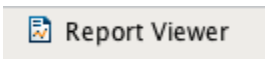
Description

This component provides a way to run and view Reports in Vision windows. Parameters added during Report creation are provided as Properties in the Viewer and can override any default values set in the Report Resource. See the Reporting Module section for more about creating dynamic reports to embed within Vision. To find documentation on the deprecated Report Viewer prior to Ignition 7.8, see the Legacy Report Viewer documentation

General



Component Palette Icon:



Properties

Name	Description	Type	Scripting	Category
Background Color	Color that lays underneath the report.	Color	.background	Appearance
Border	The border surrounding this component. NOTE that the border is unaffected by rotation.	Border	.border	Common
Font	Font of text of this component	Font	.font	Appearance
Foreground Color	The foreground color the labels on the component.	Color	.foreground	Appearance
Mouseover Text	The text that is displayed in the tooltip which pops up on mouseover of this.	String	.toolTipText	Common
Name	The name of this component.	String	.name	Common
Print Mode	Sets the printing mode. Vector is fastest and high-quality for printers that support it, but Raster mode can help spool size with older printers.	Int	.printingMode	Behavior
Retain Page on Reload	If false, always reloads report to page 1	Boolean	.retainPageOnReload	Behavior
Suggested Filename	The filename that will come up by default when the user saves the report to disk.	String	.suggestedFilename	Behavior
Visible	If disabled, the component will be hidden.	Boolean	.visible	Common

Scripting

Scripting Functions

The following print methods will only work if a report has finished loading on the Report Viewer component

print()

- Description

  Uses the system default printer and shows a print dialog.

- Keyword Args

  none

- Return

  none

- Scope

  Client

print(printerName)

- Description

  Uses the named printer and shows a print dialog.

- Keyword Args

  String printerName - The name of the printer the report should be sent to

- Return

  none

- Scope

  Client

print(printerName, showDialog)

- Description

  Uses the named printer and determine if the print dialog window should appear or not.

- Keyword Args

  String printerName - The name of the printer the report should be sent to

  Boolean showDialog - True if the dialog window should appear, False if the dialog window should be skipped.

- Return

  none

- Scope

  Client

getBytesPDF()

The following feature is new in Ignition version **7.8.3**
Click here to check out the other new features

- Description

  Return the bytes of the generated report in the Report Viewer using PDF format.

- Keyword Args
- Return

  Byte Array - The bytes of the report in PDF format.

This function will return null if the trial has expired.

- Scope

  Client

getBytesPNG()

The following feature is new in Ignition version **7.8.3**
Click here to check out the other new features

- Description

  Return the bytes of the generated report in the Report Viewer using PNG format.

- Keyword Args
- Return

  Byte Array - The bytes of the report in PNG format.

This function will return null if the trial has expired.

- Scope

  Client

saveAsPDF(fileName)

- Description

  Prompts the user to save a copy of the report as a PDF. Shows a file selection window with the extension set to PDF.

- Keyword Args

  String fileName - A suggested filename to save the report as

- Return

  none

- Scope

  Client

saveAsPNG(fileName)

- Description

  Prompts the user to save a copy of the report as a PNG. Shows a file selection window with the extension set to PNG.

- Keyword Args

  String fileName - A suggested filename to save the report as.

- Return

  none

- Scope

  Client

Extension Functions

- Description

  Called when the Report generation process has been completed.

- Keyword Args

  self - A reference to the component invoking this method.

  pdfBytes - The PDF formatted bytes generated by the Report.

- Return

  none

- Scope

  Client

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'
This event fires when a mouse button is released, if that mouse button's press happened over this component.
Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

Examples

print()

```
#calls print on a Report Viewer component located in the same window

reportViewer = event.source.parent.getComponent('Report Viewer')
reportViewer.print()
```

print() with default printer, no dialog

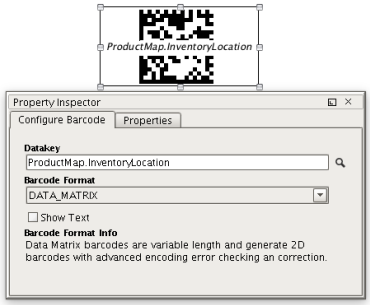
```
#calls print on a Report Viewer component located in the same window
#bypasses the print dialog window and uses the default printer

reportViewer = event.source.parent.getComponent('Report Viewer')
reportViewer.print(None, False)
```

saveAsPDF()

```
#Saves the file as a PDF to a user selected location.
#The file selection window defaults to a name of "Daily Report"

reportViewer = event.source.parent.getComponent('Report Viewer')
reportViewer.saveAsPDF("Daily Report")
```

Barcodes

The Barcode component allows reports to contain dynamically generated barcodes. With Reporting v3 we added support for a number of 2D barcodes.



Usage

To add a barcode to your report, simply drag the Barcode item off the Report Design Palette and drop onto a page. You will see a placeholder barcode with its driving data key displayed, with an example encoding of the type specified in the **Barcode Format** of the Configuration area. Data Keys can be dropped or typed into the configuration panel, and the driving key will be shown in its appropriate barcode placeholder. The actual barcode, as well as any text to be displayed under the barcode, is generated and rendered when the report is executed or previewed.

Barcodes are commonly embedded into unstructured Table Rows, or Labels for printing. When used in set-driven components such as these, each barcode can be encoded to a unique value provided by the driving data source.

Configuration

Barcodes require a **Data Key** or Text to encode. Note that each Format encoder has specific requirements for what it will encode. The barcode configuration panel displays some brief information about the selected format, but is not comprehensive.

The **Show Text** option of the barcode will add a String representation of the encoded data on the final barcode. Enabling the Show Text option will allow the configuration of Font for the displayed text.

Supported Encoders

Supported barcode formats include:

- Code 39
- Code 128
- Data Matrix
- EAN 8
- EAN 13
- ITF
- PDF 417
- QR Code
- UPC A

Examples

To see an example of embedding a barcode in a DataSource driven component, take a look at the Labels component example where we embed barcodes in the repeating Labels component to dynamically create coded labels to track items.

Images

The Image component  lets you embed images into your reports. Images can be dropped as files onto reports, pulled as binary data from a data source, or even a URL.

Usage

To create a new image, either use the Image tool on the Report Design Palette or drag an image from your desktop. If created from the toolbar, you can add the image source to the **Key** property of the Image Property Inspector table. For more flexibility, you can use a Parameter that resolves to a URL to get the image.

Labels

Labels 

can be used to print out mailing labels, create name tags, or any other generic labels. You can use standard Avery label sheets or specify your own dimensions.

Usage

The Labels component is created by selecting it and dragging it onto the Page of the Report Designer. The size of the labels component is automatically calculated based on the number and size of the labels. The label size and spacing can be set in the Configuration panel when the Labels component is the selected item. To edit the label itself, super-select the template label (always the top-left label in the component) and add text, shapes, data keys or keychain expressions, Images or Barcodes. Keep in mind that any data keys or expressions within the Label component are driven by the Data Key configured in the component (see example below).

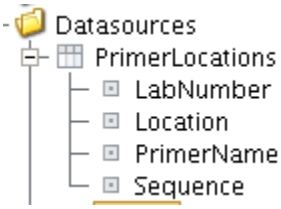
Configuration Items	Function
Data Key	Name of DataSet that will populate the labels
Label Formatting	Choose from a list of Avery Label Formats, or create a custom sized label.
Rows/Columns	Defines the number of rows and columns on the page
Label Width/Height	Width and height of labels in pixels
Spacing Width/Height	Distance between labels on the page in pixels

Example

Hypothetical Use Case

We are going to create a new set of custom labels for our University Research Lab to better track the DNA Primers we use in experiments. Primers are expensive and our researchers are getting Carpal Tunnel from the extra Grant Proposals they have to write to fund replacement of Primers that Student Interns keep losing. We have invested in a QR Code Scanner that we will use to help track the Primers. We want the labels to contain info about the Primer and where they belong. In addition, we want the barcode to contain the name, number of the lab, and the location within the lab that the samples below.

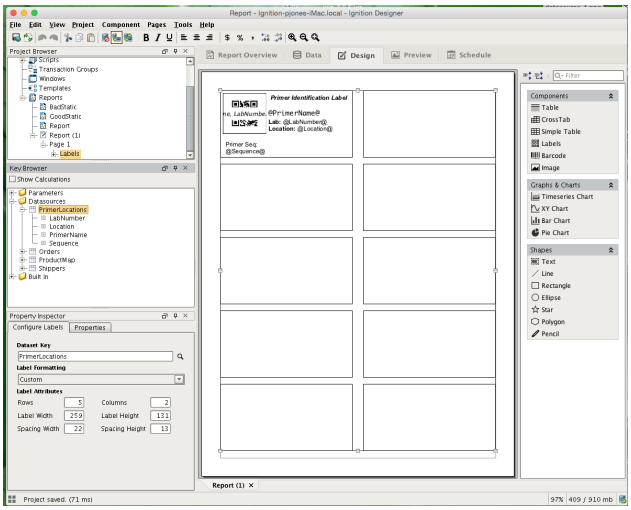
Our Datasource for this information lives in an SQL Table (or maybe an Excel CSV export used as a Static CSV Datasource) which has this structure:



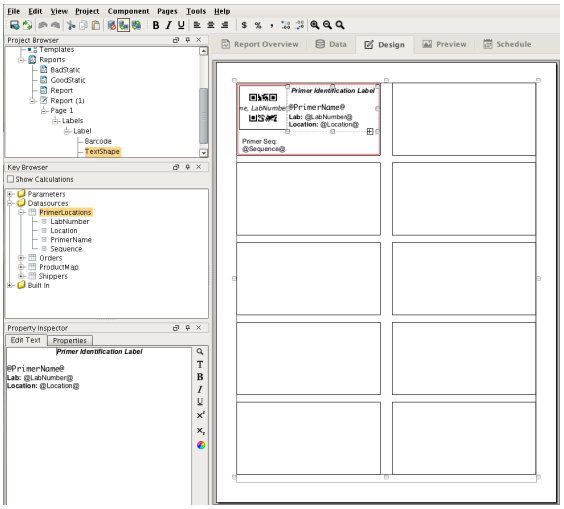
Here are the steps we take to create our labels.

1. Create a Labels component from the Report Design Palette by dragging the Labels item onto your Report.
2. Specify the Data Key that maps to the Data Source you want to drive the label
3. Choose the Appropriate Label Size (or create your own)
4. Create the template for your labels in the Label Component's driving Box
5. Verify Label layout using the Report Preview

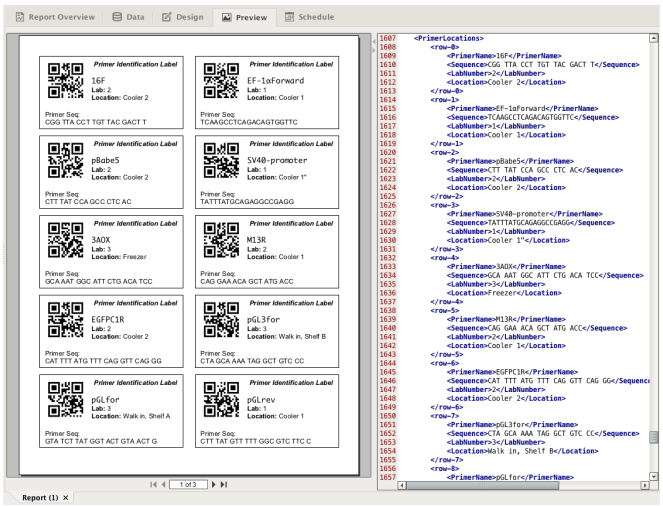
Here is how our finished Layout looks:



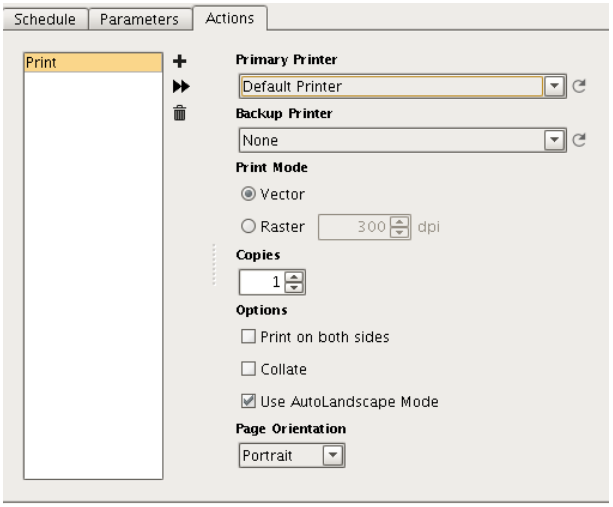
We used a Barcode embedded into the template label to create our QR Codes. Since the labels will be driven by the PrimerLocations Data Key, we can reference the child keys within the components or keychain expressions located within the label. In this case, we formatted the string we want the barcode to encode by adding *"PrimerName, LabNumber, Location"* to the Data Key field of the barcode. To make the label human readable, we added some text shapes via selecting the shape and dragging within the super-selected template label as can be seen in the next image. We mix plain text and data keys by using our @ symbols to surround things we want to resolve to data when the report is generated.



When we are happy with our design, we can switch to the Preview to take a look at the result.

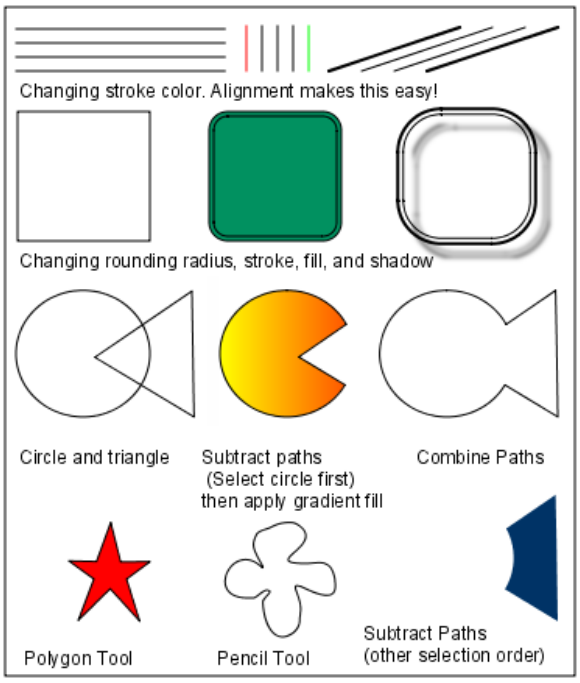


Finally, we can print our labels by creating a Print Action on the Schedule tab, and running it by clicking the "Run Immediately" button. If we know we get new shipments of Primers on a set schedule, we could have these labels print automatically. In this case, we want to simply print on demand, so we disabled the schedule on the Schedule tab before setting up our printer.



This completes this example. Feel free to comment if you have questions or recommendations!

Report Drawing Shapes



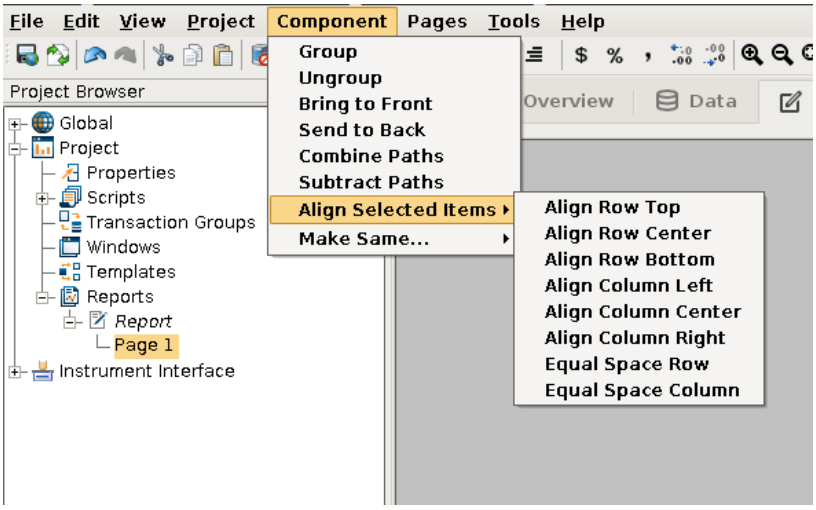
Examples using the
Drawing Shapes
Basic drawing shapes are found under Shapes on the component pallet
located in the Design Panel

Drawing Shapes

Name	Description
Text Shape	Click and drag to create text.
Line Shape	Click and drag to create a line.

Rectangle Shape	Click and drag to create a rectangle.
Ellipse Shape	Click and drag to create an ellipse.
Star Shape	Click and drag to create a star.
Polygon Shape	The polygon shape lets you click points that will be joined with straight lines. Alternatively, you can click-drag-release to position line segments interactively. Editing stops under the following conditions: clicking the same point twice, clicking close to the start point or clicking a new component in the component pallet
Pencil Shape	The pencil tool lets you click and draw free-hand path segments, automatically smoothing the curve on mouse release.

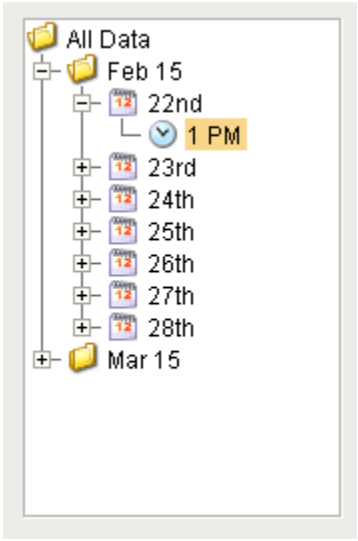
Components Menu



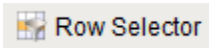
Menu Item	Function
Group/Ungroup	Allows you to merge the currently selected shapes into a single shape for convenient management. Contained shapes are still accessible, via double-click super-select. Ungroup separates grouped shapes.
Bring to Front /Send to Back	All shapes have an order on the page that determines what is drawn on top when two shapes overlap. These options allow you to alter that order.
Align Row Top /Center/Bottom	Quickly align several shapes in a row, either by their top, center, or bottom border. Useful when shapes are of different heights.
Align Column Left /Center/Right	Same as above, but for columns.
Equally Space Row/Column	Equalizes the distance between shapes horizontally or vertically.
Make Same Size, Width, Height	Make several shapes the same width, height or both.
Combine/Subtract Paths	Takes multiple overlapping shapes (such as a rectangle and an oval) and combines them into a single shape using the combined paths. A powerful tool to construct complex shapes.

Row Selector

General



Component Palette Icon:



INDUCTIVE UNIVERSITY

Row Selector

Watch the Video

Description

The row selector is a component that acts like a visual filter for datasets. It takes one dataset, chops it up into various ranges based on its configuration, and lets the user choose the splices. Then it creates a virtual dataset that only contains the rows that match the selected splices.

The most common way to splice the data is time. You could feed the row selector an input dataset that represents a large time range, and have it break it up by Month, Day, and then Shift, for example. Then you could power a report with the output dataset, and that would let the user dynamically create reports for any time range via an intuitive interface.

To configure the row selector, first you set up the appropriate bindings for its input dataset. Then you use its Customizer to alter the levels that it uses to break up the data. In the customizer, you add various filters that act upon columns in the input dataset, sorting them by various criteria. For example, you could choose a date column, and have it break that up by quarter. Then below that, you could have it use a discrete filter on a product code. This would let the user choose quarterly results for each product. Each level of filter you create in the customizer becomes a level in the selection hierarchy. Note that the output data is completely unchanged other than the fact that rows that don't match the current user selection aren't present.

This component is very handy for driving the Report Viewer, Table, and Classic Chart components, among others.

Additional information on the Row Selector can be found on the Row Selector Component page in the appendix.

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'
This event fires when a mouse button is released, if that mouse button's press happened over this component.
Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

Customizers

The row selector tree customizer allows you to customize the row filtering.

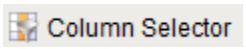
Examples

There are no examples associated with this component.

Column Selector

General

Component Palette Icon:



INDUCTIVE UNIVERSITY

Column Selector

Watch the Video

Description

The column selector component is conceptually similar to the Row Selector, except that instead of filtering rows, it filters columns from its output dataset. Each column from the input dataset is shown as a checkbox. As the user checks and un-checks columns, the output dataset has those columns added or removed. This is very handy for driving the Table and Classic Chart components.

Addition information on the Column Selector can be found on the Column Selector Component page

Properties

Unknown macro: 'sql'

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'
This event fires when a mouse button is released, if that mouse button's press happened over this component.
Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

Customizers

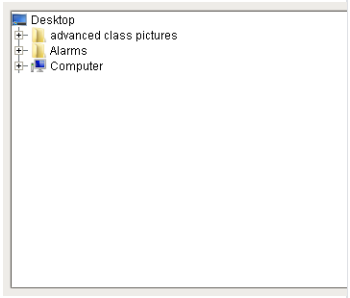
A customizer exist that allows you to configure the columns.

Examples

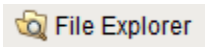
There are no examples associated with this component.

File Explorer

General



Component Palette Icon:



Description

The File Explorer component displays a filesystem tree to the user. It can be rooted at any folder, even network folders. It can also filter the types of files that are displayed by their file extension (For example, "pdf"). The path to the file that the user selects in the tree is exposed in the bindable property Selected Path.

This component is typically used in conjuction with the PDF Viewer component, in order to create a PDF viewing window. This is very useful for viewing things like maintenance manuals from within your project. To use this component to drive a PDF Viewer component, follow these steps:

- Bind the PDF Viewer's Filename property to the File Explorer's Selected Path property
- Set the File Explorer's File extension filter to "pdf"
- Set the File Explorer's Root Directory to a network folder that has your maintenance manuals in it. (Use a network folder so that all clients will be able to access the manuals).

Properties

Scripting

Scripting Functions

This component does not have scripting functions associated with it.

Extension Functions

This component does not have scripting functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'
This event fires when a mouse button is released, if that mouse button's press happened over this component.
Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'
Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

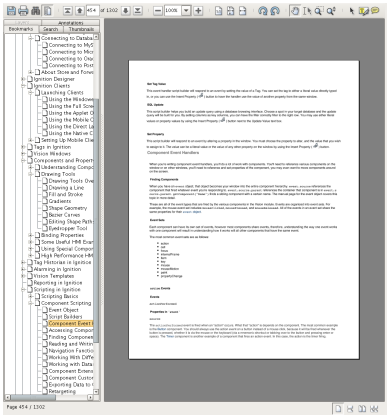
Examples

There are no examples associated with this component.

PDF Viewer

New in 7.8!

General



Component Palette Icon:

Description

Looking for documentation on the legacy PDF Viewer component? Please see the IA Labs PDF Viewer page

The PDF Viewer component displays a PDF that exists as a file in some accessible filesystem, or as a URL. Note that this component is simply for viewing existingPDFs. To create dynamic reports, or view dynamically generated reports, use Reporting Module.

This component is typically used in conjunction with the File Explorer component, in order to create a PDF viewing window. Simply bind the Selected Path property in the PDF Viewer to the File Explorer's *Selected Path* property. See the File Explorer's documentation, as well as the File Explorer and PDF Viewer page for further instructions on how to put these two components together.

Warning. Past versions of this component had some limitations on what it could present. In the Reporting Module versions 2.x, the PDF Viewer component could only be guaranteed to correctly display reports generated by the Report Viewer. In practice, it is able to view many PDFs, but it does have trouble with some, especially PDFs created by AutoCAD. Users unable to upgrade to Ignition 7.8 should consider installing the PDF-Viewer module from IA-Labs. This module is the basis on which the new 7.8 version of the PDF Viewer is built.

Properties

Name	Description	Property Type	Scripting	Category
Border	The border surrounding this component. NOTE that the border is unaffected by rotation.	Border	.border	Reporting
File Path	Path to the .pdf file to be displayed.	String	.filePath	Reporting
Footer Visible	If false, the Footer is not displayed.	Boolean	.footerVisible	Reporting
Name	The name of this component.	String	.name	Reporting
Page Fit Mode	Mode to fit the document within the viewer. (1 = Disabled, 2 = Actual Size, 3 = Fit Height, 4 = Fit Width)	Integer	.pageFitMode	Reporting
Page View Mode	How to display PDF in Viewer (1 = One Page, 2 = One Column, 3 = Two Page Left, 4 = Two Col Left, 5 = Two Page Right, 6 = Two Col Right)	Integer	.pageViewMode	Reporting
Toolbar Visible	Sets the top PDF control toolbar to visible.	Boolean	.toolBarVisible	Reporting
Utility Visible	Sets the Utility Sidebar to visible.	Boolean	.utilityPaneVisible	Reporting
Visible	If disabled, the component will be hidden.	Boolean	.visible	Reporting

Scripting

Scripting Functions

- Description

  This function will pass in the bytes of a PDF and load them into the PDF Viewer component. Please see Storing Files in a Database for more details

- Parameters

  string bytes - The bytes of the PDF to be displayed on the component

  string name - The name of the PDF

- Return

  Nothing

- Scope

  Client

- Since 7.8.2

- Description

  This function will print the PDF.

- Parameters

  boolean- If true, shows the user a print dialog. Default is true [optional]

- Return

  Nothing

- Scope

  Client

- Since 7.8.2

- Description

  This function will set the current zoom level of the PDF, adjusted to stay within the minimum / maximum zoom range. Will zoom in on center of page.

- Parameters

  float- Zoom factor to use. 1.0 is no zoom.

- Return

  Nothing

- Scope

  Client

Extension Functions

This component does not have extension functions associated with it.

Event Handlers

This event signifies a mouse click on the source component. A mouse click the combination of a mouse press and a mouse release, both of which must have occurred over the source component. Note that this event fires after the pressed and released events have fired.
Unknown macro: 'sql'

This event fires when the mouse enters the space over the source component. Unknown macro: 'sql'

This event fires when the mouse leaves the space over the source component. Unknown macro: 'sql'

This event fires when a mouse button is pressed down on the source component. Unknown macro: 'sql'

This event fires when a mouse button is released, if that mouse button's press happened over this component.
Unknown macro: 'sql'

Fires when the mouse moves over a component after a button has been pushed. Unknown macro: 'sql'

Fires when the mouse moves over a component, but no buttons are pushed. Unknown macro: 'sql'

Fires whenever a bindable property of the source component changes. This works for standard and custom (dynamic) properties.
Unknown macro: 'sql'

Customizers

This component does not have any custom properties.

Examples

There are no examples associated with this component.

Expression Functions

The expression language is used to define dynamic values for component properties and expression tags. Expressions often involve one or more other values that are used to calculate a final value. Expressions don't do anything, other than return a value.

For an overview and syntax of the expression language, see Expression Language Overview and Syntax.

Expression Overview and Syntax

Overview

The expression language is used to define dynamic values for component properties and expression tags . Expressions often involve one or more other values that are used to calculate a final value. Expressions don't do anything, other than return a value.

The classic example for an expression is to change a temperature that is stored in Celsius to Fahrenheit in order to display it. Suppose you had a tag,Tank 6/Temp that was in Celsius. If you wanted to display that tag in Fahrenheit on a Label, you would use an Expression Binding on the label's text property using the following expression:

```
1.8 * {Tank 6/Temp} + 32
```

On this page...

The expression language is used to define dynamic values for component properties and expression tags. Expressions often involve one or more other values that are used to calculate a final value. Expressions don't do anything, other than return a value.

The classic example for an expression is to change a temperature that is stored in Celsius to Fahrenheit in order to display it. Suppose you had a tag,T ank 6/Temp that was in Celsius. If you wanted to display that tag in Fahrenheit on a Label, you would use an Expression Binding on the label's text property using the following expression:

Every time that the temperature tag changes, the expression will re-calculate the value and push it into the Label's text property. Now lets say that you wanted to append a "°F" to the end of the label so that the user knew the units of the temperature. You could simply use some string concatenation in your expression, like this:

```
(1.8 * {Tank 6/Temp} + 32) + " °F"
```

The expression language is used to define dynamic values for component properties and expression tags. Expressions often involve one or more other values that are used to calculate a final value. Expressions don't do anything, other than return a value.

The classic example for an expression is to change a temperature that is stored in Celsius to Fahrenheit in order to display it. Suppose you had a tag,T ank 6/Temp that was in Celsius. If you wanted to display that tag in Fahrenheit on a Label, you would use an Expression Binding on the label's text property using the following expression:

Every time that the temperature tag changes, the expression will re-calculate the value and push it into the Label's text property. Now lets say that you wanted to append a "°F" to the end of the label so that the user knew the units of the temperature. You could simply use some string concatenation in your expression, like this:

Lets suppose that you wanted to give the user an option to display the value in Celsius or Fahrenheit, based on checking a checkbox. You could add a Check Box component to th screen called DisplayFahrenheit. Then you could use this expression to dynamically display either unit, based upon the user's selection:

```
if({Root Container.DisplayFahrenheit.selected}, (1.8 * {Tank 6/Temp} + 32) + " °F", {Tankf/Temp} +  " °C")
```

Syntax

As its name suggests, everything in the expression language is an "expression". This means that everything returns a value. 5 is an expression. So is 5+1. So are {MyTags/TankLevel} and{MyTags/TankLevel}+1. Expressions can be combined in many powerful ways. Lets take a look at how expressions are written.

More formally, an expression is any one of the following:

- Number
- Boolean
- String
- Bound tag
- Bound property
- Function call
- Dataset access
- Equation involving any of these

Literal Values

Literal values are things like numbers, booleans, and strings that are represented directly in the language. In the expression language, numbers can by typed in directly as integers, floating point values, or using hexadecimal notation with a 0x prefix. Examples:

```
42
8.456
0xFFC2
```

Strings are represented by surrounding them with double or single quotes. You can use the backslash character to escape quotes that you want to be included in the string. Examples:

```
"This is a regular string"
'This one uses single quotes'
"This string uses \"escaping\" to include quotes inside the string"
```

Operators

You can use these arithmetic, logical, and bit-shifting operators to combine expressions.

Operator	Name	Description
--	Unary Minus	Negates a number.
!	Not	Logical opposite of a boolean.
^	Power	Raises a number to the power of another number. a^b is ab.
%	Modulus	Modulus or remainder of two numbers. a%b is the remainder of a÷b.
*	Multiply	
/	Divide	
+	Add/Concatonate	If both operands are numbers, this will add them together. Otherwise treats arguments as strings and performs concatenation.
-	Subtraction	
&	Bitwise AND	
|	Bitwise OR	
xor	Bitwise XOR	
<<	Left Shift	A signed bitwise left shift.
>>	Right Shift	A signed bitwise right shift.
>	Greater Than	Logical greater-than test between two numbers. Returns a boolean.
<	Less Than	
>=	Greater Than or Equal To	
<=	Less Than or Equal To	
=	Equal	Tests for equality between two operands.
!=	Not Equal	Tests for equality, returning true when not equal.
&&	Logical AND	Returns true when both operands are true. Anything non-zero is considered true.
||	Logical OR	Returns true when either operand is true. Anything non-zero is considered true.
like	Fuzzy String Matching	Compares the left-hand value with the pattern on the right side. The pattern may consist of %,*, and ? wildcards.
//	Comments	Allows for comments following this operator.

Bound Values

Bound values are paths to other values enclosed in braces. These will appear red in the expression editor. When you are writing an expression for a E xpression Binding, you can reference tag values and property values using the brace notation. When you are writing an expression for an Expression

Tag, you can only reference other tag values. You can use the Insert Property ( ) and Insert Tag Value ( ) buttons to build these references for you.

Dataset Access

If you have an expression that returns a dataset, you can pull a value out of the datatset using the dataset access notation, which takes one of these forms:

```
Dataset_Expression ["Column_Name"]    //returns the value from the first row at the given column name
Dataset_Expression [Column_Index] //returns the value from the given column at the first row
Dataset_Expression [Row_Index, "Column_Name"] //returns the value from the given row at the given column name
Dataset_Expression [Row_Index, Column_Index] //returns the value from the given row at the given column index
```

For example, this expression would pull a value out of a Table at row 6 for column "ProductCode":

```
{Root Container.Table.data}[6, "ProductCode"]
```

Note that you'll often have to convince the expression system that what you're doing is safe. The expression language can't tell what the datatype will be for a given column, so you may have to use a type-casting function to convince the expression language to accept your expression, like this:

```
toInt({Root Container.Table.data}[6, "ProductCode"])
```

Expression Functions

The expression language's functions are where much of the real power lies. A function may take various arguments, all of which can themselves be any arbitrary expression. This means that you can use the results of one function as the argument to another function. In general, the syntax for a function call is:

```
functionName(expression1, expression2, ...)
```

Whitespace and Comments

Whitespace, such as spaces, tabs and newlines, are largely ignored in the expression language. It is often helpful to break your expression up onto multiple lines for clarity. Comments are delimited by two forward slashes. This will make the rest of that line be ignored. This example shows an if functi on spread over 4 lines with comments annotating the arguments.

```
if( {Root Container.UseTagValueOption.selected},
    {MyTags/SomeValue}, // Use the tag value
    "Not Selected",     // Use default value if the user doesn't check the box
)
```

Aggregates

- groupConcat
- max
- maxDate
- mean
- median
- min
- minDate
- stdDev
- sum

groupConcat

Description

Concatenates all of the values in the given column of the given dataset into a string, with each value separated by the string separator. Any null values in the column are ignored.

Syntax

groupConcat(dataset, column, separator)

Examples

Suppose you had a table with this dataset in it:

Product Code	Quality	Weight
BAN_002	380	3.243
BAN_010	120	9.928
APL_000	125	1.287
FWL_220	322	7.889

Code Snippet

```
groupConcat({Root Container.Table.data}, 1, " / ")  //would return the string: "380 / 120 / 125 / 322"
```

Code Snippet

```
groupConcat({Root Container.Table.data}, "ProductCode", ", ")  //would return the string: "BAN_002, BAN_010,
APL_000, FWL_220"
```

max

Description

Finds and returns the maximum value in the given column of the given dataset, or the max value in a series of numbers specified as arguments. When looking up the max in a dataset, the column may be specified as an index or as a column name. Any null values in the column are ignored. If there are no rows in the dataset, zero is returned.

Syntax

max(dataset, column OR number, number...)

Examples

For example, suppose you had a table with this dataset in it:

ProductCode	Quantity	Weight
BAN_002	380	3.243
BAN_010	120	9.928
APL_000	125	1.287
FWL_220	322	7.889

```
max({Root Container.Table.data}, 1) //would return 380
```

```
max(0, 10/2, 3.14) //would return 5. You can also use this function to find the maximum in fixed series of numbers, specified as arguments
```

```
max({SomeValue}, 0) //The following example is a great way to make sure a value never goes below zero:
```

maxDate

Description

Finds and returns the maximum date in the given column of the given dataset, or the max value in a series of dates specified as arguments. When looking up the max date in a dataset, the column may be specified as an index or as a column name. Any null values in the column are ignored. If there are no rows in the dataset, null is returned.

Syntax

maxDate(dataset, columnIndex OR date, date...)

Examples

The following table applies to the code snippet below:

AlarmTime	Path	Severity
2010-01-08 7:28:04	Tanks/Tank5/TempHiAlarm	4
2010-01-08 10:13:22	Tanks/Tank38/LoLevel	2
2010-01-08 13:02:56	Valves/Valve2/	2

```
maxDate({Root Container.Table.data}, "AlarmTime") //You could use this expression to get the date and time
for the most recent alarm:
```

mean

Description

Calculates the mean (a.k.a average) for the numbers in the given column of the given dataset or the mean of a series of numbers specified as arguments. When looking up the mean in a dataset, the column may be specified as an index or as a column name. Any null values in the column are ignored. If there are no rows in the dataset, zero is returned.

Syntax

mean(dataset, column OR number, number...)

Examples

For example, suppose you had a table with this dataset in it:

ProductCode	Quantity	Weight
BAN_002	380	3.243
BAN_010	120	9.928
APL_000	125	1.287
FWL_220	322	7.889

```
mean({Root Container.Table.data}, "Weight") //... would return 5.58675
```

```
mean(1,2,3) //... would return 2
```

median

Description

Calculates the median for the numbers in the given column of the given dataset or the median of a series of numbers specified as arguments. When looking up the median in a dataset, the column may be specified as an index or as a column name. Any null values in the column are ignored. If there are no rows in the dataset, zero is returned.

Syntax

median(dataset, column OR number, number...)

Examples

For example, suppose you had a table with this dataset in it:

ProductCode	Quantity	Weight
BAN_002	380	3.243
BAN_010	120	9.928
APL_000	125	1.287
FWL_220	322	7.889

```
median({Root Container.Table.data}, "Weight") //... would return 5.566
```

```
median(1,2,3,3,10) //... would return 3
```

min

Description

Finds and returns the minimum value in the given column of the given dataset, or the min value in a series of numbers specified as arguments. When looking up the min in a dataset, the column may be specified as an index or as a column name. Any null values in the column are ignored. If there are no rows in the dataset, zero is returned.

Syntax

min(dataset, column OR number, number...)

Examples

For example, suppose you had a table with this dataset in it:

ProductCode	Quantity	Weight
BAN_002	380	3.243
BAN_010	120	9.928
APL_000	125	1.287
FWL_220	322	7.889

```
min({Root Container.Table.data}, 1) //... would return 120
```

```
min(0, 10/2, 3.14) //... would return 0
```

```
min({SomeValue}, 180} //This example is a great way to make sure a value never goes above 180
```

minDate

Description

Finds and returns the minimum date in the given column of the given dataset, or the min value in a series of dates specified as arguments. When looking up the min date in a dataset, the column may be specified as an index or as a column name. Any null values in the column are ignored. If there are no rows in the dataset, null is returned.

Syntax

minDate(dataset, columnIndex OR date, date...)

Examples

For example, suppose you had a Table with this dataset in it:

AlarmTime	Path	Severity
2010-01-08 7:28:04	Tanks/Tank5/TempHiAlarm	4
2010-01-08 10:13:22	Tanks/Tank38/LoLevel	2
2010-01-08 13:02:56	Valves/Valve2/	2

```
 minDate({Root Container.Table.data}, "AlarmTime") //You could use this expression to get the date and time for the oldest alarm
```

stdDev

Description

Calculates the standard deviation of the values in the given column of the given dataset, or the standard deviation for a series of numbers specified as arguments. When looking up the standard deviation in a dataset, the column may be specified as an index or as a column name. Any null values in the column are ignored. If there are no rows in the dataset, zero is returned.

Syntax

stdDev(dataset, column OR number, number...)

Examples

For example, suppose you had a table with this dataset in it:

ProductCode	Quantity	Weight
BAN_002	380	3.243
BAN_010	120	9.928
APL_000	125	1.287
FWL_220	322	7.889

```
stdDev({Root Container.Table.data}, "Weight") //... would return 4.00532
```

sum

Description

Calculates the sum of the values in the given column of the given dataset, or the sum for a series of numbers specified as arguments. When looking up the sum in a dataset, the column may be specified as an index or as a column name. Any null values in the column are ignored. If there are no rows in the dataset, zero is returned.

Syntax

sum(dataset, column OR number, number...)

Examples

For example, suppose you had a table with this dataset in it.

ProductCode	Quantity	Weight
BAN_002	380	3.243
BAN_010	120	9.928
APL_000	125	1.287
FWL_220	322	7.889

```
sum({Root Container.Table.data}, 1) //... would return 947
```

```
sum(1,2,3) //... would return 6
```

Alarming Expressions

isAlarmActive

Description

Returns whether there are active alarms that match the provided criteria. The alarm name is optional, and both the tag path and alarm name support wildcards ('*'). For example, if only the tag path was specified, this function would return whether any alarm on the tag was active. The pollRate parameter is only applicable in the client scope.

Syntax

isAlarmActive(tagPath, [alarmName], [pollRate])

Examples

```
 isAlarmActive("Tanks/Temp", "Tank_Temp_High")   //when the Tank_Temp_High alarm is active then this
expression returns True.
```

Colors

brighter

Description

Returns a color that is one shade brighter than the color given as an argument. Note that if you pass in a fully saturated color, like (255,0,0), it cannot be made brighter.

Syntax

brighter(color)

Examples

```
brighter(color(100,150,250)) //returns the color (142,214,255)
```

color

Description

Creates a color using the given red, green, and blue amounts, which are integers between 0-255. The optional alpha channel to the color controls transparency.

Syntax

color(red, green, blue, [alpha])

Examples

There are no expression function examples associated with this expression function.

darker

Description

Returns a color that is one shade darker than the color given as an argument.

Syntax

darker(color)

Examples

```
darker(color(100,150,250)) //returns the color (70,105,175)
```

gradient

Description

Calculates a percentage given the three numeric arguments number, low, and high. Uses this percentage to create a color that is a mix between the two colors.

Syntax

gradient(number, low, high, lowColor, highColor)

Examples

```
gradient(0, 0, 100, toColor("red"), toColor("blue")) //returns red.
```

```
gradient(100, 0, 100, toColor("red"), toColor("blue")) //returns blue.
```

```
gradient(60, 0, 100, toColor("red"), toColor("blue")) //returns a shade of purple.
```

```
gradient({Root Container.Tank.value}, 0, 100, color(255,0,0), color(0,0,255)) //will return a gradient from red to blue based on the level of a tank.
```

Date and Time

dateArithmetic

Description

Adds or subtracts some amount of time from a date, returning the resulting date. The field argument must be a string, and must be one of these options:

- ms
- second
- sec
- minute
- hour
- hr
- day
- week
- month
- year

Syntax

dateArithmetic(date, number, field)

Examples

```
dateArithmetic(toDate("2010-01-04 8:00:00"), 5, "hour") //returns the date '2010-01-04 13:00:00'
```

```
dateArithmetic({Root Container.DatePicker.date}, -8, "days") //returns a date eight days before the date in a Popup Calendar component.
```

dateDiff

Description

Calculates the difference between the two dates, returning the result as a floating point value in the units specified by field, which must be a string matching one of these values:

- ms
- second
- sec
- minute
- min
- hour
- hr
- day
- week
- month
- year

The return value will be a floating point value, meaning that parts of units are considered. The exception to this rule is for the months and years fields, which will always return an integral difference. If the second date argument is after the first, the return value will be positive, otherwise it will be negative.

Syntax

dateDiff(date, date, field)

Examples

```
dateDiff(toDate("2008-2-24 8:00:00"), toDate("2008-2-24 8:15:30"), "minute") //returns 15.5
```

```
dateDiff(toDate("2008-2-24 8:00:00"), toDate("2008-3-12 9:28:00"), "month") //returns 1.0
```

```
dateDiff(toDate("2008-2-24 8:00:00"), toDate("2008-3-12 9:28:00"), "day") //returns 17.02
```

dateExtract

Description

Returns an integer value that is the value of the specified date field within the given date. The field must be a string, and must match one of these values:

- ms
- second
- sec
- minute
- min
- hour
- hr
- day
- week
- month
- year
- dayofweek
- dayofyear

Note: months are returned zero-indexed. That is, January is month 0, February is month 1, and so on. If this is inconvenient for you - just add one to the results.

Syntax

dateExtract(date, field)

Examples

```
dateExtract(toDate("2003-9-14 8:00:00"), "year") //returns 2003
```

```
dateExtract(toDate("2009-1-15 8:00:00"), "month") //returns 0
```

```
dateExtract(toDate("2008-1-24 8:00:00"), "month") + 1 //returns 1
```

dateFormat

Description

Returns the given date as a string, formatted according to a pattern. The pattern is a format that is full of various placeholders that will display different parts of the date. These are case-sensitive! These placeholders can be repeated for a different effect. For example, M will give you 1-12, MM will give you 01-12, MMM will give you Jan-Dec, MMMM will give you January-December.

The placeholders are:

Symbol	Description	Presentation	Examples	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	

Expert Tip: This function uses the Java class java.text.SimpleDateFormat internally, and will accept any valid format string for that class.

Syntax

dateFormat(date, pattern)

Examples

```
 dateFormat(toDate("2003-9-14 8:00:00"), "yyyy-MM-dd HH:mm:ss") //returns the string "2003-09-14 08:00:00"
This format is accepted in most databases
```

```
 dateFormat(toDate("2003-9-14 8:00:00"), "yyyy-MM-dd h a") //returns the string "2003-09-14 8 AM"
```

```
 dateFormat(toDate("2003-9-14 8:00:00"), "MMM d, yyyy") //returns the string "Sep 14, 2003"
```

now

Description

Returns the current time. The host computer's system clock is used, meaning that if this expression is being evaluated in a running client, the computer running the client's system clock is used. Note that this function is one of the few expression functions that will poll. If you do not specify a pollRate, it will default to 1,000ms. If you do not want this function to poll, use a poll rate of zero.

Syntax

now([pollRate])

Examples

```
now() //returns the current time, updates every second.
```

```
dateFormat(now(), "MMM d, h:mm a") //returns a string representing the current time, formatted like "Feb 12, 9:54 AM"
```

timeBetween

Description

Checks to see if the given time is between the start and end times. The given times are expected as strings, and may include dates. Note: dates will be parsed according to the default system culture.

Syntax

timeBetween(date,date,date)

Examples

```
timeBetween(toDate("2003-9-14 12:00:00"), toDate("2003-9-14 8:00:00"),toDate("2003-9-14 18:00:00")) //returns true
```

```
timeBetween("2:00:00 pm", "9:00:00 am", "5:00:00 pm") //returns true
```

```
timeBetween(toDate("2003-9-14 20:00:00"), toDate("2003-9-14 18:00:00"), toDate("2003-9-15 2:00:00")) //returns true
```

Logic

binEnc

Description

This function, whose name stands for "binary encoder", takes a list of booleans, and treats them like the bits in a binary number. It returns an integer representing the decimal value of the number. The digits go from least significant to most significant.This can be a very handy tool to convert bits into an integer code to drive the Component Styles feature.

Syntax

binEnc(boolean1, boolean2, ...)

Examples

```
binEnc(0,0,1,0) //returns 4 (the value of 0100)
```

```
binEnc(true,0,1,1,0) //returns 13 (the value of 01101)
```

binEnum

Description

This function, whose name stands for "binary enumeration", takes a list of booleans, and returns the index (starting at 1) of the first parameter that evaluates to true. This can be a very handy tool to convert bits into an integer code to drive the Component Styles feature.

Syntax

binEnum(boolean1, boolean2, ...)

Examples

```
binEnum(0, 1, 0) //returns 2
```

```
binEnum(0, false, 15, 0, 23) //returns 3 (the index of the 15 - any non-zero number is "true")
```

case

The following feature is new in Ignition version **7.8.1**
Click here to check out the other new features

Description

This function acts like the switch statement in C-like programming languages. It takes the value argument and compares it to each of the case1 through caseN expressions. If value is equal to caseX, then case returns valueX. If value is not equal to any of the case1..N, then returnDefault is returned.

Note that case() is similar in functionality to the switch() expression function. The difference between the two is the order in which the parameters are passed.

Syntax

case(value, case1, return1, case2, return2, ...caseN, returnN, returnDefault)

Examples

```
//The following would return 46 because the value (15) matched case 3, so the third return (46) was returned.
case(
15,        // value
1,        // case 1
44, // return 1
24,        // case 2
45, // return 2
15,        // case 3
46, // return 3
-1) // default
```

```
 //The following would return "Running".
case(
1,                        // value
0,                        // case 1
"Off",                // return 1
1,                        // case 2
"Running",        // return 2
2,                        // case 3
"Fault",        // return 3
forceQuality("!BAD STATE!",0)) // default
```

coalesce

Description

This function, which accepts any number of arguments, evaluates each in order, and returns the first non-null argument. Typically, you would call this with two arguments - the first being something dynamic, the second being a static value to use as a guard in case the dynamic value is null. The function itself detects its return type based on the type of the last argument.

Syntax

coalesce(value1, value2, ...)

Examples

```
coalesce(null, "abc") //would return "abc"
```

```
coalesce("xyz", "abc") //would return "xyz"
```

```
coalesce({Root Container.MyDataSet}["ColumnName"], 0) //would return the value in the dataset if it isn't null, but 0 if it is null.
```

getBit

Description

This function returns the bit value (an integer, 0 or 1) in the number at position position, according to its binary representation. The least significant bit in a number is position 0.

Syntax

getBit(number, position)

Examples

```
getBit(0,0) //would return 0
```

```
getBit(1,0) //would return 1
```

```
getBit(8,2)  //would return 0
```

hasChanged

Description

This function is only available in Transaction Group Expression Items.

This function returns True if the given value has changed since the last time the Expression Item was run. Setting the optional boolean argument "include quality" to true means a quality change will make this function return true.

Syntax

hasChanged(value, [include quality])

Examples

```
 hasChanged({reference to another Expression Item}) //would return true if the referenced value has changed since the last group execution
```

if

Description

This function evaluates the expression condition, and returns the value of trueReturn or falseReturn depending on the boolean value of condition.

Syntax

if(condition, trueReturn, falseReturn)

Examples

```
 if(1, "Yes", "No") //would return "Yes"
```

```
 if(0, "Yes", "No") //would return "No"
```

```
 if({Root Container.CheckBox.selected}, "Selected", "Not Selected") //would return the a description of the state of the checkbox
```

isNull

Description

Tests to see whether or not the argument value is null or not. Note that you can also check for null by simply comparing the value to the null keyword. is Null(x) is the same as x = null.

Syntax

isNull(value)

Examples

```
if(isNull({Root Container.MyProperty}), 0, 1) //returns 0 if the property is null, 1 otherwise.
```

lookup

Description

This looks for lookupValue in the lookupColumn of dataset. If it finds a match, it will return the value from the resultColumn on the same row as the match. If no match is found, noMatchValue is returned. Note: The type of the value returned will always be coerced to be the same type as the noMatch Value.

If lookupColumn is not specified, it defaults to 0. If resultColumn is not specified, it defaults to 1.

The examples are based of a table that has the following data in it:

Product	Price	Category
"Apples"	1.99	"Fruit"
"Carrots	3.5	"Vegetable"
"Walnuts"	6.25	"Nut"

Syntax

lookup(dataset, lookupValue, noMatchValue, [lookupColumn], [resultColumn])

Examples

```
lookup({Root Container.Table.data}, "Carrots", -1.0) //returns 3.50
```

```
lookup({Root Container.Table.data}, "Grapefruit", -1) //returns -1, the noMatchValue
```

```
lookup({Root Container.Table.data}, "Walnuts", "Unknown", 0, "Category") //returns "Nut"
```

```
lookup({Root Container.Table.data}, "Pecans", "Unknown", 0, 2) //returns "Unknown", the noMatchValue
```

switch

Description

This function acts like the switch statement in C-like programming languages. It takes the value argument and compares it to each of the case1 through caseN expressions. If value is equal to caseX, then switch returns valueX. If value is not equal to any of the case1..N, then returnDefault is returned.

Note that switch() is similar in functionality to the case() expression function. The difference between the two is the order in which the parameters are passed.

Syntax

switch(value, case1, ...caseN, return1, ...returnN, returnDefault)

Examples

```
//The following would return 46 because the value (15) matched case 3, so the third return (46) was returned.
switch(
15, // value
1,  // case 1
24, // case 2
15, // case 3
44, // return 1
45, // return 2
46, // return 3
-1) // default
```

```
 //The following would return "Running".
switch(
1, // value
0, 1, 2, // cases 1-3
"Off", // return 1
"Running", // return 2
"Fault", // return 3
forceQuality("!BAD STATE!",0)) // default
```

try

Description

This expression is used to swallow errors caused by other expressions. The first expression will be executed, and if it executes successfully, its value will be used. However, if there is an error evaluating it, the value of failover will be used. When the failover is used, the data quality will be set by the failover value. It is best to use a very simple return like -1 or "" (the empty string).

Syntax

try(expression, failover)

Examples

```
try(toInteger("boom"), -1) // returns -1 with a quality code of 192 (good)
```

Math

abs

Description

Returns the absolute value of number.

Syntax

abs(number)

Examples

```
abs(-4) //returns 4
```

acos

Description

Returns the arc cosine of number, which must be a number between -1 and 1. The results will be an angle expressed in radians in the range of 0.0 through pi.

Syntax

acos(number)

Examples

```
acos(.38) //returns 1.181
```

asin

Description

Returns the arc sine of number, which must be a number between -1 and 1. The results will be an angle expressed in radians in the range of -pi/2 through pi/2

Syntax

asin(number)

Examples

```
asin(.38) //returns 0.3898
```

atan

Description

Returns the arc tangent of number, which must be a number. The results will be an angle expressed in radians in the range of -pi/2 through pi/2

Syntax

atan(number)

Examples

```
atan(.38) //returns 0.3631
```

ceil

Description

Returns the smallest floating point value that is greater than or equal to the argument and is equal to a mathematical integer.

Syntax

ceil(number)

Examples

```
ceil(2.38) //returns 3.0
```

cos

Description

Returns the trigonometric cosine of number, which is interpreted as an angle expressed in radians. The results will be a floating point value.

Syntax

cos(number)

Examples

```
cos(1.89) //returns -0.31381
```

exp

Description

Returns Euler's number e raised to the power of the argument number, or e^{number}

Syntax

exp(number)

Examples

```
exp(5) //returns 148.4
```

floor

Description

Returns the largest floating point value that is less than or equal to the argument and is equal to a mathematical integer.

Syntax

floor(number)

Examples

```
floor(2.72) //returns 2.0
```

log

Description

Returns the natural logarithm (base e) of a number.

Syntax

log(number)

Examples

```
log(28) //returns 3.332
```

log10

Description

Returns the logarithm (base 10) of a number.

Syntax

log10(number)

Examples

```
log10(28) // returns 1.447
```

pow

Description

Returns a number raised to a power.

Syntax

pow(number, number)

Examples

```
pow(2,3) //returns 8
```

round

Description

Rounds a floating point number. If the decimals argument is omitted, then the number is rounded to the nearest integer value, and the result will be a long (64-bit integer). If a number of decimal places are specified, the result will be a double (64-bit floating point value), and the result will be rounded to the given number of decimal places.

Syntax

round(number, [decimals])

Examples

```
round(3.829839, 2) //returns 3.83
```

sin

Description

Returns the trigonometric sine of number, which is interpreted as an angle expressed in radians. The results will be a floating point value.

Syntax

sin(number)

Examples

```
sin(1.89) //returns 0.9495
```

sqrt

Description

Returns the square root of the argument number.

Syntax

sqrt(number)

Examples

```
sqrt(64) //returns 8.0
```

tan

Description

Returns the trigonometric tangent of number, which is interpreted as an angle expressed in radians. The results will be a floating point value.

Syntax

tan(number)

Examples

```
tan(1.89) //returns -3.026
```

todegrees

Description

Converts an angle measured in radians to an equivalent angle measured in degrees.

Syntax

todegrees(number)

Examples

```
toDegrees(3.14) //returns 179.9088
```

toradians

Description

Converts an angle measured in degrees to an equivalent angle measured in radians.

Syntax

toradians(number)

Examples

```
toRadians(180) //returns 3.141592653589793
```

String

concat

Description

Concatenates all of the strings passed in as arguments together. A null string passed as an argument will be evaluated as the word null. Rarely used, as the + operator does the same thing.

Syntax

concat(string1, string2, ...)

Examples

```
concat("The answer is: ", "42") //returns "The answer is: 42"
```

escapeSQL

Description

Returns the given string with special SQL characters escaped. This is a fairly simplistic function - it just replaces single quotes with two single quotes, and backslashes with two backslashes. See system.db.runPrepUpdate for a much safer way to sanitize user input.

Syntax

escapeSQL(string)

Examples

```
 "SELECT * FROM mytable WHERE option = '" + escapeSQL("Jim's Settings") + "'" // returns SELECT * FROM
mytable WHERE option='Jim''s Settings'
```

```
 "SELECT * FROM mytable WHERE option = 'escapeSQL({Root Container.TextField.text}) + "'" //returns a query
with sanitized user input from a text field.
```

escapeXML

Description

Returns the given string after being escaped to be valid for inclusion in XML. This means replacing XML special characters with their XML entity equivalents.

Syntax

escapeXML(string)

Examples

```
 escapeXML("Use Navigate > PB to get to the Pork&Beans section.") //returns "Use Navigate &gt; PB to get to the Pork&amp;Beans section."
```

fromBinary

Description

Returns an integer value of the binary formatted string argument. Numbers outside of the range (-231) - (231-1), and strings that are not binary numbers, return null.

Syntax

fromBinary(string)

Examples

```
fromBinary("1111") //returns 15
```

```
fromBinary("-1111") //returns -15
```

fromHex

Description

Returns an integer value of the hex formatted string argument. Numbers outside of the range (-231) - (231-1), and strings that are not hex numbers, return null.

Syntax

fromHex(string)

Examples

```
fromHex("ff") //return 255
```

```
fromHex("0xff") //returns 255
```

```
fromHex("-ff") //returns -255
```

fromOctal

Description

Returns an integer value of the octal formatted string argument. Numbers outside of the range (-231) - (231-1), and strings that are not octal numbers, return null.

Syntax

fromOctal(string)

Examples

```
fromOctal("77") //returns 63
```

```
fromOctal("-77") //returns -63
```

indexOf

Description

Searches for the first occurrence of the substring inside of string. Returns the index of where substring was found, or -1 if it wasn't found. The first position in the string is position 0.

Syntax

indexOf(string, substring)

Examples

```
indexOf("Hamburger", "urge") //returns 4
```

```
indexOf("Test", "") //returns 0
```

```
indexOf("Dysfunctional", "fun") //returns 3
```

```
indexOf("Dysfunctional", "marble") //returns -1
```

```
indexOf("banana", "n") //returns 2
```

lastIndexOf

Description

Searches for the last occurrence of the substring inside of string. Returns the index of where substring was found, or -1 if it wasn't found. The first position in the string is position 0.

Syntax

lastIndexOf(string, substring)

Examples

```
lastIndexOf("Hamburger", "urge") //returns 4
```

```
lastIndexOf("Test", "") //returns 4
```

```
lastIndexOf("Dysfunctional", "fun") //returns 3
```

```
lastIndexOf("Dysfunctional", "marble") //returns -1
```

```
lastIndexOf("banana", "n") //returns 4
```

left

Description

Returns count characters from the left side of string, where count and string are the arguments to the function.

Syntax

left(string, charCount)

Examples

```
left("hello", 2) //returns "he"
```

```
left("hello", 0) //returns ""
```

```
left("hello", 5) //returns "hello"
```

len

Description

Returns the length of the argument, which may be a string or a dataset. If the argument is a string, it returns the number of characters in the string. If the argument is a dataset, it returns the number of rows in the dataset. Will return zero if the argument is null.

Syntax

len(value)

Examples

```
len("Hello World") //returns 11
```

```
len({Root Container.Table.data}) //returns the number of rows in the table.
```

lower

Description

Takes a string and returns a lower-case version of it.

Syntax

lower(string)

Examples

```
lower("Hello World") // returns "hello world"
```

numberFormat

Description

Returns a string version of the number argument, formatted as specified by the pattern string. This is commonly used to specify the number of decimal places to display, but can be used for more advanced formatting as well. The pattern string is a numeric format string, which may include any of these characters that instruct it how to format the number.

Symbol	Description
0	Specifies a required digit
#	Specifies an optional digit
,	The grouping separator
.	The decimal separator
-	A minus sign
E	Scientific notation
;	Used to separate positive and negative patterns
%	Multiplies the value by 100 and shows as a percent
'	Used to quote special characters

This table shows some numbers, and the result of using various format strings to format them.

Number	Pattern	Result
5	0	5
5	0.0	5.0
5	00.0	05.0
123	#,##0	123
1024	#,##0	1,024
1337	#,##0.#	1,337
1337.57	#,##0.#	1,337.6
87.32	#,##0.0000	87.3200
-1234	#,##0	-1,234
-1234	#,##0;(#,##0)	(1,234)
4096	0.###E0	4.096E3
.348	#.00%	34.80%
34.8	#0.00'%'	34.80%

Syntax

numberFormat(number, pattern)

Examples

```
numberFormat(34.8, "#0.00'%'") //returns the string "34.80%"
```

repeat

Description

Repeats the given string some number of times.

Syntax

repeat(string, count)

Examples

```
repeat("hello", 2) //returns "hellohello"
```

```
repeat("hello", 0) //returns ""
```

replace

Description

Finds all occurrences of a substring inside of a source string, and replaces them with the replacement string. The first argument is the source, the second is the search string, and the third is the replacement.

Syntax

replace(string, string, string)

Examples

```
 replace("XYZ", "Y", "and") //returns "XandZ"
```

```
 replace("bob and mary went to bob's house", "bob", "judith") //returns "judith and mary went to judith's house"
```

right

Description

Returns **count** number of characters starting from the right side of **string**, where count and string are the arguments to the function.

Syntax

right(string, charCount)

Examples

```
right("hello", 2) //returns "lo"
```

```
right("filename.pdf", 3) //returns "pdf"
```

```
right("hello", 0) //returns ""
```

split

Description

This function takes the string string and splits it into a bunch of substrings. The substrings are return as a dataset with one column called "parts". The split occurs wherever the regular expression regex occurs.

The optional limit argument, if greater than zero, limits the number of times the regex pattern is applied to limit-1. Put another way, it limits the length of the resulting dataset to length limit. If limit is non-positive then the regex pattern will be applied as many times as possible and the returned dataset can have any length. If limit is zero (the default) then the pattern will be applied as many times as possible, the returned dataset can have any length, and trailing empty strings will be discarded.

Syntax

split(string, regex, [limit])

Examples

```
split("hello,world", ",") //returns dataset ["hello","parts"]
```

```
split("boo:and:foo", ":") //returns dataset ["boo", "and", "foo"]
```

```
split("boo:and:foo", ":", 2) //returns dataset ["boo", "and:foo"]
```

stringFormat

Description

Returns a formatted string using the specified format string and arguments.

Syntax

stringFormat(format, args...)

Examples

```
stringFormat("Hello %s", "world") //returns "Hello world"
```

```
stringFormat("%s, %s, %s", 1, 2, 3) //returns "1, 2, 3"
```

```
stringFormat("%d, %d, %d", 4, 5, 6) //returns "4, 5, 6"
```

substring

Description

Substring will return the portion of the string from the startIndex to the endIndex, or end of the string if endIndex is not specified. All indexes start at 0, so in the string "Test", "s" is at index 2. The start index is inclusive, while the end index, if used, is exclusive. Indexes outside of the range of the string throw a StringIndexOutOfBoundsException. Null strings return null substrings.

Syntax

substring(string, startIndex, [endIndex])

Examples

```
substring("unhappy", 2) //returns "happy"
```

```
 substring("hamburger", 4, 8) //returns "urge"
```

toBinary

Description

Returns an binary formatted string representing the unsigned integer argument. If the argument is negative, the binary string represents the value plus 2^{32}.

Syntax

toBinary(int)

Examples

```
toBinary(255) //returns "11111111"
```

```
toBinary(-255) //returns "11111111111111111111111100000001"
```

toHex

Description

Returns a hex formatted string representing the unsigned integer argument. If the argument is negative, the hex string represents the value plus 2^{32}.

Syntax

toHex(int)

Examples

```
toHex(255) //returns "FF"
```

```
toHex(-255)  //returns "FFFFFF01"
```

toOctal

Description

Returns an octal formatted string representing the unsigned integer argument. If the argument is negative, the octal string represents the value plus 2^{32}.

Syntax

toOctal(int)

Examples

```
toOctal(255) //returns "377"
```

```
toOctal(-255) //returns "37777777401"
```

trim

Description

Takes the argument string and trims of any leading and/or trailing whitespace, returning the result.

Syntax

trim(string)

Examples

```
trim("Hello Dave  ") //returns "Hello Dave"
```

```
trim(" Goodbye.") //returns "Goodbye."
```

upper

Description

Takes a string and returns an upper-case version of it.

Syntax

upper(string)

Examples

```
upper("Hello World") //returns "HELLO WORLD"
```

Translation

translate

Description

Returns a translated string, based on the current locale. If the string does not exist in the global translations, the original string will be returned. This function only exists in the client scope.

Syntax

translate(stringKey)

Examples

This expression function does not have any examples associated with it.

Type Casting

toBoolean

Description

Tries to convert value to a boolean, according to these rules:

1. If value is a number, 0 is false and anything else is true.
2. If value is a string, then the strings (case insensitive) "on", "true", "t", "yes", "y" are all true. The strings (case insensitive) "off", "false", "f", "no", "n" are considered false. If the string represents a number, the first rule applies. All other strings fail type casting.
3. All other types fail type casting.

If type casting fails, an error is thrown, unless the failover argument is specified, in which case it will be used.

Syntax

toBoolean(value, [failover])

Examples

```
toBoolean(1) //returns true
```

```
toBoolean("abc", false) //returns false
```

toBorder

Description

This function is used specifically when binding a **Border** property on a component. Typically, this is used with a Container or Label component but can be used on any component that has a Border property.

This function takes a string and converts it into a border. The string must be a semi-colon separated list of values. The first value is the name of the border, and the other values depend on the type of border you use. The following table defines the border types and the arguments they accept.

Border Type	Options	Type	Style	Font Justification
bevel	bevelType	0 = Raised 1 = Lowered 1010 = Double		
button	none			
etched	etchType	0 = Raised 1 = Lowered		
etchedtitled	title; style; fontJustification; fontPosition; fontColor; font		0 = Etched / Lowered 1 = Etched / Raised 2 = Beveled / Lowered 3 = Beveled / Raised 4 = Beveled / Double 5 = Standard	1 = Left 2 = Center 3 = Right 4 = Leading 5 = Trailing
field	none			
line	color; thickness			
linetitled	title; width; lineColor; fontJustification; fontPosition; fontColor; font			1 = Left 2 = Center 3 = Right 4 = Leading 5 = Trailing
matte	color; topWidth, leftWidth; bottomWidth; rightWidth			
paneltitled	title; style; mainColor; bgColor, shadowSize, fontJustification; fontPosition; fontColor;font		1=Gradient / West-to-East 2=Gradient / North-to-South 3=Gradient / East-to-West 4=Solid	1 = Left 2 = Center 3 = Right 4 = Leading 5 = Trailing

To use this function, you need to include the border type and then any options you want to use in the correct order. ie:

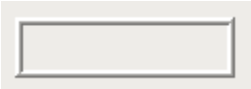
```
toBorder("paneltitled; title; style; mainColor; bgColor; shadowSize; fontJustification; fontPosition; fontColor;font")
```

Syntax

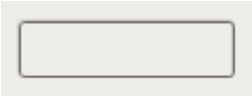
toBorder(value, [failover])

Examples

```
toBorder("bevel;1010")  //returns this...
```

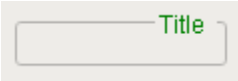


```
toBorder("button")
```



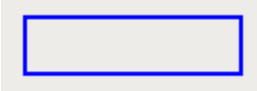
```
toBorder("etched;0")
```

```
toBorder("etchedtitled;Title;5;3;right;green;Arial")
```




```
toBorder("field")
```

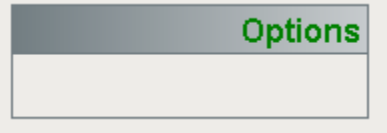
```
toBorder("line;blue;2")
```



```
toBorder("linetitled;Title") //returns this...
```


```
toBorder("matte;red;10;1;1;1") //returns this...
```

```
toBorder("paneltitled;Options;1;grey;white;0;3;0;green;Dialog,bold,16") //returns this...
```



toColor

Description

This function tries to convert value to a color. It assumes that value is a string. If you have integers representing Red, Green, and Blue values see the color expression. The string value is converted to a color according to these rules:

1. If value is a name of a color as defined in the table below, the corresponding color will be returned. Note that color names are case insensitive.
2. If value is a hex color string (with or without a leading "#", the color equivalent of that hex string will be used. Examples: "#FF0000", "556B2F"
3. If value is a list of 3 or 4 integers, a color will be created that uses the first three integers as red, green, and blue values, and the optional fourth integer as an alpha channel value. All values should be between 0 and 255. The list is free-form, any non-digit characters may be used as delimiters between the digits. Examples: "(0,0,0)", "23-99-203", "[255,255,33,127]"

Syntax

toColor(value, [failover])

Examples

```
//All of these expressions return the color red.
toColor("red")
toColor("#FF0000")
toColor("255,0,0")
```

```
//You can use the failover parameter to ensure that this expression returns something even if the input
string fails:
toColor({UserOptions/CustomColor}, "black")
```

Color Options

AliceBlue	#F0F8FF
AntiqueWhite	#FAEBD7
Aqua	#00FFFF
Aquamarine	#7FFFD4
Azure	#F0FFFF
Beige	#F5F5DC
Bisque	#FFE4C4
Black	#000000
BlanchedAlmond	#FFEBCD
Blue	#0000FF
BlueViolet	#8A2BE2
Brown	#A52A2A
BurlyWood	#DEB887
CadetBlue	#5F9EA0
Chartreuse	#7FFF00
Chocolate	#D2691E
Clear	(transparent)

Coral	#FF7F50
CornflowerBlue	#6495ED
Cornsilk	#FFF8DC
Crimson	#DC143C
Cyan	#00FFFF
DarkBlue	#00008B
DarkCyan	#008B8B
DarkGoldenRod	#B8860B
DarkGray	#A9A9A9
DarkGreen	#006400
DarkKhaki	#BDB76B
DarkMagenta	#8B008B
DarkOliveGreen	#556B2F
Darkorange	#FF8C00
DarkOrchid	#9932CC
DarkRed	#8B0000
DarkSalmon	#E9967A
DarkSeaGreen	#8FBC8F
DarkSlateBlue	#483D8B
DarkSlateGray	#2F4F4F
DarkTurquoise	#00CED1
DarkViolet	#9400D3
DeepPink	#FF1493
DeepSkyBlue	#00BFFF
DimGray	#696969
DodgerBlue	#1E90FF
Feldspar	#D19275
FireBrick	#B22222
FloralWhite	#FFFAF0
ForestGreen	#228B22
Fuchsia	#FF00FF
Gainsboro	#DCDCDC
GhostWhite	#F8F8FF
Gold	#FFD700
GoldenRod	#DAA520
Gray	#808080
Green	#008000
GreenYellow	#ADFF2F
HoneyDew	#F0FFF0
HotPink	#FF69B4
IndianRed	#CD5C5C

Indigo	#4B0082
Ivory	#FFFFF0
Khaki	#F0E68C
Lavender	#E6E6FA
LavenderBlush	#FFF0F5
LawnGreen	#7CFC00
LemonChiffon	#FFFACD
LightBlue	#ADD8E6
LightCoral	#F08080
LightCyan	#E0FFFF
LightGoldenRodYellow	#FAFAD2
LightGreen	#90EE90
LightGrey	#D3D3D3
LightPink	#FFB6C1
LightSalmon	#FFA07A
LightSeaGreen	#20B2AA
LightSkyBlue	#87CEFA
LightSlateBlue	#8470FF
LightSlateGray	#778899
LightSteelBlue	#B0C4DE
LightYellow	#FFFFE0
Lime	#00FF00
LimeGreen	#32CD32
Linen	#FAF0E6
Magenta	#FF00FF
Maroon	#800000
MediumAquaMarine	#66CDAA
MediumBlue	#0000CD
MediumOrchid	#BA55D3
MediumPurple	#9370D8
MediumSeaGreen	#3CB371
MediumSlateBlue	#7B68EE
MediumSpringGreen	#00FA9A
MediumTurquoise	#48D1CC
MediumVioletRed	#C71585
MidnightBlue	#191970
MintCream	#F5FFFA
MistyRose	#FFE4E1
Moccasin	#FFE4B5
NavajoWhite	#FFDEAD
Navy	#000080

OldLace	#FDF5E6
Olive	#808000
OliveDrab	#6B8E23
Orange	#FFA500
OrangeRed	#FF4500
Orchid	#DA70D6
PaleGoldenRod	#EEE8AA
PaleGreen	#98FB98
PaleTurquoise	#AFEEEE
PaleVioletRed	#D87093
PapayaWhip	#FFEFD5
PeachPuff	#FFDAB9
Peru	#CD853F
Pink	#FFC0CB
Plum	#DDA0DD
PowderBlue	#B0E0E6
Purple	#800080
Red	#FF0000
RosyBrown	#BC8F8F
RoyalBlue	#4169E1
SaddleBrown	#8B4513
Salmon	#FA8072
SandyBrown	#F4A460
SeaGreen	#2E8B57
SeaShell	#FFF5EE
Sienna	#A0522D
Silver	#C0C0C0
SkyBlue	#87CEEB
SlateBlue	#6A5ACD
SlateGray	#708090
Snow	#FFFAFA
SpringGreen	#00FF7F
SteelBlue	#4682B4
Tan	#D2B48C
Teal	#008080
Thistle	#D8BFD8
Tomato	#FF6347
Transparent	#FFFFFF
Turquoise	#40E0D0
Violet	#EE82EE
VioletRed	#D02090

Wheat	#F5DEB3
White	#FFFFFF
WhiteSmoke	#F5F5F5
Yellow	#FFFF00
YellowGreen	#9ACD32

toDataSet

Description

Tries to coerce value into a dataset. Not many things can be coerced into datasets. Namely, only DataSets and PyDataSets can be coerced into DataSets. This is useful for the runScript() expression, to convince the expression compiler to let you assign the return value of a scripting function to a DataSet property.

Syntax

toDataSet(value, [failover])

Examples

```
 toDataSet(runScript("app.funcs.runSomeFunction()")) //coerces the value returned by the a project scripting function into a dataset.
```

toDate

Description

Tries to coerce value into a Date. If value is a number or a string that represents a number, the number is treated as the number of milliseconds since the epoch, January 1, 1970, 00:00:00 GMT. If value is a string, it is parsed to see if it represents a date in one of these two formats: "yyyyMMdd.HHmmssSSSZ" or "yyyy-MM-dd HH:mm:ss". If not, type casting fails.The failover value must be a number or string with the same restrictions.

Syntax

toDate(value, [failover])

Examples

```
toDate("2007-04-12 16:28:22") //returns April 12th, 2007, 4:28:22 PM
```

toDouble

Description

Tries to coerce value into a double (64-bit floating point value). If value is a number, the conversion is direct. If value is a string, it is parsed to see if it represents a double. If not, type casting fails.

Syntax

toDouble(value, [failover])

Examples

```
 toDouble("38.772") //returns 38.772
```

```
 toDouble({Root Container.TextField.text}, 0.0) //returns the value in the text box as a double, or 0.0 if
the value doesn't represent an number.
```

toFloat

Description

Tries to coerce value into a float (32-bit floating point vaule). If value is a number, the conversion is direct. If value is a string, it is parsed to see if it represents a float. If not, type casting fails.

Syntax

toFloat(value, [failover])

Examples

```
 toFloat("38.772") //returns 38.772
```

```
 toFloat({Root Container.TextField.text}, 0.0) //returns the value in the text box as a float, or 0.0 if the value doesn't represent an number.
```

toFont

Description

Coerces a string into a font. The string must be in the format:

font(fontName, fontType, fontSize)

fontName is the name of the font to use. Note that special care must be taken with fonts, because of the web-launched nature of the clients. You can only use font names that exist on the client machines. The following font names are known as logical fonts, meaning that they are guaranteed to exist on all systems, mapped to the most appropriate real, or physical font that exists on the host system:

- Serif
- SansSerif
- Monospaced
- Dialog
- DialogInput

fontType is a string, that should match one of these (case-insensitive):

- Plain
- Bold
- Italic
- BoldItalic

fontSize is an integer that represent the font's point size.

Syntax

toFont(value, [failover])

Examples

```
toFont("font(Dialog,Bold,12)") //returns the standard font used in most clients.
```

toInt

Description

Tries to coerce value into an integer (32-bit integer). If value is a number, the conversion is direct (with possible loss of precision). If value is a string, it is parsed to see if it represents an integer. If not, type casting fails. Will round if appropriate.

Syntax

toInt(value, [failover])

Examples

```
toInt("38") //returns 38
```

```
toInt("33.9") // returns 34
```

```
toInt({Root Container.TextField.text}, -1) //returns the value in the text box as an int, or -1 if the value doesn't represent an number.
```

toInteger

Description
Identical to the toInt expression function.

Syntax
toInteger(value, [failover])

Examples
This expression function does not have examples associated with it.

toLong

Description

Tries to coerce value into a long (64-bit integer). If value is a number, the conversion is direct. If value is a string, it is parsed to see if it represents a long. If not, type casting fails. Will round if appropriate.

Syntax

toLong(value, [failover])

Examples

```
toLong("38") //returns 38
```

```
toLong("33.9") //returns 34
```

```
toLong({Root Container.TextField.text}, -1) //returns the value in the text box as an long, or -1 if the value doesn't represent an number.
```

toStr

Description
Identical to the toString expression function.

Syntax
toStr(value, [failover])

Examples
There are no examples associated with this expression function.

toString

Description

Represents the value as a string. Will succeed for any type of value.

Syntax

toString(value, [failover])

Examples

```
toString(1/3.0) // returns "0.3333333333333333"
```

```
toString({Root Container.Table.data}) //returns something like: "Dataset [150R x 3C]"
```

Users

hasRole

Description

Returns true if the user has the given role. The username and usersource parameters are optional in the client scope, but required in the gateway scope.

Syntax

hasRole(role, [username], [usersource])

Examples

```
// This is an example using the default user and userSource:
hasRole("Administrator", "admin", "default")
```

```
// This is an example using the current user and default userSource in the client scope:
hasRole("Administrator")
```

Advanced

columnRearrange

Description

Returns a view of the given dataset with the given columns in the order specified. Columns may be omitted in order to filter out columns from the original dataset. Since 7.8.1.

Syntax

columnRearrange(dataset, col, col, col...)

Examples

```
columnRearrange(fiveColDataset, "secondCol", "thirdCol", "firstCol") // returns a 3 column Dataset, where the columns are in the given order
```

columnRename

Description

Returns a view of the given dataset with the columns renamed. The number of new names must match exactly with the existing column count. Since 7.8.1.

Syntax

columnRename(dataset, newName, newName, newName...)

Examples

```
columnRename(twoColDataset, "colOne", "colTwo") // returns a Dataset with columns ["colOne", "colTwo"]
```

forceQuality

Description

Returns the given value, but overwrites the quality of that value. If the quality argument is omitted, the quality will be GOOD (192). This is a way to have expressions opt-out of the quality overlay system. You can also force a specific quality code here by including the quality argument.

Syntax

forceQuality(value, [qualityCode])

Examples

```
forceQuality({Tanks/Tank15})   //returns the value of the Tank15 tag, but always with a good quality code.
```

```
forceQuality({Tanks/Tank15}, 410) //returns the value of the Tank15 tag, but always with a TAG_DISABLED quality.
```

runScript

Description

Runs a single line of Python code as an expression. If a poll rate is specified, the function will be run repeatedly at the poll rate. This is a very powerful way for you to add extensions to the expression language. For example, one could write a project script module function called shared.weather.getTempAt(zip) that queried a web service for the current temperature at a given zipcode, and then bind the value of a label to the return value of that function.

The **scriptFunction** is a entered as a **string** and the **pollRate** is in **milliseconds**. You can optionally add any function arguments after the poll rate.

runScript Polling in Tags

The runScript function can be used in expression tags, but the poll rate doesn't work exactly the same as in an expression binding. All Tags have a Scan Class that dictates the minimum amount of time between each evaluation. The runScript poll rate only polls **up to** the rate of the scan class set on the tag.

For example, if an Expression Tag is configured with runScript to run at a poll rate of 60 seconds and is using the "default" (1 second) scan class, the Tag's Expression will still execute every 1 second. So a scan class rate of 60 seconds will be necessary for a runScript expression to poll at any rate between 0 and 60 seconds.

Syntax

```
runScript(scriptFunction, [pollRate], [arguments...])
```

Examples

Code Snippet

```
#You could implement shared.weather.getTempAt(zip) with this Python script:
# This function would query Yahoo Weather for the temperature at
# the given zipcode and find the temperature using a regular expression
def getTempAt(zipCode):
        import system
        import re #Regular Expression library
        try:
                yahooURL = "http://xml.weather.yahoo.com/forecastrss?p="
                response = system.net.httpGet(yahooURL + str(zipCode))
                # NOTE - if you've never seen regular expressions before, don't worry, they look
                # confusing even to people who use them frequently.
                pattern = re.compile('.*?<yweather:condition (.*?)/>', re.DOTALL)
                match = pattern.match(response)
                if match:
                        subText = match.group(1)
                        temp = re.compile('.*?temp="(.*?)"').match(subText).group(1)
                        return int(temp)
        except:
                system.gui.errorBox("Yahoo weather service changed")
                return -1
```

Expression Code Snippet

```
// run a shared function with this expression
runScript("shared.weather.getTempAt('95818')", 15000) //This would bind a property to the temperature in
sunny Sacramento, CA, and would refresh itself every 15 seconds.
```

Expression Code Snippet

```
// run a shared function dynamically with this expression
// using string concatenation
runScript("shared.weather.getTempAt("+{Root Container.Numeric Text Field.intValue}+")", 0) // binds a
property with a dynamic zip code and does not refresh automatically
```

Expression Code Snippet

```
// run a shared function dynamically with this expression
// using optional arguments
// Note the missing "()" at the end of the scriptFunction string
runScript("shared.weather.getTempAt", 0, {Root Container.Numeric Text Field.intValue})
```

sortDataset

Description

Returns a new dataset based on the rows in the given dataset. Sort order is natural if the Class of keyColumn implements java.lang.Comparable, otherwise sorting is done by the toString() value.

Syntax

sortDataset(dataset, keyColumn, [ascending])

Examples

```
sortDataset(dataset, 0, true) // returns a Dataset sorted ascending on column 0.
```

```
sortDataset(dataset, "Column 1", false) // returns a Dataset sorted descending on the column named "Column 1".
```

tag

Description

Returns the value of the tag at the path specified. Normally, you'd use the expression language's built-in bound-value syntax to use a tag value in an expression. What makes this function useful is that the path itself can be the result of an expression, meaning it can be dynamic.

Syntax

tag(tagPath)

Examples

```
tag("Tanks/Tank5") //returns Tank5's value.
```

```
tag("Tanks/Tank" + {Root Container.TankNum}) //returns the value for the tank represented by the dynamic property TankNum on the Root Container.
```

Scripting Functions

The Ignition scripting API, which is available under the module name "system", is full of functions that are useful when designing projects in Ignition. From running database queries, manipulating components, to exporting data, scripting functions can help. Some of these functions only work in the Gateway scope, and other only work in the Client scope, while the rest will work in any scope.

FactoryPMI

"I'm upgrading from FactoryPMI - will my calls to fpmi. still work?"*

Yes. Ignition's scripting API is backwards compatible. You'll probably want to gradually move your "fpmi" references to "system" but you don't need to.

For an overview and syntax of the scripting functions, see Scripting Overview and Syntax.

system.db

system.alarm

system.opc

system.print

system.serial

system.user

system.sfc

system.twilio

system.gui

system.util

system.nav

system.report

system.dnp3

system.tag

system.dataset

system.file

system.net

system.security

system.device

system.date

Scripting Overview and Syntax

Overview

Scripting is used in many places in Ignition to add a significant degree of flexibility and customization where pre-canned options fall short. There are two major scripting languages in Ignition, Python and the Expression Language. It is important to understand the differences between the two, and to know where each is used.

On this page...

Basic Syntax

Hello World

Lets get right to everyone's favorite example: the following script will print out `"Hello World"` to the out put console.

```
print "Hello World"
```

The `print` keyword is a handy tool in Python, allowing you to put text into the output console. This is useful for debugging your scripts. You can print multiple things by separating them with commas.

Variables

Variables are created by simply assigning a value to them. Variables do not need to be declared, because Python has a dynamic type system. That means Python figures out the type of the variable on the fly, when the script is executed.

The following script would print out: 15

```
x=5
y=3
print x*y
```

Quotations

Python makes limited distinction between single and double quotes. As long as they are used consistently then there are few times when which type of quotation mark you use matters. Some of the rules are shown here:

```
print "This is my text"           #Using double quotation marks
print 'This is my text'                #Using single quotation marks
print "This is my text'                #This will not work because python
does not allow mixing the single and double quotation marks
print "My name is 'David'"        #This will print My name is 'David'
print 'My name is "David"'        #This will print My name is "David"
```

Strings, Numbers, and Booleans

Literal strings can be typed in using either double quotes or single quotes. This can be handy when your string contains one quote or the other. You can also use the backslash character to escape special characters.

Strings that contain characters beyond 7-bit ASCII, such as é or need to be marked as unicode strings by placing the letter u in front of the string. There is no harm in marking all strings as unicode strings. The following prints a unicode string:

```
print u'été'
```

Numbers can just be typed in normally, like `42` or `3.14159`. Python does not have a boolean type. 0 is false and 1 is true. For convenience, Python also has constants named False and True which can be used in place of 0 and 1. This is an oversimplification, but should suffice for now. The following prints out `"True"`.

```
x="isn't this grand"
y='isn\'t this grand'
print x==y
```

None

There is a special value in Python called `None` (with a capital N). This is simply a special value that means: no value. This value is equivalent to Java's `null` value.

List

In Python, lists (arrays) are a built-in type that contains multiple other values. Lists can contain any type of items, and the items in a list do not all need to be the same type. You can create a list by enclosing multiple items in square brackets (`[]`), separated with commas. You can pull items out of a list with the square-bracket list index notation. Note that lists are zero-indexed, meaning that the first item in the list is at position 0. This code will print out `"a list"`.

```
a = ['this', 'is', 'a list', 8, 93.928]
print a[2]
```

Basic Operators

Python has all of the normal arithmetic operators you'd expect, addition(+), subtraction(-), division(/), multiplication(*), modulus(%), etc.

The comparison operators are just like in C: equals(==), not equals(!=) greater than (>), greater than or equal(>=), etc.

The logical operators are just typed in plain text: and, or, not.

Comments

Comments start with a hash sign. Add comments to your code so that when you go back to it after a long time, you know what the code is trying to do.

```
# Prints out 'Hello World' 5 times.
for x in range(5):
   print 'Hello world'
```

Whitespace

Perhaps its most unique feature, logical blocks are defined by indentation in Python. A colon (:) starts a new block, and the next line must be indented (typically using a tab of 4 spaces). The block ends when the indentation level returns to the previous level. For example, the following will print out `"5 4 3 2 1 Blast-off"`. The final print isnot part of the loop, because it isn't indented.

```
countdown=5
while countdown > 0:
   print countdown,
   countdown = countdown - 1
print "Blast-off!"
```

Control Flow Statements

Control flow statements, that is the ifs and loops, make the language do things differently based on the various conditions. Python has all of the basic control flow statements that you'd expect from a programming language.

`if` Statement

The `if` statement is 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 if a certain condition is true. For example, this script won't do anything.

```
x = 15
if x < 10:
   print "this will never show"
```

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. This script will print out `"this will show!"`

```
 x = 15
if x < 10:
   print "this will never show"
else:
   print "this will show!"
```

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"`:

```
x = 3
if x == 1:
   print "one"
elif x == 2:
   print "two"
elif x == 3:
   print "three"
else:
   print "not 1-3"
```

`while` Loops

A `while` loop will repeat a block of statements while a condition is true. This code will print out the contents of the items in the list. This code uses a fu nction called `len`, which is a built-in function that returns the length of a list or string.

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

`for` Loops

Python's `for` loop may be a bit different than what you're used to if you've programmed any C. The `for` loop is specialized to iterate over the elements of any sequence, like a list. So, we could re-write the example above using a `for` loop eliminating the counter x:

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

Much more graceful! You'll often see the `for` loop used instead of the `while` loop, even when you simply want to iterate a given number of times. To do this with the `for` loop, you can use the built-in function `range`. The `range` function returns a variable-size list of integers starting at zero. Calling ra nge(4) will return the list [0, 1, 2, 3]. So, to have a `for` loop repeat 4 times, you can simply do:

```
for x in range(4):
   print "this will print 4 times"
```

`break` and `continue` in Loops

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 `"Fin ished"`.

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

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

```
for x in range(10):
   if x == 4:
      continue
   print x
```

String Formatting

Although string formatting is a minor feature of Python, but it is incredibly useful in Ignition. String formatting is used to manipulate strings, specifically to insert the values of variables inside a string without a bunch of concatenation.

The % operator is used in Python not just for modulus, but also for string formatting. Suppose we wanted to print a weather report. We could use concatenation, like this:

```
temp = 65.8
city = "Sacramento"
windSpeed = 25
windDir = "east"

print city
print "Weather: " + str(temp) + "°F, wind "+ str(windSpeed) + "mph from the "+ windDir
```

Yuck! This kind of concatenation is really a pain to write and to read. With string formatting, we could have written it like this:

```
temp = 65.8
city = "Sacramento"
windSpeed = 25
windDir = "east"

print "%s weather: %f°F, wind %dmph from the %s" % (city, temp, windSpeed, windDir)
```

Ah, that's much easier on the eyes. What is happening here is that the % operator is applying the variables on its right side to the format string on its left side. It looks for placeholders (called format specifiers) inside the format string, and replaces them with corresponding values from the variables on the right side. There are various format specifiers that can be used for different types of variable types. If you actually want a % sign inside the final string, use the special format specifier: "%%"

Format Specifier	Meaning
%%	Inserts a % sign into the final string.
%c	A single character. Value must be a string of length 1 or an integer.
%d or %i	Signed integer
%f	Floating point, decimal format
%s	A String, converts the value to a string using `str()`.
%u	Unsigned decimal
%x or %X	Unsigned hexadecimal

You can also put some extra information in the format specifiers between the % and the format specifier character. The most useful thing to do is to specify the number of decimal places to use to print floating point numbers. For example, `"%.3f"` would always put three digits after the decimal point.

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.

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 tests whether or not a number is odd. It returns a true value (1) if the number is odd. It is then used in a loop to print out the odd numbers between 0 and 9.

```
def isOdd(num):
   return num % 2 == 1 # uses the modulus (or remainder) operator

for x in range(10):
   if isOdd(x):
      print x
```

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. In the example above, the value of `x` inside the `for` loop gets passed to the `isOdd` function, and becomes the value of the `num` argument.

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 `calledcap`, which will take a number, and make sure it is within an upper and lower limit. The limits default to 0 and 100.

```
def cap(x, min=0, max=100):
   if x < min:
      return min
   elif x > max:
      return max
   else:
      return x

# This will print out "0"
print cap(-1)

# This will print out "100"
print cap(150)

# this will print out "150", because it uses a max of 200
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 below, 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`.

```
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.

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++).

Lets go back to the `isOdd` example above. Suppose we wanted a more general way to filter a list. Maybe sometimes we want the odd entries, while other times we want even ones, or entries less than 3, etc. We can define a function called `extract` that takes a list and another function, and returns only entries that "pass" through the other function.

```
def isOdd(num):
   return num % 2 == 1

def isEven(num):
   return num % 2 == 0

def isLessThan(num, max=3):
   return num < max

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))
```

Now, it just so happens that Python has a built-in function that does exactly what our `extract` function does - its called `filter`.

We would also be remiss at this point if we didn't mention another language feature called `list` comprehensions. This is a great little bit of syntax that helps make new lists out of other lists. Instead of using our filter function, we could have simply done this:

```
def isEven(num):
   return num % 2 == 0

print [x for x in range(10) if isEven(x)]
```

In Ignition, you'll most commonly see functions used as objects when using the `system.util.invokeLater` function. This function takes a function and executes it after all pending event handling has finished processing.

Scope and Import

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.

Definition by Assignment

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.

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

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

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

Function Scope

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.

```
def myFunction():
   x = 15 # x is local to myFunction()
   print x # This will work, because it is part of the function

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

Import Keyword

You can import the namespaces defined in other scopes into your scope with the `import` keyword. Most commonly, you'll import from global library sources, like `system` (the Ignition standard libraries), `project` (your project's script library), `java` (importing from the Java standard library), and the various python standard libraries. When you're writing component event handlers, system, shared, and project are imported for you automatically.

The import keyword can be used in a variety of forms:

- `import X`
- `from X import Y`

For example, suppose you wanted to use the `java.util.Calendar` class for some date manipulations. You could import this in a number of different ways. These examples are equivalent, printing out a date 8 hours before the current date.

```
import java
cal = java.util.Calendar.getInstance()
cal.add(java.util.Calendar.HOUR, -8)
print cal.getTime()
```

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

Sequences and Dictionaries

Python offers a variety of sequence types. We've already seen one - the List. There are other kinds of sequences, most notably *tuples* and *strings*. There is also the *dictionary* type, which contains a list of key-value pairs.

List

Lists are a very handy kind of sequence. They can hold any number of items and can be resized on the fly. After creating a list using the square bracket notation, there are a number of functions that you can call on the list. Some common list functions are listed here.

`append(x)`
Takes a single argument, which will be appended to the end of the list.

`insert(i,x)`
Inserts an item x at index i

`remove(x)`
Will remove the given item from the list.

`index(x)`
Returns the index of the value x. Throws an error if the list doesn't contain the item. Use the in operator to check if an item is contained in a sequence.

`sort()`
Sorts the items in the list.

```
myList = ['a', 'b', 'c', 'd']
print myList # --> [a, b, c, d]

myList.append("Q")
print myList # --> [a, b, c, d, Q]

myList.insert(1, "Z")
print myList # --> [a, Z, b, c, d, Q]

myList.remove("c")
print myList # --> [a, Z, b, d, Q]

print myList[2] # --> b
print myLIst.index("b") # --> 2

if 'Z' in myList:
   print 'Z is in the list'

if 'c' not in myList:
   print 'c was removed from the list'
```

Tuples

A tuple is similar to a list, but you use parenthesis instead of square brackets to define one. The major difference between a tuple and a list is that tuple's are *immutable*. That is, once created, they cannot be altered. Tuples are very useful for passing multiple things to and from functions. For example, you could pass a point to a function using a tuple like so:

```
def printPoint(point):
   print "x = ", point[0]
   print "y = ", point[1]

printPoint((28,89))
```

This can also be handy for returning multiple values from a function. For example, if you had a mouse event, you could write a function that found the component's center point, and return that point as a tuple. You could then use unpacking assignment to extract the values into separate values.

```
def findCenter(event):
   w = event.source.width
   h = event.source.height
   return (w/2, h/2)

# point will be a tuple
point = findCenter(event)

# x and y will be numbers, using unpacking assignment
x,y = findCenter(event)
```

Dictionaries

A dictionary is a very useful type that holds a set of key-value pairs. You may have used these in other languages and know them as hashmaps, maps, associative memories or associative arrays. Dictionaries are not ordered sequences - you reference any item via its key value. The keys can be numbers, strings, or tuples of these types. Any given key may only appear once in a dictionary. Trying to set another value for that key will overwrite any previous value for that key.

Dictionaries are created using braces ({}). Key-value pairs are separated by commas, and the keys are separated from the values with a colon. You can use the .keys() function to have a set of the keys. For example:

```
myDict = {'Bob': 89.9, 'Joe': 188.72, 'Sally': 21.44}

print myDict['Bob'] # --> 89.9

myDict['Amir']=45.89 # Adds a key for 'Amir'

names = myDict.keys()
names.sort()
print names # --> ['Amir', 'Bob', 'Joe', 'Sally']
```

You can loop through dictionaries using a for loop. You can use the keys() to loop through the dictionary, and then use the key values to look up the value. For example:

```
for name in myDict.keys():
   print name, myDict[name]
```

Exception Handling

Exception handling is a language feature of many high-level languages that allows you to "catch" a runtime error and deal with it as you see fit. On the flip side, it allows you to "raise" or "throw" an error in your code, which will break out of whatever code is currently executing and jump to the nearest enclosing catch block that knows how to handle your error.

For example, dividing by zero raises a ZeroDivisionError. You can catch this error using a try...except block, like this:

```
try:
   result = 8 / 0
   print "this will never get called"
except ZeroDivisionError:
   print "oops - can't divide by zero"
```

You don't have to specify a particular type of error to catch - you can use the except keyword by itself to catch any kind of exception. You can also assign the details of the exception to a tuple of variables, which you can use in your error reporting. You can also have multiple except blocks for one try block, each that handle different kinds of exceptions. This example shows these variations:

```
def someDangerousFunction():
   raise IOError(42,"oh no")
try:
   someDangerousFunction()
except IOError, (errno, description):
   print "An I/O error occurred: "+description
except:
   print "An unexpected error occurred"
```

Accessing Java

When programming Python in Ignition, your code executes in the Jython implementation of Python. (See About Scripting - 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 withJava code, *as if it were Python code*. This means that your Python code has access to the entire Java standard library, which is saying a lot.

To use Java classes, you simple 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.

```
from java.lang import System
from java.io import File

homePath = System.getProperty("user.home")
homeDir = File(homePath)

for filename in homeDir.list():
   print filename
```

You can find the reference documentation for the Java standard class libraray (also known as, the "*JavaDocs*") at: http://docs.oracle.com/javase/6/docs/api/

Subclassing Java

You can even create Python classes that implement Java interfaces. If this is greek to you - don't worry, it isn't crucial. You'd 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 `aFilenameFilter`, 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.

```
from java.lang import System
from java.io import *

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)

homePath = System.getProperty("user.home")
homeDir = File(homePath)

# prints out all .txt files
for filename in homeDir.list(ExtensionFilter()):
   print filename

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

system.alarm

system.alarm.acknowledge

Description

Acknowledges any number of alarms, specified by their event ids. The event id is generated for an alarm when it becomes active, and is used to identify a particular event from other events for the same source. The alarms will be acknowledged by the logged in user making the call. Additionally, acknowledgement notes may be included and will be stored along with the acknowledgement.

This function uses different parameters based on the scope of the script calling it. Both versions are listed below.

Syntax - Client Scripts

system.alarm.acknowledge(alarmIds, [notes])

- Parameters

  String[] **alarmIds** - List of alarm event ids (uuids) to acknowledge.

  String **notes** - *Optional*. Notes that will be stored on the acknowledged alarm events.

- Returns

  Nothing

- Scope

  Client

Syntax - Gateway Scripts

system.alarm.acknowledge(alarmIds, [notes], username)

- Parameters

  String[] **alarmIds** - List of alarm event ids (uuids) to acknowledge.

  String **notes** - *Optional*. Notes that will be stored on the acknowledged alarm events.

  String **username** - The user that acknowledged the alarm. **NOTE** that this parameter is only used when called from a gateway scoped script. This parameter should be omitted from any client-based scripts.

- Returns

  Nothing

- Scope

  Client

Examples

Code Snippet

```
#This example shows the basic syntax for acknowledging an alarm from a client-based script
system.alarm.acknowledge(['c27c06d8-698f-4814-af89-3c22944f58c5'],'Saw this alarm, did something about
it.')
```

Code Snippet

```
#This example shows the basic syntax for acknowledging an alarm from a gateway-based script
system.alarm.acknowledge(['c27c06d8-698f-4814-af89-3c22944f58c5'],'Saw this alarm, did something about
it.', 'admin')
```

Code Snippet

```
#This code snippet could be used as a mouseReleased event handler on a Table component whose data was the
return value of thesystem.alarm.queryAlarmStatus function.
#It would present a right-click menu to acknowledge the currently selected alarms (for more than #one,
the table must be set to allow multiple selection).
#This example does not ask for an ack message, and therefore might fail if the alarms we're attempting to
acknowledge require notes.
#Also, note that the system will ignore any alarms that have already #been acknowledged.

if event.button==3:
   rows = event.source.selectedRows
   data = event.source.data
   if len(rows)>0:
      uuids = [str(data.getValueAt(r,'EventId')) for r in rows]
      def ack(event, uuids=uuids):
         import system
         system.alarm.acknowledge(uuids, None)
      menu = system.gui.createPopupMenu({'Acknowledge':ack})
      menu.show(event)
```

system.alarm.cancel

Description

Cancels any number of alarms, specified by their event ids. The event id is generated for an alarm when it becomes active, and is used to identify a particular event from other events for the same source. The alarm will still be active, but will drop out of alarm pipelines.

Syntax

system.alarm.cancel(alarmIds)

- Parameters

  String[] alarmIds - List of alarm event ids (uuids) to cancel.

- Returns

  nothing

- Scope

  All

Examples

Code Snippet

```
#This example shows the basic syntax for cancelling an alarm.

system.alarm.cancel(['c27c06d8-698f-4814-af89-3c22944f58c5'])
```

Code Snippet

```
#To cancel all currently active alarms:

ids = []
results = system.alarm.queryStatus(state=["ActiveUnacked", "ActiveAcked"])
for result in results:
 id = result.getId()
 ids.append(str(id))

system.alarm.cancel(ids)
```

system.alarm.createRoster

Description

This function creates a new roster. Users may be added to the roster through the Gateway or the Roster Management component

Syntax

system.alarm.createRoster(name, description)

- Parameters

  String name - The name for the new roster

  String description - A description for the new roster. Required, but can be blank.

- Returns

  Nothing

- Scope

  All

Code Examples

Code In Action

```
# This example creates a new roster
name = 'MyRoster'
description = 'A roster created by scripting'
system.alarm.createRoster(name, description)
```

system.alarm.getRosters

Description

This function returns a mapping of roster names to a list of usernames contained in the roster.

Syntax

system.alarm.getRosters()

- Parameters

  None

- Returns

  PyDict - A dictionary that maps roster names to a List of usernames in the roster. The List of usernames may be empty if no users have been added to the roster.

- Scope

  All

Code Examples

Code In Action

```
# this script will get all the rosters and list the users in them
rosters = system.alarm.getRosters()
for key, values in rosters.iteritems():
        # key is the roster name, values is a dict of usernames
        print 'Roster', key, 'contains these users:'
        for value in values:
                print '  ', value
```

Output

```
Roster Admins contains these users:
   admin
Roster Supervisors contains these users:
   asmith
   jdoe
```

system.alarm.getShelvedPaths

Description

Returns a list of ShelvedPath objects, which each represent a shelved alarm.

Syntax

system.alarm.getShelvedPaths()

- Parameters

  Nothing

- Returns

  List - A list of ShelvedPath objects. ShelvedPath objects can be examined with getExpiration, getHitCount, getPath, getShelveTime, getUser, and isExpired.

- Scope

  All, Client, or Gateway

Examples

Code Snippet

```
#The following code prints a list of the shelved alarms paths and prints them to the console.
paths = system.alarm.getShelvedPaths()
for p in paths:
   print "Path: %s, Shelved by: %s, expires: %s, is expired? %s" % (p.getPath(), p.getUser(), p.
getExpiration(), p.isExpired())
```

system.alarm.listPipelines

The following feature is new in Ignition version **7.8.1**
Click here to check out the other new features

Syntax

system.alarm.listPipelines()

- Parameters

  None

- Returns

  List - A list of pipeline names. The list may be empty if no pipelines exist. Unsaved name changes will not be reflected in the list.

- Scope

  All

Code Examples

Code In Action

```
# This script will print a list of all alarm pipeline names.
pipelines = system.alarm.listPipelines()
for pipeline in pipelines:
        print pipeline
```

Output

```
Emergency_Pipeline
Test
SMS_Pipeline
```

system.alarm.queryJournal

Description

Queries the specified journal for historical alarm events. The result is a list of alarm events, which can be queried for individual properties. The result object also has a getDataset() function that can be used to convert the query results into a normal dataset, with the columns: EventId, Source, DisplayPath, EventTime, EventState, Priority, IsSystemEvent

Click here for more information on alarm properties

Syntax

system.alarm.queryJournal(startDate, endDate, journalName, priority, state, path, source, displaypath, all_properties, any_properties, defined, includeData, includeSystem, isSystem)

- Parameters

  Date startDate - The start of the time range to query. Defaults to 8 hours previous to now if omitted. Time range is inclusive.

  Date endDate - The end of the time range to query. Defaults to "now" if omitted.

  String journalName - The journal name to query.

  String[] priority - A list of possible priorities to match. Priorities can be specified by name or number, with the values: Diagnostic(0), Low (1), Medium(2), High(3), Critical(4).

  String[] state - A list of the event state types to match. Valid values are "ClearUnacked", "ClearAcked", "ActiveUnacked", and "ActiveAcked".

  String[] path - A list of possible source paths to search at. The wildcard "*" may be used.

  String[] source - A list of possible source paths to search at. The wildcard "*" may be used.

  String[] displaypath - A list of display paths to search at. Display paths are separated by "/", and if a path ends in "/*", everything below that path will be searched as well.

  Object[][] all_properties - A set of property conditions, all of which must be met for the condition to pass. This parameter is a list of tuples, in the form ("propName", "condition", value). Valid condition values: "=","!=","<","<=",">",">=". Only the first two conditions may be used for string values.

  Object[][] any_properties - A set of property conditions, any of which will cause the overall the condition to pass. This parameter is a list of tuples, in the form ("propName", "condition", value). Valid condition values: "=","!=","<","<=",">",">=". Only the first two conditions may be used for string values.

  String[] defined - A list of string property names, all of which must be present on an event for it to pass.

  Boolean includeData - Whether or not event data should be included in the return. If this parameter is false, and if there are no conditions specified on associated data, the properties table will not be queried.

  Boolean includeSystem - Specifies whether system events are included in the return.

  Boolean isSystem - Specifies whether the returned event must or must not be a system event.

- Returns

  List - A list of matching AlarmEvent objects. AlarmEvent objects can be examined with getAckData, getActiveData, getClearedData, getCount, getDisplayPath, getDisplayPathOrSource, getId, getLastEventState, getName, getNotes, getOrDefault, getOrElse, getPriority, getProperties, getRawValueMap, getSource, getState, getValues, isAcked, isCleared, isExtended, isInherited, and isShelved.

> **Important**
>
> Each item in the resulting list is a separate alarm event: an alarm becoming active is one item, while the same alarm becoming acknowledged is a separate item. This differs from system.alarm.queryStatus() which groups each event into a single item.

- Scope

  All

Code Examples

Code Snippet

```
#This example shows the basic syntax for querying from the journal in a button's actionPerformed event,
with a date range selector ("Range"), storing the results back to a table called "Table":

table = event.source.parent.getComponent("Table")
range= event.source.parent.getComponent("Range")

results = system.alarm.queryJournal(journalName="Journal", startDate=range.startDate, endDate=range.
endDate)
table.data = results.getDataset()
```

Code Snippet

```
#This example extends the previous to only include non-acknowledged events of High or Critical severity,
who have associated data called "Department", set to "maintenance". It also excludes system events
(shelving notifications, etc):

table = event.source.parent.getComponent("Table")
range= event.source.parent.getComponent("Range")

results = system.alarm.queryJournal(journalName="Journal", startDate=range.startDate, endDate=range.
endDate, state=['ActiveUnacked', 'ClearUnacked'], all_properties=[("Department","=","maintenance")],
priority=["High", "Critical"], includeSystem=False)
table.data = results.getDataset()
```

system.alarm.queryStatus

Description

Queries the current state of alarms. The result is a list of alarm events, which can be queried for individual properties. The result object also has a getDataset() function that can be used to convert the query results into a normal dataset, with the columns: EventId, Source, DisplayPath, EventTime, State, Priority

Click here for more information on Alarm Properties

Syntax

system.alarm.queryStatus(priority, state, path, source, displaypath, all_properties, any_properties, defined, includeShelved)

- Parameters

  String[] priority - A list of possible priorities to match. Priorities can be specified by name or number, with the values: Diagnostic(0), Low (1), Medium(2), High(3), Critical(4).

  String[] state - A list of states to allow. Valid values: "ClearUnacked", "ClearAcked", "ActiveUnacked", "ActiveAcked".

  String[] path - A list of possible source paths to search at. The wildcard "*" may be used.

  String[] source - A list of possible source paths to search at. The wildcard "*" may be used.

  String[] displaypath - A list of display paths to search at. Display paths are separated by "/", and if a path ends in "/*", everything below that path will be searched as well.

  Object[][] all_properties - A set of property conditions, all of which must be met for the condition to pass. This parameter is a list of tuples, in the form ("propName", "condition", value). Valid condition values: "=","!=","<","<=",">",">=". Only the first two conditions may be used for string values.

  Object[][] any_properties - A set of property conditions, any of which will cause the overall the condition to pass. This parameter is a list of tuples, in the form ("propName", "condition", value). Valid condition values: "=","!=","<","<=",">",">=". Only the first two conditions may be used for string values.

  String[] defined - A list of string property names, all of which must be present on an event for it to pass.

  Boolean includeShelved - A flag indicating whether shelved events should be included in the results. Defaults to "false".

- Returns

  List - A list of matching AlarmEvent objects. AlarmEvent objects can be examined with getAckData, getActiveData, getClearedData, getCount, getDisplayPath, getDisplayPathOrSource, getId, getLastEventState, getName, getNotes, getOrDefault, getOrElse, getPriority, getProperties, getRawValueMap, getSource, getState, getValues, isAcked, isCleared, isExtended, isInherited, and isShelved.

Important

Each item in the resulting list is a combination of each alarm event for the same alarm: details for when the alarm became active, acknowledged, and cleared are combined into a single item. This differs from system.alarm.queryJournal() which splits these events into separate items.

- Scope

  All

Code Examples

Code Snippet

```
#This example queries the state of all tags named "HiAlarm", and puts the results in a table named
"Table" (this assumes it's being run from a button on the same screen):

table = event.source.parent.getComponent("Table")

results = system.alarm.queryStatus(source=["*HiAlarm*"])
table.data = results.getDataset()
```

system.alarm.shelve

Description

This function shelves the specified alarms for the specified amount of time. The paths may be either source paths, or display paths. The time can be specified in minutes (timeoutMinutes) or seconds (timeoutSeconds). If an alarm is already shelved, this will overwrite the remaining time. To unshelve alarms, this function may be used with a time of "0".

Syntax

system.alarm.shelve(path, timeoutSeconds, timeoutMinutes)

- Parameters

  String[] path - A list of possible source paths to search at. If a path ends in "/*", the results will include anything below that path.

  Integer timeoutSeconds - The amount of time to shelve the matching alarms for, specified in seconds. 0 indicates that matching alarm events should now be allowed to pass.

  Integer timeoutMinutes - The amount of time to shelve the matching alarms for, specified in minutes. 0 indicates that matching alarm events should now be allowed to pass.

- Returns

  Nothing

- Scope

  All

Code Examples

Code In Action

```
#This example assumes that data has been loaded into a table ("Table") from system.alarm.queryStatus, and
it shelves the selected alarms for 5 minutes. It also assumes that it is being executed from a button's
actionPerformed event.

table = event.source.parent.getComponent('Table')
rows = table.selectedRows
data = table.data
if len(rows)>0:
   sourcePaths = [str(data.getValueAt(r,'Source')) for r in rows]
   system.alarm.shelve(path=sourcePaths,timeoutMinutes=5)
```

system.alarm.unshelve

Description

Unselves alarms in accordance with the path parameter.

Syntax

system.alarm.unshelve(path)

- Parameters

  String[] path - A list of possible source paths to search at. If a path ends in "/*", the results will include anything below that path.

- Returns

  Nothing

- Scope

  All

Code Examples

There are not code examples for this function.

system.dataset

system.dataset.addColumn

Description

Takes a dataset and returns a new dataset with a new column added or inserted into it. Datasets are immutable, so it is important to realize that this function does not actually add a column to a dataset. You'll need to do something with the new dataset that this function creates to achieve something useful. If the columnIndex argument is omitted, the column will be appended to the end of the dataset.

Syntax

system.dataset.addColumn(dataset [, colIndex], col, colName, colType)

- Parameters

  Dataset dataset - The starting dataset. Please be aware that this dataset will not actually be modified (datasets are immutable), but rather will be the starting point for creating a new dataset.

  int colIndex - The index (starting at 0) at which to insert the new column. Will throw an IndexError if less than zero or greater than the length of the dataset. If omitted, the new column will be appended to the end. [optional]

  PySequence col - A Python sequence representing the data for the new column. Its length must equal the number of rows in the dataset.

  String colName - The name of the column.

  PyType colType - The type of the of the column. The type can be the Python equivalent of String, Long, Double, Short, Integer, Float, Boolean, null, or java.util.Date if they exist.

- Returns

  Dataset - A new dataset with the new column inserted or appended.

- Scope

  All

Code Examples

Code Snippet

```
#This example takes the dataset from Bar Chart 1, adds a column of integers called Center Area to the end
of the existing data, and displays the new dataset in Bar Chart 2.

ds1 = event.source.parent.getComponent('Bar Chart 1').data
colCount = ds1.getColumnCount()
columnName = "Center Area"
columnData = []
for i in range(ds1.getRowCount()):
 columnData.append(i* 10)

ds2 = system.dataset.addColumn(ds1, colCount, columnData, columnName, int)
event.source.parent.getComponent('Bar Chart 2').data = ds2
```

system.dataset.addRow

Description

Takes a dataset and returns a new dataset with a new row added or inserted into it. Datasets are immutable, so it is important to realize that this function does not actually add a row to a dataset. You'll need to do something with the new dataset that this function creates to achieve something useful. If the rowIndex argument is omitted, the row will be appended to the end of the dataset.

Syntax

system.dataset.addRow(dataset [, rowIndex], row)

- Parameters

  Dataset dataset - The starting dataset. Please be aware that this dataset will not actually be modified (datasets are immutable), but rather will be the starting point for creating a new dataset.

  int rowIndex - The index (starting at 0) at which to insert the new row. Will throw an IndexError if less than zero or greater than the length of the dataset. If omitted, the new row will be appended to the end. [optional]

  PySequence row - A Python sequence representing the data for the new row. Its length must equal the number of columns in the dataset.

- Returns

  Dataset - A new dataset with the new row inserted or appended.

- Scope

  All

Code Examples

Code Snippet

```
#This snippet would add a new option into a Dropdown component just like above, but at the beginning:
dropdown = event.source.parent.getComponent("Dropdown")
newRow = [5, "New Option"]
dropdown.data = system.dataset.addRow(dropdown.data, 0, newRow)
```

Code Snippet

```
#This example would add a new option to a Dropdown List by adding a row to its underlying dataset. Notice
how the last line assigns the return value of the addRow function to the dropdown's data property.
dropdown = event.source.parent.getComponent("Dropdown")
newRow = [5, "New Option"]
dropdown.data = system.dataset.addRow(dropdown.data, newRow)
```

system.dataset.dataSetToExcel

Description

Formats the contents of one or more datasets as an excel spreadsheet, returning the results as a string. Each dataset specified will be added as a worksheet in the Excel workbook. This function uses an xml-format for Excel spreadsheets, not the native Excel file format.

Syntax

system.dataset.dataSetToExcel(showHeaders, datasets)

- Parameters

  boolean showHeaders - If true (1), the spreadsheet will include a header row.

  Object[] datasets - A sequence of datasets, one for each sheet in the resulting workbook.

- Returns

  String - An Excel-compatible XML-based workbook, with one worksheet per dataset.

- Scope

  All

Code Examples

Code Snippet

```
#This snippet would run a SQL query against a database, and turn the results into a string that is XML
that Excel can open. It then writes the string to a file on the local hard drive.

results = system.db.runQuery("SELECT * FROM example1 LIMIT 100")
results = system.dataset.toDataSet(results)
spreadsheet = system.dataset.dataSetToExcel(1, [results])
filePath = "C:\\output\\results.xls"
system.file.writeFile(filePath, spreadsheet)
```

system.dataset.dataSetToHTML

Description

Formats the contents of a dataset as an HTML page, returning the results as a string. Uses the <table> element to create a data table page.

Syntax

system.dataset.dataSetToHTML(showHeaders, dataset, title)

- Parameters

  boolean showHeaders - If true(1), the HTML table will include a header row.

  Dataset dataset - The dataset to export

  String title - The title for the HTML page.

- Returns

  String - The HTML page as a string.

- Scope

  All

Code Examples

Code Snippet

```
#This snippet would run a SQL query against a database, and turn the results into a string containing
HTML. It then writes the string to a file on the local hard drive.

results = system.db.runQuery("SELECT * FROM example1 LIMIT 100")
results = system.dataset.toDataSet(results)
html = system.dataset.dataSetToHTML(1, results, "Production Report")
filePath = "C:\\output\\results.html"
system.file.writeFile(filePath, html)
```

system.dataset.deleteRow

Description

Takes a dataset and returns a new dataset with a row removed. Datasets are immutable, so it is important to realize that this function does not actually remove the row from the argument dataset. You'll need to do something with the new dataset that this function creates to achieve something useful.

Syntax

system.dataset.deleteRow(dataset, rowIndex)

- Parameters

  Dataset dataset - The starting dataset. Please be aware that this dataset will not actually be modified (datasets are immutable), but rather will be the starting point for creating a new dataset.

  int rowIndex - The index (starting at 0) of the row to delete. Will throw an IndexError if less than zero or greater than len(dataset)-1.

- Returns

  Dataset - A new dataset with the specified row removed.

- Scope

  All

Code Examples

Code Snippet

```
#This example would remove the selected row from a List component, by re-assigning the List's data
property to the new dataset returned by thedeleteRow function.

myList = event.source.parent.getComponent("List")
row = myList.selectedIndex
if row != -1: # make sure there is something selected
   myList.data = system.dataset.deleteRow(myList.data, row)
```

system.dataset.deleteRows

Description

Takes a dataset and returns a new dataset with one or more rows removed. Datasets are immutable, so it is important to realize that this function does not actually remove the rows from the argument dataset. You'll need to do something with the new dataset that this function creates to achieve something useful.

Syntax

system.dataset.deleteRows(dataset, rowIndices)

- Parameters

  Dataset dataset - The starting dataset. Please be aware that this dataset will not actually be modified (datasets are immutable), but rather will be the starting point for creating a new dataset.

  int[] rowIndices - The indices (starting at 0) of the rows to delete. Will throw an IndexError if any element is less than zero or greater than len(dataset)-1.

- Returns

  Dataset - A new dataset with the specified rows removed.

- Scope

  All

Code Examples

Code Snippet

```
#This example would remove several rows from a Table component, by re-assigning the Table's data property
to the new dataset returned by thedeleteRows function.

ds = event.source.parent.getComponent('Table').data
rows = [0,2,3,4]
ds = system.dataset.deleteRows(ds, rows)
event.source.parent.getComponent('Table').data = ds
```

system.dataset.exportCSV

Description

Exports the contents of a dataset as a CSV file, prompting the user to save the file to disk.

Syntax

system.dataset.exportCSV(filename, showHeaders, dataset)

- Parameters

  String filename - A suggested filename to save as.

  boolean showHeaders - If true (1), the CSV file will include a header row.

  Dataset dataset - The dataset to export.

- Returns

  String - The path to the saved file, or None if the action was canceled by the user.

- Scope

  Client

Code Examples

Code Snippet

```
#This snippet would prompt the user to save the data currently displayed in a Table component to a CSV
file, and would open the file (in an external program, presumably Excel) after a successful save.

table = event.source.parent.getComponent("Table")
filePath = system.dataset.exportCSV("data.csv", 1, table.data)
if filePath != None:
        system.net.openURL("file:///"+filePath.replace('\\','/'))
```

system.dataset.exportExcel

Description

Exports the contents of a dataset as an Excel spreadsheet, prompting the user to save the file to disk. Uses the same format as the dataSetToExcel function.

Syntax

system.dataset.exportExcel(filename, showHeaders, dataset)

- Parameters

  String filename - A suggested filename to save as.

  boolean showHeaders - If true (1), the spreadsheet will include a header row.

  Object[] dataset - A sequence of datasets, one for each sheet in the resulting workbook.

- Returns

  String - The path to the saved file, or None if the action was canceled by the user.

- Scope

  Client

Code Examples

Code Snippet

```
#This snippet would prompt the user to save the data currently displayed in a Table component to an Excel-compatible spreadsheet file, and would open the file after a successful save.

table = event.source.parent.getComponent("Table")
filePath = system.dataset.exportExcel("data.xls", 1, table.data)
if filePath != None:
   system.net.openURL("file://"+filePath)
```

system.dataset.exportHTML

Description

Exports the contents of a dataset to an HTML page. Prompts the user to save the file to disk.

Syntax

system.dataset.exportHTML(filename, showHeaders, dataset, title)

- Parameters

  String filename - A suggested filename to save as.

  boolean showHeaders - If true (1), the HTML table will include a header row.

  Dataset dataset - The dataset to export.

  String title - The title for the HTML page.

- Returns

  String - The path to the saved file, or None if the action was canceled by the user.

- Scope

  Client

Code Examples

Code Snippet

```
#This snippet would prompt the user to save the data currently displayed in a Table component to an HTML
file, and would open the file in the default web  browser after a successful save.

table = event.source.parent.getComponent("Table")
filePath = system.dataset.exportHTML("data.html", 1, table.data, "Production Report")
if filePath != None:
   system.net.openURL("file://"+filePath)
```

system.dataset.filterColumns

Description

Takes a dataset and returns a view of the dataset containing only the columns found within the given list of columns. Since 7.8.1.

Syntax

system.dataset.filterColumns(dataset, columns)

- Parameters

  Dataset dataset - The starting dataset.

  PySequence columns - A list of columns to keep in the returned dataset. The columns may be in integer index form (starting at 0), or the name of the columns as Strings.

- Returns

  Dataset - A new dataset containing the filtered columns.

- Scope

  All

Code Examples

Code Snippet

```
# This example takes the dataset from a five column Bar Chart and displays a subset of the data in two
separate tables.  This is performed in a button component actionPerformed script.

chartData = event.source.parent.getComponent('Bar Chart').data

northSouth = [1, 2] # North Area, South Area cols
eastWest = ["East Area", "West Area"]

filteredData = system.dataset.filterColumns(chartData, northSouth)
event.source.parent.getComponent('NorthSouthTable').data = filteredData

filteredData = system.dataset.filterColumns(chartData, eastWest)
event.source.parent.getComponent('EastWestTable').data = filteredData
```

system.dataset.fromCSV

Description

Converts a dataset stored in a CSV formatted string to a dataset that can be immediately assignable to a dataset property in your project. Usually this is used in conjunction with system.file.readFileAsString when reading in a CSV file that was exported using system.dataset.toCSV. The CSV string must be formatted in a specific way:

```
"#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"
"78","Test Row 9","34.57127071614745"
"20","Test Row 10","76.41114659745085"
"21","Test Row 13","13.880548366871926"
```

The first line must be "#NAMES"

The second line must list the names of the columns of the datset, each in quotes and separated by commas

The third line must be "#TYPES"

The fourth line must list the type of each column of the dataset in order

Integer = "I"

String = "str"

Double = "D"

Date = "date"

Long = "L"

Short = "S"

Float = "F"

Boolean = "B"

The fifth line must be "#ROWS" followed by a comma and then the number of rows of data in quotes (i.e. "#ROWS", "6")

The following lines will be your data, each column value surrounded in quotes and separated by a comma; each row on a separate line. The number of rows must match what was specified on line 5

Syntax

system.dataset.fromCSV(csv)

- Parameters

  String csv - A string holding a CSV dataset.

- Returns

  Dataset - A new dataset.

- Scope

  All

Code Examples

Code Snippet

```
#In this example it is assumed that the CSV file being read was a dataset that was previously exported
using system.dataset.toCSV:
#Specify file path
file_path = "C:\\my_dataset.csv"
#Read in the file as a string
data_string = system.file.readFileAsString(file_path)
#Convert the string to a dataset and store in a variable
data = system.dataset.fromCSV(data_string)
#Assign the dataset to a table
event.source.parent.getComponent('Table').data = data
```

system.dataset.getColumnHeaders

Description

Returns the headers from a dataset and returns a python dataset without the headers.

Syntax

system.dataset.getColumnHeaders(dataset)

- Parameters

  Dataset dataset - The input dataset.

- Returns

  PyList - A list of column header strings.

- Scope

  All

Code Examples

There are no code examples available for this function.

system.dataset.setValue

Description

Takes a dataset and returns a new dataset with a one value altered. Datasets are immutable, so it is important to realize that this function does not actually set a value in the argument dataset. You'll need to do something with the new dataset that this function creates to achieve something useful.

Syntax

system.dataset.setValue(dataset, rowIndex, columnName, value)

- Parameters

  Dataset dataset - The starting dataset. Will not be modified (datasets are immutable), but acts as the basis for the returned dataset.

  int rowIndex - The index of the row to set the value at (starting at 0)

  String columnName - The name of the column to set the value at. Case insensitive.

  PyObject value - The new value for the specified row/column.

- Returns

  Dataset - A new dataset, with the new value set at the given location.

- Scope

  All

Syntax

system.dataset.setValue(dataset, rowIndex, columnIndex, value)

- Parameters

  Dataset dataset - The starting dataset. Will not be modified (datasets are immutable), but acts as the basis for the returned dataset.

  int rowIndex - The index of the row to set the value at (starting at 0)

  String columnIndex - The index of the column to set the value at (starting at 0)

  PyObject value - The new value for the specified row/column.

- Returns

  Dataset - A new dataset, with the new value set at the given location.

- Scope

  All

Code Examples

Code Snippet

```
#This snippet could be used for a Button's actionPerformed event to change the selected cell's value in a
Table component to zero.

table = event.source.parent.getComponent("Table")
selRow = table.getSelectedRow()
selCol = table.getSelectedColumn()
if selRow != -1 and selCol != -1:
   newData = system.dataset.setValue(table.data, selRow, selCol, 0.0)
   table.data = newData
```

system.dataset.sort

Description

Sorts a dataset and returns the sorted dataset. This works on numeric, as well as alphanumeric columns. It will go character by character, going from 0-9, A-Z, a-z.

Syntax

system.dataset.sort(dataset, keyColumn [, ascending])

- Parameters

  Dataset dataset - The dataset to sort.

  int keyColumn - The index or column name of the column to sort on.

  boolean ascending - True for ascending order, False for descending order. If omitted, ascending order will be used. [optional]

- Returns

  Dataset - A new sorted dataset.

- Scope

  All

Code Examples

Code Snippet

```
#This code will take the data in a table component, sort it based on the column with index 1,
#and then reinsert the sorted data into the same table.

data = event.source.parent.getComponent('Table').data
newData = system.dataset.sort(data, 1)
event.source.parent.getComponent('Table').data = newData
```

Code Snippet

```
#This code will create a dataset in scripting, and then sort it based on the name of one of the columns.
#It then inserts the sorted dataset into a table component.

headers = ["City", "Population", "Timezone", "GMTOffset"]
data = []

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])

cities = system.dataset.toDataSet(headers, data)
newData = system.dataset.sort(cities, "City")
event.source.parent.getComponent('Table').data = newData
```

system.dataset.toCSV

Description

Formats the contents of a dataset as CSV (comma separated values), returning the resulting CSV as a string. If the "forExport" flag is set, then the format will be appropriate for parsing using the system.dataset.fromCSV function.

Syntax

system.dataset.toCSV(dataset, showHeaders, forExport, localized)

- Parameters

  Dataset dataset - The dataset to export to CSV.

  Boolean showHeaders - If set to true(1), a header row will be present in the CSV. Default is true(1).

  Boolean forExport - If set to true(1), extra header information will be present in the CSV data which is necessary for the CSV to be compatible with the fromCSV method. Overrides showHeaders. Default is false(0).

  Boolean localized - If set to true(1), the string representations of the values in the CSV data will be localized.

- Returns

  String - The CSV data as a string

- Scope

  All

Code Examples

Code Snippet

```
#This snippet would run a SQL query against a database, and turn the results into a CSV string. It would
then store resulting CSV to a file on the local hard drive.

results = system.db.runQuery("SELECT * FROM example1 LIMIT 100")
results = system.dataset.toDataSet(results)
csv = system.dataset.toCSV(dataset = results, showHeaders = True, forExport = False)
filePath = "C:\\output\\results.csv"
system.file.writeFile(filePath, csv)
```

system.dataset.toDataSet

Description

This function is used to 1) convert PyDataSets to DataSets, and 2) create new datasets from raw Python lists

Syntax

system.dataset.toDataSet(dataset)

- Parameters

  PyDataSet dataset - A PyDataSet object to convert.

- Returns

  Dataset - The newly created dataset.

- Scope

  All

Syntax

system.dataset.toDataSet(headers, data)

- Parameters

  PySequence headers - The column names for the dataset to create.

  PySequence data - A list of rows for the new dataset. Each row must have the same length as the headers list, and each value in a column must be the same type.

- Returns

  Dataset - The newly created dataset.

- Scope

  All

Code Examples

Code Snippet

```
#This first example shows how this function can be used to convert from a PyDataSet (which is what system.
db.runQuery returns) to a normal DataSet, which is the datatype of a Table component's data property.

pyDataSet = system.db.runQuery("SELECT * FROM example1 LIMIT 100")
table = event.source.parent.getComponent("Table")
normalDataSet = system.dataset.toDataSet(pyDataSet)
table.data = normalDataSet
```

system.dataset.toPyDataSet

Description

This function converts from a normal DataSet to a PyDataSet, which is a wrapper class which makes working with datasets more Python-esque.

Syntax

system.dataset.toPyDataSet(dataset)

- Parameters

  Dataset dataset - A DataSet object to convert into a PyDataSet.

- Returns

  PyDataSet - The newly created PyDataSet.

- Scope

  All

Code Examples

Code Snippet

```
#This example script would be added to a button that is in the same container as the table you are
working with.
#It grabs the data of the Table component, and adds up the values in the column named "Value", displaying
the result to the user.

# Get a Table component's data
table = event.source.parent.getComponent("Table")
data = system.dataset.toPyDataSet(table.data)

# Loop through the data, summing the Value column
value = 0.0
for row in data:
   value += row["Value"]

# Show the user the sum of the Value column
system.gui.messageBox("The value is: %f" % value)
```

system.dataset.updateRow

Description

Takes a dataset and returns a new dataset with a one row altered. Datasets are immutable, so it is important to realize that this function does not actually change the row in the argument dataset. You'll need to do something with the new dataset that this function creates to achieve something useful.

To alter the row, this function takes a Python dictionary to represent the changes to make to the specified row. The keys in the dictionary are used to find the columns to alter.

Syntax

system.dataset.updateRow(dataset, rowIndex, changes)

- Parameters

  Dataset dataset - The starting dataset. Will not be modified (datasets are immutable), but acts as the basis for the returned dataset.

  int rowIndex - The index of the row to update (starting at 0)

  PyDictionary changes - A Dictionary of changes to make. They keys in the dictionary should match column names in the dataset, and their values will be used to update the row.

- Returns

  Dataset - A new dataset with the values at the specified row updated according to the values in the dictionary.

- Scope

  All

Code Examples

Code Snippet

```
#This example could be used to dynamically change the data that an Easy Chart displays. In this simple
example, we assume that the chart is always configured to display a single tank's level.
#This script would update the pen being displayed using a dynamic tank number.

# Generate new tag name and tag path
tankNumber = 5
newName = "Tank%d Level" % tankNumber
newPath = "Tanks/Tank%d/Level" % tankNumber

# Consolidate changes into a dictionary
updates = {"NAME": newName, "TAG_PATH":newPath}

# Update the Easy Chart
chart = event.source.parent.getComponent("Easy Chart")
newPens = system.dataset.updateRow(chart.tagPens, 0, updates)
chart.tagPens = newPens
```

system.date

system.date.add*

Description

This function is a set of functions that include:

Function	Description
system.date.addMillis	Add or subtract an amount of milliseconds to a given date and time.
system.date.addSeconds	Add or subtract an amount of seconds to a given date and time.
system.date.addMinutes	Add or subtract an amount of minutes to a given date and time.
system.date.addHours	Add or subtract an amount of hours to a given date and time.
system.date.addDays	Add or subtract an amount of days to a given date and time.
system.date.addWeeks	Add or subtract an amount of weeks to a given date and time.
system.date.addMonths	Add or subtract an amount of months to a given date and time. This function is unique since each month can have a variable number of days. For example, if the date passed in is March 31st, and we add one month, April does not have a 31st day, so the returned date will be the proper number of months rounded down to the closest available day, in this case April 30th.
system.date.addYears	Add or subtract an amount of years to a given date and time.

Syntax

system.date.add*(date, value)

- Parameters

  Date date- The starting date.

  Int value - The number of units to add, or subtract if the value is negative.

- Returns

  Date - A new date object offset by the integer passed to the function.

- Scope

  All

Code Examples

Code Snippet

```
#This example would add two days to the date passed in.

today = system.date.now()
twoDaysFromToday = system.date.addDays(today, 2)
```

Code Snippet

```
#This example would subtract twenty minutes from a date object we create.
#Even though our original date starts on the 28th, it starts at midnight,
#so subtracting 20 minutes puts it at the previous day.

date = system.date.getDate(2018, 4, 28) #This would print out like Mon May 28 00:00:00 PDT 2018
print system.date.addMinutes(date, -20) #This will print Sun May 27 23:40:00 PDT 2018
```

Code Snippet

```
#This example can be placed on the property change script of one calendar component.
#It will then automatically set a second calendar component two weeks in advanced of the first calendars
selected date.
if event.propertyName == "date":
        date = event.newValue
        event.source.parent.getComponent('End Date Calendar').date = system.date.addWeeks(date, 2)
```

system.date.*Between

Description

This function is a set of functions that include:

Function	Description
system.date.millisBetween	Calculates the number of whole milliseconds between two dates.
system.date.secondsBetween	Calculates the number of whole seconds between two dates.
system.date.minutesBetween	Calculates the number of whole minutes between two dates.
system.date.hoursBetween	Calculates the number of whole hours between two dates.
system.date.daysBetween	Calculates the number of whole days between two dates. Daylight savings changes are taken into account.
system.date.weeksBetween	Calculates the number of whole weeks between two dates.
system.date.monthsBetween	Calculates the number of whole months between two dates. Daylight savings changes are taken into account.
system.date.yearsBetween	Calculates the number of whole years between two dates. Daylight savings changes are taken into account.

Order does matter for the two dates passed in that we are calculating how much time has passed from date 1 to date 2. So, if date 2 is further in time than date 1, then a positive amount of time has passed. If date 2 is backwards in time from date 1, then a negative amount of time has passed.

Syntax

system.date.*between(date_1, date_2)

- Parameters

  Date date_1 - The first date to use.

  Date date_2 - The second date to use.

- Returns

  Int - An integer that is representative of the difference between two dates.

- Scope

  All

Code Examples

Code Snippet

```
#This example would grab the current time, and add 119 minutes to it, then calculate the number
#of hours between the two dates.

first = system.date.now()
second = system.date.addMinutes(first, 119)
print system.date.hoursBetween(first, second)#This would print 1 since it is only 1 whole hour.
```

Code Snippet

```
#This example would create two date objects, one on the 28th of May,
#and one on the 22nd of April, both in 2018. Because the second date is
#before the first date, a negative number will be returned.

first = system.date.getDate(2018, 4, 28)
second = system.date.getDate(2018, 3, 22)
print system.date.daysBetween(first, second) #This will print -36
```

Code Snippet

```
#This example can be placed on the action performed event of a button.
#It will then grab the week difference of two calendar components,
#and enter the value returned into a numeric text field.

first = event.source.parent.getComponent('Start Date Calendar').date
second = event.source.parent.getComponent('End Date Calendar').date
event.source.parent.getComponent('Numeric Text Field').intValue = system.date.weeksBetween(first, second)
```

system.date.format

Description

Returns the given date as a string, formatted according to a pattern. The pattern is a format that is full of various placeholders that will display different parts of the date. These are case-sensitive! These placeholders can be repeated for a different effect. For example, M will give you 1-12, MM will give you 01-12, MMM will give you Jan-Dec, MMMM will give you January-December.

The placeholders are:

Symbol	Description	Presentation	Examples	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	

Expert Tip: This function uses the Java class java.text.SimpleDateFormat internally, and will accept any valid format string for that class.

Syntax

system.date.format(date, format)

- Parameters

  Date date- The date to format.

  String format - A format string such as "yyyy-MM-dd HH:mm:ss".

- Returns

  String - A string representing the formatted datetime

- Scope

  All

Code Examples

Code Snippet

```
#This example would format the current date to look like "01/01/01"

today = system.date.now()
print system.date.format(today, "yy/MM/dd") #This printed 16/04/01
```

Code Snippet

```
#This example would format the current date to look like "2001-01-31 16:59:59"
#This is a standard format that all databases recognize.

today = system.date.now()
print system.date.format(today, "yyyy-MM-dd HH:mm:ss")
```

system.date.fromMillis

Description

Creates a date object given a millisecond value

Syntax

system.date.fromMillis(millis)

- Parameters

  Long millis- The number of milliseconds elapsed since January 1, 1970, 00:00:00 UTC (GMT)

- Returns

  Date - A new date object

- Scope

  All

Code Examples

Code Snippet

```
#This example will print out the date "Fri Aug 18 14:35:25 PDT 2017"

print system.date.fromMillis(1503092125000)
```

system.date.get*

The following feature is new in Ignition version **7.8.1**
Click here to check out the other new features

Some of these functions are new in 7.8.1: getMillis, getSecond, getMinute, getHour12, getHour24, getDayOfWeek, getDayOfYear, getMonth, getQuarter, getYear

Description

This function is a set of functions that include:

Function	Description
system.date.getMillis	Extracts the milliseconds from a date, ranging from 0-999.
system.date.getSecond	Extracts the second from a date, ranging from 0-59.
system.date.getMinute	Extracts the minutes from a date, ranging from 0-59.
system.date.getHour12	Extracts the hour from a date. Uses a 12 hour clock, so noon and midnight are returned as 0.
system.date.getHour24	Extracts the hour from a date. Uses a 24 hour clock, so midnight is zero.
system.date.getDayOfWeek	Extracts the day of the week from a date. Sunday is day 1, Saturday is day 7.
system.date.getDayOfMonth	Extracts the day of the month from a date. The first day of the month is day 1.
system.date.getDayOfYear	Extracts the day of the year from a date. The first day of the year is day 1.
system.date.getMonth	Extracts the month from a date, where January is month 0.
system.date.getQuarter	Extracts the quarter from a date, ranging from 1-4.
system.date.getYear	Extracts the year from a date.
system.date.getAMorPM	Returns a 0 if the time is before noon, and a 1 if the time is after noon.

Syntax

system.date.get*(date)

- Parameters

  Date date - The date to use.

- Returns

  Int - An integer that is representative of the extracted value.

- Scope

  All

Code Examples

Code Snippet

```
#This example would grab the current time, and print the current month.

date = system.date.now()
print system.date.getMonth(date) #This would print the current month.
```

Code Snippet

```
#This example would create a date object, and print out the quarter of that date.

date = system.date.getDate(2017, 3, 15) #This would print "Mon April 15 00:00:00 PDT 2016"
print system.date.getQuarter(date) #This will print 2
```

Code Snippet

```
#This example can be placed on the action performed event of a button.
#It will then grab the day of the week of the calendar component,
#and enter the value returned into a numeric text field.

date = event.source.parent.getComponent('Calendar').date
event.source.parent.getComponent('Numeric Text Field').intValue = system.date.getDayOfWeek(date)
```

system.date.getDate

Description

Creates a new Date object given a year, month and a day. The time will be set to midnight of that day.

Syntax

system.date.getDate(year, month, day)

- Parameters

  Int year - The year for the new date.

  Int month - The month of the new date. January is month 0.

  Int day - The day of the month for the new date. The first day of the month is day 1.

- Returns

  Date - A new date, set to midnight of that day.

- Scope

  All

Code Examples

Code Snippet

```
#This example will create a new date object set to January 1st, 2017.

date = system.date.getDate(2017, 0, 1)
print date
```

system.date.getTimezone

Description

Returns the ID of the current timezone.

*This list is subject to change depending on the exact version of java that is installed.

Africa/Abidjan
Africa/Accra
Africa/Addis_Ababa
Africa/Algiers
Africa/Asmara
Africa/Asmera
Africa/Bamako
Africa/Bangui
Africa/Banjul
Africa/Bissau
Africa/Blantyre
Africa/Brazzaville
Africa/Bujumbura
Africa/Cairo
Africa/Casablanca
Africa/Ceuta
Africa/Conakry
Africa/Dakar
Africa/Dar_es_Salaam
Africa/Djibouti
Africa/Douala
Africa/El_Aaiun
Africa/Freetown
Africa/Gaborone
Africa/Harare
Africa/Johannesburg
Africa/Juba
Africa/Kampala
Africa/Khartoum
Africa/Kigali
Africa/Kinshasa
Africa/Lagos
Africa/Libreville
Africa/Lome
Africa/Luanda
Africa/Lubumbashi
Africa/Lusaka
Africa/Malabo
Africa/Maputo
Africa/Maseru
Africa/Mbabane
Africa/Mogadishu
Africa/Monrovia
Africa/Nairobi
Africa/Ndjamena
Africa/Niamey
Africa/Nouakchott
Africa/Ouagadougou
Africa/Porto-Novo
Africa/Sao_Tome
Africa/Timbuktu
Africa/Tripoli
Africa/Tunis
Africa/Windhoek
America/Adak
America/Anchorage
America/Anguilla
America/Antigua
America/Araguaina
America/Argentina/Buenos_Aires
America/Argentina/Catamarca
America/Argentina/ComodRivadavia
America/Argentina/Cordoba
America/Argentina/Jujuy
America/Argentina/La_Rioja
America/Argentina/Mendoza
America/Argentina/Rio_Gallegos

America/Argentina/Salta
America/Argentina/San_Juan
America/Argentina/San_Luis
America/Argentina/Tucuman
America/Argentina/Ushuaia
America/Aruba
America/Asuncion
America/Atikokan
America/Atka
America/Bahia
America/Bahia_Banderas
America/Barbados
America/Belem
America/Belize
America/Blanc-Sablon
America/Boa_Vista
America/Bogota
America/Boise
America/Buenos_Aires
America/Cambridge_Bay
America/Campo_Grande
America/Cancun
America/Caracas
America/Catamarca
America/Cayenne
America/Cayman
America/Chicago
America/Chihuahua
America/Coral_Harbour
America/Cordoba
America/Costa_Rica
America/Creston
America/Cuiaba
America/Curacao
America/Danmarkshavn
America/Dawson
America/Dawson_Creek
America/Denver
America/Detroit
America/Dominica
America/Edmonton
America/Eirunepe
America/El_Salvador
America/Ensenada
America/Fort_Wayne
America/Fortaleza
America/Glace_Bay
America/Godthab
America/Goose_Bay
America/Grand_Turk
America/Grenada
America/Guadeloupe
America/Guatemala
America/Guayaquil
America/Guyana
America/Halifax
America/Havana
America/Hermosillo
America/Indiana/Indianapolis
America/Indiana/Knox
America/Indiana/Marengo
America/Indiana/Petersburg
America/Indiana/Tell_City
America/Indiana/Vevay
America/Indiana/Vincennes
America/Indiana/Winamac
America/Indianapolis
America/Inuvik
America/Iqaluit
America/Jamaica
America/Jujuy
America/Juneau
America/Kentucky/Louisville
America/Kentucky/Monticello
America/Knox_IN
America/Kralendijk
America/La_Paz
America/Lima
America/Los_Angeles

America/Louisville
America/Lower_Princes
America/Maceio
America/Managua
America/Manaus
America/Marigot
America/Martinique
America/Matamoros
America/Mazatlan
America/Mendoza
America/Menominee
America/Merida
America/Metlakatla
America/Mexico_City
America/Miquelon
America/Moncton
America/Monterrey
America/Montevideo
America/Montreal
America/Montserrat
America/Nassau
America/New_York
America/Nipigon
America/Nome
America/Noronha
America/North_Dakota/Beulah
America/North_Dakota/Center
America/North_Dakota/New_Salem
America/Ojinaga
America/Panama
America/Pangnirtung
America/Paramaribo
America/Phoenix
America/Port-au-Prince
America/Port_of_Spain
America/Porto_Acre
America/Porto_Velho
America/Puerto_Rico
America/Rainy_River
America/Rankin_Inlet
America/Recife
America/Regina
America/Resolute
America/Rio_Branco
America/Rosario
America/Santa_Isabel
America/Santarem
America/Santiago
America/Santo_Domingo
America/Sao_Paulo
America/Scoresbysund
America/Shiprock
America/Sitka
America/St_Barthelemy
America/St_Johns
America/St_Kitts
America/St_Lucia
America/St_Thomas
America/St_Vincent
America/Swift_Current
America/Tegucigalpa
America/Thule
America/Thunder_Bay
America/Tijuana
America/Toronto
America/Tortola
America/Vancouver
America/Virgin
America/Whitehorse
America/Winnipeg
America/Yakutat
America/Yellowknife
Antarctica/Casey
Antarctica/Davis
Antarctica/DumontDUrville
Antarctica/Macquarie
Antarctica/Mawson
Antarctica/McMurdo
Antarctica/Palmer

Antarctica/Rothera
Antarctica/South_Pole
Antarctica/Syowa
Antarctica/Troll
Antarctica/Vostok
Arctic/Longyearbyen
Asia/Aden
Asia/Almaty
Asia/Amman
Asia/Anadyr
Asia/Aqtau
Asia/Aqtobe
Asia/Ashgabat
Asia/Ashkhabad
Asia/Baghdad
Asia/Bahrain
Asia/Baku
Asia/Bangkok
Asia/Beirut
Asia/Bishkek
Asia/Brunei
Asia/Calcutta
Asia/Chita
Asia/Choibalsan
Asia/Chongqing
Asia/Chungking
Asia/Colombo
Asia/Dacca
Asia/Damascus
Asia/Dhaka
Asia/Dili
Asia/Dubai
Asia/Dushanbe
Asia/Gaza
Asia/Harbin
Asia/Hebron
Asia/Ho_Chi_Minh
Asia/Hong_Kong
Asia/Hovd
Asia/Irkutsk
Asia/Istanbul
Asia/Jakarta
Asia/Jayapura
Asia/Jerusalem
Asia/Kabul
Asia/Kamchatka
Asia/Karachi
Asia/Kashgar
Asia/Kathmandu
Asia/Katmandu
Asia/Khandyga
Asia/Kolkata
Asia/Krasnoyarsk
Asia/Kuala_Lumpur
Asia/Kuching
Asia/Kuwait
Asia/Macao
Asia/Macau
Asia/Magadan
Asia/Makassar
Asia/Manila
Asia/Muscat
Asia/Nicosia
Asia/Novokuznetsk
Asia/Novosibirsk
Asia/Omsk
Asia/Oral
Asia/Phnom_Penh
Asia/Pontianak
Asia/Pyongyang
Asia/Qatar
Asia/Qyzylorda
Asia/Rangoon
Asia/Riyadh
Asia/Saigon
Asia/Sakhalin
Asia/Samarkand
Asia/Seoul
Asia/Shanghai

Asia/Singapore
Asia/Srednekolymsk
Asia/Taipei
Asia/Tashkent
Asia/Tbilisi
Asia/Tehran
Asia/Tel_Aviv
Asia/Thimbu
Asia/Thimphu
Asia/Tokyo
Asia/Ujung_Pandang
Asia/Ulaanbaatar
Asia/Ulan_Bator
Asia/Urumqi
Asia/Ust-Nera
Asia/Vientiane
Asia/Vladivostok
Asia/Yakutsk
Asia/Yekaterinburg
Asia/Yerevan
Atlantic/Azores
Atlantic/Bermuda
Atlantic/Canary
Atlantic/Cape_Verde
Atlantic/Faeroe
Atlantic/Faroe
Atlantic/Jan_Mayen
Atlantic/Madeira
Atlantic/Reykjavik
Atlantic/South_Georgia
Atlantic/St_Helena
Atlantic/Stanley
Australia/ACT
Australia/Adelaide
Australia/Brisbane
Australia/Broken_Hill
Australia/Canberra
Australia/Currie
Australia/Darwin
Australia/Eucla
Australia/Hobart
Australia/LHI
Australia/Lindeman
Australia/Lord_Howe
Australia/Melbourne
Australia/NSW
Australia/North
Australia/Perth
Australia/Queensland
Australia/South
Australia/Sydney
Australia/Tasmania
Australia/Victoria
Australia/West
Australia/Yancowinna
Brazil/Acre
Brazil/DeNoronha
Brazil/East
Brazil/West
CET
CST6CDT
Canada/Atlantic
Canada/Central
Canada/East-Saskatchewan
Canada/Eastern
Canada/Mountain
Canada/Newfoundland
Canada/Pacific
Canada/Saskatchewan
Canada/Yukon
Chile/Continental
Chile/EasterIsland
Cuba
EET
EST5EDT
Egypt
Eire
Etc/GMT
Etc/GMT+0

Etc/GMT+1
Etc/GMT+10
Etc/GMT+11
Etc/GMT+12
Etc/GMT+2
Etc/GMT+3
Etc/GMT+4
Etc/GMT+5
Etc/GMT+6
Etc/GMT+7
Etc/GMT+8
Etc/GMT+9
Etc/GMT-0
Etc/GMT-1
Etc/GMT-10
Etc/GMT-11
Etc/GMT-12
Etc/GMT-13
Etc/GMT-14
Etc/GMT-2
Etc/GMT-3
Etc/GMT-4
Etc/GMT-5
Etc/GMT-6
Etc/GMT-7
Etc/GMT-8
Etc/GMT-9
Etc/GMT0
Etc/Greenwich
Etc/UCT
Etc/UTC
Etc/Universal
Etc/Zulu
Europe/Amsterdam
Europe/Andorra
Europe/Athens
Europe/Belfast
Europe/Belgrade
Europe/Berlin
Europe/Bratislava
Europe/Brussels
Europe/Bucharest
Europe/Budapest
Europe/Busingen
Europe/Chisinau
Europe/Copenhagen
Europe/Dublin
Europe/Gibraltar
Europe/Guernsey
Europe/Helsinki
Europe/Isle_of_Man
Europe/Istanbul
Europe/Jersey
Europe/Kaliningrad
Europe/Kiev
Europe/Lisbon
Europe/Ljubljana
Europe/London
Europe/Luxembourg
Europe/Madrid
Europe/Malta
Europe/Mariehamn
Europe/Minsk
Europe/Monaco
Europe/Moscow
Europe/Nicosia
Europe/Oslo
Europe/Paris
Europe/Podgorica
Europe/Prague
Europe/Riga
Europe/Rome
Europe/Samara
Europe/San_Marino
Europe/Sarajevo
Europe/Simferopol
Europe/Skopje
Europe/Sofia
Europe/Stockholm

Europe/Tallinn
Europe/Tirane
Europe/Tiraspol
Europe/Uzhgorod
Europe/Vaduz
Europe/Vatican
Europe/Vienna
Europe/Vilnius
Europe/Volgograd
Europe/Warsaw
Europe/Zagreb
Europe/Zaporozhye
Europe/Zurich
GB
GB-Eire
GMT
GMT0
Greenwich
Hongkong
Iceland
Indian/Antananarivo
Indian/Chagos
Indian/Christmas
Indian/Cocos
Indian/Comoro
Indian/Kerguelen
Indian/Mahe
Indian/Maldives
Indian/Mauritius
Indian/Mayotte
Indian/Reunion
Iran
Israel
Jamaica
Japan
Kwajalein
Libya
MET
MST7MDT
Mexico/BajaNorte
Mexico/BajaSur
Mexico/General
NZ
NZ-CHAT
Navajo
PRC
PST8PDT
Pacific/Apia
Pacific/Auckland
Pacific/Bougainville
Pacific/Chatham
Pacific/Chuuk
Pacific/Easter
Pacific/Efate
Pacific/Enderbury
Pacific/Fakaofo
Pacific/Fiji
Pacific/Funafuti
Pacific/Galapagos
Pacific/Gambier
Pacific/Guadalcanal
Pacific/Guam
Pacific/Honolulu
Pacific/Johnston
Pacific/Kiritimati
Pacific/Kosrae
Pacific/Kwajalein
Pacific/Majuro
Pacific/Marquesas
Pacific/Midway
Pacific/Nauru
Pacific/Niue
Pacific/Norfolk
Pacific/Noumea
Pacific/Pago_Pago
Pacific/Palau
Pacific/Pitcairn
Pacific/Pohnpei
Pacific/Ponape

Pacific/Port_Moresby
Pacific/Rarotonga
Pacific/Saipan
Pacific/Samoa
Pacific/Tahiti
Pacific/Tarawa
Pacific/Tongatapu
Pacific/Truk
Pacific/Wake
Pacific/Wallis
Pacific/Yap
Poland
Portugal
ROK
Singapore
SystemV/AST4
SystemV/AST4ADT
SystemV/CST6
SystemV/CST6CDT
SystemV/EST5
SystemV/EST5EDT
SystemV/HST10
SystemV/MST7
SystemV/MST7MDT
SystemV/PST8
SystemV/PST8PDT
SystemV/YST9
SystemV/YST9YDT
Turkey
UCT
US/Alaska
US/Aleutian
US/Arizona
US/Central
US/East-Indiana
US/Eastern
US/Hawaii
US/Indiana-Starke
US/Michigan
US/Mountain
US/Pacific
US/Pacific-New
US/Samoa
UTC
Universal
W-SU
WET
Zulu
EST
HST
MST
ACT
AET
AGT
ART
AST
BET
BST
CAT
CNT
CST
CTT
EAT
ECT
IET
IST
JST
MIT
NET
NST
PLT
PNT
PRT
PST
SST
VST

Syntax

system.date.getTimezone()

- Parameters

  none

- Returns

  String - A representation of the current timezone.

- Scope

  All

Code Examples

Code Snippet

```
#This example will print out your current Timezone ID.
#If your Client and Gateway are in different timezones, the returned value will be
#dependent on where this script is run.
#IE: in a button on a client will return the client timezone. On a Gateway script will
#return the Gateway timezone.

print system.date.getTimezone()
```

system.date.getTimezoneOffset

Description

Returns the current timezone's offset versus UTC for a given instant, taking Daylight Savings Time into account.

Syntax

system.date.getTimezoneOffset([date])

- Parameters

  Date date- The instant in time for which to calculate the offset. Uses now() if omitted. [optional]

- Returns

  Double - The timezone offset compared to UTC, in hours.

- Scope

  All

Code Examples

Code Snippet

```
#This example will give the timezone offset using the date February 22, 2017
#and the computers current timezone.

date = system.date.getDate(2017, 1, 22)
print system.date.getTimezoneOffset(date) # returns -8.0 (if you are in Pacific Daylight Time)
```

system.date.getTimezoneRawOffset

The following feature is new in Ignition version **7.8.1**
Click here to check out the other new features

Description

Returns the current timezone offset versus UTC, not taking daylight savings into account.

Syntax

system.date.getTimezoneRawOffset()

- Parameters

  none

- Returns

  Double - The timezone offset.

- Scope

  All

Code Examples

Code Snippet

```
#This example will give the Raw timezone offset (ignoring daylight savings) for the computer's current timezone.

print system.date.getTimezoneRawOffset() # returns -8.0 (if you are in the Pacific Timezone)
```

system.date.isAfter

The following feature is new in Ignition version **7.8.1**
Click here to check out the other new features

Description

Compares to dates to see if date_1 is after date_2.

Syntax

system.date.isAfter(date_1, date_2)

- Parameters

  Date date_1 - The first date.

  Date date_2 - The second date.

- Returns

  Bool - True (1) if date_1 is after date_2, false (0) otherwise.

- Scope

  All

Code Examples

Code Snippet

```
#This will compare if the first date is after the second date, which it is.

first = system.date.getDate(2018, 4, 28)
second = system.date.getDate(2018, 3, 22)
print system.date.isAfter(first, second) #Will print true.
```

system.date.isBefore

The following feature is new in Ignition version **7.8.1**
Click here to check out the other new features

Description

Compares to dates to see if date_1 is before date_2.

Syntax

system.date.isBefore(date_1, date_2)

- Parameters

  Date date_1 - The first date.

  Date date_2 - The second date.

- Returns

  Bool - True (1) if date_1 is before date_2, false (0) otherwise.

- Scope

  All

Code Examples

Code Snippet

```
#This will compare if the first date is before the second date, which it is not.

first = system.date.getDate(2018, 4, 28)
second = system.date.getDate(2018, 3, 22)
print system.date.isBefore(first, second) #Will print false.
```

system.date.isBetween

The following feature is new in Ignition version **7.8.1**
Click here to check out the other new features

Description

Compares to dates to see if a target date is between two other dates.

Syntax

system.date.isBefore(target_date, start_date, end_date)

- Parameters

  Date target_date - The date to compare.

  Date start_date - The start of a date range.

  Date end_date - The end of a date range. This date myst be after the start date.

- Returns

  Bool - True (1) if target_date is >= start_date and target_date <= end_date, false (0) otherwise.

- Scope

  All

Code Examples

Code Snippet

```
#This will compare if the first date is between the other dates, which it is not.
#Note that if the end date is before the start date, this function will always return False

target = system.date.getDate(2017, 4, 28)
start = system.date.getDate(2017, 3, 22)
end = system.date.getDate(2017, 4, 22)
print system.date.isBetween(target, start, end) #Will print false.
```

system.date.isDaylightTime

Description

Checks to see if the current timezone is using daylight savings time during the date specified.

Syntax

system.date.isDaylightTime([date])

- Parameters

  Date date - The date you want to check if the current timezone is observing daylight savings time. Uses now() if omitted. [optional]

- Returns

  Bool - True (1) if date is observing dalight savings time in the current timezone, false (0) otherwise.

- Scope

  All

Code Examples

Code Snippet

```
#This will compare if the first date is before the second date, which it is not.

date = system.date.getDate(2018, 6, 28)
print system.date.isDaylightTime(date) #Will print True in the US Pacific Timezone.
```

system.date.midnight

Description

Returns a copy of a date with the hour, minute, second, and millisecond fields set to zero.

Syntax

system.date.midnight(date)

- Parameters

  Date date- The starting date.

- Returns

  Date - A new date, set to midnight of the day provided

- Scope

  All

Code Examples

Code Snippet

```
#This example will print out the current date with the time set to midnight.

date = system.date.now()
print system.date.midnight(date)
```

system.date.now

Description

Returns a java.util.Date object that represents the current time according to the local system clock.

Syntax

system.date.now()

- Parameters

  none

- Returns

  Date - A new date, set to the current date and time.

- Scope

  All

Code Examples

Code Snippet

```
#This example will set a calendar component to the current date and time.

event.source.parent.getComponent('Calendar').date = system.date.now()
```

system.date.setTime

The following feature is new in Ignition version **7.8.1**
Click here to check out the other new features

Description

Takes in a date, and returns a copy of it with the time fields set as specified.

Syntax

system.date.setTime(date, hour, minute, second)

- Parameters

  Date date - The starting date.

  Int hour - The hours (0-23) to set.

  Int minute - The minutes (0-59) to set.

  Int second - The seconds (0-59) to set.

- Returns

  Date - A new date, set to the appropriate time.

- Scope

  All

Code Examples

Code Snippet

```
#This example will set the date object to the current date with the time set to 01:37 in the morning and 44
seconds.

date = system.date.getDate(2018, 6, 29)
print system.date.setTime(date, 1, 37, 44) #This will print Fri June 29 01:37:44 PDT 2018
```

system.date.toMillis

Description

Converts a Date object to its millisecond value elapsed since January 1, 1970, 00:00:00 UTC (GMT)

Syntax

system.date.toMillis(date)

- Parameters

  Date date - The date object to convert.

- Returns

  Long - an 8-byte integer representing the number of millisecond elapsed since January 1, 1970, 00:00:00 UTC (GMT)

- Scope

  All

Code Examples

Code Snippet

```
#This example will take the date Fri Aug 18 14:35:25 PDT 2017,
#and print out 1503092125000

date = system.date.getDate(2017, 7, 18)
datetime = system.date.setTime(date, 14, 35, 25)
print system.date.toMillis(datetime)
```

system.db

system.db.addDatasource

Description

Adds a new database connection in Ignition.

Syntax

system.db.addDatasource(jdbcDriver, name, description, connectUrl, username, password, props, validationQuery, maxConnections)

- Parameters

  String jdbcDriver - The name of the JDBC driver in Ignition. Required.

  String name - The datasource name. Required.

  String description

  String connectUrl - Default is the connect URL for JDBC driver.

  String username

  String password

  String props - The extra connection parameters.

  String validationQuery - Default is the validation query for the JDBC driver.

  Integer maxConnections - Default is 8.

- Returns

  Nothing

- Scope

  All

Code Examples

Code Snippet

```
#The following code would add a MySQL connection to the Gateway.
system.db.addDatasource(jdbcDriver="MySQL ConnectorJ", name="MySQL",
connectURL="jdbc:mysql://localhost:3306/test", username="root",
password="password", props="zeroDateTimeBehavior=convertToNull;")
```

system.db.beginTransaction

Description

Begins a new database transaction. Database transactions are used to execute multiple queries in an atomic fashion. After executing queries, you must either commit the transaction to have your changes take effect, or rollback the transaction which will make all operations since the last commit not take place. The transaction is given a new unique string code, which is then returned. You can then use this code as the tx argument for other system.db.* function calls to execute various types of queries using this transaction.

An open transaction consumes one database connection until it is closed. Because leaving connections open indefinitely would exhaust the connection pool, each transaction is given a timeout. Each time the transaction is used, the timeout timer is reset. For example, if you make a transaction with a timeout of one minute, you must use that transaction at least once a minute. If a transaction is detected to have timed out, it will be automatically closed and its transaction id will no longer be valid.

Syntax

system.db.beginTransaction(database, isolationLevel, timeout)

- Parameters

  String database - The name of the database connection to create a transaction in.

  Integer isolationLevel - The transaction isolation level to use. Use one of the four constants: system.db.READ_COMMITTED, system.db.READ_UNCOMMITTED, system.db.REPEATABLE_READ, or system.db.SERIALIZABLE

  Long timeout - The amount of time, in milliseconds, that this connection is allowed to remain open without being used. Timeout counter is reset any time a query or call is executed against the transaction, or when committed or rolled-back.

- Returns

  String - The new transaction ID. You'll use this ID as the "tx" argument for all other calls to have them execute against this transaction.

- Scope

  Gateway

Syntax

system.db.beginTransaction(database, isolationLevel, timeout)

- Parameters

  String database - The name of the database connection to create a transaction in. Use "" for the project's default connection.

  Integer isolationLevel - The transaction isolation level to use. Use one of the four constants: system.db.READ_COMMITTED, system.db.READ_UNCOMMITTED, system.db.REPEATABLE_READ, or system.db.SERIALIZABLE

  Long timeout - The amount of time, in milliseconds, that this connection is allowed to remain open without being used. Timeout counter is reset any time a query or call is executed against the transaction, or when committed or rolled-back.

- Returns

  String - The new transaction ID. You'll use this ID as the "tx" argument for all other calls to have them execute against this transaction.

- Scope

  Vision Client

Isolation Level Values

The following table lists each value of the isolationLevel parameter and its associated level. Either the integer value or constant may be passed. Note that some JDBC drivers only support some levels, so the driver's documentation should be consulted. Isolation levels are well documented online, but the following link is a great starting point: Data Concurrency and Consistency

Isolation Level	Int Value	Constant
Read Uncommitted	1	system.db.READ_UNCOMMITTED
Read Committed	2	system.db.READ_COMMITTED
Repeatable Read	4	system.db.REPEATABLE_READ
Serializable	8	system.db.SERIALIZABLE

Code Examples

Code Snippet

```
#This example would start a transaction with a 5 second timeout against the project's default database,
using the default isolation level. Then it executes a series of update calls, and commits and closes the
transaction.

txId = system.db.beginTransaction(timeout=5000)
status=2

for machineId in range(8):
   system.db.runPrepUpdate("UPDATE MachineStatus SET status=? WHERE ID=?",
      args=[status, machineId], tx=txId)

system.db.commitTransaction(txId)
system.db.closeTransaction(txId)
```

system.db.closeTransaction

Description

Closes the transaction with the given ID. Note that you must commit or rollback the transaction before you close it. Closing the transaction will return its database connection to the pool. The transaction ID will no longer be valid.

Syntax

system.db.closeTransaction(tx)

- Parameters

  String tx - The transaction ID.

- Returns

  Nothing

- Scope

  All

Code Examples

There are no example available for this function.

system.db.commitTransaction

Description

Performs a commit for the given transaction. This will make all statements executed against the transaction since its beginning or since the last commit or rollback take effect in the database. Until you commit a transaction, any changes that the transaction makes will not be visible to other connections. Note that if you are done with the transaction, you must close it after you commit it.

Syntax

system.db.commitTransaction(tx)

- Parameters

  String tx - The transaction ID.

- Returns

  Nothing

- Scope

  All

Code Examples

There are no code examples available for this function.

system.db.createSProcCall

Description

Creates an SProcCall object, which is a stored procedure call context. This is an object that is used to configure a call to a stored procedure. Once configured, you'd use system.db.execSProcCall to call the stored procedure. The call context object then holds any results from the stored procedure. The SProcCall object has the following functions used for registering parameters:

SPRocCall.registerInParam(index OR name, typeCode, value)

SPRocCall.registerOutParam(index OR name, typeCode)

SPRocCall.registerReturnParam(typeCode)

These functions are used to register any in/out parameters for the stored procedure. Parameters can be referenced by index (starting at 1, not 0), or by name. To register an in/out parameter, you simply register it twice - once as an input parameter with the value you'd like to pass to the stored procedure, and once as an output parameter. N.B. not all JDBC drivers support named procedure parameters. If your function returns a value, you must use registerReturnParam to specify the datatype of the returned value. Note that this is different from stored procedures that return a result set , which doesn't require any setup on the SProcCall object. Some database systems call stored procedures that return a value "functions" instead of "procedures". For all of these functions, you'll need to specify a type code. These are codes defined by the JDBC specification. For your convenience, the codes exist as constants in the system.db namespace. Each type code will be mapped to a database-specific type by the JDBC driver. Not all type codes will be recognized by all JDBC drivers. The following type code constants are available for use in createSProcCall:

BIT	REAL	LONGVARCHAR	LONGVARBINARY	BLOB
TINYINT	DOUBLE	DATE	NULL	CLOB
SMALLINT	NUMERIC	TIME	OTHER	JAVA_OBJECT
INTEGER	DECIMAL	TIMESTAMP	SQLXML	DATALINK
BIGINT	CHAR	BINARY	NCLOB	BOOLEAN
FLOAT	VARCHAR	VARBINARY	ARRAY	ROWID
NCHAR	NVARCHAR	LONGNVARCHAR		

The following type code constants are available for other uses, but are not supported by createSProcCall:

ORACLE_CURSOR	DISTINCT	STRUCT	REF

Once the call context has been executed, you can retrieve the result set, return value, and output parameter values (if applicable) by calling the following functions:

SProcCall.getResultSet() - returns a dataset that is the resulting data of the stored procedure, if any.

SProcCall.getUpdateCount() - returns the number of rows modified by the stored procedure, or -1 if not applicable.

SProcCall.getReturnValue() - returns the return value, if registerReturnParam had been called.

SProcCall.getOutParamValue(index OR name) - returns the value of the previously registered out-parameter.

Syntax

system.db.createSProcCall(procedureName, database, tx, skipAudit)

- Parameters

  String procedureName - The named of the stored procedure to call.

  String database - The name of the database connection to execute against.

  String tx - A transaction identifier. If omitted, the call will be executed in its own transaction.

  boolean skipAudit - A flag which, if set to true, will cause the procedure call to skip the audit system. Useful for some queries that have fields which won't fit into the audit log.

- Returns

  SProcCall - A stored procedure call context, which can be configured and then used as the argument to system.db.execSProcCall.

- Scope

  Gateway

Syntax

system.db.createSProcCall(procedureName, database, tx, skipAudit)

- Parameters

  String procedureName - The named of the stored procedure to call.

  String database - The name of the database connection to execute against. If omitted or "", the project's default database connection will be used.

  String tx - A transaction identifier. If omitted, the call will be executed in its own transaction.

  boolean skipAudit - A flag which, if set to true, will cause the procedure call to skip the audit system. Useful for some queries that have fields which won't fit into the audit log.

- Returns

  SProcCall - A stored procedure call context, which can be configured and then used as the argument to system.db.execSProcCall.

- Scope

  Vision Client

Code Examples

Code Snippet

```
#This example would call a stored procedure named "start_batch" against the current project's default
database connection that had no input or output parameters, and did not return any values or results:

call = system.db.createSProcCall("start_batch")
system.db.execSProcCall(call)
```

Code Snippet

```
#This example would call a stored procedure "get_shift_workers" with no arguments, which returned a
result set of employees for the current shift. It then pushes the resulting dataset into a Table
component:

call = system.db.createSProcCall("get_shift_workers")
system.db.execSProcCall(call)

results = call.getResultSet()
table = event.source.parent.getComponent("Table")
table.data = results
```

Code Snippet

```
#This example would call a stored procedure that took two arguments, the first an integer and the second
a string. It also is configured to return an integer value.

call = system.db.createSProcCall("perform_calculation")
call.registerReturnParam(system.db.INTEGER)
call.registerInParam(1, system.db.INTEGER, 42)
call.registerInParam(2, system.db.VARCHAR, "DC-MODE")

system.db.execSProcCall(call)

#Print the result to the console
print call.getReturnValue()
```

Code Snippet

```
#This example would do the same as the one above, except for a stored procedure that returned its value
using an out-parameter. It also uses named argument names instead of indexed arguments.

call = system.db.createSProcCall("perform_calculation")
call.registerInParam("arg_one", system.db.INTEGER, 42)
call.registerInParam("arg_two", system.db.VARCHAR, "DC-MODE")
call.registerOutParam("output_arg", system.db.INTEGER)

system.db.execSProcCall(call)

#Print the result to the console
print call.getOutParamValue("output_arg")
```

system.db.dateFormat

Description

This function is used to format Dates nicely as strings. It uses a format string to guide its formatting behavior. Learn more about date formatting in Working with Datatypes / Dates

Expert Tip: This function uses the Java class java.text.SimpleDateFormat internally, and will accept any valid format string for that class.

Syntax

system.db.dateFormat(date, formatPattern)

- Parameters

  Date date - The Date object that you'd like to format

  String formatPattern - A format pattern string to apply.

- Returns

  String - The date as a string formatted according to the format pattern.

- Scope

  All

Code Examples

Code Snippet

```
#This example will display a message box on a button press that displays the selected date (without the
time) from a Calendar component, in a format like "Feb 3, 2009"
date = event.source.parent.getComponent("Calendar").latchedDate
toDisplay = system.db.dateFormat(date, "MMM d, yyyy")
system.gui.messageBox("The date you selected is: %s" % toDisplay)
```

Code Snippet

```
#This example would do the same as the one above, but also display the time, in a format like: "Feb 3,
2009 8:01pm"
date = event.source.parent.getComponent("Calendar").latchedDate
toDisplay = system.db.dateFormat(date, "MMM d, yyyy")
system.gui.messageBox("The date you selected is: %s" % toDisplay)
```

Code Snippet

```
#This example would take two dates from two Popup Calendar components, format them in a manner that the
database understands, and then use them in a SQL query to limit the results to a certain date range.
startDate = event.source.parent.getComponent("StartDate").date
endDate = event.source.parent.getComponent("EndDate").date
startDate = system.db.dateFormat(startDate, "yyyy-MM-dd HH:mm:ss")
endDate = system.db.dateFormat(endDate, "yyyy-MM-dd HH:mm:ss")
query = ("SELECT * FROM mytable WHERE t_stamp >= '%s' AND t_stamp <= '%s'" % (startDate, endDate))
results = system.db.runQuery(query)
event.source.parent.getComponent("Table").data = results
```

Code Snippet

```
#This example would show how to get the current date in scripting, and extract the month and year into
separate variables, and
#assign them to a "Month View" calendar object, in case you changed the viewing date and needed to return.

# first import the java.util.Date object required for getting the current date
from java.util import Date

# get calendar object we will be editing (you would have to place one by this name on your window)
cal = event.source.parent.parent.getComponent('Month View')


# get current date and separate into month and year, using dateFormat. Month View object requires an
integer, so this is wrapped in int() currentDate = Date()
currentMonth = int(system.db.dateFormat(currentDate, "M"))
currentYear = int(system.db.dateFormat(currentDate, "Y"))


# change our Month View calendar object to the current month and year (provided it was not on the current
month/year)
cal.month = currentMonth
cal.year = currentYear
```

system.db.execSProcCall

Description

Executes a stored procedure call. The one parameter to this function is an SProcCall - a stored procedure call context. See the description of system.db.createSProcCall for more information and examples.

Syntax

system.db.execSProcCall(callContext)

- Parameters

  SProcCall callContext - A stored procedure call context, with any input, output, and/or return value parameters correctly configured. Use system.db.createSProcCall to create a call context.

- Returns

  Nothing

- Scope

  All

Code Examples

There are no code examples available for this function.

system.db.getConnectionInfo

Description

Returns a dataset of information about a single database connection, as specified by the name argument.

Syntax

system.db.getConnectionInfo(name)

- Parameters

  String name - The name of the database connection to find information about.

- Returns

  Dataset - A dataset containing information about the named database connection, or an empty dataset if the connection wasn't found.

- Scope

  All

Code Examples

There are no code examples available for this function.

system.db.getConnections

Description

Returns a dataset of information about each configured database connection. Each row represents a single connection.

Syntax

system.db.getConnections()

- Parameters

  None

- Returns

  Dataset - A dataset, where each row represents a database connection.

- Scope

  All

Code Examples

There are no code examples available for this function.

system.db.refresh

Description

This function will programmatically cause a SQL Query or DB Browse property binding to execute immediately. This is most often used for bindings that are set to Polling - Off. In this way, you cause a binding to execute on demand, when you know that the results of its query will return a new result. To use it, you simply specify the component and name of the property on whose binding you'd like to refresh.

Syntax

system.db.refresh(component, propertyName)

- Parameters

  JComponent component - The component whose property you want to refresh

  String propertyName - The name of the property that has a SQL Query binding that needs to be refreshed

- Returns

  boolean - True (1) if the property was found and refreshed successfully.

- Scope

  All

Code Examples

Code Snippet

```
#This example could be placed in the actionPerformed event of a Button, to be used to refresh the data of
a Table.
#Remember to use the scripting name of the property that you're trying to refresh, and that the property
names are case-sensitive.

table = event.source.parent.getComponent("Table")
system.db.refresh(table, "data")
```

system.db.removeDatasource

Description

Removes a database connection from Ignition.

Syntax

system.db.removeDatasource(name)

- Parameters

  String name - The name of the database connection in Ignition.

- Returns

  Nothing

- Scope

  All

Code Examples

Code Snippet

```
#This will result in the connection named MySQL being removed
system.db.removeDatasource("MySQL")
```

system.db.rollbackTransaction

Description

Performs a rollback on the given connection. This will make all statements executed against this transaction since its beginning or since the last commit or rollback undone. Note that if you are done with the transaction, you must also close it afterward you do a rollback on it.

Syntax

system.db.rollbackTransaction(tx)

- Parameters

  String tx - The transaction ID.

- Returns

  Nothing

- Scope

  All

Code Examples

There are no examples available for this code function.

system.db.runPrepQuery

Description

Runs a prepared statement against the database, returning the results in a PyDataSet.. Prepared statements differ from regular queries in that they can use a special placeholder, the question-mark character (?) in the query where any dynamic arguments would go, and then use an array of values to provide real information for those arguments. Make sure that the length of your argument array matches the number of question-mark placeholders in your query.

This call should be used for SELECT queries.This is a useful alternative to system.db.runQuery because it allows values in the WHERE clause, JOIN clause, and other clauses to be specified without having to turn those values into strings. This is safer because it protects against a problem known as a SQL injection attack, where a user can input data that affects the query's semantics.

The "?" placeholder refers to variables of the query statement that help the statement return the correct information. The "?" placeholder cannot reference column names, table names, or the underlying syntax of the query. This is because the SQL standard for handling the "?" placeholder excludes these items.

Syntax

system.db.runPrepQuery(query, args, database, tx)

- Parameters

  String query - A query (typically a SELECT) to run as a prepared statement with placeholders (?) denoting where the arguments go.

  Object[] args - A list of arguments. Will be used in order to match each placeholder (?) found in the query.

  String database - The name of the database connection to execute against.

  String tx - A transaction identifier. If omitted, the query will be executed in its own transaction.

- Returns

  PyDataSet - The results of the query as a PyDataSet

- Scope

  Gateway

Syntax

system.db.runPrepQuery(query, args, database, tx)

- Parameters

  String query - A query (typically a SELECT) to run as a prepared statement with placeholders (?) denoting where the arguments go.

  Object[] args - A list of arguments. Will be used in order to match each placeholder (?) found in the query.

  String database - The name of the database connection to execute against. If omitted or "", the project's default database connection will be used.

  String tx - A transaction identifier. If omitted, the query will be executed in its own transaction.

- Returns

  PyDataSet - The results of the query as a PyDataSet

- Scope

  Vision Client

Code Examples

Code Snippet

```
#This example would search for all records in a LogEntry table where the message contained a user-entered
search term.

search = event.source.parent.getComponent("SearchFor").text
# Wrap the term in % signs for LIKE-style matching
search = '%' + search + '%'

results= system.db.runPrepQuery("SELECT * FROM LogEntry WHERE EntryText LIKE ?", [search])
event.source.parent.getComponent("Table").data = results
```

system.db.runPrepUpdate

Description

Runs a prepared statement against the database, returning the number of rows that were affected. Prepared statements differ from regular queries in that they can use a special placeholder, the question-mark character (?) in the query where any dynamic arguments would go, and then use an array of values to provide real information for those arguments. Make sure that the length of your argument array matches the number of question-mark placeholders in your query. This call should be used for UPDATE, INSERT, and DELETE queries.

This is extremely useful for two purposes:

- This method avoids the problematic technique of concatenating user input inside of a query, which can lead to syntax errors, or worse, a nasty security problem called a SQL injection attack. For example, if you have a user-supplied string that is used in a WHERE clause, you use single-quotes to enclose the string to make the query valid. What happens in the user has a single-quote in their text? Your query will fail. Prepared statements are immune to this problem.
- This is the only way to write an INSERT or UPDATE query that has binary or BLOB data. Using BLOBs can be very hand for storing images or reports in the database, where all clients have access to them.

The "?" placeholder refers to variables of the query statement that help the statement return the correct information. The "?" placeholder cannot reference column names, table names, or the underlying syntax of the query. This is because the SQL standard for handling the "?" placeholder excludes these items.

Syntax

system.db.runPrepUpdate(query, args, database, [tx], [getKey], [skipAudit])

- Parameters

  String query - A query (typically an UPDATE, INSERT, or DELETE) to run as a prepared statement with placeholders (?) denoting where the arguments go.

  Object[] args - A list of arguments. Will be used in order to match each placeholder (?) found in the query.

  String database - The name of the database connection to execute against.

  String tx - Optional, A transaction identifier. If omitted, the update will be executed in its own transaction.

  Boolean getKey - Optional, A flag indicating whether or not the result should be the number of rows returned (getKey=0) or the newly generated key value that was created as a result of the update (getKey=1). Not all databases support automatic retrieval of generated keys.

  Boolean skipAudit - Optional, A flag which, if set to true, will cause the prep update to skip the audit system. Useful for some queries that have fields which won't fit into the audit log.

- Returns

  Integer - The number of rows affected by the query, or the key value that was generated, depending on the value of the getKey flag.

- Scope

  Gateway

Syntax

system.db.runPrepUpdate(query, args, [database], [tx], [getKey], [skipAudit])

- Parameters

  String query - A query (typically an UPDATE, INSERT, or DELETE) to run as a prepared statement with placeholders (?) denoting where the arguments go.

  Object[] args - A list of arguments. Will be used in order to match each placeholder (?) found in the query.

  String database - Optional, The name of the database connection to execute against. If omitted or "", the project's default database connection will be used.

  String tx - Optional, A transaction identifier. If omitted, the update will be executed in its own transaction.

  Boolean getKey - Optional, A flag indicating whether or not the result should be the number of rows returned (getKey=0) or the newly generated key value that was created as a result of the update (getKey=1). Not all databases support automatic retrieval of generated keys.

  Boolean skipAudit - Optional, A flag which, if set to true, will cause the prep update to skip the audit system. Useful for some queries that have fields which won't fit into the audit log.

- Returns

  Integer - The number of rows affected by the query, or the key value that was generated, depending on the value of the getKey flag.

- Scope

  Vision Client

Code Examples

Code Snippet

```
#This example would gather some user entered text and insert it into the database.

userText = event.source.parent.getComponent("TextArea").text
userName = system.security.getUsername()
system.db.runPrepUpdate("INSERT INTO Comments (Name, UserComment) VALUES (?,?)", [userName, userText])
```

Code Snippet

```
#This example would gather some user entered text and insert it into the database.
#The difference between this example and the previous example is that this example is explicitly
declaring which database connection to run the query againt.
#Sometimes you need to run a query against a database connection that is not the default connection.

userText = event.source.parent.getComponent("TextArea").text
userName = system.security.getUsername()
databaseConnection = "AlternateDatabase"
system.db.runPrepUpdate("INSERT INTO Comments (Name, UserComment) VALUES (?,?)", [userName, userText],
databaseConnection)
```

Code Snippet

```
#This code would read a file and upload it to the database

filename = system.file.openFile() # Ask the user to open a file
if filename != None:
   filedata = system.file.readFileAsBytes(filename)
   system.db.runPrepUpdate("INSERT INTO Files (file_data) VALUES (?)", [filedata])
```

Code Snippet

```
#This example inserts a new user and gives it the 'admin' role. Demonstrates the ability to retrieve a
newly created key value.

#get the username/password
name = event.source.parent.getComponent('Name').text
desc = event.source.parent.getComponent('Description').text
building = event.source.parent.getComponent('Building').selectedValue

#insert the value
id = system.db.runPrepUpdate("INSERT INTO machines (machine_name, description) VALUES (?, ?)", [name,
desc], getKey=1)

#add a row to the user role mapping table
system.db.runPrepUpdate("INSERT INTO machine_building_mapping (machine_id, building) VALUES (?, ?)", [id,
building])
```

system.db.runQuery

Description

Runs a SQL query, usually a SELECT query, against a database, returning the results as a dataset. If no database is specified, or the database is the empty-string "", then the current project's default database connection will be used. The results are returned as a PyDataSet, which is a wrapper around the standard dataset that is convenient for scripting.

Syntax

system.db.runQuery(query, database, tx)

- Parameters

  String query - A SQL query, usually a SELECT query, to run.

  String database - The name of the database connection to execute against.

  String tx - A transaction identifier. If omitted, the query will be executed in its own transaction.

- Returns

  PyDataSet - The results of the query as a PyDataSet.

- Scope

  Gateway

Syntax

system.db.runQuery(query, database, tx)

- Parameters

  String query - A SQL query, usually a SELECT query, to run.

  String database - The name of the database connection to execute against. If omitted or "", the project's default database connection will be used.

  String tx - A transaction identifier. If omitted, the query will be executed in its own transaction.

- Returns

  PyDataSet - The results of the query as a PyDataSet.

- Scope

  Vision Client

Code Examples

Assuming the following dataset:

ID	Value
1	3.55
2	67.2
3	9.87

If you executed the following code:

Code Snippet

```
table = system.db.runQuery("SELECT * FROM TEST")
```

Table[2] would access the third row (rows are zero-indexed), and both table[2][0] and table[2]["ID"] would access the ID value of the third row. As further example of how to use the results of runQuery, here are seven different ways to print out the table, and their results follow. Note that some of the later methods exercise some more advanced Jython concepts such as list comprehensions and string formatting, but their intent should be obvious. Generally speaking, the more concise Jython code becomes, the more readable it is.

Code Snippet

```
table = system.db.runQuery("SELECT * FROM Test")

print "Printing TEST Method 1..."
for row in table:
   for col in row:
      print col,
   print ""
print ""

print "Printing TEST Method 2..."
for row in table:
   print row[0], row[1]
print ""

print "Printing TEST Method 3..."
for row in table:
   print row["ID"], row["VALUE"]
print ""

print "Printing TEST Method 4..."
for rowIdx in range(len(table)):
   print "Row ",str(rowIdx)+": ", table[rowIdx][0], table[rowIdx][1]
print ""

print "Printing TEST Method 5..."
print [str(row[0])+", "+ str(row[1]) for row in table]
print ""

print "Printing TEST Method 6..."
print ["%s, %s" % (row["ID"],row["VALUE"]) for row in table]
print ""

print "Printing TEST Method 7..."
print [[col for col in row] for row in table]
print ""
```

The result would be:

Printing TEST Method 1...

0 3.55

1 67.2

2 9.87

Printing TEST Method 2...

0 3.55

1 67.2

2 9.87

Printing TEST Method 3...

0 3.55

1 67.2

2 9.87

Printing TEST Method 4...

Row 0: 0 3.55

Row 1: 1 67.2

Row 2: 2 9.87

Printing TEST Method 5...

['0, 3.55', '1, 67.2', '2, 9.87']

Printing TEST Method 6...

['0, 3.55', '1, 67.2', '2, 9.87']

Printing TEST Method 7...

[[0, 3.55], [1, 67.2], [2, 9.87]]

system.db.runScalarPrepQuery

Description

Runs a prepared statement against a database connection just like the runPrepQuery function, but only returns the value from the first row and column. If no results are returned from the query, the special value None is returned.

Syntax

system.db.runScalarPrepQuery(query, args, database, tx)

- Parameters

  String query - A SQL query (typically a SELECT) to run as a prepared statement with placeholders (?) denoting where the arguments go, that should be designed to return one row and one column.

  Object[] args - A list of arguments. Will be used in order to match each placeholder (?) found in the query.

  String database - The name of the database connection to execute against.

  String tx - A transaction identifier. If omitted, the query will be executed in its own transaction.

- Returns

  Object - The value from the first row and first column of the results. Returns None if no rows were returned.

- Scope

  Gateway

Syntax

system.db.runScalarPrepQuery(query, args, database, tx)

- Parameters

  String query - A SQL query (typically a SELECT) to run as a prepared statement with placeholders (?) denoting where the arguments go, that should be designed to return one row and one column.

  Object[] args - A list of arguments. Will be used in order to match each placeholder (?) found in the query.

  String database - The name of the database connection to execute against. If omitted or "", the project's default database connection will be used.

  String tx - A transaction identifier. If omitted, the query will be executed in its own transaction.

- Returns

  Object - The value from the first row and first column of the results. Returns None if no rows were returned.

- Scope

  Vision Client

Code Examples

There are no examples associated with this scripting function.

system.db.runScalarQuery

Description

Runs a query against a database connection just like the runQuery function, but only returns the value from the first row and column. If no results are returned from the query, the special value None is returned.

Syntax

system.db.runScalarQuery(query, database, tx)

- Parameters

  String query - A SQL query that should be designed to return one row and one column.

  String database - The name of the database connection to execute against.

  String tx - A transaction identifier. If omitted, the query will be executed in its own transaction.

- Returns

  Object - The value from the first row and first column of the results. Returns None if no rows were returned.

- Scope

  Gateway

Syntax

system.db.runScalarQuery(query, database, tx)

- Parameters

  String query - A SQL query that should be designed to return one row and one column.

  String database - The name of the database connection to execute against. If omitted or "", the project's default database connection will be used.

  String tx - A transaction identifier. If omitted, the query will be executed in its own transaction.

- Returns

  Object - The value from the first row and first column of the results. Returns None if no rows were returned.

- Scope

  Vision Client

Code Examples

Code Snippet

```
# This code would count the number of active alarms, and acknowledge them all if there is at least one.
numAlarms = system.db.runScalarQuery("SELECT COUNT(*) FROM alarmstatus " + "WHERE unacknowledged = 1")
if numAlarms > 0:
   # There are alarms - acknowledge all of them
   system.db.runUpdateQuery("UPDATE alarmstatus SET unacknowledged = 0")
```

Code Snippet

```
#This code would read a single value from a table and show it to the user an a popup box.
level = system.db.runScalarQuery("SELECT Level FROM LakeInfo WHERE LakeId='Tahoe'")
system.gui.messageBox("The lake level is: %d feet" % level)
lakeLevel = system.db.runScalarQuery(query)
system.gui.messageBox("The lake level is: %d feet" % lakeLevel)
```

system.db.runSFPrepUpdate

Description

Runs a prepared statement query through the store and forward system and to multiple datasources at the same time. Prepared statements differ from regular queries in that they can use a special placeholder, the question-mark character (?) in the query where any dynamic arguments would go, and then use an array of values to provide real information for those arguments. Make sure that the length of your argument array matches the number of question-mark placeholders in your query. This call should be used for UPDATE, INSERT, and DELETE queries.

This is extremely useful for two purposes:

- This method avoids the problematic technique of concatenating user input inside of a query, which can lead to syntax errors, or worse, a nasty security problem called a SQL injection attack. For example, if you have a user-supplied string that is used in a WHERE clause, you use single-quotes to enclose the string to make the query valid. What happens in the user has a single-quote in their text? Your query will fail. Prepared statements are immune to this problem.
- This is the only way to write an INSERT or UPDATE query that has binary or BLOB data. Using BLOBs can be very handy for storing images or reports in the database, where all clients have access to them.

Syntax

system.db.runSFPrepUpdate(query, args, datasources)

- Parameters

  String query - A query (typically an UPDATE, INSERT, or DELETE) to run as a prepared statement, with placeholders (?) denoting where the arguments go.

  Object[] args - A list of arguments. Will be used in order to match each placeholder (?) found in the query.

  String[] datasources - List of datasources to run the query through.

- Returns

  boolean - Returns true if successfully sent to store-and-forward system.

- Scope

  All

Code Examples

Code Snippet

```
#Example 1: Run through single datasource
print system.db.runSFPrepUpdate("INSERT INTO recipes (name, sp1, sp2, sp3) VALUES (?,?,?,?)", ['A Name',
1032, 234, 1], datasources=["MySQLDatasource"])
```

Code Snippet

```
#Example 2: Run through two datasources
print system.db.runSFPrepUpdate("INSERT INTO recipes (name, sp1, sp2, sp3) VALUES (?,?,?,?)", ['A Name',
1032, 234, 1], datasources=["MySQLDatasource", "SQLServerDatasource"])
```

system.db.runSFUpdateQuery

Description

Runs an query through the store and forward system and to multiple datasources at the same time.

Syntax

system.db.runSFUpdateQuery(query, datasources)

- Parameters

  String query - A query (typically an UPDATE, INSERT, or DELETE) to run.

  String[] datasources - List of datasources to run the query through.

- Returns

  Boolean - Returns true if successful and false if not.

- Scope

  All

Code Examples

Code Snippet

```
#Example 1: Run through single datasource
print system.db.runSFUpdateQuery("INSERT INTO recipes (name, sp1, sp2, sp3) VALUES ('A Name', 1032, 234,
1)", ["MySQLDatasource"])
```

Code Snippet

```
#Example 2: Run through 2 datasources
print system.db.runSFUpdateQuery("INSERT INTO recipes (name, sp1, sp2, sp3) VALUES ('A Name', 1032, 234,
1)", ["MySQLDatasource", "SQLServerDatasource"])
```

system.db.runUpdateQuery

Description

Runs a query against a database connection, returning the number of rows affected. Typically this is an UPDATE, INSERT, or DELETE query. If no database is specified, or the database is the empty-string "", then the current project's default database connection will be used.

Note that you may want to use the runPrepUpdate query if your query is constructed with user input (to avoid the user's input from breaking your syntax) or if you need to insert binary or BLOB data.

Syntax

system.db.runUpdateQuery(query, database, tx, getKey, skipAudit)

- Parameters

  String query - A SQL query, usually an INSERT, UPDATE, or DELETE query, to run.

  String database - The name of the database connection to execute against.

  String tx - A transaction identifier. If omitted, the update will be executed in its own transaction.

  Boolean getKey - A flag indicating whether or not the result should be the number of rows returned (getKey=0) or the newly generated key value that was created as a result of the update (getKey=1). Not all databases support automatic retrieval of generated keys.

  Boolean skipAudit - A flag which, if set to true, will cause the update query to skip the audit system. Useful for some queries that have fields which won't fit into the audit log.

- Returns

  Integer - The number of rows affected by the query, or the key value that was generated, depending on the value of the getKey flag.

- Scope

  Gateway

Syntax

system.db.runUpdateQuery(query, database, tx, getKey, skipAudit)

- Parameters

  String query - A SQL query, usually an INSERT, UPDATE, or DELETE query, to run.

  String database - The name of the database connection to execute against. If omitted or "", the project's default database connection will be used.

  String tx - A transaction identifier. If omitted, the update will be executed in its own transaction.

  Boolean getKey - A flag indicating whether or not the result should be the number of rows returned (getKey=0) or the newly generated key value that was created as a result of the update (getKey=1). Not all databases support automatic retrieval of generated keys.

  Boolean skipAudit - A flag which, if set to true, will cause the update query to skip the audit system. Useful for some queries that have fields which won't fit into the audit log.

- Returns

  Integer - The number of rows affected by the query, or the key value that was generated, depending on the value of the getKey flag.

- Scope

  Vision Client

Code Examples

Code Snippet

```
#This code would acknowledge all unacknowledged alarms # and show the user how many alarms were
acknowledged.
rowsChanged = system.db.runUpdateQuery("UPDATE alarmstatus SET unacknowledged = 0")
system.gui.messageBox("Acknowledged %d alarms" % rowsChanged)
```

Code Snippet

```
#This code would insert a new recipe step into a recipe table, after asking the user how many gallons of
syrup should be added on this recipe step.
inputText = system.db.inputBox("How many gallons?", "12.3")
# Make sure the user didn't hit cancel
if inputText != None:
   # Make sure the input is a number
   gallons = float(inputText)
   # Detect the next step number by adding 1 to the last step number
   nextStepNum = system.db.runScalarQuery("SELECT MAX(StepNum) + 1 FROM RecipeSteps")
   # Insert recipe step
   system.db.runUpdateQuery("INSERT INTO RecipeSteps (StepNum, Gallons) VALUES (%d, %f)" % (nextStepNum,
gallons))
   insertQuery = "INSERT INTO RecipeSteps (StepNum, Gallons) VALUES (%d, %f)"
   system.db.runUpdateQuery(insertQuery % (nextStepNum, gallons))
```

Code Snippet

```
#This example inserts a new user and gives it the 'admin' role.  Demonstrates the ability to retrieve a
newly created key value.
#get the username/password
name = event.source.parent.getComponent('Name').text
desc = event.source.parent.getComponent('Description').text
building = event.source.parent.getComponent('Building').selectedValue

#insert the value
id = system.db.runUpdateQuery("INSERT INTO machines (machine_name, description) " + "VALUES ('%s', '%s')"
%(name, desc), getKey=1)

#add a row to the user role mapping table
system.db.runUpdateQuery("INSERT INTO machine_building_mapping " + "(machine_id, building) VALUES (%d, %
d)" %(id, building))
```

system.db.setDatasourceConnectURL

Description

Changes the connect URL for a given database connection.

Syntax

system.db.setDatasourceConnectURL(name, connectUrl)

- Parameters

  String name - The name of the database connection in Ignition.

  String connectUrl - The new connect URL.

- Returns

  nothing

- Scope

  All

Code Examples

Code Snippet

```
#Example 1:

system.db.setDatasourceConnectURL("MySQL", "jdbc:mysql://localhost:3306/test")
```

system.db.setDatasourceEnabled

Description

Enables/disables a given database connection.

Syntax

system.db.setDatasourceEnabled(name, enabled)

- Parameters

  String name - The name of the database connection in Ignition.

  Boolean enabled

- Returns

  nothing

- Scope

  All

Code Examples

Code Snippet

```
#Example 1: Enable a connection

system.db.setDatasourceEnabled("MySQL", 1)
```

Code Snippet

```
#Example 2: Disable a connection

system.db.setDatasourceEnabled("MySQL", 0)
```

system.db.setDatasourceMaxConnections

Description

Sets the *Max Active* and *Max Idle* parameters of a given database connection.

Syntax

system.db.setDatasourceMaxConnections(name, maxConnections)

- Parameters

  String name - The name of the database connection in Ignition.

  Integer maxConnections

- Returns

  nothing

- Scope

  All

Code Examples

Code Snippet

```
#Example 1: Enable a connection
system.db.setDatasourceMaxConnections("MySQL", 20)
```

system.device

system.device.addDevice

The following feature is new in Ignition version **7.8.0**
Click here to check out the other new features

Description

Adds a new device connection in Ignition. Accepts a dictionary of parameters to configure the connection. Acceptable parameters differ by device type: i.e., a Modbus/TCP connection requires a hostname and port, but a simulator doesn't require any parameters.

When using this function, the arguments **MUST** be passed as keyword arguments.

Syntax - Using deviceType

system.device. addDevice(deviceType, deviceName, deviceProps)

- Parameters

  String driverType - The device driver type. Possible values are listed in the Device Types table below.

  String deviceName - The name that will be given to the the new device connection.

  PyDictionary deviceProps - A dictionary of device connection properties and values. Each deviceType has different properties, but most require at least a hostname. Keys in the dictionary are **case-insensitive**, spaces are omitted, and the names of the properties that appear when manually creating a device connection.

- Returns

  nothing

- Scope

  All

Device Types

Note that this function may be called to add devices using 3rd party drivers: you simply need the driver type, which the module developer will be able to provide.

Driver Name	Driver Type
Legacy Allen-Bradley CompactLogix	CompactLogix
Legacy Allen-Bradley ControlLogix	ControlLogix
Simulators Dairy Demo Simulator	DairyDemoSimulator
DNP3 Driver	Dnp3Driver
Allen-Bradley Logix Driver	LogixDriver
Allen-Bradley MicroLogix	MicroLogix
Modbus RTU	ModbusRtu
Modbus TCP	ModbusTcp
Omron NJ Driver	com.inductiveautomation.omron.NjDriver
Allen-Bradley PLC5	PLC5

Driver Name	Driver Type
Siemens S7-300	S7300
Siemens S7-400	S7400
Siemens S7-1200	S71200
Siemens S7-1500	S71500
Allen-Bradley SLC	SLC
Simulators SLC Simulator	SLCSimulator
Simulators Generic Simulator	Simulator
TCP Driver	TCPDriver
UDP Driver	UDPDriver

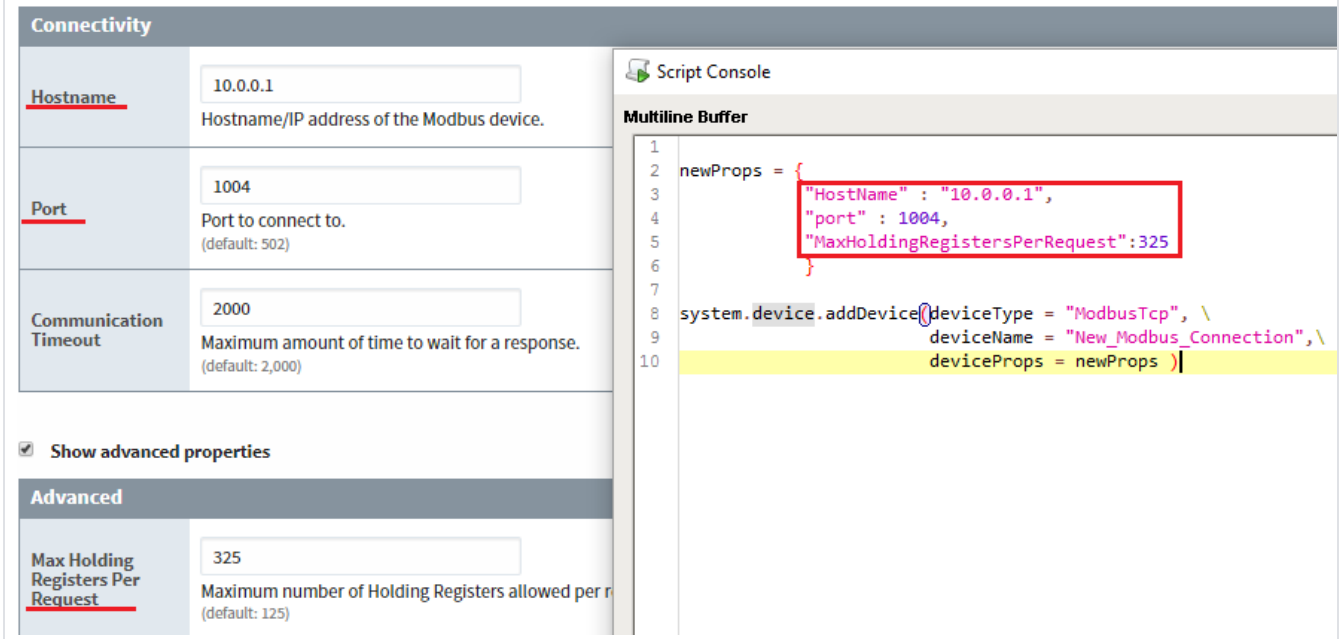
Device Properties

The `deviceProps` parameter is where you supply configuration values to the new connection. Value properties depend on which `deviceType` was specified. To find the valid properties for each type, simply check the OPC-UA and Device Connections section of the manual, and find the property list. The keys in the `deviceProps` parameter are case-insensitive, and have the spaces omitted. Device properties not specified in the `deviceProps` parameter will fallback to default values if not specified (where applicable: i.e., "hostname" typically does not have a default value).

Modbus Example

Below we see an example using the ModbusTcp `deviceType`. The left portion is from the newly created device connection on the Gateway, and the right portion is the function that created the connection. The **Hostname**, **Port**, and **Max Holding Registers Per Request** properties were specified in the call. Note the following:

- `port` in the dictionary does not have to match the casing of the **Port** property on the device page.
- The **Communication Timeout** property was not specified in the dictionary, so it is set to the default value of 2000.
- **Max Holding Registers Per Request** was included in the dictionary with all spaces removed, and successfully configured with a value of 325.



Code Examples

Code Snippet

```
#Below is an example of creating a new Generic Simulator device connection.
#Note that we MUST pass a dictionary as the 3rd parameter, even if it's empty.

#Call the function
system.device.addDevice(deviceType = "Simulator", deviceName = "New_Generic_Simulator", deviceProps = {} )
```

Code Snippet

```
#Add a device using the Allen-Bradley Logix Driver for firmware v21+ devices
deviceProps = {}
deviceProps["Hostname"] = "192.168.1.2"
system.device.addDevice(deviceName="Test1", deviceType="LogixDriver", deviceProps=deviceProps)
```

Code Snippet

```
#Below is an example of creating a new S7-1500 device connection.

#Build a Dictionary of parameters
newProps = {
                        "HostName" : "10.0.0.1",
                        "Port" : 102 #<---If adding additional parameters, make sure to add a comma.
                        }

#Call the function
system.device.addDevice(deviceType = "S71500", \
                                              deviceName = "My_S7_1500_Device",\
                                              deviceProps = newProps )
```

system.device.listDevices

Description

Returns a dataset of information about each configured device. Each row represents a single device.

Syntax

system.device.listDevices()

- Parameters

  none

- Returns

  Dataset - A dataset, where each row represents a device. Contains 4 columns *Name*, *Enabled*, *State*, and *Driver*.

- Scope

  All

Code Examples

Code Snippet

```
//Use the following expression binding on a table to show the list of devices. The binding polls every minute.

runScript("system.device.listDevices()", 60000)
```

system.device.refreshBrowse

Description

Forces Ignition to browse the controller. Only works for Allen-Bradley controllers.

Syntax

system.device.refreshBrowse(deviceName)

- Parameters

  String deviceName - The name of the device in Ignition.

- Returns

  nothing

- Scope

  All

Code Examples

Code Snippet

```
#Example 1:
system.device.refreshBrowse("CLX")
```

system.device.removeDevice

Description

Removes a given device from Ignition.

Syntax

system.device.removeDevice(deviceName)

- Parameters

  String deviceName - The name of the device in Ignition.

- Returns

  nothing

- Scope

  All

Code Examples

Code Snippet

```
#Example 1:
system.device.removeDevice("CLX")
```

system.device.setDeviceEnabled

Description

Enables/disables a device in Ignition.

Syntax

system.device.setDeviceEnabled(deviceName, enabled)

- Parameters

  String deviceName - The name of the device in Ignition.

  Boolean enabled

- Returns

  nothing

- Scope

  All

Code Examples

Code Snippet

```
#Example 1: Enable a device

system.device.setDeviceEnabled("CLX", 1)
```

Code Snippet

```
#Example 2: Disable a device

system.device.setDeviceEnabled("CLX", 0)
```

system.device.setDeviceHostname

Description

Changes the hostname of a device. Used for all ethernet based drivers.

Syntax

system.device.setDeviceHostname(deviceName, hostname)

- Parameters

  String deviceName - The name of the device in Ignition.

  String hostname - The new IP address or hostname.

- Returns

  nothing

- Scope

  All

Code Examples

Code Snippet

```
#Example 1:

system.device.setDeviceHostname("CLX", "10.10.1.20")
```

system.dnp3

Constants

`system.dnp3.NUL = 0`

`system.dnp3.PULSE_ON = 1`

`system.dnp3.PULSE_OFF = 2`

`system.dnp3.LATCH_ON = 3`

`system.dnp3.LATCH_OFF = 4`

`system.dnp3.CLOSE = 1`

`system.dnp3.TRIP = 2`

Scripting Functions

system.dnp3.directOperateAnalog

Description

Issues a Select-And-Operate command to set an analog value in an analog output point.

Syntax

system.dnp3.directOperateAnalog(deviceName, index, value, [variation])

- Parameters

  String deviceName-The name of the DNP3 device driver.

  Integer index-The index of the object to be modified in the outstation.

  Numeric value-The analog value that is requested (of type int, short, float, or double).

  Integer variation-The DNP3 object variation to use in the request.

- Returns

  The DNP3 status code of the response, as an integer.

- Scope

  All

Code Examples

```
# This example shows setting the analog output at index 0 to the
#   Double value 3.14

system.dnp3.directOperateAnalog("Dnp3", 0, 3.14)


# This example shows setting the analog output at index 2 to the
#   Integer value 300

system.dnp3.directOperateAnalog("Dnp3", 2, 300)


# This example shows setting the analog output at index 15 to the
#   Short value 33.  The value sent in the request is converted
#   for the object variation, 2.

system.dnp3.directOperateAnalog("Dnp3", 15, 33.3333, variation=2)


# This example shows setting the analog output at index 1 to the
#   Float value 15.0.  The value sent in the request is converted
#   for the object variation, 3.

system.dnp3.directOperateAnalog("Dnp3", index=1, value=15, variation=3)
```

system.dnp3.directOperateBinary

Description

Issues a Direct-Operate command for digital control operations at binary output points (CROB).

Syntax

system.dnp3.directOperateBinary(deviceName, indexes, opType, tcCode, count, onTime, offTime)

- Parameters

  String deviceName-The name of the DNP3 device driver.

  List indexes-A list of indexes of the objects to be modified in the outstation.

  Integer opType-The type of the operation. 0=NUL, 1=PULSE_ON, 2=PULSE_OFF, 3=LATCH_ON, 4=LATCH_OFF

  Integer tcCode-The Trip-Close code, used in conjunction with the opType. 0=NUL, 1=CLOSE, 2=TRIP

  Integer count-The number of times the outstation shall execute the operation.

  Long onTime-The duration that the output drive remains active, in millis.

  Long offTime-The duration that the output drive remains non-active, in millis.

- Returns

  The DNP3 status code of the response, as an integer.

- Scope

  All

Code Examples

```
# This example shows latching off 3 binary output points with the Direct-Operate command.

off = system.dnp3.LATCH_OFF
system.dnp3.directOperateBinary("Dnp3", [0, 1, 2], off)


# This example shows setting a binary output point at index 3 to pulse at 5 second intervals
#   with the Direct-Operate command.

pulse = system.dnp3.PULSE_OFF
trip = system.dnp3.TRIP
system.dnp3.directOperateBinary("Dnp3", [3], pulse, trip, onTime=5000, offTime=5000)
```

system.dnp3.freezeAnalogs

Description

Issues a freeze command on the given analog outputs.

Syntax

system.dnp3.freezeAnalogs(deviceName, [indexes])

- Parameters

  `String deviceName` - The name of the DNP3 device driver.

  `List indexes` - An optional list of specific indexes on which to issue the freeze command.

- Returns

  Nothing

- Scope

  All

Code Examples

```
# This example shows a request to freeze all analog inputs in the outstation.
system.dnp3.freezeAnalogs("Dnp3")

# This example shows a request to freeze analog inputs at indexes 1, 3, and 5.
system.dnp3.freezeAnalogs("Dnp3", [1, 3, 5])
```

system.dnp3.freezeAnalogsAtTime

Description

Issues a freeze command on the given analog outputs at the given time for the specified duration.

Syntax

system.dnp3.freezeAnalogsAtTime(deviceName, absoluteTime, intervalTime, indexes)

- Parameters

  `String` `deviceName` - The name of the DNP3 device driver.

  `Integer` `absoluteTime` - The absolute time at which to freeze, in millis.

  `Integer` `intervalTime` - The interval at which to periodically freeze, in millis.

  `List` `indexes` - A list of specific indexes on which to issue the freeze command. An empty list will freeze all analogs.

- Returns

  Nothing

- Scope

  All

Code Examples

```
# This example shows a request to freeze analog inputs at indexes 2 and 4,
#  5 minutes from the current time, with no interval.
from time import *

fiveMikes = (60 * 1000 * 5) + int(time() * 1000) #ms
system.dnp3.freezeAnalogsAtTime("Dnp3", fiveMikes, 0, [2, 4])
```

system.dnp3.freezeCounters

Description

Issues a freeze command on the given counters.

Syntax

system.dnp3.freezeCounters(deviceName, [indexes])

- Parameters

  `String deviceName` - The name of the DNP3 device driver.

  `List indexes` - An optional list of specific indexes on which to issue the freeze command.

- Returns

  Nothing

- Scope

  All

Code Examples

```
# This example shows a request to freeze all counters in the outstation.
system.dnp3.freezeCounters("Dnp3")

# This example shows a request to freeze counters at indexes 1, 3, and 5.
system.dnp3.freezCounters("Dnp3", [1, 3, 5])
```

system.dnp3.freezeCountersAtTime

Description

Issues a freeze command on the given counters at the given time for the specified duration.

Syntax

system.dnp3.freezeCountersAtTime(deviceName, absoluteTime, intervalTime, indexes)

- Parameters

  `String` `deviceName` - The name of the DNP3 device driver.

  `Integer` `absoluteTime` - The absolute time at which to freeze, in millis.

  `Integer` `intervalTime` - The interval at which to periodically freeze, in millis.

  `List` `indexes` - A list of specific indexes on which to issue the freeze command. An empty list will freeze all counters.

- Returns

  Nothing

- Scope

  All

Code Examples

```
# This example shows a request to freeze counters at indexes 2 and 4,
#  5 minutes from the current time, with no interval.
from time import *

fiveMikes = (60 * 1000 * 5) + int(time() * 1000) #ms
system.dnp3.freezeCountersAtTime("Dnp3", fiveMikes, 0, [2, 4])
```

system.dnp3.selectOperateAnalog

Description

Issues a Select-And-Operate command to set an analog value in an analog output point.

Syntax

system.dnp3.selectOperateAnalog(deviceName, index, value, [variation])

- Parameters

  String deviceName - The name of the DNP3 device driver.

  Integer index - The index of the object to be modified in the outstation.

  Numeric value - The analog value that is requested (of type int, short, float, or double).

  Integer variation - The DNP3 object variation to use in the request.

- Returns

  The DNP3 status code of the response, as an integer.

- Scope

  All

Code Examples

```
# This example shows setting the analog output at index 0 to the
#   Double value 3.14

system.dnp3.selectOperateAnalog("Dnp3", 0, 3.14)


# This example shows setting the analog output at index 2 to the
#   Integer value 300

system.dnp3.selectOperateAnalog("Dnp3", 2, 300)


# This example shows setting the analog output at index 15 to the
#   Short value 33.  The value sent in the request is converted
#   for the object variation, 2.

system.dnp3.selectOperateAnalog("Dnp3", 15, 33.3333, variation=2)


# This example shows setting the analog output at index 1 to the
#   Float value 15.0.  The value sent in the request is converted
#   for the object variation, 3.

system.dnp3.selectOperateAnalog("Dnp3", index=1, value=15, variation=3)
```

system.dnp3.selectOperateBinary

Description

Issues a Select-And-Operate command for digital control operations at binary output points (CROB).

Syntax

system.dnp3.selectOperateBinary(deviceName, indexes, opType, tcCode, count, onTime, offTime)

- Parameters

  String deviceName - The name of the DNP3 device driver.

  List indexes - A list of indexes of the objects to be modified in the outstation.

  Integer opType - The type of operation. 0=NUL, 1=PULSE_ON, 2=PULSE_OFF, 3=LATCH_ON, 4=LATCH_OFF

  Integer tcCode - The Trip-Close code, used in conjunction with the opType. 0=NUL, 1=CLOSE, 2=TRIP

  Integer count - The number of times the outstation shall execute the operation.

  Integer onTime - The duration that the output drive remains active, in millis.

  Integer offTime - The duration that the output drive remains non-active, in millis.

- Returns

  The DNP3 status code of the response, as an integer.

- Scope

  All

Code Examples

```
# This example shows latching on 3 binary output points with the Select-And-Operate command.

latch = system.dnp3.LATCH_ON
system.dnp3.selectOperateBinary("Dnp3", [0, 1, 2], latch)


# This example shows setting a binary output point at index 3 to pulse at 5 second intervals
# with the Select-And-Operate command.

pulse = system.dnp3.PULSE_ON
trip = system.dnp3.TRIP
system.dnp3.selectOperateBinary("Dnp3", [3], pulse, trip, count=2, onTime=5000, offTime=5000)
```

system.eam

system.eam.getGroups

Description

Returns the names of the defined agent organizational groups in the Gateway.

Syntax

system.eam.getGroups()

Parameters

- none

Returns

- A string list of group names

Scope

- All

Examples

Code Snippet

```
groups = system.eam.getGroups()
for group in groups:
        print group
```

system.eam.queryAgentHistory

Description

Returns a list of the most recent agent events

Syntax

system.eam.getAgentEvents(groupIds, agentIds, startDate, endDate, limit)

- Parameters

  List groupIds - A list of groups to restrict the results to. If not specified, all groups will be included.

  List agentIds - A list of agent ids to restrict the results to. If not specified, all agents will be allowed.

  Date startDate - The starting time for history events. If null, defaults to 8 hours previous to now.

  Date endDate - The ending time for the query range. If null, defaults to "now".

  int limit - The limit of results to return. Defaults to 100. A value of 0 means "no limit".

- Returns

  Dataset - A list of agent events, arranged by time ascending.

- Scope

  All

Examples

Code Snippet

```
results=system.eam.queryAgentHistory()
for row in range(results.rowCount):
        eventId=results.getValueAt(row, "id")
        agentName=results.getValueAt(row, "agent_name")
        agentRole=results.getValueAt(row, "agent_role")
        eventTime=results.getValueAt(row, "event_time")
        eventCategory=results.getValueAt(row, "event_category")
        eventType=results.getValueAt(row, "event_type")
        eventSource=results.getValueAt(row, "event_source")
        eventLevel=results.getValueAt(row, "event_level")
        eventLevelInt=results.getValueAt(row, "event_level_int")
        message=results.getValueAt(row, "message")
```

system.eam.queryAgentStatus

Description

Returns the current state of the matching agents.

Syntax

system.eam.queryAgentStatus(groupIds, agentIds, isConnected)

- Parameters

  List groupIds - A list of groups to restrict the results to. If not specified, all groups will be included.

  List agentIds - A list of agent ids to restrict the results to. If not specified, all agents will be allowed.

  Boolean isConnected - If True, only returns agents that are currently connected. If False, only agents that are considered down will be returned, and if not specified, all agents will be returned.

- Returns

  Dataset - A list of AgentStatus objects.

- Scope

  All

Examples

Code Snippet

```
results=system.eam.queryAgentStatus()
for row in range(results.rowCount):
        agentName=results.getValueAt(row, "AgentName")
        nodeRole=results.getValueAt(row, "NodeRole")
        agentGroup=results.getValueAt(row, "AgentGroup")
        lastComm=results.getValueAt(row, "LastCommunication")
        isConnected=results.getValueAt(row, "IsConnected")
        isRunning=results.getValueAt(row, "IsRunning")
        runningState=results.getValueAt(row, "RunningState")
        runningStateInt=results.getValueAt(row, "RunningStateInt")
        licenseKey=results.getValueAt(row, "LicenseKey")
        platformVersion=results.getValueAt(row, "Version")
```

system.file

system.file.fileExists

Description

Checks to see if a file or folder at a given path exists.

Syntax

system.file.fileExists(filepath)

- Parameters

  String filepath - The path of the file or folder to check.

- Returns

  boolean - True (1) if the file/folder exists, false (0) otherwise.

- Scope

  All

Code Examples

Code Snippet

```
#This basic example shows how the fileExists function is used in its simplest form:
if system.file.fileExists("C:\\temp_file.txt"):
   system.gui.messageBox("Yes, the file exists")
else:
   system.gui.messageBox("No, it doesn't exist")
```

Code Snippet

```
#This code uses the fileExists function, along with other system.file.* functions, to prompt the user to
confirm that they want to overwrite an existing file.
filename = system.file.saveFile(name)
if filename != None:
   reallyWrite = 1
   if system.file.fileExists(filename):
   overwriteMessage = "File '%s' already exists. Overwrite?"
      reallyWrite = system.gui.confirm(overwriteMessage % filename)
   if reallyWrite:
      system.file.writeFile(filename, "This will be the contents of my new file")
```

system.file.getTempFile

Description

Creates a new temp file on the host machine with a certain extension, returning the path to the file. The file is marked to be removed when the Java VM exits.

Syntax

system.file.getTempFile(extension)

- Parameters

  String extension - An extension, like ".txt", to append to the end of the temporary file.

- Returns

  String - The path to the newly created temp file.

- Scope

  All

Code Examples

Code Snippet

```
#This code writes some data to a temorary file, and then opens that file. Assume that the data variable
holds the contents of an excel (xls) file.

filename = system.file.getTempFile("xls")
system.file.writeFile(filename, data)
system.net.openURL("file://" + filename)
```

system.file.openFile

Description

Shows an "Open File" dialog box, prompting the user to choose a file to open. Returns the path to the file that the user chose, or None if the user canceled the dialog box. An extension can optionally be passed in that sets the filetype filter to that extension.

Syntax

system.file.openFile([extension], [defaultLocation])

- Parameters

  String extension - A file extension, like "pdf", to try to open. [optional]

  String defaultLocation - A folder location, like "C:\MyFiles", to use as the default folder to store in. [optional] - Added in 7.8.1

- Returns

  String - The path to the selected file, or None if canceled.

- Scope

  All

Code Examples

Code Snippet

```
#This code would prompt the user to open a file of type 'gif'. If None is returned, it means the user
canceled the open dialog box.

path = system.file.openFile('gif')
if path != None:
   # do something with the file
```

Code Snippet

```
#This code would prompt the user to open a file of type 'pdf' from their stored documents folder. If None
is returned, it means the user canceled the open dialog box.
#Note: the computer running this code needs to have network access to the "fileserver" computer.
path = system.file.openFile('pdf', '\\fileserver\PDF_Storage')
if path != None:
   # do something with the file
```

system.file.readFileAsBytes

Description

Opens the file found at path filename, and reads the entire file. Returns the file as an array of bytes. Commonly this array of bytes is uploaded to a database table with a column of type BLOB (Binary Large OBject). This upload would be done through an INSERT or UPDATE SQL statement run through the system.db.runPrepUpdate function. You could also write the bytes to another file using thesystem.file.writeFile function, or send the bytes as an email attachment using system.net.sendEmail.

Syntax

system.file.readFileAsBytes(filepath)

- Parameters

  String filepath - The path of the file to read.

- Returns

  byte[] - The contents of the file as an array of bytes.

- Scope

  All

Code Examples

Code Snippet

```
#This code would prompt the user to choose a file. If the user chooses a file, it would then read that
file and upload it to a database table called Files into a BLOB column called file_data.

path = system.file.openFile()
if path != None:
   bytes = system.file.readFileAsBytes(filename)
   system.db.runPrepUpdate("INSERT INTO Files (file_data) VALUES (?)", [bytes])
```

system.file.readFileAsString

Description

Opens the file found at path filename, and reads the entire file. Returns the file as a string. Common things to do with this string would be to load it into the text property of a component, upload it to a database table, or save it to another file using system.file.writeFile function.

Syntax

system.file.readFileAsString(filepath)

- Parameters

  String filepath - The path of the file to read.

- Returns

  String - The contents of the file as a string.

- Scope

  All

Syntax

system.file.readFileAsString(filepath, encoding)

- Parameters

  String filepath - The path of the file to read.

  String encoding - The character encoding of the file to be read. Will throw an exception if the string does not represent a supported encoding. Common encodings are "UTF-8", "ISO-8859-1" and "US-ASCII".

- Returns

  String - The contents of the file as a string.

- Scope

  All

Code Examples

Code Snippet

```
#This code would prompt the user to choose a text file. If the user chooses a file, it would then set a
text area on the screen to display the file.

path = system.file.openFile("txt")
if path != None:
   contents = system.file.readFileAsString(path)
   event.source.parent.getComponent("Text Area").text = contents
```

system.file.saveFile

Description

Prompts the user to save a new file named filename. The optional extension and typeDesc arguments will be used for a file type filter, if any. If the user accepts the save, the path to that file will be returned. If the user cancels the save, None will be returned.

Syntax

system.file.saveFile(filename [, extension] [, typeDesc])

- Parameters

  String filename - A file name to suggest to the user.

  String extension - The appropriate file extension, like "jpeg", for the file. [optional]

  String typeDesc - A description of the extension, like "JPEG Image" [optional]

- Returns

  String - The path to the file that the user decided to save to, or None if they canceled.

- Scope

  All

Code Examples

Code Snippet

```
#This code would prompt the user to save the text in a text area to a file.

path = system.file.saveFile("myfile.txt")
if path != None:
   system.file.writeFile(path, event.source.parent.getComponent("Text Area").text)
```

system.file.writeFile

Description

Writes the given data to the file at file path filename. If the file exists, the append argument determines whether or not it is overwritten (the default) or appended to. The data argument can be either a string or an array of bytes (commonly retrieved from a BLOB in a database or read from another file using system.file.readFileAsBytes).

Syntax

system.file.writeFile(filepath, data [, append])

- Parameters

  String filepath - The path of the file to write to.

  byte[] data - The binary content to write to the file.

  boolean append - If true(1), the file will be appended to if it already exists. If false(0), the file will be overwritten if it exists. The default is false(0). [optional]

- Returns
- Scope

  All

Syntax

system.file.writeFile(filepath, charData [, append])

- Parameters

  String filepath - The path of the file to write to.

  String charData - The character content to write to the file.

  boolean append - If true(1), the file will be appended to if it already exists. If false(0), the file will be overwritten if it exists. The default is false(0). [optional]

- Returns
- Scope

  All

Code Examples

Code Snippet

```
#This code would download a BLOB from a database and save it to a file.

resultSet = system.db.runQuery("SELECT file_data FROM Files WHERE id=12")
   if len(resultSet) > 0: # if the query returned anything...
      data = resultSet[0][0] # grab the BLOB at the 0th row and 0th column
      filename = system.file.saveFile("MyDownloadedFile.xyz")
      if filename != None:
         system.file.writeFile(filename, data)
```

Code Snippet

```
#This code would write the contents of a text area to a file.

data = event.source.parent.getComponent("Text Area").text
filename = system.file.saveFile("MyDownloadedFile.txt")
if filename != None:
   system.file.writeFile(filename, data)
```

system.groups

system.groups.loadFromFile

Description

```
Loads the configuration from an xml export, into the specified project (creating the project if necessary).
The mode parameter dictates how overwrites occur.
```

Syntax

system.groups.loadFromFile(filePath, projectName, mode)

- Parameters

  String filePath - The path to a valid transaction group xml or csv file.

  String projectName - The name of the project to load into.

  int mode - How duplicates will be handled. 0 = Overwrite, 1 = Ignore, 2 = Replace the existing project with this one.

- Returns

  none

- Scope

  Gateway

system.groups.removeGroups

Description

```
Removes the specified groups from the project. The group paths are "Folder/Path/To/GroupName", separated by
forward slashes.
```

Syntax

system.groups.removeGroups(projectName, paths)

- Parameters

  String projectName - The project to remove from. If the project does not exist, throws an IllegalArgumentException

  Collection<String> paths - A collection of paths to remove. The group paths are "Folder/Path/To/GroupName", separated by forward slashes.

- Returns

  none

- Scope

  Gateway

system.gui

Constants

```
system.gui.ACCL_NONE = 0
system.gui.ACCL_CONSTANT = 1
system.gui.ACCL_FAST_TO_SLOW = 2
system.gui.ACCL_SLOW_TO_FAST = 3
system.gui.ACCLE_EASE = 4
system.gui.COORD_SCREEN = 0
system.gui.COORD_DESIGNER = 1
```

Scripting Functions

- system.gui.chooseColor
- system.gui.color
- system.gui.confirm
- system.gui.convertPointToScreen
- system.gui.createPopupMenu
- system.gui.errorBox
- system.gui.findWindow
- system.gui.getOpenedWindowNames
- system.gui.getOpenedWindows
- system.gui.getParentWindow
- system.gui.getScreens
- system.gui.getSibling
- system.gui.getWindow
- system.gui.getWindowNames
- system.gui.inputBox
- system.gui.isTouchscreenModeEnabled
- system.gui.messageBox
- system.gui.moveComponent
- system.gui.openDiagnostics
- system.gui.passwordBox
- system.gui.reshapeComponent
- system.gui.resizeComponent
- system.gui.setScreenIndex
- system.gui.setTouchscreenModeEnabled
- system.gui.showNumericKeypad
- system.gui.showTouchscreenKeyboard
- system.gui.transform
- system.gui.warningBox

system.gui.chooseColor

Description

Prompts the user to pick a color using the default color-chooser dialog box.

Syntax

system.gui.chooseColor(initialColor [, dialogTitle])

- Parameters

  Color initialColor - A color to use as a starting point in the color choosing popup.

  String dialogTitle - The title for the color choosing popup. Defaults to "Choose Color" [optional]

- Returns

  Color - The new color chosen by the user.

- Scope

  All

Code Examples

Code Snippet

```
#This code would be placed in the actionPerformed event of a button, and would change the background
color of the container the button was placed in.

parent = event.source.parent
newColor = system.gui.chooseColor(parent.background)
parent.background = newColor
```

system.gui.color

Description

Creates a new color object, either by parsing a string or by having the RGB[A] channels specified explicitly.

Syntax

system.gui.color(color)

- Parameters

  String color - A string that will be coerced into a color. Can accept many formats, such as "red" or "#FF0000" or "255,0,0"

- Returns

  Color - The newly created color.

- Scope

  All

Syntax

system.gui.color(red, green, blue [, alpha])

- Parameters

  int red - The red component of the color, an integer 0-255.

  int green - The green component of the color, an integer 0-255.

  int blue - The blue component of the color, an integer 0-255.

  int alpha - The alpha component of the color, an integer 0-255. [optional]

- Returns

  Color - The newly created color.

- Scope

  All

Code Examples

Code Snippet

```
#This example changes the background color of a component to red.

myComponent = event.source
myComponent.background = fpmi.gui.color(255,0,0) # turn the component red
```

system.gui.confirm

Description

Displays a confirmation dialog box to the user with "Yes" and "No" options, and a custom message.

Syntax

system.gui.confirm(message [, title] [, allowCancel])

- Parameters

  String message - The message to show in the confirmation dialog.

  String title - The title for the confirmation dialog. [optional]

  Boolean allowCancel - Show a cancel button in the dialog. [optional]

- Returns

  Boolean - True (1) if the user selected "Yes", false (0) if the user selected "No", None if the user selected "Cancel".

- Scope

  All

Code Examples

Code Snippet

```
#By using the confirm function in an if statement, we can let the user confirm an action. In this case,
we shut down the plant if the user confirms it, otherwise, we don't do anything.

if system.gui.confirm("Are you sure you want to shutdown the plant?",
 "Really Shutdown?"):
   system.db.runUpdateQuery("UPDATE ControlTable SET Shutdown=1")
```

system.gui.convertPointToScreen

Description

Converts a pair of coordinates that are relative to the upper-left corner of some component to be relative to the upper-left corner of the entire screen.

Syntax

system.gui.convertPointToScreen(x, y, event)

- Parameters

  int x - The X-coordinate, relative to the component that fired the event.

  int y - The Y-coordinate, relative to the component that fired the event.

  EventObject event - An event object for a component event.

- Returns

  PyTuple - A tuple of (x,y) in screen coordinates.

- Scope

  All

Code Examples

Code Snippet

```
#This example will get the coordinates where the mouse is (from the corner of the monitor) and display
them in a label.
#Get the screen coordinates of the pointer and write them to a label.
coords = system.gui.convertPointToScreen(event.x, event.y, event)
event.source.getComponent('Label').text = "x: %s y: %s" %(coords[0], coords[1])
```

system.gui.createPopupMenu

Description

Creates a new popup menu, which can then be shown over a component on a mouse event. To use this function, first create a Python sequence w hose entries are strings, and another sequence whose entries are function objects. The strings will be the items that are displayed in your popup menu, and when an item is clicked, its corresponding function will be run. Passing in a function of None will cause a separator line to appear in the popup menu, and the corresponding string will not be displayed. Your functions must accept an event object as an argument. See also: Functions to show the popup menu, store the menu object that this function returns, and then call its show(event) function, where event is the event object for a mousePressed or mouseReleased event on the component you wish the popup menu to be shown on. Best Practices. It is best to have the menu object created only once via an application specific library function. Then, call the show(event) function on both the mousePressed and mouseRele asedevents on your component. The reason for this is that different operating systems (Windows, Linux, MacOS) differ in when they like to show the popup menu. The show(event) function detects when the right time is to show itself, either on mouse press or release. See the examples for more.

Syntax

system.gui.createPopupMenu(itemNames, itemFunctions)

- Parameters

  PySequence itemNames - A list of names to create popup menu items with.

  PySequence itemFunctions - A list of functions to match up with the names.

- Returns

  JPopupMenu - The javax.swing.JPopupMenu that was created.

- Scope

  Client

Code Examples

Event Handlers

A popup menu must be created on either the mouse released or mouse pressed event handlers. This function is not appropriate for invoking on the property change event.

Also, the mouse motions that invoke the popup menu are dependent on the operating system and may behave differently depending on which button you press on the mouse. Because of the different popup-trigger settings on different operating systems, the example code may behave differently on a Linux or a Mac. The way around this is to do the same code in both themousePressed and mouseReleased events. In order to avoid code duplication, consider placing the code in a custom method.

Code Snippet

```
#This first example is a very basic to demonstrate the fundamentals of making a popup menu. Put the
following script in the mouseReleased event of a component.
#This will only work on Windows - continue on for cross-platform instructions.
#Right click on the component to see the resulting pop-up menu that is created with this code.

def sayHello(event):
   system.gui.messageBox("Hello World")
menu = system.gui.createPopupMenu(["Click Me"], [sayHello])
menu.show(event)
```

Code Snippet

```
#The following code demonstrates how to edit a component's custom property after you right clicked the
component.
#This code makes use of functions in order to edit the components custom properties.
#The following code should be located in the mouse released event handler.
#Also, there must be custom properties present on the component in order to handle these functions.
#For example, there must be a custom property called 'DatabaseProvider' that takes a string.
if event.button == event.BUTTON3:
        def editDatabaseProvider(event):
                result = system.gui.inputBox("Database Provider",event.source.parent.DatabaseProvider)
                event.source.parent.DatabaseProvider = result

        def editTable(event):
                result = system.gui.inputBox("Table Name",event.source.parent.Table)
                event.source.parent.Table = result

        def editColumn(event):
                result = system.gui.inputBox("Column Name",event.source.parent.Column)
                event.source.parent.Column = result

        def editKeyColumn(event):
                result = system.gui.inputBox("Key Column Name",event.source.parent.KeyColumn)
                event.source.parent.KeyColumn = result

        names = ["Edit DB Provider", "Edit Table Name", "Edit Column Name", "Edit Key Column"]
        functions = [editDatabaseProvider, editTable, editColumn, editKeyColumn]
        menu = system.gui.createPopupMenu(names, functions)
        menu.show(event)
```

Code Snippet

```
#This example shows a nested popup menu, with menus within menus.  All menu items call sayHello().
def sayHello(event):
   system.gui.messageBox("Hello World")
subMenu = [["Click Me 2", "Click Me 3"], [sayHello, sayHello]]
menu = system.gui.createPopupMenu(["Click Me", "SubMenu"], [sayHello, subMenu])
menu.show(event)
```

system.gui.errorBox

Description

Displays an error-style message box to the user.

Syntax

system.gui.errorBox(message [, title])

- Parameters

  String message - The message to display in an error box.

  String title - The title for the error box. [optional]

- Returns

  Nothing

- Scope

  Client

Code Examples

Code Snippet

```
#Turn on compressor #12, but only if the user has the right credentials.

if 'Supervisor' in system.security.getRoles():
 updateQuery = "UPDATE CompressorControl SET running=1 WHERE compNum = 12"
    system.db.runUpdateQuery(updateQuery)
else:
 errorMessage = "Unable to turn on Compressor 12."
 errorMessage += " You don't have proper security privileges."
    system.gui.errorBox(errorMessage)
```

system.gui.findWindow

Description

Finds and returns a list of windows with the given path. If the window is not open, an empty list will be returned. Useful for finding all instances of an open window that were opened withsystem.gui.openWindowInstance

Syntax

system.gui.findWindow(path)

- Parameters

  String path - The path of the window to search for

- Returns

  List - A list of window objects. May be empty if window is not open, or have more than one entry if multiple windows are open.

- Scope

  Client

Code Examples

Code Snippet

```
#This example finds all of the open instances of the window named "Popup" and closes them all.

allInstances = system.gui.findWindow("Popup")
for window in allInstances:
    system.nav.closeWindow(window)
```

system.gui.getOpenedWindowNames

Description

Finds all of the currently open windows, returning a tuple of their paths.

Syntax

system.gui.getOpenedWindowNames()

- Parameters

  None

- Returns

  PyTuple - A tuple of strings, representing the path of each window that is open.

- Scope

  All

Code Examples

Code Snippet

```
#This example prints out into the console the full path for each opened window.

windows = system.gui.getOpenedWindowNames()
print 'There are %d windows open' % len(windows)
for path in windows:
    print path
```

system.gui.getOpenedWindows

Description

Finds all of the currently open windows, returning a tuple of references to them.

Syntax

system.gui.getOpenedWindows()

- Parameters

  None

- Returns

  PyTuple - A tuple of the opened windows. Not their names, but the actual window objects themselves.

- Scope

  Client

Code Examples

Code Snippet

```
#This example prints out the path of each currently opened window to the console.

windows = system.gui.getOpenedWindows()
print 'There are %d windows open' % len(windows)
for window in windows:
    print window.getPath()
```

system.gui.getParentWindow

Description

Finds the parent (enclosing) window for the component that fired an event, returning a reference to it.

Syntax

system.gui.getParentWindow(event)

- Parameters

  EventObject event - A component event object.

- Returns

  PyObject - The window that contains the component that fired the event.

- Scope

  Client

Code Examples

Code Snippet

```
#Use this in an event script to change the window's title.

window = system.gui.getParentWindow(event)
window.title='This is a new title'
```

system.gui.getScreens

Description

Get a list of all the monitors on the computer this client is open on. Use with system.gui.setScreenIndex() to move the client.

Syntax

system.gui.getScreens()

- Parameters

  Nothing

- Returns

  PySequence - A sequence of tuples of the form (index, width, height) for each screen device (monitor) available.

- Scope

  Client

Code Examples

Code Snippet

```
#This example fetches monitor data and pushes it to a table in the same container

screens = system.gui.getScreens()
pyData = []
for screen in screens:
        pyData.append([screen[0], screen[1], screen[2]])

#Push data to 'Table'
event.source.parent.getComponent('Table').data = system.dataset.toDataSet(["screen","width","height"],
pyData)
```

system.gui.getSibling

Description

Given a component event object, looks up a sibling component. Shortcut for event.source.parent.getComponent("siblingName"). If no such sibling is found, the special value None is returned.

Syntax

system.gui.getSibling(event, name)

- Parameters

  EventObject event - A component event object.

  String name - The name of the sibling component.

- Returns

  PyObject - The sibling component itself.

- Scope

  Client

Code Examples

Code Snippet

```
#This example will get its sibling Text Field's text, and use it.

textField = system.gui.getSibling(event, 'TextField (1)')
if textField is None:
   system.gui.errorBox("There is no text field!")
else:
   system.gui.messageBox("You typed: %s" % textField.text)
```

system.gui.getWindow

Description

Finds a reference to an open window with the given name. Throws a ValueError if the named window is not open or not found.

Syntax

system.gui.getWindow(name)

- Parameters

  String name - The path to the window to field.

- Returns

  PyObject - A reference to the window, if it was open.

- Scope

  Client

Code Examples

Code Snippet

```
#This example will get the window named 'Overview' and then close it.

try:
   window = system.gui.getWindow('Overview')
   system.gui.closeWindow(window)

except ValueError:
   system.gui.warningBox("The Overview window isn't open")
```

Code Snippet

```
#This example will set a value on a label component in the 'Header' window.

try:
   window = system.gui.getWindow('Header')
   window.getRootContainer().getComponent('Label').text = "Machine 1 Starting"

except ValueError:
   system.gui.warningBox("The Header window isn't open")
```

system.gui.getWindowNames

Description

Returns a list of the paths of all windows in the current project, sorted alphabetically.

Syntax

system.gui.getWindowNames()

- Parameters

  None

- Returns

  PyTuple - A tuple of strings, representing the path of each window defined in the current project.

- Scope

  Client

Code Examples

Code Snippet

```
#This example would open windows that begin with "Motor" and pass in the currently selected motor number.

motor = event.source.parent.number
windows = system.gui.getWindowNames()
for path in windows:
   if name[:5] == "Motor":
      system.gui.openWindow(path, {"motorNumber":motor})
```

system.gui.inputBox

Description

Opens up a popup input dialog box. This dialog box will show a prompt message, and allow the user to type in a string. When the user is done, they can press "OK" or "Cancel". If OK is pressed, this function will return with the value that they typed in. If Cancel is pressed, this function will return the value None.

Syntax

system.gui.inputBox(message, defaultText)

- Parameters

  String message - The message to display for the input box.

  String defaultText - The default text to initialize the input box with.

- Returns

  String - The string value that was entered in the input box.

- Scope

  Client

Code Examples

Code Snippet

```
#This could go in the mouseClicked event of a label to allow the user to change the label's text.

txt = system.gui.inputBox("Enter text:", event.source.text)
if txt != None:
   event.source.text = txt
```

system.gui.isTouchscreenModeEnabled

Description

Checks whether or not the running client's touchscreen mode is currently enabled.

Syntax

system.gui.isTouchscreenModeEnabled()

- Parameters

  None

- Returns

  boolean - True(1) if the client currently has touchscreen mode activated.

- Scope

  All

Code Examples

Code Snippet

```
#This example should be used in the Client Startup Script to check if this client is being run on a touch
screen computer (judged by an IP address) and set touchscreen mode.

ipAddress = system.net.getIpAddress()
query = "SELECT COUNT(*) FROM touchscreen_computer_ips WHERE ip_address = '%s' "
isTouchscreen = system.db.runScalarQuery(query %(ipAddress))
if isTouchscreen and not system.gui.isTouchscreenModeEnabled():
   system.gui.setTouchscreenModeEnabled(1)
```

system.gui.messageBox

Description

Displays an informational-style message popup box to the user.

Syntax

- Parameters

  String message - The message to display.

  String title - A title for the message box. [optional]

- Returns

  Nothing

- Scope

  Client

Code Examples

Code Snippet

```
#This example will show how many hours a motor has been running when it is clicked.

# get the motor number
motorNumber = event.source.getPropertyValue('MotorNumber')
# retrieve the hours running from the database
query = "SELECT HoursRunning FROM MotorStatus WHERE motor=%d"
hours = system.db.runScalarQuery(query % motorNumber)

system.gui.messageBox("The motor has been running for %d hours" % motorNumber)
```

system.gui.moveComponent

> ⚠ **Deprecated**
>
> As of 7.8.1. See system.gui.transform() instead.

Description

Alters a component's position to a new pair of coordinates, (x,y), a point relative to the upper-left corner of the component's parent. Note that when using relative layout, these coordinates are evaluated as if the component's size was the same size as the last time the component was saved in the Designer. This effectively means that your argument coordinates will automatically scale with relative layout.

Syntax

system.gui.moveComponent(component, x, y)

- Parameters

  JComponent component - The component to move.

  int x - The x-coordinate to move to, relative to the upper-left corner of the component's parent container.

  int y - The y-coordinate to move to, relative to the upper-left corner of the component's parent container.

- Returns

  Nothing

- Scope

  Client

Code Examples

Code Snippet

```
#This code would go in a Timer's propertyChange script for animation.

if event.propertyName == "value":
   newX = event.newValue;
   rect = event.source.parent.getComponent("Rectangle")
   system.gui.moveComponent(rect, newX, 250)
```

system.gui.openDiagnostics

Description

Opens the client runtime diagnostics window, which provides information regarding performance, logging, active threads, connection status, and the console.

Syntax

system.gui.openDiagnostics()

- Parameters

  None

- Returns

  None

- Scope

  Client

Code Examples

Code Snippet

```
#Opens the diagnostics window in a running client
system.gui.openDiagnostics()
```

system.gui.passwordBox

Description

Pops up a special input box that uses a password field, so the text isn't echoed back in clear-text to the user. Returns the text they entered, or None if they canceled the dialog box.

Syntax

system.gui.passwordBox(message [, title] [, echoChar])

- Parameters

  String message - The message for the password prompt.

  String title - A title for the password prompt. [optional]

  String echoChar - A custom echo character. Defaults to: * [optional]

- Returns

  String - The password that was entered, or None if the prompt was canceled.

- Scope

  Client

Code Examples

Code Snippet

```
#This example would prompt a user for a password before opening the 'Admin' Screen.

password = system.gui.passwordBox("Please enter the password.")
if password == "open sesame":
   system.nav.openWindow("Admin")
```

system.gui.reshapeComponent

> ⚠ **Deprecated**
>
> As of 7.8.1

Description

Sets a component's position and size at runtime. The coordinates work in the same way as the system.gui.moveComponent function.

Syntax

system.gui.reshapeComponent(component, x, y, width, height)

- Parameters

  JComponent component - The component to move and resize

  int x - The x-coordinate to move to, relative to the upper-left corner of the component's parent container.

  int y - The y-coordinate to move to, relative to the upper-left corner of the component's parent container.

  int width - The new width for the component

  int height - The new height for the component

- Returns

  Nothing

- Scope

  Client

Code Examples

Code Snippet

```
#This code would go in a Timer's propertyChange script for animation.

if event.propertyName == "value":
   newX = event.newValue;
   newWidth = int(event.newValue*1.5)
   rect = event.source.parent.getComponent("Rectangle")
   system.gui.reshapeComponent(rect, newX, 150, newWidth, 80)
```

system.gui.resizeComponent

> ⚠ **Deprecated**
>
> As of 7.8.1

Description

Sets a component's size at runtime. The coordinates work in the same way as the system.gui.moveComponent function.

Syntax

system.gui.resizeComponent(component, width, height)

- Parameters

  JComponent component - The component to resize

  int width - The new width for the component

  int height - The new height for the component

- Returns

  Nothing

- Scope

  Client

Code Examples

Code Snippet

```
#This code would go in a Timer's propertyChange script for animation \

if event.propertyName == "value":
   newWidth = event.newValue;
   rect = event.source.parent.getComponent("Rectangle")
   system.gui.resizeComponent(rect, newWidth, 80)
```

system.gui.setScreenIndex

Description

Moves an open client to a specific monitor. Use with system.gui.getScreens() to identify monitors before moving.

Syntax

system.gui.setScreenIndex(index)

- Parameters

  integer index - The new monitor index for this client to move to. 0 based.

- Returns

  Nothing

- Scope

  Client

Code Examples

Code Snippet

```
#This example could be used on a startup script to move the client to a 2nd monitor.

system.gui.setScreenIndex(1)
```

system.gui.setTouchscreenModeEnabled

Description

Alters a running client's touchscreen mode on the fly.

Syntax

system.gui.setTouchscreenModeEnabled(enabled)

- Parameters

  boolean enabled - The new value for touchscreen mode being enabled.

- Returns

  Nothing

- Scope

  Client

Code Examples

Code Snippet

```
#This example could be used on an input heavy window's internalFrameActivated event to remove touch
screen mode.

if system.gui.isTouchscreenModeEnabled():
   system.gui.setTouchscreenModeEnabled(0)
```

system.gui.showNumericKeypad

Description

Displays a modal on-screen numeric keypad, allowing for arbitrary numeric entry using the mouse, or a finger on a touchscreen monitor. Returns the number that the user entered.

Syntax

system.gui.showNumericKeypad(initialValue [, fontSize][, usePasswordMode])

- Parameters

  Number initialValue - The value to start the on-screen keypad with.

  int fontSize - The font size to display in the keypad. [optional]

  boolean usePasswordMode - If True, display a * for each digit. [optional] (since 7.8.1)

- Returns

  Number - The value that was entered in the keypad.

- Scope

  Client

Code Examples

Code Snippet

```
#This function is a holdover for backwards compatibility. Input components now know when the client is in
touchscreen mode and respond accordingly.
#This script would go in the MouseClicked or MousePressed action of a Text Field or Numeric Text Field.

# For Integer Numeric Text Field:
if system.gui.isTouchscreenModeEnabled():
   event.source.intValue = system.gui.showNumericKeypad(event.source.intValue)

# For Double Numeric Text Field:
if system.gui.isTouchscreenModeEnabled():
   event.source.doubleValue = system.gui.showNumericKeypad(event.source.doubleValue)

# For Text Field:
# notice the str() and int() functions used to convert the text to a number and
# vice versa.
# str() and int() are built-in Jython functions
if system.gui.isTouchscreenModeEnabled():
   event.source.text = str(system.gui.showNumericKeypad(int(event.source.text)))
```

system.gui.showTouchscreenKeyboard

Description

Displays a modal on-screen keyboard, allowing for arbitrary text entry using the mouse, or a finger on a touchscreen monitor. Returns the text that the user "typed".

Syntax

system.gui.showTouchscreenKeyboard(initialText [, fontSize] [, passwordMode])

- Parameters

  String initialText - The text to start the on-screen keyboard with.

  int fontSize - The font size to display in the keyboard. [optional]

  boolean passwordMode - True (1) to activate password mode, where the text entered isn't echoed back clear-text. [optional]

- Returns

  String - The text that was "typed" in the on-screen keyboard.

- Scope

  Client

Code Examples

Code Snippet

```
#This function is a holdover for backwards compatibility. Input components now know when the client is in
touchscreen mode and respond accordingly.
#This would go in the MouseClicked or MousePressed action of a Text Field or similar component.

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

system.gui.transform

Description

Sets a component's position and size at runtime. Additional arguments for the duration, framesPerSecond, and acceleration of the operation exist for animation. An optional callback argument will be executed when the transformation is complete. Note: The transformation is performed in Designer coordinate space on components which are centered or have more than 2 anchors. Since 7.8.1.

Syntax

system.gui.transform(component [, newX, newY, newWidth, newHeight, duration, callback, framesPerSecond, acceleration, coordSpace])

- Parameters

  JComponent component - The component to move or resize.

  int newX - An optional x-coordinate to move to, relative to the upper-left corner of the component's parent container.

  int newY - An optional y-coordinate to move to, relative to the upper-left corner of the component's parent container.

  int newWidth - An optional width for the component.

  int newHeight - An optional height for the component.

  int duration - An optional duration over which the transformation will take place. If omitted or 0, the transform will take place immediately.

  PyObject callback- An optional function to be called when the transformation is complete.

  int framesPerSecond - An optional frame rate argument which dictates how often the transformation updates over the given duration. The default is 60 frames per second.

  int acceleration - An optional modifier to the acceleration of the transformation over the given duration. See system.gui constants for valid arguments.

  int coordSpace- The coordinate space to use. When the default Screen Coordinates are used, the given size and position are absolute, as they appear in the client at runtime. When Designer Coordinates are used, the given size and position are pre-runtime adjusted values, as they would appear in the Designer. See system.gui constants for valid arguments.

- Returns

  PyObject animation - An animation object that the script can use to pause(), resume(), or cancel() the transformation.

- Scope

  Client

Code Examples

```
# This example changes the size the a component to 100x100
# This script should be run from the component that will be changed (ie: on the mouseEntered event)

system.gui.transform(component=event.source, newWidth=100, newHeight=100)
```

```
# This example moves a component to coordinates 0,0 over the course of 1 second.
# When the animation is complete, the component is moved back to its original position
#   over the course of 2 seconds, slowing in speed as it approaches the end.

component = event.source.parent.getComponent('Text Field')
origX = component.x
origY = component.y

system.gui.transform(
        component,
        0, 0,
        duration=1000,
        callback=lambda: system.gui.transform(
                component,
                origX, origY,
                duration=2000,
                acceleration=system.gui.ACCL_FAST_TO_SLOW
        )
)
```

system.gui.warningBox

Description

Displays a message to the user in a warning style pop-up dialog.

Syntax

system.gui.warningBox(message [, title])

- Parameters

  String message - The message to display in the warning box.

  String title - The title for the warning box. [optional]

- Returns

  Nothing

- Scope

  Client

Code Examples

Code Snippet

```
#This code show a yellow popup box similar to the system.gui.messageBox function.
# Start the motor, or, warn the user if in wrong mode
runMode = event.source.parent.getPropertyValue('RunMode')

# Cannot start the motor in mode #1
if runMode == 1:
   system.gui.warningBox("Cannot start the motor, current mode is <B>VIEW MODE</B>")
else:
   system.db.runUpdateQuery("UPDATE MotorControl SET MotorRun=1")
```

system.nav

system.nav.centerWindow

Description

Given a window path, or a reference to a window itself, it will center the window. The window should be floating an non-maximized. If the window can't be found, this function will do nothing.

Syntax

system.nav.centerWindow(windowPath)

- Parameters

  String windowPath - The path of the window to center.

- Returns

  Nothing

- Scope

  All

Syntax

system.nav.centerWindow(window)

- Parameters

  FPMIWindow window - A reference to the window to center.

- Returns

  Nothing

- Scope

  All

Code Examples

Code Snippet

```
#This example centers the window named 'Overview'.
system.nav.centerWindow('Overview')
```

system.nav.closeParentWindow

Description

Closes the parent window given a component event object.

Syntax

system.nav.closeParentWindow(event)

- Parameters

  EventObject event - A component event object. The enclosing window for the component will be closed.

- Returns

  nothing

- Scope

  Client

Code Examples

Code Snippet

```
 # This code would be placed in the actionPerformed event of a button,
# and would close the window that contained the button.
system.nav.closeParentWindow(event)
```

system.nav.closeWindow

Description

Given a window path, or a reference to a window itself, it will close the window. If the window can't be found, this function will do nothing.

Syntax

system.nav.closeWindow(window)

- Parameters

  FPMIWindow window - A reference to the window to close.

- Returns

  nothing

- Scope

  Client

Syntax

system.nav.closeWindow(windowPath)

- Parameters

  String windowPath - The path of a window to close.

- Returns

  nothing

- Scope

  Client

Code Examples

Code Snippet

```
#This example would get the window named 'Overview' and then close it.
# If the window isn't open, show a warning
try:
   window = system.gui.getWindow('Overview')
   system.nav.closeWindow(window)
except ValueError:
   system.gui.warningBox("The Overview window isn't open")
```

Code Snippet

```
#This example would close the window named 'Overview' in one step.
# If the window isn't open, the call to closeWindow will have no effect
system.nav.closeWindow('Overview')
```

system.nav.getCurrentWindow

Description

Returns the path of the current "main screen" window, which is defined as the maximized window. With the Typical Navigation Strategy, there is only ever one maximized window at a time.

Syntax

system.nav.getCurrentWindow()

- Parameters

  none

- Returns

  String - The path of the current "main screen" window - the maximized window.

- Scope

  Client

Code Examples

Code Snippet

```
# This code could run in a global timer script.
# After a 5-minute timeout, navigate back to the home screen
if system.util.getInactivitySeconds()>300 and system.nav.getCurrentWindow()!="Home":
   system.nav.swapTo("Home")
```

system.nav.goBack

Description

When using the Typical Navigation Strategy, this function will navigate back to the previous main screen window.

Syntax

system.nav.goBack()

- Parameters

  none

- Returns

  PyObject - The window that was returned to

- Scope

  Client

Code Examples

Code Snippet

```
#This code would go in a button to move to the previous screen.
system.nav.goBack()
```

system.nav.goForward

Description

When using the Typical Navigation Strategy, this function will navigate "forward" to the last main-screen window the user was on when they executed a system.nav.goBack().

Syntax

system.nav.goForward()

- Parameters

  none

- Returns

  PyObject - The window that was returned to

- Scope

  Client

Code Examples

Code Snippet

```
#This code would go in a button to move to the last screen that used system.nav.goBack().
system.nav.goForward()
```

system.nav.goHome

Description

When using the Typical Navigation Strategy, this function will navigate to the "home" window. This is automatically detected as the first main-screen window shown in a project.

Syntax

system.nav.goHome()

- Parameters

  none

- Returns

  PyObject - A reference to the home window that was navigated to.

- Scope

  Client

Code Examples

Code Snippet

```
#This code would go in a button to move to the Home screen.
system.nav.goHome()
```

system.nav.openWindow

Description

Opens the window with the given path. If the window is already open, brings it to the front. The optional params dictionary contains key:value pairs which will be used to set the target window's root container's dynamic variables.

For instance, if the window that you are opening is named "TankDisplay" has a dynamic variable in its root container named "TankNumber", then callingsystem.nav.openWindow("TankDisplay", {"TankNumber" : 4}) will open the "TankDisplay" window and set Root Container.TankNumber to four. This is useful for making parameterized windows, that is, windows that are re-used to display information about like pieces of equipment. See also: Parameterized Popup Windows.

Syntax

system.nav.openWindow(path [, params])

- Parameters

  String path - The path to the window to open.

  PyDictionary params - A dictionary of parameters to pass into the window. The keys in the dictionary must match dynamic property names on the target window's root container. The values for each key will be used to set those properties. [optional]

- Returns

  PyObject - A reference to the opened window.

- Scope

  Client

Code Examples

Code Snippet

```
# This is the simplest form of openWindow
system.nav.openWindow("SomeWindowName")
```

Code Snippet

```
# A more complex example - a setpoint screen for multiple valves that opens centered
titleText = "Third Valve Setpoints"
tankNo = system.nav.openWindow("ValveSetPts", {"valveNum":3, "titleText":titleText})
system.nav.centerWindow("ValveSetPts")
```

system.nav.openWindowInstance

Description

Operates exactly like system.nav.openWindow, except that if the named window is already open, then an additional instance of the window will be opened. There is no limit to the number of additional instances of a window that you can open.

Syntax

system.nav.openWindowInstance(path [, params])

- Parameters

  String path - The path to the window to open.

  PyDictionary params - A dictionary of parameters to pass into the window. The keys in the dictionary must match dynamic property names on the target window's root container. The values for each key will be used to set those properties. [optional]

- Returns

  PyObject - A reference to the opened window.

- Scope

  Client

Code Examples

Code Snippet

```
#This example would open three copies of a single HOA popup screen.

system.nav.openWindowInstance("HOA" {machineNum:3})
system.nav.openWindowInstance("HOA" {machineNum:4})
system.nav.openWindowInstance("HOA" {machineNum:5})
```

system.nav.swapTo

Description

Performs a window swap from the current main screen window to the window specified. Swapping means that the opened window will take the place of the closing window - in this case it will be maximized. See also: Navigation Strategies.

This function works like system.nav.swapWindow except that you cannot specify the source for the swap

Syntax

system.nav.swapTo(path [, params])

- Parameters

  String path - The path of a window to swap to.

  PyDictionary params - A dictionary of parameters to pass into the window. The keys in the dictionary must match dynamic property names on the target window's root container. The values for each key will be used to set those properties. [optional]

- Returns

  PyObject - A reference to the swapped-to window.

- Scope

  Client

Code Examples

Code Snippet

```
#This code would go in a button's ActionPerformed event to swap out of the current window and into a
window named MyWindow
system.nav.swapTo("MyWindow")
```

Code Snippet

```
#This code would go in a button's ActionPerformed event to swap out of the current window and into a
window named MyWindow.
#It also looks at the selected value in a dropdown menu and passes that value into the new window.

# MyWindow's Root Container must have a dynamic property named "paramValue"
dropdown = event.source.parent.getComponent("Dropdown")
system.nav.swapTo("MyWindow", {"paramValue":dropdown.selectedValue)
```

Code Snippet

```
#This code cycles through a dictionary of windows. This could be placed on a Client Event Timer Script to
cycle through some windows.
#The below code assumes that each of the windows are in the same folder (named "Main Windows")
#If the windows are in different folders, then the script would need to be modified to prepend the
correct folder name on the last line of code.


#Build a dictionary of window names without directories.
windowDict = {"Overview":"Motors", "Motors":"Alarming", "Alarming":"Scripting", "Scripting":"Overview"}
#Find the current window
currentWin = system.nav.getCurrentWindow()
winObj = system.gui.getWindow(currentWin)
#Find the next window in the dictionary based on the name of the current window (winObj)
nextWindow = windowDict[winObj.name]
#Swap to the next window
system.nav.swapTo("Main Windows/" + nextWindow)
```

system.nav.swapWindow

Description

Performs a window swap. This means that one window is closed, and another is opened and takes its place - assuming its size, floating state, and maximization state. This gives a seamless transition - one window seems to simply turn into another.

This function works like system.nav.swapTo except that you can specify the source and destination for the swap

Syntax

system.nav.swapWindow(swapFromPath, swapToPath [, params])

- Parameters

  String swapFromPath - The path of the window to swap from. Must be a currently open window, or this will act like an openWindow.

  String swapToPath - The name of the window to swap to.

  PyDictionary params - A dictionary of parameters to pass into the window. The keys in the dictionary must match dynamic property names on the target window's root container. The values for each key will be used to set those properties. [optional]

- Returns

  PyObject - A reference to the swapped-to window.

- Scope

  Client

Syntax

system.nav.swapWindow(event, swapToPath [, params])

- Parameters

  EventObject event - A component event whose enclosing window will be used as the "swap-from" window.

  String swapToPath - The name of the window to swap to.

  PyDictionary params - A dictionary of parameters to pass into the window. The keys in the dictionary must match dynamic property names on the target window's root container. The values for each key will be used to set those properties. [optional]

- Returns

  PyObject - A reference to the swapped-to window.

- Scope

  Client

Code Examples

Code Snippet

```
# This code would go in a button's ActionPerformed event to swap out of the
# window containing the button and into a window named MyWindow
system.nav.swapWindow(event, "MyWindow")
```

Code Snippet

```
# This code would swap from window named WindowA to a window named WindowB
system.nav.swapWindow("WindowA", "WindowB")
```

Code Snippet

```
# This code would swap from window named WindowA to a window named WindowB.
# It also looks at the two calendar popup controls and passes the two selected
# dates to WindowB. WindowB's Root Container must have dynamic properties named
# "startDate" and "endDate"
date1 = event.source.parent.getComponent("Start Date").date
date2 = event.source.parent.getComponent("End Date").date
system.nav.swapWindow("WindowA", "WindowB", {"startDate":date1, "endDate":date2})
```

system.net

system.net.getExternalIpAddress

Description

Returns the client's IP address, as it is detected by the Gateway. This means that this call will communicate with the Gateway, and the Gateway will tell the clienth what IP address its incoming traffic is coming from. If you have a client behind a NAT router, then this address will be the WAN address of the router instead of the LAN address of the client, which is what you'd get with system.net.getIpAddress.

Syntax

system.net.getExternalIpAddress()

- Parameters

  None

- Returns

  String - A text representation of the client's IP address, as detected by the Gateway

- Scope

  Client

Code Examples

Code Snippet

```
#Put this script on a navigation button to restrict users from opening a specific page.

ip = sytem.net.getExternalIpAddress()
#check if this matches the CEO's IP address
if ip == "66.102.7.104":
   system.nav.swapTo("CEO Dashboard")
else:
   system.nav.swapTo("Manager Dashboard")
```

system.net.getHostName

Description

Returns the host name of the computer that the client is currently running on. On Windows, this is typically the "computer name". For example, might return EAST_WING_WORKSTATION or bobs-laptop.

Syntax

system.net.getHostName()

- Parameters

  none

- Returns

  String - The hostname of the local machine. This is the computer that the script is being executed on - may be a Client or the Gateway depending on the script context.

- Scope

  All

Code Examples

Code Snippet

```
#Put this script on a navigation button to link dedicated machines to specific screens.
comp = system.net.getHostName()
#check which line this client is tied to
if comp == "Line1Computer":
   system.nav.swapTo("Line Detail", {"line":1})
elif comp == "Line2Computer":
   system.nav.swapTo("Line Detail", {"line":2})
else:
   system.nav.swapTo("Line Overview")
```

system.net.getIpAddress

Description

Returns the IP address of the computer the client is running on, as it appears to the client. See also: system.net.getExternalIpAddress().

Syntax

system.net.getIpAddress()

- Parameters

  none

- Returns

  String - Returns the IP address of the local machine, as it sees it.

- Scope

  All

Code Examples

Code Snippet

```
#Put this script on a navigation button to link dedicated machines to specific screens.
ip = sytem.net.getIpAddress()
#check which line this client is tied to
if ip == "10.1.10.5":
   system.nav.swapTo("Line Detail", {"line":1})
elif ip == "10.1.10.6":
   system.nav.swapTo("Line Detail", {"line":2})
else:
   system.nav.swapTo("Line Overview")
```

system.net.getRemoteServers

Description

This function returns a List of Gateway Network servers that are visible from the local Gateway.

Syntax

system.net.getRemoteServers([runningOnly])

- Parameters

  Boolean runningOnly- [Optional] If set to True, only servers on the Gateway Network that are running will be returned. Servers that have lost contact with the Gateway Network will be filtered out.

- Returns

  String[] - A List of Strings representing Gateway Network server ids.

- Scope

  All

Code Examples

Code Snippet

```
#The following will create a list of running servers on the Gateway Network, and show the list in a
message box.

#Collect the list of running servers
runningServers = system.net.getRemoteServers(True)

#initialize the start of the message
serverStatusText = "The following servers are running:\n "
#add each running server to the message
for server in runningServers:
        serverStatusText += "%s \n" % server

#Show the message
system.gui.messageBox(serverStatusText)
```

system.net.httpDelete

This function is used in **Python Scripting.**

Description

Performs an HTTP DELETE to the given URL.

Keep in mind that JRE proxy settings will influence how these functions conduct their network activities. When using a non-bundled JRE, this is typically set via the Java Control Panel.

Syntax

This function accepts keyword arguments.

system.net.httpDelete(url, [contentType], [connectTimeout], [readTimeout], [username], [password], [headerValues], [bypassCertValidation])

- Parameters

  String url - The URL to send the request to.

  String contentType - [Optional] The MIME type used in the HTTP 'Content-type' header.

  Int connectTimeout - [Optional] The timeout for connecting to the URL in milliseconds. Default is 10,000

  Int readTimeout - [Optional] The read timeout for the operation in milliseconds. Default is 60,000.

  String username - [Optional] If specified, the call will attempt to authenticate with basic HTTP authentication.

  String password - [Optional] The password used for basic HTTP authentication, if the username parameter is also present.

  PyDictionary headerValues - [Optional] A dictionary of name/value pairs that will be set in the HTTP header.

  Boolean bypassCertValidation - [Optional] If the target address in an HTTPS address, and this parameter is TRUE, the system will bypass all SSL certificate validation. This is not recommended, though is sometimes necessary for self-signed certificates.

- Returns

  Object - The content returned for the DELETE operation.

- Scope

  All

Code Examples

```
# This example attempts to perform a DELETE operation
URL = "http://myURL/folder.resource"
system.net.httpDelete(URL)
```

system.net.httpGet

Description

Retrieves the document at the given URL using the HTTP GET protocol. The document is returned as a string. For example, if you use the URL of a website, you'll get the same thing you'd get by going to that website in a browser and using the browser's "View Source" function.

Keep in mind that JRE proxy settings will influence how these functions conduct their network activities. When using a non-bundled JRE, this is typically set via the Java Control Panel.

Syntax

system.net.httpGet(url, connectTimeout, readTimeout, username, password, headerValues, bypassCertValidation)

- Parameters

  String url - The URL to retrieve.

  Integer connectTimeout - The timeout for connecting to the url. In millis. Default is 10,000.

  Integer readTimeout - The read timeout for the get operation. In millis. Default is 60,000.

  String username - If specified, the call will attempt to authenticate with basic HTTP authentication.

  String password - The password used for basic http authentication, if the username parameter is also present.

  PyDictionary headerValues - Optional - A dictionary of name/value pairs that will be set in the http header.

  Boolean bypassCertValidation - Optional - If the target address is an HTTPS address, and this parameter is True, the system will bypass all SSL certificate validation. This is not recommended, though is sometimes necessary for self-signed certificates.

- Returns

  String - The content found at the given URL.

- Scope

  All

Code Examples

Code Snippet

```
# This code would return the source for Google's homepage
source = system.net.httpGet("http://www.google.com")
print source
```

Code Snippet

```
# This code would query Yahoo Weather for the temperature in Sacramento, CA
# and then find the current temperature using a regular expression

response = system.net.httpGet("http://xml.weather.yahoo.com/forecastrss?p=95818")

# import Python's regular expression library
import re

# NOTE - if you've never seen regular expressions before, don't worry, they look
# confusing even to people who use them frequently.
pattern = re.compile('.*?<yweather:condition (.*?)/>', re.DOTALL)
match = pattern.match(response)
if match:
   subText = match.group(1)
   condition = re.compile('.*?text="(.*?)"').match(subText).group(1)
   temp = re.compile('.*?temp="(.*?)"').match(subText).group(1)
   print "Condition: ", condition
   print "Temperature (F): ", temp
else:
   print 'Weather service format changed'
```

system.net.httpPost

Description

Retrieves the document at the given URL using the HTTP POST protocol. If a parameter dictionary argument is specified, the entries in the dictionary will encoded in "application/x-www-form-urlencoded" format, and then posted. You can post arbitrary data as well, but you'll need to specify the MIME type. The document is then returned as a string.
Keep in mind that JRE proxy settings will influence how these functions conduct their network activities. When using a non-bundled JRE, this is typically set via the Java Control Panel.

Syntax

system.net.httpPost(url, postParams)

- Parameters

  String url - The URL to post to.

  PyDictionary postParams - A dictionary of name: value key pairs to use as the post data.

- Returns

  String - The content returned for the POST operation.

- Scope

  All

Syntax

system.net.httpPost(url, contentType, postData, connectTimeout, readTimeout, username, password, headerValues, bypassCertValidation)

- Parameters

  String url - The URL to post to.

  String contentType - Optional - The MIME type to use in the HTTP "Content-type" header.

  String postData - The raw data to post via HTTP.

  Integer connectTimeout - The timeout for connecting to the url. In millis. Default is 10,000.

  Integer readTimeout - The read timeout for the get operation. In millis. Default is 60,000.

  String username - If specified, the call will attempt to authenticate with basic HTTP authentication.

  String password - The password used for basic http authentication, if the username parameter is also present.

  PyDictionary headerValues - Optional - A dictionary of name/value pairs that will be set in the http header.

  Boolean bypassCertValidation - Optional - If the target address is an HTTPS address, and this parameter is True,the system will bypass all SSL certificate validation. This is not recommended, though is sometimes necessary for self-signed certificates.

- Returns

  String - The content returned for the POST operation.

- Scope

  All

Code Examples

Code Snippet

```
# This code posts a name (first and last) to the post testing page at
# "http://www.snee.com/xml/crud/posttest.cgi", and returns the resulting page
# as a string.
page = system.net.httpPost("http://www.snee.com/xml/crud/posttest.cgi", {"fname":"Billy", "lname":"Bob"})
print page
```

Code Snippet

```
# This code sends an XML message to a hypothetical URL.
message = "<MyMessage><MyElement>here is the element</MyElement></MyMessage>"
system.net.httpPost("http://www.posttome.xyz/posthere", "text/xml", message)
```

system.net.httpPut

This function is used in **Python Scripting.**

Description

Performs an HTTP PUT to the given URL. Encodes the given dictionary of parameters using "applications/x-www-form-urlencoded" format.

Keep in mind that JRE proxy settings will influence how these functions conduct their network activities. When using a non-bundled JRE, this is typically set via the Java Control Panel.

Syntax

This function accepts keyword arguments.

system.net.httpPut(url, [contentType], putData, [connectTimeout], [readTimeout], [username], [password], [headerValues], [bypassCertValidation])

- Parameters

  String url - The URL to put to.

  String contentType - [Optional] The MIME type used in the HTTP 'Content-type' header.

  String putData - The raw data to put via HTTP.

  Int connectTimeout - [Optional] The timeout for connecting to the URL in milliseconds. Default is 10,000

  Int readTimeout - [Optional] The read timeout for the operation in milliseconds. Default is 60,000.

  String username - [Optional] If specified, the call will attempt to authenticate with basic HTTP authentication.

  String password - [Optional] The password used for basic HTTP authentication, if the username parameter is also present.

  PyDictionary headerValues - [Optional] A dictionary of name/value pairs that will be set in the HTTP header.

  Boolean bypassCertValidation - [Optional] If the target address in an HTTPS address, and this parameter is TRUE, the system will bypass all SSL certificate validation. This is not recommended, though is sometimes necessary for self-signed certificates.

- Returns

  String - The content returned for the PUT operation.

- Scope

  All

Syntax

system.net.httpPut(url, putParams)

- Parameters

  String url - The URL to send the request to.

  PyDictionary putParams - A dictionary of name/value key pairs to use as the put data.

- Returns

  String - The content returned for the PUT operation.

- Scope

  All

Code Examples

Code Example - Simple Test

```
# The following example uses a test URL to echo back the data used in the PUT request.
# Test URL courtesy of:
# http://stackoverflow.com/questions/5725430/http-test-server-that-accepts-get-post-calls?
answertab=votes#tab-top

# Specify URL and parameters to pass in the PUT call
URL = "http://httpbin.org/put"
params = {"testkey":"testValue"}

# Make the PUT request and print the results to the console
print system.net.httpPut(URL, params)
```

Code Example - Keyword Arguments

```
 """
This example attempts to authenticate with a username and password, as well as specify a MIME type.
The Username and password are static in this example, but could easily make use of other components to
allow user input
or fetch data out of a database instead.
"""

URL = "http://httpbin.org/put"
params = {"testkey":"testValue"}
user = "myUser"
userPass = "password"

# Make the PUT request and print the results to the console
print system.net.httpPut(URL, params, username = user, password = userPass, contentType = "text/html")
```

system.net.openURL

Description

Opens the given URL or URI scheme outside of the currently running Client in whatever application the host operating system deems appropriate. For example, the URL:

"http://www.google.com"

... will open in the default web browser, whereas this one:

"file://C:/Report.pdf"

... will likely open in Adobe Acrobat. The Windows network-share style path like:

"\\Fileserver\resources\machine_manual.pdf"

... will work as well (in Windows).

Be careful not to use this function in a full-screen client, as launching an external program will break your full-screen exclusive mode.

Syntax

system.net.openURL(url [, useApplet])

- Parameters

  String url - The URL to open in a web browser.

  boolean useApplet - If set to true (1), and the client is running as an Applet, then the browser instance that launched the applet will be used to open the URL. [optional]

- Returns

  nothing

- Scope

  Client

Code Examples

Code Snippet

```
# This code would open a web page
system.net.openURL("http://www.google.com")
```

Code Snippet

```
# This code would open a PDF document located at C: on the client computer
# Note the double backslashes are needed because backslash is the escape character
# for Python
system.net.openURL("file://C:\\myPDF.pdf")
```

Code Snippet

```
# This code would open a PDF document from a Windows-based file server
# Note the double backslashes are needed because backslash is the escape character
# for Python
system.net.openURL("\\\\MyServer\\MyDocs\\document.pdf")
```

system.net.sendEmail

Description

Sends an email through the given SMTP server. Note that this email is relayed first through the Gateway - the client host machine doesn't need network access to the SMTP server.

You can send text messages to cell phones and pagers using email. Contact your cell carrier for details. If you had a Verizon cell phone with phone number (123) 555-8383, for example, your text messaging email address would be: 1235558383@vtext.com. Try it out!

Syntax

This function accepts keyword arguments.

system.net.sendEmail(smtp, fromAddr, subject, body, html, to, attachmentNames, attachmentData, timeout, username, password, priority, smtpProfile, cc, bcc, retries)

- Parameters

  String smtp - The address of an SMTP server to send the email through, like "mail.example.com". A port can be specified, like "mail.example.com:25". SSL can also be forced, like "mail.example.com:25:tls".

  String fromAddr - An email address to have the email come from.

  String subject - The subject line for the email

  String body - The body text of the email.

  Boolean html - A flag indicating whether or not to send the email as an HTML email. Will auto-detect if omitted.

  String[] to - A list of email addresses to send to.

  String[] attachmentNames - A list of attachment names. Attachment names must have the correct extension for the file type or an error will occur.

  byte[][] attachmentData - A list of attachment data, in binary format.

  Integer timeout - A timeout for the email, specified in milliseconds. Defaults to 300,000 milliseconds (5 minutes).

  String username - If specified, will be used to authenticate with the SMTP host.

  String password - If specified, will be used to authenticate with the SMTP host.

  String priority - Priority for the message, from "1" to "5", with "1" being highest priority. Defaults to "3" (normal) priority.

  String smtpProfile - If specified, the named SMTP profile defined in the Gateway will be used. If this keyword is present, the smtp, username, and password keywords will be ignored.

  String[] cc - A list of email addresses to carbon copy. Only available if a smtpProfile is used.

  String[] bcc - A list of email addresses to blind carbon copy. Only available if a smtpProfile is used.

  Integer retries - The number of additional times to retry sending on failure. Defaults to 0. Only available if a smtpProfile is used.

- Returns

  nothing

- Scope

  All

Code Examples

Code Snippet

```
# This code would send a simple plain-text email to a single recipient, with no attachments
body = "Hello, this is an email."
recipients = ["bobsmith@mycompany.com"]
system.net.sendEmail("mail.mycompany.com",
 "myemail@mycompany.com", "Here is the email!", body, 0, recipients)
```

Code Snippet

```
# This code would send an HTML-formatted email to multiple recipients (including
# cellphones) with no attachments
body = "<HTML><BODY><H1>This is a big header</H1>"
body += "And this text is <font color='red'>red</font></BODY></HTML>"
recipients = ["bobsmith@mycompany.com", "1235558383@vtext.com", "sally@acme.org", "1235557272@vtext.com"]
myuser = "mycompany"
mypass = "1234"
system.net.sendEmail(smtp="mail.mycompany.com", fromAddr="myemail@mycompany.com",
subject="Here is the email!", body=body, html=1, to=recipients, username=myuser, password=mypass)
```

Code Snippet

```
# This code ask the user for an attachment file and attaches the file.
filePath = system.file.openFile()
if filePath != None:
   # This gets the filename without the C:\folder stuff
   fileName = filePath.split("\\")[-1]
   fileData = fpmi.file.readFileAsBytes(filePath)
   smtp = "mail.mycompany.com"
   sender = "myemail@mycompany.com"
   subject = "Here is the file you requested"
   body = "Hello, this is an email."
   recipients = ["bobsmith@mycompany.com"]
   system.net.sendEmail(smtp, sender, subject, body, 0, recipients, [fileName], [fileData])
```

Code Snippet

```
# This code would send an HTML-formatted email to multiple recipients, including a cc, with no
attachments,
# using an smtp server defined in the Gateway
body = "<HTML><BODY><H1>This is a big header</H1>"
body += "And this text is <font color='red'>red</font></BODY></HTML>"
recipients = ["bobsmith@mycompany.com", "1235558383@vtext.com", "sally@acme.org", "1235557272@vtext.com"]
cc_recipients = ["annejones@mycompany.com"]
smtp_server = "mySmtpServer"
system.net.sendEmail(smtpProfile=smtp_server, fromAddr="myemail@mycompany.com", subject="Here is the
email!", body=body, html=1, to=recipients, cc=cc_recipients)
```

system.opc

system.opc.browse

Description

Allows browsing of the OPC servers in the runtime, returning a list of tags.

Syntax

This function accepts keyword arguments.

system.opc.browse(opcServer, device, folderPath, opcItemPath)

- Parameters

  String opcServer - The name of the OPC server to browse

  String device - The name of the device to browse

  String folderPath - Filters on a folder path. Use * as a wildcard for any number of characters and a ? for a single character.

  String opcItemPath - Filters on a OPC item path. Use * as a wildcard for any number of characters and a ? for a single character.

- Returns

  OPCBrowseTag[] - An array of OPCBrowseTag objects. OPCBrowseTag has the following functions: getOpcServer(), getOpcItemPath(), getType(), getDisplayName(), getDisplayPath(), getDataType().

- Scope

  All

Code Examples

Code Snippet

```
#Example 1: Browse every OPC server

tags = system.opc.browse()
for row in tags:
    print row.getOpcServer(), row.getOpcItemPath(), row.getType(),
    print row.getDisplayName(), row.getDisplayPath(), row.getDataType()
```

Code Snippet

```
#Example 2: Browse Ignition OPC-UA

tags = system.opc.browse(opcServer="Ignition OPC-UA Server")
```

Code Snippet

```
#Example 3: Browse Specific Device

server = "Ignition OPC-UA Server"
tags = system.opc.browse(opcServer=server, device="Dairy Demo Simulator")
```

Code Snippet

```
#Example 4: Browse Specific Folder Path (not OPC item path)

server = "Ignition OPC-UA Server"
tags = system.opc.browse(opcServer=server, folderPath="*Overview/AU 1*")
```

system.opc.browseServer

Description

When called from a Vision Client, returns a list of OPCBrowseElement objects for the given server. Otherwise returns a list of OPCBrowseElements.

The OPCBrowseElement object has the following methods:

- getDisplayName() - returns the display name of the object
- getElementType() - returns the element type. Element types are server, device, view, folder, object, datavariable, property and method.
- getNodeId() - returns a string representing the server node ID

The PyOPCTag object has the following methods to retrieve information:

- getDisplayName() - returns the display name of the object
- getElementType() - returns the element type. Element types are server, device, view, folder, object, datavariable, property and method.
- getServerName() - returns the server name as a string.

Syntax

system.opc.browseServer(opcServer, nodeId)

- Parameters

  String opcServer - The name of the OPC server connection.

  String nodeId - The node ID to browse.

- Returns

  List - A list of PyOPCTag objects.

- Scope

  Gateway

Syntax - Vision Client Scope

system.opc.browseServer(opcServer, nodeId)

- Parameters

  String opcServer - The name of the OPC server connection.

  String nodeId - The node ID to browse.

- Returns

  List - A list of OPCBrowseElement objects.

- Scope

  Vision Client

Code Examples

Code Snippet

```
# Print the name of all devices on Ignition OPC-UA
opcServer="Ignition OPC-UA Server"
nodeId = "Devices"
devices = system.opc.browseServer(opcServer, nodeId)
for device in devices:
 print device.getDisplayName()
```

system.opc.browseSimple

Description

Allows browsing of OPC servers in the runtime returning a list of tags. browseSimple() takes mandatory parameters, which can be null, while browse() uses keyword-style arguments.

> ⚠ The spelling on the opcServer and device parameters must be exact.

Syntax

system.opc.browseSimple(opcServer, device, folderPath, opcItemPath)

- Parameters

  String opcServer - The name of the OPC server to browse

  String device - The name of the device to browse

  String folderPath - Filters on a folder path. Use * as a wildcard for any number of characters and a ? for a single character.

  String opcItemPath - Filters on a OPC item path. Use * as a wildcard for any number of characters and a ? for a single character.

- Returns

  OPCBrowseTag[] - An array of OPCBrowseTag objects. OPCBrowseTag has the following functions: getOpcServer(), getOpcItemPath(), getType(), getDisplayName(), getDisplayPath(), getDataType().

- Scope

  All

Code Examples

Code Snippet

```
#This example will print out the the OPC item path for each item in a specific folder,

#Browse Ignition's OPC-UA Server. This can be changed to match any connected OPC server.
server = "Ignition OPC-UA Server"

#Focus on the "SLC" device connection. This must match a valid device connection in the OPC server.
device = "SLC"

#Specify that the folder path should contain "B3".
folderPath = "*B3*"

#This example is not filtering on a specific OPCItemPath, so it pass Python's None for this parameter
opcItemPath = None

#Call browseSimple and store the results in a variable. Note that it may take some time to complete the
browse.
OpcObjects = system.opc.browseSimple(server, device, folderPath, opcItemPath)

#For each returned address, print out the
for address in OpcObjects:
        print address.getOpcItemPath()
```

system.opc.getServers

Description

Returns a list of server names.

Syntax

system.opc.getServers()

- Parameters

  none

- Returns

  List - A list of server name strings. If no servers are found, returns an empty list.

- Scope

  All

Code Examples

Code Snippet

```
# print a list of all server names found
servers = system.opc.getServers()
if not servers:
 print "No servers found"
else:
 for server in servers:
     print server
```

system.opc.getServerState

Description

Retreives the current state of the given OPC server connection. If the given server is not found, the return value will be None. Otherwise, the return value will be one of these strings:

- UNKNOWN
- FAULTED
- CONNECTING
- CLOSED
- CONNECTED
- DISABLED

Syntax

system.opc.getServerState(opcServer)

- Parameters

  String opcServer - The name of an OPC server connection.

- Returns

  String - A string representing the current state of the connection, or None if the connection doesn't exist.

- Scope

  All

Code Examples

There are not examples associated with this scripting function.

system.opc.readValue

Description

Reads a single value directly from an OPC server connection. The address is specified as a string, for example, [MyDevice]N11/N11:0The object returned from this function has three attributes: value, quality, and timestamp. The value attribute represents the current value for the address specified.

The quality attribute is an OPC-UA status code. You can easily check a good quality vs a bad quality by calling the isGood()function on the quality object. The timestamp attribute is Date object that represents the time that the value was retrieved at.

Syntax

system.opc.readValue(opcServer, itemPath)

- Parameters

  String opcServer - The name of the OPC server connection in which the item resides.

  String itemPath - The item path, or address, to read from.

- Returns

  QualifiedValue - An object that contains the value, quality, and timestamp returned from the OPC server for the address specified.

- Scope

  All

Code Examples

Code Snippet

```
server = "Ignition OPC-UA Server"
path = "[SLCSim]_Meta:N7/N7:0"
qualifiedValue = system.opc.readValue(server, path)
print "Value: " + str(qualifiedValue.getValue())
print "Quality: " + qualifiedValue.getQuality().toString()
print "Timestamp: " + qualifiedValue.getTimestamp().toString()
```

system.opc.readValues

Description

This function is equivalent to the system.opc.readValue function, except that it can operate in bulk. You can specify a list of multiple addresses to read from, and you will receive a list of the same length, where each entry is the qualified value object for the corresponding address.

Syntax

system.opc.readValues(opcServer, itemPaths)

- Parameters

  String opcServer - The name of the OPC server connection in which the items reside.

  String[] itemPaths - A list of strings, each representing an item path, or address to read from.

- Returns

  QualifiedValue[] - A sequence of objects, one for each address specified, in order. Each object will contains the value, quality, and timestamp returned from the OPC server for the corresponding address.

- Scope

  All

Code Examples

There are no examples associated with this scripting function.

system.opc.writeValue

Description

Writes a value directly through an OPC server connection. Will return an OPC-UA status code object. You can quickly check if the write succeeded by calling isGood() on the return value from this function.

Syntax

system.opc.writeValue(opcServer, itemPath, value)

- Parameters

  String opcServer - The name of the OPC server connection in which the item resides.

  String itemPath - The item path, or address, to write to.

  Object value - The value to write to the OPC item.

- Returns

  Quality - The status of the write. Use returnValue.isGood() to check if the write succeeded.

- Scope

  All

Code Examples

Code Snippet

```
server = "Ignition OPC-UA Server"
path = "[SLCSim]_Meta:N7/N7:0"
oldQualifiedValue = system.opc.readValue(server, path)
newValue = oldQualifiedValue.getValue() + 1
returnQuality = system.opc.writeValue(server, path, newValue)
if returnQuality.isGood():
 print "Write was successful"
else:
 print "Write failed"
```

system.opc.writeValues

Description

This function is a bulk version of system.opc.writeValue. It takes a list of addresses and a list of objects, which must be the same length. It will write the corresponding object to the corresponding address in bulk. It will return a list of status codes representing the individual write success or failure for each corresponding address.

Syntax

system.opc.writeValues(opcServer, itemPaths, values)

- Parameters

  String opcServer - The name of the OPC server connection in which the items reside.

  String[] itemPaths - A list of item paths, or addresses, to write to.

  Object[] values - A list of values to write to each address specified.

- Returns

  Quality[] - An array of Quality objects, each entry corresponding in order to the addresses specified.

- Scope

  All

Code Examples

There are no examples associated with this scripting function.

system.print

system.print.createImage

Description

Advanced Function. Takes a snapshot of a component and creates a Java BufferedImage out of it. You can use javax.imageio.ImageIO to turn this into bytes that can be saved to a file or a BLOB field in a database.

Syntax

system.print.createImage(component)

- Parameters

  Component component - The component to render.

- Returns

  BufferedImage - A java.awt.image.BufferedImage representing the component.

- Scope

  Client

Code Examples

There are no examples associated with this scripting function.

system.print.createPrintJob

Description

Provides a general printing facility for printing the contents of a window or component to a printer. The general workflow for this function is that you create the print job, set the options you'd like on it, and then call print() on the job. For printing reports or tables, use those components' dedicated print() functions.

The PrintJob object that this function returns has the following properties that can be set:

Property	Description
showPrintDialog	If true (1), then the print dialog window will be shown before printing. This allows users to specify printing options like orientation, printer, paper size, margins, etc. [default: 1]
fitToPage	If the component is too wide or tall to fit on a page, it will be proportionately zoomed out until it fits into the page. [default: 1]
zoomFactor	If greater than zero, this zoom factor will be used to zoom the printed image in or out. For example, if this is 0.5, the printed image will be half size. If used, this zoom factor overrides the fitToPage parameter. [default: -1.0]
orientation	Either system.print.PORTRAIT or system.print.LANDSCAPE [default: system.print.PORTRAIT]
pageWidth	The width of the paper in inches. [default: 8.5]
pageHeight	The height of the paper in inches. [default: 11]
leftMargin, rightMargin, **topMargin, bottomMargin**	The margins, specified in inches. [default: 0.75]

You can set all of the margins at once with job.setMargins(number), and you initiate the printing with job.print().

Syntax

system.print.createPrintJob(component)

- Parameters

  Component component - The component that you'd like to print.

- Returns

  JythonPrintJob - A print job that can then be customized and started.

- Scope

  Client

Code Examples

Code Snippet

```
#Put this code on a button to print out an image of the container the button is in
job = system.print.createPrintJob(event.source.parent)
job.setMargins(0.5)
job.zoomFactor = 0.75
job.print()
```

system.print.printToImage

Description

This function prints the given component (such as a graph, container, entire window, etc) to an image file, and saves the file where ever the operating system deems appropriate. A filename and path may be provided to determine the name and location of the saved file.

While not required, it is highly recommended to pass in a filename and path. The script may fail if the function attempts to save to a directory that the client does not have access rights to.

Syntax

system.print.printToImage(component)

- Parameters

  Component component - The component to render.

- Returns

  nothing

- Scope

  Client

Syntax

system.print.printToImage(component [, filename])

- Parameters

  Component component - The component to render.

  String filename - A filename to save the image as. [optional]

- Returns

  nothing

- Scope

  Client

Code Examples

Code Snippet

```
#This code would go on a button and save an image of the container that it is in.
system.print.printToImage(event.source.parent, "C:\\temp\\Screen.jpg")
```

Code Snippet - User Selected Location

```
#Again, this example would save an image of the container, but prompts the user for a location and filename
with system.file.saveFile()

#Ask the user for a location. Uses a default filename of "image.png"
path = system.file.saveFile("image.png")

#If the path is not None...
if path != None:
        #Save the file
        system.print.printToImage(event.source.parent, path)
```

system.report

system.report.executeAndDistribute

Description

Executes and distributes a report.

The action parameter supports the following keys as strings:

- email
- print
- save
- ftp

The action settings parameter supports an optional dictionary of settings particular to the action. Missing values will use the default value for that action.

- email
  - Setting Keys: "smtpServerName", "from", "subject", "body", "attachmentName", "retries", "fileType", "to", "cc", "bcc", "useRoles", "roles", "userSource".
  - Note: *To*, *cc*, and *bcc* must be Python lists. If *useRoles* is True, *to*, *cc* and *bcc* will be ignored and all email addresses for all users matching *roles* in userSource (which defaults to the project's current user source) will be in the *to* field. If *useRoles* is true but no *roles* are listed, all user email addresses in *userSource* will be in the *to* field. *fileType* can be pdf, html, csv, rtf, jpeg, png, or xml and is not case-sensitive. If omitted, *fileType* defaults to pdf. *smtpServerName* refers not to the actual name of a SMTP server, but the name given to a SMTP server profile set up in the Gateway.
- print
  - Setting Keys: "primaryPrinterName", "backupPrinterName", "copies", "printBothSides", "collate", "useRaster", "rasterDPI", "useAutoLandscape", "pageOrientation".
  - Note: *primaryPrinterName* defaults to the default printer. *backupPrinterName* defaults to "none", but can also have the special value of "default". *printBothSides, collate,* and *useRaster* are booleans which default to false. *rasterDPI* is only used if *useRaster* is true. *useAutoLandscape* defaults to true. If *useAutoLandscape* is false, *pageOrientation*, which can have values of "portrait" or "landscape" (default is "portrait"), is used.
- save
  - Setting Keys: "path", "fileName" and "format".
  - Note: Since the script is sent to the gateway for execution, path and fileName must be relative to the gateway.
- ftp
  - Setting Keys: "server", "port", "username", "password", "useSSL", "path", "fileName", and "format".
  - Note: Server and fileName are required. If omitted, fileType defaults to pdf, port defaults to 21, and useSSL defaults to false.

Syntax

This function accepts keyword arguments.

system.report.executeAndDistribute(path, project, [parameters], action, [actionSettings])

- Parameters

  String path - The path to the existing report.

  String project - The name of the project where the report is located. Optional in client scope.

  PyDictionary parameters - A optional dictionary of parameter overrides, in the form name:value.

  String action - The name of the distribution action to use.

  PyDictionary actionSettings - An optional dictionary of settings particular to the action. Missing values will use the default value for that action.

- Returns

  None

- Throws

  IllegalArgumentException - Thrown when any of the following occurs: If the file type is not recognized, path does not exist, project does not exist, or a key is not valid.

- Scope

  All

Code Examples

Code Snippet

```
#Executes and distributes the report to an email address.
system.report.executeAndDistribute(path="My Report Path", project="My Project", action= "email",
  actionSettings = {"to":["plantmanager@myplant.com"], "smtpServerName":"myplantMailServer", "from":"
reporting@myplant.com", "subject":"Production Report"})
```

Code Snippet

```
#Executes and distributes the report to all users in the default user source who are Supervisors or
Managers.
system.report.executeAndDistribute(path="My Report Path", project="My Project", action= "email",
  actionSettings = {"useRoles":True, "roles":["Supervisor", "Manager"], "smtpServerName":"
myplantMailServer", "from":"reporting@myplant.com", "subject":"Production Report"})
```

Code Snippet

```
#Executes and distributes the report to an ftp server
settings = {"server":"10.20.1.80", "port":22, "username":"Ignition", "password":"Secret", "useSSL":
False, "path":"C:\\FTP", "fileName":"Ignition Report", and "format":"pdf"}
system.report.executeAndDistribute(path="My Report Path", project="My Project", action= "ftp",
actionSettings = settings)
```

Code Snippet

```
#Executes and distributes the report to save a PDF
settings = {"path":"C:\\Ignition Reports", "fileName":"Report.pdf", "format":"pdf"}
system.report.executeAndDistribute(path="My Report Path", project="My Project", action="save",
actionSettings=settings)
```

system.report.executeReport

Description

Immediately executes an existing report and returns a byte[] of the output.

Syntax

This function accepts keyword arguments.

system.report.executeReport(path, project, [parameters], fileType)

- Parameters

  String path - The path to the existing report.

  String project - The name of the project where the report is located. Optional in client scope.

  PyDictionary parameters - A optional dictionary of parameter overrides, in the form name:value.

  String fileType - The file type the resulting byte array should represent. Acceptable values are "pdf", "html", "csv", "rtf", "jpeg", "png", or "xml". Defaults to "pdf". Not case-sensitive

- Returns

  byte[] - A byte array of the resulting report.

- Throws

  IllegalArgumentException - Thrown when any of the following occurs: If the file type is not recognized, path does not exist, project does not exist.

- Scope

  All

Code Examples

Code Snippet

```
#Executes the report, overriding two parameters
overrides = {"myStringParam":"Hello world", "myIntParam":3}
bytesArray = system.report.executeReport(path="My Path", project="My Project", parameters=overrides,
fileType="pdf")
```

system.report.getReportNamesAsDataset

Description

Gets a data of all reports for a project. This dataset is particularly suited for display in a Tree View component

Syntax

Since 7.8.1

This function accepts keyword arguments.

system.report.getReportNamesAsDataset(project)

- Parameters

  String project - The name of the project where the reports are located. Optional in client scope.

- Returns

  List - A dataset of report paths and names for the project. Returns an empty dataset if the project has no reports.

- Throws

  IllegalArgumentException - Thrown when any of the following occurs: If the project name is omitted in the Gateway scope, project does not exist.

- Scope

  All

Code Examples

Code Snippet

```
# Gets a dataset of reports for the current project and displays
# them in a Tree View component.

event.source.parent.getComponent('Tree View').data = system.report.getReportNamesAsDataset()
```

system.report.getReportNamesAsList

Description

Gets a list of all reports for a project.

Syntax

Since 7.8.1

This function accepts keyword arguments.

system.report.getReportNamesAsList(project)

- Parameters

  String project - The name of the project where the reports are located. Optional in client scope.

- Returns

  List - A list of report paths for the project. Returns an empty list if the project has no reports.

- Throws

  IllegalArgumentException - Thrown when any of the following occurs: If the project name is omitted in the Gateway scope, project does not exist.

- Scope

  All

Code Examples

Code Snippet

```
# Gets a list of reports for the current project and prints it
reports = system.report.getReportNamesAsList()
for report in reports:
    print report
```

Output

```
Comparisons
Line Reports/Line 1/Defect rates
Line Reports/Line 1/Production
Line Reports/Line 2/Defect Rates
```

system.security

system.security.getRoles

Description

Finds the roles that the currently logged in user has, returns them as a Python tuple of strings.

Syntax

system.security.getRoles()

- Parameters

  None

- Returns

  PyTuple - A list of the roles (strings) that are assigned to the current user.

- Scope

  All

Code Examples

Code Snippet

```
#This would run on a button to prevent certain users from opening a window

if "Supervisor" in system.security.getRoles():
   system.nav.openWindow("ManagementOnly")
else:
   system.gui.errorBox("You don't have sufficient privileges to continue")
```

system.security.getUsername

Description

Returns the currently logged-in username.

Syntax

system.security.getUsername()

- Parameters

  none

- Returns

  String - The current user name.

- Scope

  All

Code Examples

Code Snippet

```
#This code would run on a startup script and does special logic based upon who was logging in
name = system.security.getUsername()
if name == 'Bob':
   system.nav.openWindow("BobsHomepage")
else:
   system.nav.openWindow("NormalHomepage")
```

system.security.getUserRoles

Description

Fetches the roles for a user from the Gateway. This may not be the currently logged in user. Requires the password for that user. If the authentication profile name is omitted, then the current project's default authentication profile is used.

Syntax

system.security.getUserRoles(username, password, authProfile, timeout)

- Parameters

  String username - The username to fetch roles for

  String password - The password for the user

  String authProfile - The name of the authentication profile to run against. Optional. Leaving this out will use the project's default profile.

  Integer timeout - Timeout for client-to-gateway communication. (default: 60,000ms)

- Returns

  PyTuple - A list of the roles that this user has, if the user authenticates successfully. Otherwise, returns None.

- Scope

  All

Code Examples

Code Snippet

```
#Fetch the roles for a given user, and check to see if the role "Admin" is in them.

reqRole = "Admin"
username = "Billy"
password= "Secret"
roles = system.security.getUserRoles(username, password)
if reqRole in roles:
   # do something requiring "Admin" role.
```

system.security.isScreenLocked

Description

Returns whether or not the screen is currently locked.

Syntax

system.security.isScreenLocked()

- Parameters

  none

- Returns

  boolean - A flag indicating whether or not the screen is currently locked.

- Scope

  All

Code Examples

Code Snippet

```
#This would run in a timer script to lock the screen after 15 seconds of inactivity, and then log the
user out after 30 seconds of inactivity.

if system.util.getInactivitySeconds() > 15 and not system.security.isScreenLocked():
   system.security.lockScreen()
elif system.util.getInactivitySeconds() > 30:
   system.security.logout()
```

system.security.lockScreen

Description

Used to put a running client in lock-screen mode. The screen can be unlocked by the user with the proper credentials, or by scripting via thesystem.security.unlockScreen() function.

Syntax

system.security.lockScreen([obscure])

- Parameters

  boolean obscure - If true(1), the locked screen will be opaque, otherwise it will be partially visible. [optional]

- Returns

  nothing

- Scope

  Client

Code Examples

Code Snippet

```
#This would run in a timer script to lock the screen after 15 seconds of inactivity, and then log the
user out after 30 seconds of inactivity.

if system.util.getInactivitySeconds() > 15 and not system.security.isScreenLocked():
   system.security.lockScreen()
elif system.util.getInactivitySeconds() > 30:
   system.security.logout()
```

system.security.logout

Description

Shuts-down the currently running client and brings the client to the login screen.

Syntax

system.security.logout()

- Parameters

  none

- Returns

  nothing

- Scope

  All

Code Examples

Code Snippet

```
#This would run in a timer script to log the user out after 30 seconds of inactivity.
if system.util.getInactivitySeconds() > 30:
   system.security.logout()
```

system.security.switchUser

Description

Attempts to switch the current user on the fly. If the given username and password fail, this function will return false. If it succeeds, then all currently opened windows are closed, the user is switched, and windows are then re-opened in the states that they were in.

If an event object is passed to this function, the parent window of the event object will not be re-opened after a successful user switch. This is to support the common case of having a switch-user screen that you want to disappear after the switch takes place.

Syntax

system.security.switchUser(username, password, event, hideError)

- Parameters

  String username - The username to try and switch to.

  String password - The password to authenticate with.

  EventObject event - If specified, the enclosing window for this event's component will be closed in the switch user process.

  Boolean hideError - If true (1), no error will be shown if the switch user function fails. (default: 0)

- Returns

  boolean - false(0) if the switch user operation failed, true (1) otherwise.

- Scope

  Client

Code Examples

Code Snippet

```
#This script would go on a button in a popup window used to switch users without logging out of the
client.

# Pull the username and password from the input components
uname = event.source.parent.getComponent("Username").text
pwd = event.source.parent.getComponent("Password").text

# Call switchUser. The event object is passed to this
# function so that if the username and password work,
# this window will be closed before the switch occurs.
success= system.security.switchUser(uname,pwd,event)

# If the login didn't work, give input focus back to the
# username component, so that the user can try again
if not success:
   event.source.parent.getComponent("Username").requestFocusInWindow()
```

system.security.unlockScreen

Description

Unlocks the client, if it is currently in lock-screen mode.

Syntax

system.security.unlockScreen()

- Parameters

  none

- Returns

  nothing

- Scope

  Client

Code Examples

Code Snippet

```
#This code would go in a global script to automatically unlock the screen on a specific computer

comp = system.net.getHostName()
if comp == 'Line 1':
   system.security.unlockScreen()
```

system.security.validateUser

Description

Tests credentials (username and password) against an authentication profile. Returns a boolean based upon whether or not the authentication profile accepts the credentials. If the authentication profile name is omitted, then the current project's default authentication profile is used.

Syntax

system.security.validateUser(username, password, authProfile, timeout)

- Parameters

  String username - The username to validate

  String password - The password for the user

  String authProfile - The name of the authentication profile to run against. Optional. Leaving this out will use the project's default profile.

  Integer timeout - Timeout for client-to-gateway communication. (default: 60,000ms)

- Returns

  boolean - false(0) if the user failed to authenticate, true(1) if the username/password was a valid combination.

- Scope

  Client

Syntax

system.security.validateUser(username, password, authProfile, timeout)

- Parameters

  String username - User name to validate. Required.

  String password - User's password. Required.

  String authProfile - Authorization profile to user for validation.

  Integer timeout - Not used in gateway scripts.

- Returns

  boolean - True if valid username/password combination.

- Scope

  Gateway

Code Examples

Code Snippet

```
#This would require the current user to enter their password again before proceeding.

currentUser = system.security.getUsername()
password = system.gui.passwordBox("Confirm Password")
valid = system.security.validateUser(currentUser, password)
if valid:
   # do something
else:
   system.gui.errorBox("Incorrect password")
```

system.serial

system.serial.closeSerialPort

Description

Closes a previously opened serial port. Returns without doing anything if the named serial port is not currently open. Will throw an exception if the port is open and cannot be closed.

Syntax

system.serial.closeSerialPort(port)

- Parameters

  String port - The name of the serial port, e.g., "COM1" or "dev/ttyS0".

- Returns

  Nothing

- Scope

  All

Code Examples

There are no examples associated with this scripting function.

system.serial.configureSerialPort

Description

Configure a serial port for use in a later call. This only needs to be done once unless the configuration has changed after the initial call. All access to constants must be prefixed by "system.serial.".

Syntax

This function accepts keyword arguments.

system.serial.configureSerialPort(port, bitRate, dataBits, handshake, hardwareFlowControl, parity, stopBits)

- Parameters

  String port - The name of the serial port, e.g., "COM1" or "/dev/ttyS0". This parameter is required.

  Integer bitRate - Configure the bit rate.Valid values are defined by the following constants: BIT_RATE_110, BIT_RATE_150, BIT_RATE_300, BIT_RATE_600, BIT_RATE_1200, BIT_RATE_2400, BIT_RATE_4800, BIT_RATE_9600, BIT_RATE_19200, BIT_RATE_38400, BIT_RATE_57600, BIT_RATE_115200, BIT_RATE_230400, BIT_RATE_460800, BIT_RATE_921600.

  Integer dataBits - Configure the data bits.Valid values are defined by the following constants: DATA_BITS_5, DATA_BITS_6, DATA_BITS_7, DATA_BITS_8.

  Integer handshake - Configure the handshake.Valid values are defined by the following constants: HANDSHAKE_CTS_DTR, HANDSHAKE_CTS_RTS, HANDSHAKE_DSR_DTR, HANDSHAKE_HARD_IN, HANDSHAKE_HARD_OUT, HANDSHAKE_NONE, HANDSHAKE_SOFT_IN, HANDSHAKE_SOFT_OUT, HANDSHAKE_SPLIT_MASK, HANDSHAKE_XON_XOFF.

  Boolean hardwareFlowControl - Configure hardware flow control. On or off.

  Integer parity - Configure parity.Valid values are defined by the following constants: PARITY_EVEN, PARITY_ODD, PARITY_MARK, PARITY_SPACE, PARITY_NONE.

  Integer stopBits - Configure stop bits.Valid values are defined by the following constants: STOP_BITS_1, STOP_BITS_2.

- Returns

  SerialConfigurator - A SerialConfigurator that can be used to configure the serial port instead of or in addition to the given keyword arguments.

- Scope

  All

Code Examples

Code Snippet

```
#Configure a serial port using keyword args.
#The "port" keyword is mandatory.

system.serial.configureSerialPort(\
port="COM1",\
bitRate=system.serial.BIT_RATE_9600,\
dataBits=system.serial.DATA_BITS_8,\
handshake=system.serial.HANDSHAKE_NONE,\
hardwareFlowControl=False,\
parity=system.serial.PARITY_NONE,\
stopBits=system.serial.STOP_BITS_1)
```

Code Snippet

```
#Configure a serial port using a SerialConfigurator (returned by configureSerialPort()):

system.serial.configureSerialPort("COM1")\
.setBitRate(system.serial.BIT_RATE_9600)\
.setDataBits(system.serial.DATA_BITS_8)\
.setHandshake(system.serial.HANDSHAKE_NONE)\
.setHardwareFlowControl(False)\
.setParity(system.serial.PARITY_NONE)\
.setStopBits(system.serial.STOP_BITS_1)
```

system.serial.openSerialPort

Description

Opens a previously configured serial port for use. Will throw an exception if the serial port cannot be opened.

Syntax

system.serial.openSerialPort(port)

- Parameters

  String port - The name of the serial port, e.g., "COM1" or "dev/ttyS0".

- Returns

  nothing

- Scope

  All

Code Examples

There are no examples associated with this scripting function.

system.serial.readBytes

Description

Read numberOfBytes bytes from a serial port.

Syntax

system.serial.readBytes(port, numberOfBytes [, timeout])

- Parameters

  String port - The previously configured serial port to use.

  int numberOfBytes - The number of bytes to read.

  int timeout - Maximum amount of time, in milliseconds, to block before returning. Default is 5000. [optional]

- Returns

  byte[] - A byte[] containing bytes read from the serial port.

- Scope

  All

Code Examples

There are no examples associated with this scripting function.

system.serial.readBytesAsString

Description

Read numberOfBytes bytes from a serial port and convert them to a String. If a specific encoding is needed to match the source of the data, use system.serial.readBytes and use the desired encoding to decode the byte array returned.

Syntax

system.serial.readBytesAsString(port, numberOfBytes [, timeout])

- Parameters

  String port - The previously configured serial port to use.

  int numberOfBytes - The number of bytes to read.

  int timeout - Maximum amount of time, in milliseconds, to block before returning. Default is 5000. [optional]

- Returns

  String - A String created from the bytes read.

- Scope

  All

Code Examples

There are no examples associated with this scripting function.

system.serial.readLine

Description

Read one line from a serial port.

Syntax

system.serial.readLine(port [, timeout] [, encoding])

- Parameters

  String port - The previously configured serial port to use.

  int timeout - Maximum amount of time, in milliseconds, to block before returning. Default is 5000. [optional]

  String encoding - The String encoding to use. Default is UTF8. [optional]

- Returns

  String - A line of text. A line is considered to be terminated by any one of a line feed ('\n'), a carriage return ('\r'), or a carriage return followed immediately by a line feed.

- Scope

  All

Code Examples

There are no examples associated with this scripting function.

system.serial.readUntil

Description

Reads a byte at a time from a serial port until a delimiter character is encountered. The read will block for up to *timeout* milliseconds before returning.

Syntax

system.serial.readUntil(port, delimiter, includeDelimiter, timeout)

- Parameters

  String port - The previously configured serial port to use.

  char delimiter - The delimiter to read until.

  boolean includeDelimiter - If true, the delimiter will be included in the return value.

  int timeout - Optional timeout in milliseconds. Default is 5000. (Since 7.8.0)

- Returns

  String - Returns a String containing all 8-bit ASCII characters read until the delimiter was reached, and including the delimiter if the "includeDelimiter" parameter was true.

- Scope

  All

Code Examples

There are no examples associated with this scripting function.

system.serial.sendBreak

Description

Sends a break signal for approximately millis milliseconds.

Syntax

system.serial.sendBreak(port, millis)

- Parameters

  String port - The name of the serial port, e.g., "COM1" or "dev/ttyS0".

  int millis - Approximate length of break signal, in milliseconds.

- Returns

  nothing

- Scope

  All

Code Examples

There are no examples associated with this scripting function.

system.serial.write

Description

Write a String to a serial port using the platforms default character encoding.

Syntax

system.serial.write(port, toWrite)

- Parameters

  String port - The previously configured serial port to use.

  String toWrite - The String to write.

- Returns

  nothing

- Scope

  All

Code Examples

There are no examples associated with this scripting function.

system.serial.writeBytes

Description

Write a byte[] to a serial port.

Syntax

system.serial.writeBytes(port, toWrite)

- Parameters

  String port - The previously configured serial port to use.

  byte[] toWrite - The byte[] to write.

- Returns

  nothing

- Scope

  All

Code Examples

There are no examples associated with this scripting function.

system.sfc

Chart Scope Variables

Certain chart scoped variables may interfere with the internal functions of the chart. For example, creating a variable like chart.values will conflict with a pyDictionary's values() method and therefore the chart will show an error. Since SFC charts use Python Dictionaries to manage chart scoped variables the methods associated with Python Dictionary's act like reserved words.

There are a number of built-in variables maintained by the SFC engine that can be read through the `chart` scope.

SFC built-in Variables	Description
chart.instanceId	The string UUID of the running chart instance
chart.startTime	A `java.util.Date` object that indicates when the chart instance started running.
chart.runningTime	An integer representing the number of seconds the chart has been running for.
chart.parent	The chart scope of the enclosing chart (if any). null if this chart was not executed as part of an enclosing step.

Scripting system.sfc.* Functions

See the following links for a description of each function:

system.sfc.cancelChart

Description

Cancels the execution of a running chart instance. Any running steps will be told to stop, and the chart will enter Canceling state.

Syntax

system.sfc.cancelChart(id)

- Parameters

  id -The ID of the chart instance to cancel

- Returns

  Nothing

- Scope

  All

- Throws

  Will throw a KeyError if the ID does not match any running chart instance.

Code Examples

Code Snippet

```
#The following will attempt to stop an SFC but will alert the user if the id of the chart is not
currently running
id = 'Some long string value obtained from earlier in the script'
try:
        system.sfc.cancelChart(id)
except:
        system.gui.messageBox("Could not stop the SFC")
```

system.sfc.getRunningCharts

Description

Retrieves information about all of the charts currently running.

Syntax

system.sfc.getRunningCharts()

- Parameters

  nothing

- Returns

  Dataset

- Scope

  All

Code Examples

There are no examples associated with this scripting function.

system.sfc.pauseChart

Description

Pauses a running chart instance. Any running steps will be told to pause, and the chart will enter Pausing state.

Syntax

system.sfc.pauseChart(id)

- Parameters

  id - The ID of the chart instance to pause

- Returns

  Nothing

- Scope

  All

- Throws

  Will throw a KeyError if the ID does not match any running chart instance.

Code Examples

There are no examples associated with this scripting function.

system.sfc.resumeChart

Description

Resumes a chart that was paused. Steps which were previously paused will be resumed, and chart will enter Resuming state.

Syntax

system.sfc.resumeChart(id)

- Parameters

  id - The ID of the chart instance to resume.

- Returns

  Nothing

- Scope

  All

- Throws

  Will throw a KeyError if the ID does not match any running chart instance.

Code Examples

There are no examples associated with this scripting function.

system.sfc.setVariable

Description

Sets a variable inside a currently running chart.

Syntax

system.sfc.setVariable(instanceId, [stepId], variableName, variableValue)

- Parameters

  String instanceId - The instance identifier of the chart.

  String stepId - [Optional] The id for a step inside of a chart. If omitted the function will target a chart scoped variable.

  String variableName - The name of the variable to set.

  Object variableValue - The value for the variable to be set to.

- Returns

  Nothing

- Scope

  All

Omitting the **stepId** parameter will cause the function to target a chart scoped variable. If the variable is persistent to the whole chart, or used in multiple different steps, then this parameter should be omitted.

If a stepId parameter is used, then the function will target a step scoped variable. **The step associated with the stepId must be the currently active step**.

Code Examples

Code Snippet

```
#The following passes the chart instance ID and step ID to a client message Handler. The message handler
can then wait
#for user input, and then write back to the step variables.

#The example assumes there is a chart scoped variable called confirmEndchart, and a step scoped variable
called "messageSent".

#Get the instanceId of the current chart
chartID = chart.get("instanceId")

#Get the id of the step
stepID = step.get("id")

#Create a payload to pass to the client.
#Include the instanceId and stepId so the script from the message handler knows which
#chart and step to write to
payload = {"chartID" : chartID, "stepID" : stepID}

#Send the message
system.util.sendMessage(project = "SFC", messageHandler = "SFCMessage", payload = payload

############

#The following script would be placed on a client message handler. This receives the payload,
#and sets a variable on either the chart or step depending on user selection

#Read items out of the payload
id = payload['chartID']
stepId = payload['stepID']

#Ask the user to end the chart
if system.gui.confirm("Would you like to end the process"):
        #If yes, end the chart. confirmEndChart is chart scoped, so only 3 parameters are passed
        system.sfc.setVariable(id,"confirmEndChart",True)
else:
        #If no, reset the step.messageSent variable so that the user will be prompted again
        #messageSent is step scoped, so 4 parameters are passed
        system.sfc.setVariable(id,stepId,"messageSent",False)
```

system.sfc.setVariables

Description

Sets any number of variables inside a currently running chart.

Syntax

system.sfc.setVariables(instanceId, [stepId], variableMap)

- Parameters

  String instanceId - The instance identifier of the chart.

  String stepId - [Optional] The id for a step inside of a chart. If omitted the function will target a chart scoped variable.

  PyObject variablesMap - A dictionary containing the name:value pairs of the variables to set.

- Returns

  Nothing

- Scope

  All

Omitting the **stepId** parameter will cause the function to target a chart scoped variable. If the variable is persistent to the whole chart, or used in multiple different steps, then this parameter should be omitted.

If a stepId parameter is used, then the function will target a step scoped variable. **The step associated with the stepId must be the currently active step**.

Code Examples

Code Snippet

```
#Get the instance ID from the selected chart on a SFC Monitor component
id = event.source.parent.getComponent('SFC Monitor').instanceId

#Create a Python dictionary of values. This example assumes there are variables on the
#chart named chartParam and counter. The script will set these to 1, and 0 respectively
dict = {"chartParam":1, "counter":0}

#Set the variables on the chart
system.sfc.setVariables( id, dict)
```

system.sfc.startChart

Description

Starts a new instance of a chart. The chart must be set to "Callable" execution mode.

Syntax

system.sfc.startChart(path, arguments)

- Parameters

  Path - The path to the chart, for example "ChartFolder/ChartName"

  Arguments - A dictionary of arguments. Each key-value pair in the dictionary becomes a variable in the chart scope and will override any default.

- Returns

  String - The unique ID of this chart.

- Scope

  All

Code Examples

Code Snippet

```
#The following will start an SFC with a dictionary of values to use inside the chart
args= {"var1":10, "Var2":15,"Var3":1}
path = "ChartFolder/ChartName"
sfcID = system.sfc.startChart(path, args)
```

system.tag

system.tag.addTag

Description

Adds a new tag in Ignition. You can add OPC, memory, expression, query, folder, and UDT instance tags.

This function accepts keyword arguments.

Syntax

system.tag.addTag(parentPath, name, tagType, dataType, accessRights, enabled, value, attributes, parameters, overrides, alarmList, alarmConfig)

- Parameters

  String parentPath - The folder to add the tag to. Leave blank for the root folder.

  String name - The name of the tag.

  String tagType - The type of tag to create. Possible values are OPC, MEMORY, EXPRESSION, QUERY, Folder, and UDT_INST.

  String dataType - The data type of the tag. Not used for UDT instances or folders.

    Possible basic values are Int1, Int2, Int4, Int8, Float4, Float8, Boolean, String, DataSet, and DateTime.

    Possible array values are Int4Array, Int8Array, Float8Array, BooleanArray, StringArray, DateTimeArray.

  String accessRights - The access rights for a tag. Possible values are Read_Only, Read_Write, and Custom.

  boolean enabled - If true, the tag will be enabled.

  Object value - The value of the tag. Used for memory tags.

  PyDictionary attributes - The tag's configuration attributes.

  PyDictionary parameters - The parameters for a UDT instance tag.

  PyDictionary overrides - All of the overrides for a UDT instance tag.

  String alarmList - List of alarms for the tag.

  PyDictionary alarmConfig - The alarm configuration for the tag.

- Returns

  Nothing

- Scope

  All

If called in the gateway scope, a tag provider must be specified.

Associated attributes:

- Complete list of the acceptable Tag attributes.
- Complete list of the alarmConfig

Code Examples

Code Snippet

```
#Example: Add OPC tag

system.tag.addTag(parentPath='', name="TagOPC", tagType="OPC", dataType="Int2",attributes={"OPCServer":"
Ignition OPC-UA Server", "OPCItemPath":"[MLX]N7:0"})
```

Code Snippet

```
#Example: Add OPC tag with alarm

system.tag.addTag(parentPath='', name="TagOPCAlarm", tagType="OPC", dataType="Int2", attributes=
{"OPCServer":"Ignition OPC-UA Server", "OPCItemPath":"[MLX]N7:0"}, alarmConfig={"Alarm 1":[["name",
"Value", "Alarm 1"], ["setpointA","Value", 1.0], ["CustomEmailSubject","Value","My Subject"]], "Alarm":
[["name", "Value", "Alarm"], ["Something", "Value", "sdfsdfs"], ["enabled", "Expression", "1=2"],
["CustomEmailMessage","Value","My Message"]]})
```

Code Snippet

```
#Example: Add Folder

system.tag.addTag(parentPath='', name="Folder", tagType="Folder")
```

Code Snippet

```
#Example: Add Memory tag

system.tag.addTag(parentPath='', name="TagMemory", tagType="MEMORY", dataType="Int2", value=25)
```

Code Snippet

```
#Example: Add Expression tag

system.tag.addTag(parentPath='', name="TagExpression", tagType="EXPRESSION", dataType="Int2", attributes=
{"Expression":"{[~]Tag1} * 20.5"})
```

Code Snippet

```
#Example: Add Query tag
system.tag.addTag(parentPath='', name="TagQuery", tagType="QUERY", dataType="DateTime", attributes=
{"Expression":"SELECT CURRENT_TIMESTAMP", "SQLBindingDatasource":"MySQL"})
```

Code Snipet

```
#Example: Add Memory tag to a provider other than the default tag provider

system.tag.addTag(parentPath="[ProviderName]Folder", name="TagMemoryProvider", tagType="MEMORY",
dataType="Int2", value=42)
```

Code Snippet

```
#Example: Add UDT instance tag

# Before running this script, there must be a UDT named 'Motor'
# that has a string parameter 'DeviceName' and an integer parameter
# 'MotorNumber'.
system.tag.addTag(parentPath='', name="TagUDT", tagType="UDT_INST", attributes={"UDTParentType":"Motor"},
parameters={"DeviceName":"CLX", "MotorNumber":1})
```

Code Snippet

```
#Example: Add UDT instance tag and override multiple parameters on status tag.

# Before running this script, there must be a UDT named 'SecondUDT'
# that has a string parameter 'DeviceName', an integer parameter
# 'MotorNumber', and a tag named "STATUS".
system.tag.addTag(parentPath='', name="TagUDTParameters", tagType="UDT_INST", attributes=
{"UDTParentType":"SecondUDT"}, parameters={"DeviceName":"CLX", "MotorNumber":2}, overrides={"STATUS":
{"ScanClass":"Default Historical", "Enabled":"false"}})
```

Code Snippet

```
#Example: Add UDT instance tag and override the scan class of a tag inside of another UDT

# Before running this script, there must be a UDT named 'ThirdUDT'.
# ThirdUDT must contain another UDT named 'SecondUDT', which
# that has a string parameter 'DeviceName', an integer parameter
# 'MotorNumber' and a tag named "STATUS".
system.tag.addTag(parentPath='', name="TagUDTScanClass", tagType="UDT_INST", attributes={"UDTParentType":"
ThirdUDT"}, parameters={"Motor/DeviceName":"CLX", "Motor/MotorNumber":1}, overrides={"Motor/STATUS":
{"ScanClass":"Default Historical"}})
```

Code Snippet

```
#Example: Create a memory tag with a type of DataSet, and pass in a dataset as the initial value of the
newly created tag

#create an intial dataset to pass the tag
initData=system.dataset.fromCSV('"#NAMES"\n"Col 1"\n"#TYPES"\n"I"\n"#ROWS","1"\n"0"\n')

#create the tag
system.tag.addTag(parentPath='', value=initData, name="TagMemoryDataset", tagType="MEMORY", dataType="
DataSet")
```

Code Snippet

```
#Example: Add an Array type Memory tag to the default tag provider

# create a dataset
dataset = system.dataset.toDataSet(["values"], [[1],[2],[3]])

# create the tag
system.tag.addTag(parentPath='', name="TagMemoryArray", tagType="MEMORY", dataType="Int4Array",
value=dataset)
```

Tag Attributes

Description
Tags have many attributes that define how the tag is configured. This are often referred to when creating tags using the system.tag.addTag and system.tag.editTag scripting functions. See below for the complete list of attributes.

Attributes

Value Object
Name String
Quality DataQuality
Enabled Boolean
AccessRights AccessRightsType
OPCServer String
OPCItemPath String
OPCWriteBackServer String
OPCWriteBackItemPath String
ScaleMode Integer
RawLow Double
RawHigh Double
ScaledLow Double
ScaledHigh Double
ClampMode Integer
Deadband Double
FormatString String
EngUnit String
Tooltip String
EngHigh Double
EngLow Double
Documentation String
Expression String
ExpressionType Integer
AlertMode Integer
AlertAckMode Integer
AlertSendClear Integer
AlertMessageMode Integer
AlertMessage String
AlertNotes String
AlertDisplayPath String
AlertDeadband Double
LastChange Date.class
DriverName String
ScanClass String.class
PollRate PollingRate.class
SQLBindingDatasource String
PrimaryHistoryProvider String
HistoricalDeadband Double
HistoryEnabled Boolean
HistoricalScanclass String
InterpolationMode Integer
AlertExecEnabled Boolean
AlertActive Boolean
AlertCurrentState String
AlertCurrentSeverity Integer
AlertAcknowledged Boolean
AlertAcknowledgeUser String
AlertActiveTime Date
AlertAcknowledgedTime Date
AlertClearedTime Date
AlertTimestampSource Integer
AlertMessageSubject String
HistoryTimestampSource Integer
HistoryMaxAgeMode Integer
HistoryMaxAge Integer
DataType DataType.class
TagTypeSubCode Integer
TagType Integer
UDTParentType String
ExtendedProperties PropertySet
PropertyOverrides OverrideMap
UDTMemberUID String.class
AlarmActiveUnackCount Integer
AlarmActiveAckCount Integer
ScaleFactor Double
EngLimitMode Integer
AlarmClearUnackCount Integer
AlarmHighestUnackPriority Integer
AlarmHighestUnackName String
AlarmHighestAckName String
AlarmHighestAckPriority Integer
DeadbandMode Integer
HistoricalDeadbandMode Integer

system.tag.browseHistoricalTags

Description

Browses and returns the historical tags.

Syntax

system.tag.browseHistoricalTags(path, [nameFilters], [maxSize], [continuationPoint])

- Parameters

  String path

  String[] nameFilters

  Integer maxSize

  Object continuationPoint

- Returns

  BrowseResults

- Scope

  All

Code Examples

There are no examples associated with this scripting function.

system.tag.browseTags

Description

Returns an array of tags from a specific folder. The function supports filtering and recursion. Leave filters blank to return all tags.

Note: This function cannot browse Client tags.

Syntax

system.tag.browseTags(parentPath, tagPath, tagType, dataType, udtParentType, recursive, sort)

- Parameters

  String parentPath - The parent folder path. Leave blank for the root folder. Note: you can specify the tag provider name in square brackets at the beginning of the parentPath string. Example: "[myTagProvider]MyTagsFolder". If the tag provider name is left off then the project default provider will be used.

  String tagPath - Filters on a tag path. Use * as a wildcard for any number of characters and a ? for a single character.

  String tagType - The type of tag to create. Possible values are OPC, MEMORY, EXPRESSION, QUERY, FOLDER, and UDT_INST.

  String dataType - The data type of the tag. Not used for UDT instances or folders. Possible values are Int1, Int2, Int4, Int8, Float4, Float8, Boolean, String, and DateTime.

  String udtParentType - The name of the parent UDT.

  boolean recursive - Recursively search for tags inside of folders.

  String sort - Sets the sort order, possible values are ASC and DESC. Sorting is done on the full path of the tag.

- Returns

  BrowseTag[] - An array of BrowseTag. BrowseTag has the following variables: name, path, fullPath, type, dataType, and the following functions: isFolder(), isUDT(), isOPC(), isMemory(), isExpression(), isQuery().

- Scope

  All

If called in the gateway scope, a tag provider must be specified.

Code Examples

Code Snippet

```
#Example 1: Browse all tags in a specific folder

tags = system.tag.browseTags(parentPath="")
for tag in tags:
    print tag.name, tag.path, tag.fullPath, tag.isFolder(), tag.isUDT(),
 print tag.isOPC(), tag.isMemory(), tag.isExpression(), tag.isQuery(),
 print tag.isDB(), tag.type, tag.dataType
```

Code Snippet

```
#Example 2: Browse tags of a the same data type

tags = system.tag.browseTags(parentPath="", dataType="Int2")
```

Code Snippet

```
#Example 3: Browse tags of a the same UDT parent type

tags = system.tag.browseTags(parentPath="", udtParentType="Motor")
```

Code Snippet

```
#Example 4: Browse tags of a the same type

tags = system.tag.browseTags(parentPath="", tagType="OPC")
```

Code Snippet

```
#Example 5: Browse tags using a tag path filter

tags = system.tag.browseTags(parentPath="", tagPath="*Folder1")
```

Code Snippet

```
#Example 6: Recursively browse tags

tags = system.tag.browseTags(parentPath="", recursive=True)
```

Code Snippet

```
#Example 7: Sort tag in DESC order

tags = system.tag.browseTags(parentPath="", sort="DESC")
```

system.tag.browseTagsSimple

Description

Returns a sorted array of tags from a specific folder.

Syntax

system.tag.browseTagsSimple(parentPath, sort)

- Parameters

  String parentPath - The parent folder path. Leave blank for the root folder. Note: you can specify the tag provider name in square brackets at the beginning of the parentPath string. Example: "[myTagProvider]MyTagsFolder". If the tag provider name is left off then the project default provider will be used.

  String sort - Sets the sort order, possible values are ASC and DESC.

- Returns

  BrowseTag[] - An array of BrowseTag. BrowseTag has the following variables: name, path, fullPath, type, dataType, and the following functions: isFolder(), isUDT(), isOPC(), isMemory(), isExpression(), isQuery().

- Scope

  All

If called in the gateway scope, a tag provider must be specified.

Code Examples

The following script will print out the names of all tags and the tag type:

Code Snippet

```
tags = system.tag.browseTagsSimple("", "ASC")
for tag in tags:
 print "Name:",tag.name, "\tType:", tag.dataType
```

system.tag.editAlarmConfig

Description

Edit the alarm configuration of multiple existing tags in Ignition with a single call.

Syntax

system.tag.editAlarmConfig(tagPaths, alarmConfig)

- Parameters

  String[] tagPaths - The full path to the tag you want to edit. Note: you can specify the tag provider name in square brackets at the beginning of the parentPath string. Example: "[myTagProvider]MyTagsFolder". If the tag provider name is left off then the project default provider will be used.

  PyDictionary alarmConfig - A dictionary of multi-dimensional lists containing the new alarm configuration. The key in the dictionary will be the name of the alarm being to edit, and the value is a list of lists. The nested lists use the format ["name", "Value", "newValue"]. Note that item 1 is always "Value".

  Example: `{"targetAlarm":[["propertyToChange","Value","newValue"],["anotherProperty","Value","anotherNewValue"]]}`

  If the name of the alarm does not match a pre-existing alarm, **a new alarm will be created**. Note that alarm names are case sensitive. Beware of typos.

  A list of scripting names for each alarm property can be found on the Configure Alarm on a Tag page.

- Returns

  nothing

- Scope

  All

Code Examples

Code Snippet - Single Tag, Multiple properties

```
#The following example will alter the alarm configuration on a single tag.
#The tag currently has an alarm named "High Temp". The code below will change the name of the alarm
#to "Low Temp", and change the Setpoint to 100.

#Build a list of tag paths. Only a single tag will be altered, so create a list of one item and store the
list in a variable
tagPaths = ["sensors/B3:0"]

#Build the dictionary of alarm names, and properties to change.
alarmConfig = {"High Temp":[["name","Value","Low Temp"],["setpointA","Value","100"]]}

#Edit the alarm configuration.
system.tag.editAlarmConfig(tagPaths, alarmConfig)
```

Code Snippet - Multiple Tags, Single Alarm per Tag

```
#The following example will disable alarms on multiple tags.
#The tags below both have an alarm named "Alarm"
#This code will edit the Enabled property for both alarms.

#Build a list of tag paths.
tagPaths = ["Tanks/Tank_1/level_PV", "Tanks/Tank_2/level_PV"]

#Build a dictionary of alarms. Both alarms have the name "Alarm", so a dictionary of one item
#will be able to alter both alarms in a single call.
alarmConfig = {"Alarm":[["enabled","Value","0"]]}

#Edit the alarm configuration.
system.tag.editAlarmConfig(tagPaths, alarmConfig)
```

Code Snippet - Multiple Tags, Multiple Alarms per Tag

```
#The following example will create two alarms each on two different tags.
#The code assumes there are not pre-existing alarms on the tags by the name of "High Level" and "Low
Level"
#The Name, Mode, and Setpoint properties will be modified for each alarm.

#Build a list of tag paths.
tagPaths = ["Tanks/Tank_1/level_PV","Tanks/Tank_2/level_PV"]

#Configure two alarms on the tags.
#Mode value of 2 = Above Setpoint
#Mode value of 3 = Below Setpoint
alarmConfig = {"High Level":[["mode","Value","2"],["setpointA","Value",80]], "Low Level":[["mode","
Value","3"],["setpointA","Value",15]]}

#Edit the alarm configuration.
system.tag.editAlarmConfig(tagPaths, alarmConfig)
```

system.tag.editTag

Description

Edits an existing tag in Ignition.

Syntax

system.tag.editTag(tagPath, attributes, parameters, accessRights, overrides, alarmList, alarmConfig)

- Parameters

  String tagPath - The full path to the tag you want to edit. Note: you can specify the tag provider name in square brackets at the beginning of the parentPath string. Example: "[myTagProvider]MyTagsFolder". If the tag provider name is left off then the project default provider will be used.

  PyDictionary attributes - The tag's configuration attributes.

  PyDictionary parameters - The parameters for a UDT instance tag.

  String accessRights - The access rights for the tags. Possible values are Read_Only, Read_Write, and Custom.

  PyDictionary overrides - All of the overrides for a UDT instance tag.

  String alarmList - List of alarms for the tags.

  PyDictionary alarmConfig - The alarm configuration for the tag.

- Returns

  nothing

- Scope

  All

If called in the gateway scope, a tag provider must be specified.

Associated attributes:

- Complete list of the acceptable Tag attributes.
- Complete list of the alarmConfig values coming soon.

Code Examples

Code Snippet

```
#Example 1: Edit OPC tag

system.tag.editTag(tagPath="Tag1",
attributes={"OPCServer":"Ignition OPC-UA Server", "OPCItemPath":"[MLX]N7:2"})
```

Code Snippet

```
#Example 2: Edit UDT instance parameters

system.tag.editTag(tagPath="Tag5", parameters={"DeviceName":"CLX", "MotorNumber":2})
```

Code Snippet

```
#Example 3: Edit UDT instance and override certain parameters

system.tag.editTag(tagPath="Tag8", overrides={"STATUS":{"ScanClass":"Default"}})
```

Code Snippet

```
#Example 4: Edit UDT instance and override multiple parameters

system.tag.editTag(tagPath="Tag8",
overrides={"STATUS":{"ScanClass":"Default", "Enabled":"false"}})
```

Code Snippet

```
#Example 5: Edit UDT instance and remove certain overrides

system.tag.editTag(tagPath="Tag8", parameters={"Param":"Something"},
overrides={"STATUS":{"ScanClass":None}})
```

Code Snippet

```
#Example 6: Enable history on a tag, set the historical scanclass to "Default Historical", and set the
History Provider to the "Data" provider

system.tag.editTag(tagPath="Folder/Tag",attributes={"HistoryEnabled":True, "HistoricalScanclass":"Default
Historical", "PrimaryHistoryProvider":"Data"})
```

system.tag.editTags

Description

Edit multiple existing tags in Ignition with a single call.

Syntax

system.tag.editTags(tagPaths, attributes, parameters, accessRights, overrides, alarmList, alarmConfig)

- Parameters

  String[] tagPaths - The full path to the tag you want to edit. Note: you can specify the tag provider name in square brackets at the beginning of the parentPath string. Example: "[myTagProvider]MyTagsFolder". If the tag provider name is left off then the project default provider will be used.

  PyDictionary attributes - The tag's configuration attributes.

  PyDictionary parameters - The parameters for a UDT instance tag.

  String accessRights - The access rights for a tag. Possible values are Read_Only, Read_Write, and Custom.

  PyDictionary overrides - All of the overrides for a UDT instance tag.

  String alarmList - List of alarms for the tag.

  PyDictionary alarmConfig - The alarm configuration for the tag.

- Returns

  nothing

- Scope

  All

Code Examples

There are no examples associated with this scripting function.

system.tag.exists

Description

Checks whether or not a tag with a given path exists.

Syntax

system.tag.exists(tagPath)

- Parameters

  String tagPath - The path of the tag to look up.

- Returns

  boolean - True if a tag exists for the given path, false otherwise.

- Scope

  All

Code Examples

Code Snippet

```
#This code would write a 1 to the tag "Compressors/C28/ClearFault" if that tag exists.

if system.tag.exists("Compressors/C28/ClearFault"):
   system.tag.write("Compressors/C28/ClearFault", 1)
```

system.tag.getAlarmStates

Description

Returns an array of alarm definitions for a specific tag.

Syntax

system.tag.getAlarmStates(tagPath)

- Parameters

  String tagPath - The full path to the tag. Note: you can specify the tag provider name in square brackets at the beginning of the parentPath string. Example: "[myTagProvider]MyTagsFolder". If the tag provider name is left off then the project default provider will be used.

- Returns

  TagAlarmDefinition[] - An array of TagAlarmDefinition.

- Scope

  All

The properties on the alarm definitions returned by this function do not list what are considered "default" properties; calling getAlarmProperties on an alarm with a Low properity will not return the priority property of the alarm, but if the priority is changed to Medium, then the priority property would be returned. See the Configure Alarm on a Tag page for a full list of properties.

Code Examples

Code Snippet

```
# Get the alarm configuration for a tag. The prop.type represents whether the
# value is static, bound to an expression, or bound to a UDT parameter
tagDefs = system.tag.getAlarmStates("TagAlarm")
for tagDef in tagDefs:
    print tagDef.alarm
    for prop in tagDef.getAlarmProperties():
        print prop.property, prop.type, prop.value
```

system.tag.getAttribute - Deprecated

Description

Returns an any attribute for a specific tag.

Syntax

system.tag.getAttribute(tagPath, attribute)

- Parameters

  String tagPath - The full path to the tag you want to edit. **Note:** you can specify the Tag provider name in square brackets at the beginning of the parentPath string. Example: "[myTagProvider]MyTagsFolder". If the tag provider name is left off then the project default provider will be used.

  String attribute - The name of a tag attribute. See Tag Attributes for more information.

- Returns

  Object - The value of the attribute.

- Scope

  All

Code Examples

Code Snippet

```
#Example 1:

expression = system.tag.getAttribute("Tag3", "Expression")
print expression
```

system.tag.isOverlaysEnabled

Description

Returns whether or not the current client's quality overlay system is currently enabled.

Syntax

system.tag.isOverlaysEnabled()

- Parameters

  none

- Returns

  boolean - True (1) if overlays are currently enabled.

- Scope

  All

Code Examples

There are no examples associated with this scripting function.

system.tag.loadFromFile

Description

This function locates an exported tag file and loads the tags into the specified tag provider.

Syntax

system.tag.loadFromFile(filePath, provider, mode)

- Parameters

  String filePath - The path of the tag file to import from.

  String provider - The name of the provider to import to.

  Integer mode - Dictates what happens if the tag already exists. 0 = overwrite, 1 = ignore.

- Returns

  nothing

- Scope

  All

Code Examples

Code Snippet

```
#The following example loads a file named 'tags.xml' located at C:\
#The backslash character is an escape character in python, so we have to use it twice for the filePath
variable


#Create variables to pass into the function
filePath = 'C:\\tags.xml'
provider = 'default'


#Load the file
system.tag.loadFromFile(filePath,provider,0)
```

system.tag.queryTagCalculations

Description

Queries various calculations (aggregations) for a set of tags over a specified range. Returns a dataset with a row per tag, and a column per calculation.

Syntax

This function accepts keyword arguments.

system.tag.queryTagCalculations(paths, calculations, startDate, endDate, rangeHours, rangeMinutes, aliases, includeBoundingValues, validatesSCExec, noInterpolation, ignoreBadQuality)

- Parameters

  PySequence paths - An array of tag paths (strings) to query calculations for. The resulting dataset will have a row for each tag, and a column for each calculation.

  PySequence calculations - An array of calculations (aggregation functions) to execute for each tag. Valid values are: "Average" (time-weighted), "MinMax", "LastValue", "SimipleAverage", "Sum", "Minimum", and "Maximum".

  Date startDate - The starting point for the calculation window. If omitted, and range is not used, 8 hours before the current time is used.

  Date endDate - The end of the calculation window. If omitted, and range is not used, uses the current time.

  Integer rangeHours - Allows you to specify the query range in hours, instead of using start and end date. Can be positive or negative, and can be used in conjunction with startDate or endDate.

  Integer rangeMinutes - Same as rangeHours, but in minutes.

  PySequence aliases - Aliases that will be used to override the tag path names in the result dataset. Must be 1-to-1 with the tag paths. If not specified, the tag paths themselves will be used.

  Boolean includeBoundingValues - A boolean flag indicating that the system should attempt to load values before and after the query bounds for the purpose of interpolation. The effect depends on the aggregates used. The default is "true".

  Boolean validatesSCExec - A boolean flag indicating whether or not data should be validated against the scan class execution records. If false, calculations may include data that is assumed to be good, even though the system may not have been running. Default is "true"

  Boolean noInterpolation - A boolean flag indicating that the system should not attempt to interpolate values in situations where it normally would, such as for analog tags. Default is "false"

  Boolean ignoreBadQuality - A boolean flag indicating that bad quality values should not be used in the query process. If set, any value with a "bad" quality will be completely ignored in calculations. Default is "false".

- Returns

  Dataset - A dataset representing the calculations over the specified range. There is a row per tag id, and a column per requested calculation. Tag path is returned in the first column.

- Scope

  All

Code Examples

Code Snippet

```
 system.tag.queryTagCalculations(paths=['Historical Tag'], calculations=['Average'],
noInterpolation=False)
```

system.tag.queryTagDensity

Description

Queries the tag history system for information about the density of data. In other words, how much data is available for a given time span.

This function is called with a list of tag paths, and a start and end date. The result set is a two column dataset specifying the timestamp, and a relative weight. Each row is valid from the given time until the next row. The weight is normalized to a value of 1.0 for each tag with data during that time. Thus, for three tag paths passed in, if all tags were present during the span, the result would be 3.0.

Syntax

system.tag.queryTagDensity(paths, startDate, endDate)

- Parameters

  PySequence paths - An array of tag paths (strings) to query.

  Date startDate - The start of the range to query.

  Date endDate - The end of the range to query.

- Returns

  Dataset - A 2-column dataset consisting of a timestamp and a weight. Each row is valid until the next row. The weight is 1 point for each tag with data present.

- Scope

  All

Code Examples

There are no examples associated with this scripting function.

system.tag.queryTagHistory

Description

Issues a query to the Tag Historian. Querying tag history involves specifying the tags and the date range, as well as a few optional parameters. The Tag historian will find the relevant history and then interpolate and aggregate it together into a coherent, tabular result set.

This function takes a list of strings, where each string is a tag path, like "Tanks/Tank5" or "[OracleProvider]Sump/Out2". See also: Tag Paths.

The return size determines how the underlying data is aggregated and/or interpolated. If a distinct return size is specified, that will be the number of rows in the resulting dataset. The special numbers 0 and -1 mean "Natural" and "On-Change", respectively. "Natural" calculates a return size based on the rate of the logging historical scan classes. For example, if you query 1 hour of data for a scan class logging every minute, the natural return size is 60. "On-Change means that you'll get an entry whenever any of the tags under consideration have changed.

Instead of defining a fixed return size, the parameters intervalHours and intervalMinutes can be used. These parameters can be used independently or together to define a "window size". For example, if you defined a 1 hour range, with intervalMinutes=15, you would get 4 rows as a result.

The span of the query can be specified using startDate and endDate. You can also use rangeHours and rangeMinutes in conjunction with either start or end date to specify the range in dynamic terms. For example, you could specify only "rangeHours=-8" to get the last 8 hours from the current time. Or you could use "startDate='2012-05-30 00:00:00', rangeHours=12" to get the first half of the day for May 30th, 2012. The aggregation mode is used when the data is denser than what you asked for. This happens when using fixed return sizes, as there will often be multiple raw values for the window interval defined. Another common operation is to set the return size to 1, in order to use these aggregate functions for calculation purposes. The available functions are:

- "MinMax" - will return two entries per time slice - the min and the max.
- "Average" - will return the time-weighted average value of all samples in that time slice.
- "LastValue" - returns the most recent actual value to the end of the window.
- "SimpleAverage" - returns the simple mathematical average of the values - ((V1+V2+...+Vn)/n)
- "Maximum" - the maximum value of the window.
- "Minimum" - the minimum value of the window.

Syntax

This function accepts keyword arguments.

system.tag.queryTagHistory(paths, startDate, endDate, returnSize, aggregationMode, returnFormat, columnNames, intervalHours, intervalMinutes, rangeHours, rangeMinutes, aggregationModes, includeBoundingValues, validateSCExec, noInterpolation, ignoreBadQuality)

- Parameters

  PySequence paths - An array of tag paths (strings) to query. Each tag path specified will be a column in the result dataset.

  Date startDate - The earliest value to retrieve. If omitted, 8 hours before current time is used.

  Date endDate - The latest value to retrieve. If omitted, current time is used.

  Integer returnSize - The number of samples to return. -1 will return values as they changed, and 0 will return the "natural" number of values based on the logging rates of the scan class(es) involved. -1 is the default.

  String aggregationMode - The mode to use when aggregating multiple samples into one time slice. Valid values are: "Average" (time-weighted), "MinMax", "LastValue", "SimpleAverage", "Sum", "Minimum", and "Maximum". Default is "Average" (time-weighted).

  String returnFormat - Use "Wide" to have a column per tag queried, or "Tall" to have a fixed-column format. Default is "Wide".

  PySequence columnNames - Aliases that will be used to override the column names in the result dataset. Must be 1-to-1 with the tag paths. If not specified, the tag paths themselves will be used as column titles.

  Integer intervalHours - Allows you to specify the window interval in terms of hours, as opposed to using a specific return size.

  Integer intervalMinutes - Same as intervalHours, but in minutes. Can be used on its own, or in conjunction with intervalHours.

  Integer rangeHours - Allows you to specify the query range in hours, instead of using start and end date. Can be positive or negative, and can be used in conjunction with startDate or endDate.

  Integer rangeMinutes - Same as rangeHours, but in minutes.

  PySequence aggregationModes - A one-to-one list with paths specifying an aggregation mode per column.

  Boolean includeBoundingValues - A boolean flag indicating that the system should attempt to include values for the query bound times if possible. The default for this property depends on the query mode, so unless a specific behavior is desired, it is best to not include this parameter.

  Boolean validateSCExec - A boolean flag indicating whether or not data should be validated against the scan class execution records. If false, data will appear flat (but good quality) for periods of time in which the system wasn't running. If true, the same data would be bad quality during downtime periods.

  Boolean noInterpolation - A boolean flag indicating that the system should not attempt to interpolate values in situations where it normally would. This will also prevent the return of rows that are purely interpolated.

  Boolean ignoreBadQuality - A boolean flag indicating that bad quality values should not be used in the query process. If set, any value with a "bad" quality will be completely ignored in calculations and in the result set.

- Returns

  Dataset - A dataset representing the historian values for the specified tag paths. The first column will be the timestamp, and each column after that represents a tag.

- Scope

  All

Code Examples

Code Snippet

```
#The following example will return a dataset with one row detailing maximum value of a tag named 'Sine'
for the past 5 minutes.
import datetime
startTime = datetime.datetime.now() - datetime.timedelta(minutes=5)
endTime = datetime.datetime.now()
dataSet = system.tag.queryTagHistory(paths=['Sine'], startDate=startTime, endDate=endTime, returnSize=1,
aggregationMode="Maximum", returnFormat='Wide')
```

system.tag.read

Description

Reads the value of the tag at the given tag path. Returns a qualified value object. You can read the value, quality, and timestamp from this object. If the tag path does not specify a tag property, then the Value property is assumed.

You can also read the value of tag attributes by appending the attribute to the tagPath parameter. See the Tag Attributes page for a list of available attributes.

Syntax

system.tag.read(tagPath)

- Parameters

  String tagPath - Reads from the given tag path. If no property is specified in the path, the Value property is assumed.

- Returns

  QualifiedValue - A qualified value. This object has three sub-members: value, quality, and timestamp.

- Scope

  All

Code Examples

Code Snippet

```
#This example would read a value and display it in a message box.

qv = system.tag.read("[]EastSection/ValveG/HOA_bit")
system.gui.messageBox("The value is %d" % qv.value)
```

Code Snippet

```
#This example would check the quality of a tag value, and write it to the database if the quality is good

qv = system.tag.read("[]EastSection/ValveG/HOA_bit")
if qv.quality.isGood():
   system.db.runPrepQuery("INSERT INTO VALVE_TABLE (HOA) VALUES (?)", qv.value)
```

Code Snippet

```
#This example would check the value of the EngHigh attribute on the tag

qv = system.tag.read("[]EastSection/ValveG/HOA_bit.EngHigh")
system.gui.messageBox("The EngHigh value is %d" % qv.value)
```

system.tag.readAll

Description

Reads the values of each tag in the tag path list. Returns a sequence of qualified value objects. You can read the value, quality, and timestamp from each object in the return sequence. Reading in bulk like this is more efficient than calling read() many times.

Syntax

system.tag.readAll(tagPaths)

- Parameters

  String[] tagPaths - A sequence of tag paths to read from.

- Returns

  QualifiedValue[] - A sequence of qualified values corresponding to each tag path given. Each qualified value will have three sub-members: value, quality, and timestamp.

- Scope

  All

Code Examples

Code Snippet

```
#This example would read 5 tags at once and print each of their values

tags = ["Tags/T1", "Tags/T2", "Tags/T3", "Tags/T4", "Tags/T5"]
values = system.tag.readAll(tags)
for x in range(len(tags)):
   print "%s = %s" % (tags[x], values[x])
```

system.tag.removeTag

Description

Removes a tag from Ignition.

Syntax

system.tag.removeTag(tagPath)

- Parameters

  String tagPath - The path to the tag you want to remove. Note: you can specify the tag provider name in square brackets at the beginning of the parentPath string. Example: "[myTagProvider]MyTagsFolder". If the tag provider name is left off then the project default provider will be used.

- Returns

  nothing

- Scope

  All

Code Examples

Code Snippet

```
system.tag.removeTag("Tag1")
```

system.tag.removeTags

Description

Removes multiple tags from Ignition with a single call.

Syntax

system.tag.removeTags(tagPaths)

- Parameters

  String[] tagPaths - An array of the paths to the tags you want to remove. Note: you can specify the tag provider name in square brackets at the beginning of the parentPath string. Example: "[myTagProvider]MyTagsFolder". If the tag provider name is left off then the project default provider will be used.

- Returns

  nothing

- Scope

  All

Code Examples

Code Snippet

```
system.tag.removeTags(["Tag1", "Tag2"])
```

system.tag.setOverlaysEnabled

Description

Enables or disables the component quality overlay system.

Syntax

system.tag.setOverlaysEnabled(enabled)

- Parameters

  boolean enabled - True (1) to turn on tag overlays, false (0) to turn them off.

- Returns

  nothing

- Scope

  All

Code Examples

There are no examples associated with this scripting function.

system.tag.storeTagHistory

Description

Inserts data into the tag history system, allowing Tag history to be recorded via scripting.

The Tag paths are associated with a historical and realtime provider, but they do not necessarily need to exist in the realtime provider. This means records from non-existent Tags can be stored in the Tag History system. Because of this, it is imperative that Tag paths passed to the function are typed precisely, otherwise the history will be stored at an incorrect path.

Syntax

system.tag.storeTagHistory(historyprovider, tagprovider, paths, values [, qualities, timestamps])

- Parameters

  String historyprovider - The historical provider to store to.

  String tagprovider - The name of the realtime tag provder to associate these tags with. The tag provider does not need to exist, and the tag paths do not need to exist in it.

  String[] paths - A list of paths to store. The values, qualities, and timestamps are one-to-one with the paths. A single path may be present multiple times in order to store multiple values.

  Object[] values - A list of values to store.

  Integer[] qualities - A list of integer quality codes corresponding to the values. Quality codes can be found on the Tag Quality and Overlays page. If omitted, GOOD quality will be used.

  Date[] timestamps - A list of Date timestamps corresponding to the values. If omitted, the current time will be used. A java.util.date object may be passed, so the system.date funcitions can be used to return a timestamp.

- Returns

  nothing

- Scope

  All

Code Examples

Example - Single Tag

```
"""
This example stores history for a fictitious tag path in a non-existent Tag provider, but both could be
substituted for actual resources in the project.
Note that the History Provider specified must exist in the system.
"""

histProv = "My History Provider"
tagProv = "My Tag Provider"
paths = ["folder/tag"]
values = [ 10]

#Store the history with the variables declared above.
system.tag.storeTagHistory(histProv, tagProv, paths, values)
```

Example - Single Tag, Multiple Entries

```
"""
Stores multiple records for a single tag path. Could be modified to store more records by increasing the
number of items in each list.
Additionally, different tag paths could be used for each record.
"""

paths = ["folder/tag","folder/tag"]
values = [15, 300]
quals = [192, 192]

#Generate the date: Jan 19th 2017 10:02:44 AM local time
date = system.date.getDate(2017, 0, 19)
histDate = system.date.setTime(date, 10, 02, 44)
dates = [system.date.now(), histDate]

#Store the history with the variables declared above.
system.tag.storeTagHistory("My History Provider", "My Tag Provider", paths, values, quals, dates)
```

system.tag.write

Description

Writes a value to a tag. Note that this function writes asynchronously. This means that the function does not wait for the write to occur before returning - the write occurs sometime later on a different thread.

Syntax

system.tag.write(tagPath, value, suppressErrors)

- Parameters

  String tagPath - The path of the tag to write to.

  Object value - The value to write.

  Boolean suppressErrors - A flag indicating whether or not to suppress errors. (optional, client-only).

- Returns

  int - 0 if the write failed immediately, 1 if it succeeded immediately, and 2 if it is pending.

- Scope

  All

Code Examples

Code Snippet

```
#This code would go on a property change event for a numeric text field to calculate and write a value to
a tag.

if event.propertyName == intValue:
 calcValue = event.newValue * 2.5
 system.tag.write("Tanks/tankHiSP",calcValue)
```

system.tag.writeAll

Description

Performs an asynchronous bulk write. Takes two sequences that must have the same number of entries. The first is the list of tag paths to write to, and then second is a list of values to write. This function is dramatically more efficient than calling write multiple times.

Syntax

system.tag.writeAll(tagPaths, values)

- Parameters

  String[] tagPaths - The paths of the tags to write to.

  Object[] values - The values to write.

- Returns

  int[] - Array of ints with an element for each tag written to: 0 if the write failed immediately, 1 if it succeeded immediately, and 2 if it is pending.

- Scope

  All

Code Examples

Code Snippet

```
#This code write to 5 tags at once.

tags = ["Tags/T1", "Tags/T2", "Tags/T3", "Tags/T4", "Tags/T5"]
values = [2, 4, 8, 16, 32]
values = system.tag.writeAll(tags,values)
```

system.tag.writeAllSynchronous

Description

Performs a synchronous bulk write to tags. This means that you know at the end of this function whether or not the writes succeeded or not. Writes that fail or time out will throw errors. However, this function cannot be called from the event dispatch thread, which means that it cannot be called directly from a GUI event like a button press, without wrapping it in a system.util.invokeAsynchronous. You can call this from project event scripts like timer scripts.

Syntax

system.tag.writeAllSynchronous(tagPaths, values [, timeout])

- Parameters

  String[] tagPaths - The paths of the tags to write to.

  Object[] values - The values to write.

  int timeout - How long to wait in milliseconds before timing out pending writes. The default is 45000 milliseconds. [optional]

- Returns

  nothing

- Scope

  All

Code Examples

Code Snippet

```
#This code write to 5 tags at once, waiting up to 30 seconds for any pending writes to complete.

tags = ["Tags/T1", "Tags/T2", "Tags/T3", "Tags/T4", "Tags/T5"]
values = [2, 4, 8, 16, 32]
timeout = 30000
system.tag.writeAllSynchronous(tags,values,timeout)
```

system.tag.writeSynchronous

Description

Performs a write to a tag, synchronously. This means that you know at the end of this function whether or not the write succeeded or not. A write that fails or times out will throw an error. However, this function cannot be called from the event dispatch thread, which means that it cannot be called directly from a GUI event like a button press, without wrapping it in a system.util.invokeAsynchronous. You can call this from project event scripts like timer scripts.

Syntax

system.tag.writeSynchronous(tagPath, value [, timeout])

- Parameters

  String tagPath - The path of the tag to write to.

  Object value - The value to write.

  int timeout - How long to wait in milliseconds before timing out pending writes. The default is 45000 milliseconds. [optional]

- Returns

  nothing

- Scope

  All

Code Examples

Code Snippet

```
#This code would write the value 1 to a tag. It will continue immediately on success or failure, or wait
up to 38 seconds if the write is pending.

system.tag.writeSynchronous("Tags/T5", 1, 38000)
# This line will not be reached until the tag write succeeds, fails, or has been
#  pending for at least 38 seconds.
```

system.twilio

system.twilio.getAccounts

Description

Return a list of Twilio accounts that have been configured in the gateway

Syntax

system.twilio.getAccounts()

- Parameters

  None

- Returns

  List - A list of configured Twilio accounts

- Scope

  All

Code Examples

Code Snippet

```
#Retrieves a list of twilio accounts and then iterates through the resulting list
#Call system.twilio.getAccounts() and store the returned list into a variable
twilioAccounts = system.twilio.getAccounts()

#Iterate through the list of accounts
for account in twilioAccounts:

	#Prints the account name to the console, but could do something more useful with each account
	print account
```

system.twilio.getAccountsDataset

Description

Return a list of Twilio accounts that have been configured in the gateway as a single-column Dataset

Syntax

system.twilio.getAccounts()

- Parameters

  None

- Returns

  Dataset - A list of configured Twilio accounts as a single-column Dataset

- Scope

  All

Code Examples

Code Snippet

```
#Retrieves a list of Twilio accounts and then passes the data to a Table component's Data property

#Call system.twilio.getAccountsDataset() and store the returned list into a variable
twilioAccounts = system.twilio.getAccounts()


#Pass the dataset to a Table component. The Table is located in the same container as the
#component calling this script
event.source.parent.getComponent('Table').data = twilioAccounts
```

system.twilio.getPhoneNumbers

Description

Returns a list of outgoing phone numbers for a Twilio account. Note that these numbers are supplied by Twilio, and are not defined on a user in Ignition.

Syntax

system.twilio.getPhoneNumbers(accountName)

- Parameters

  String accountName - The Twilio account to retrieve phone numbers for

- Returns

  List - A list of phone numbers for the given Twilio account

- Scope

  All

Code Examples

Code Snippet

```
#Retrieves a list of phone numbers associated with a twilio account and then iterates through the
resulting list
#Checks against a Twilio Profile configured on the gateway by the name of "Twilio Account"

#Call system.twilio.getPhoneNumbers() and store the returned list into a variable
twilioNumbers = system.twilio.getPhoneNumbers("Twilio Account")

#Iterate through the list of numbers
for number in twilioNumbers:

    #Prints the numbers to the console, but could do something more useful with each number
    print number
```

system.twilio.getPhoneNumbersDataset

Description

Return a list of outgoing phone numbers for a Twilio account as a single-column Dataset. Note that these numbers are supplied by Twilio, and are not defined on a user in Ignition.

Syntax

system.twilio.getPhoneNumbersDataset(accountName)

- Parameters

  String accountName - The Twilio account to retrieve phone numbers for

- Returns

  Dataset - A list of phone numbers for the given Twilio account as a single-column Dataset

- Scope

  All

Code Examples

Code Snippet

```
#Retrieves a list of phone numbers associated with a twilio account and then passes the resulting list to
a Table component's Data property.
#Checks against a Twilio Profile configured on the gateway by the name of "Twilio Account"


#Call system.twilio.getPhoneNumbers() and store the returned list into a variable
twilioNumbers = system.twilio.getPhoneNumbersDataset("Twilio Account")

#Pass the dataset to a Table component. The Table is located in the same container as the
#component calling this script
event.source.parent.getComponent('Table').data = twilioNumbers
```

system.twilio.sendSms

Description

Sends a SMS message

Syntax

system.twilio.sendSms(accountName, fromNumber, toNumber, message)

- Parameters

  String accountName - The Twilio account to send the SMS from

  String fromNumber- The outbound phone number belonging to the Twilio account to use

  String toNumber - The phone number of the recipient

  String message - The body of the SMS

- Returns

  Nothing

- Scope

  All

Code Examples

Code Snippet

```
#Send a SMS message.
#Fetch the Twilio account name
#getAccounts() returns a list, so the "[0]" operator is refering to the first item in the list
account = system.twilio.getAccounts()[0]

#Fetch the number associated with the account
fromNumber = system.twilio.getPhoneNumbers(account)

#Fetch a specific user's contact information
#A static value is used below, but system.user.getUser() could be used to retrieved a user's phone number
toNumber = "+19165550101"

#Define the text message
#A static message is used below, but multiple messages could be stored in a database table and retrieved
here
textMessage = "This is the body of a text message"

#Send the message
system.twilio.sendSms(account, fromNumber, toNumber, textMessage)
```

system.user

system.user.addHoliday

The following feature is new in Ignition version **7.8.0**
Click here to check out the other new features

Description

Allows a holiday to be added.

Syntax

- Parameters

  HolidayModel holiday - The holiday to add.

- Returns

  UIResponse - an object with lists of warnings, errors, and info about the success or failure of the add.

- Scope

  All

Code Examples

Code Snippet

```
# This example adds a holiday
def printResponse(responseList):
    if len(responseList) > 0:
        for response in responseList:
            print "", response
    else:
        print " None"

from com.inductiveautomation.ignition.common.user.schedule import HolidayModel
from java.util import Date
holidayName = "Groundhog Day"
d = Date(2016 - 1900, 2, 2)                        # java dates start in 1900
repeatAnnually = False
myHoliday = HolidayModel(holidayName, d, repeatAnnually)
response = system.user.addHoliday(myHoliday)

warnings = response.getWarns()
print "Warnings are:"
printResponse(warnings)

errors = response.getErrors()
print "Errors are:"
printResponse(errors)

infos = response.getInfos()
print "Infos are:"
printResponse(infos)
```

Output

```
Warnings are:
 None
Errors are:
 None
Infos are:
 New holiday "Groundhog Day" added.
```

system.user.addSchedule

The following feature is new in Ignition version **7.8.0**
Click here to check out the other new features

Description

Allows a schedule to be added.

Syntax

- Parameters

  AbstractScheduleModel schedule - The schedule to add.

- Returns

  UIResponse - an object with lists of warnings, errors, and info about the success or failure of the add.

- Scope

  All

Code Examples

Code Snippet

```
# This example tries to add the schedule NewSchedule based on an existing schedule MySchedule, and prints
the results of the action.

# This function prints the response received
def printResponse(responseList):
    if len(responseList) > 0:
        for response in responseList:
            print "", response
    else:
        print " None"

# The main function
mySchedule = system.user.getSchedule("Always")
if mySchedule != None and mySchedule.getType() == "basic schedule":
        mySchedule.setObserveHolidays(False)
        mySchedule.setName("NewSchedule")
        response = system.user.addSchedule(mySchedule)
        warnings = response.getWarns()
        print "Warnings are:"
        printResponse(warnings)

        errors = response.getErrors()
        print "Errors are:"
        printResponse(errors)

        infos = response.getInfos()
        print "Infos are:"
        printResponse(infos)
```

Output

```
Warnings are:
 None
Errors are:
 None
Infos are:
 New schedule "NewSchedule" added.
```

system.user.editHoliday

The following feature is new in Ignition version **7.8.0**
Click here to check out the other new features

Description

Allows a holiday to be edited.

Syntax

- Parameters

  String holidayName - The name of the holiday to edit. Name is case-sensitive.

  HolidayModel holiday - The edited holiday.

- Returns

  UIResponse - an object with lists of warnings, errors, and info about the success or failure of the edit.

- Scope

  All

Code Examples

Code Snippet

```
# This example gets a holiday and edits it

# This function prints the response received
def printResponse(responseList):
    if len(responseList) > 0:
        for response in responseList:
            print "", response
    else:
        print " None"

# The main function
holidayName = "Labor Day"
myHoliday = system.user.getHoliday(holidayName)
if myHoliday != None:
        myHoliday.setRepeatAnnually(False)
        response = system.user.editHoliday(holidayName, myHoliday)

        warnings = response.getWarns()
        print "Warnings are:"
        printResponse(warnings)

        errors = response.getErrors()
        print "Errors are:"
        printResponse(errors)

        infos = response.getInfos()
        print "Infos are:"
        printResponse(infos)
```

Output

```
Warnings are:
 None
Errors are:
 None
Infos are:
 Holiday "Labor Day" updated.
```

system.user.editSchedule

The following feature is new in Ignition version **7.8.0**
Click here to check out the other new features

Description

Allows a schedule to be edited.

Syntax

- Parameters

  String scheduleName - The name of the schedule to edit. Name is case-sensitive.

  AbstractScheduleModel schedule - The edited schedule.

- Returns

  UIResponse - an object with lists of warnings, errors, and info about the success or failure of the edit.

- Scope

  All

Code Examples

Code Snippet

```
# This example tries to edit the schedule MySchedule, and prints the results of the action.

# This function prints the response received
def printResponse(responseList):
    if len(responseList) > 0:
        for response in responseList:
            print "", response
    else:
        print " None"

# The main function
oldScheduleName = "MySchedule"
mySchedule = system.user.getSchedule(oldScheduleName)
if mySchedule != None and mySchedule.getType() == "basic schedule":
        mySchedule.setObserveHolidays(False)
        mySchedule.setName("MyEditedSchedule")
        mySchedule.setDescription("A modified description")
        response = system.user.editSchedule(oldScheduleName, mySchedule)
        warnings = response.getWarns()
        print "Warnings are:"
        printResponse(warnings)

        errors = response.getErrors()
        print "Errors are:"
        printResponse(errors)

        infos = response.getInfos()
        print "Infos are:"
        printResponse(infos)
else:
        print "Basic schedule", oldScheduleName, "not found."
```

Output

```
Warnings are:
 None
Errors are:
 None
Infos are:
 Schedule "MyEditedSchedule" updated.
```

system.user.getHoliday

The following feature is new in Ignition version **7.8.0**
Click here to check out the other new features

Description

Returns a specific holiday.

Syntax

- Parameters

  String holidayName - The name of the holiday to return. Case-sensitive

- Returns

  HolidayModel - The holiday, or None if not found.

- Scope

  All

Code Examples

Code Snippet

```
# This example will get a holiday and print info about it
holidayName = "Labor Day"
holiday = system.user.getHoliday(holidayName)
if holiday == None:
        print holidayName, "not found"
else:
        print holiday.getName(), holiday.getDate(), holiday.isRepeatAnually()
```

Output

```
Labor Day 2015-09-07 00:00:00.0 False
```

system.user.getHolidayNames

The following feature is new in Ignition version **7.8.0**
Click here to check out the other new features

Description

Returns a collection of Strings of all holiday names.

Syntax

- Parameters

  None

- Returns

  List - A list of all holiday names, or an empty list if no holidays are defined.

- Scope

  All

Code Examples

Code Snippet

```
# This example prints the name of every holiday

holidayNames = system.user.getHolidayNames()
for holidayName in holidayNames:
        print holidayName
```

Output

```
Labor Day
Groundhog Day
```

system.user.getHolidays

The following feature is new in Ignition version **7.8.0**
Click here to check out the other new features

Description

Returns a sequence of all of the holidays available.

Syntax

- Parameters

  none

- Returns

  List - A list of holidays.

- Scope

  All

Code Examples

Code Snippet

```
# This example prints information about every holiday
holidays = system.user.getHolidays()
if len(holidays) == 0:
        print "No holidays defined"
for holiday in holidays:
        print holiday.getName(), holiday.getDate(), holiday.isRepeatAnnually()
```

Output

```
Labor Day 2015-09-07 00:00:00.0 False
Groundhog Day 2016-03-02 00:00:00.0 False
```

system.user.getRoles

Description

Returns a sequence of strings representing all of the roles configured in a specific user source.

Syntax

system.user.getRoles(userSource)

- Parameters

  String userSource - The user source to fetch the roles for.

- Returns

  List - A List of Strings that holds all the roles in the user source.

- Scope

  All

Code Examples

Code Snippet

```
#This example will print a list of all user roles in the default datasource:

roles = system.user.getRoles("")
for role in roles:
    print role
```

system.user.getSchedule

The following feature is new in Ignition version **7.8.0**
Click here to check out the other new features

Description

Returns a specific schedule.

Syntax

- Parameters

  String scheduleName - The name of the schedule to return. Case-sensitive

- Returns

  AbstractScheduleModel - The schedule, which can be a BasicSchedule, CompositeSchedule, or another type registered by a module, or None if not found.

- Scope

  All

Code Examples

Code Snippet

```
# This example will get a schedule and print info about it:

# This function handles recursive printing of the different schedule types. Modules can register more
types than listed here.
def printScheduleInfo(aSchedule):
        if aSchedule.getType() == "basic schedule":
                print "Basic schedule type: ",aSchedule.getName(), aSchedule.getDescription(), aSchedule.
isAllDays(), aSchedule.isObserveHolidays()
        elif aSchedule.getType() == "composite schedule":
                compositePieces = aSchedule.getModels()
                print "Composite schedule type:",aSchedule.getName(), aSchedule.getDescription(), " which
is made up of..."
                for piece in compositePieces:
                        printScheduleInfo(piece)
        else:
                print "Other schedule type: ", aSchedule.getName(), aSchedule.getDescription(), aSchedule.
getType(), aSchedule.isObserveHolidays()

# The main function
scheduleName = "MySchedule"
schedule = system.user.getSchedule(scheduleName)
if schedule == None:
        print "Schedule", scheduleName, "was not found"
else:
        printScheduleInfo(schedule)
```

Output

```
Basic schedule type:  MySchedule A description False True
```

system.user.getScheduleNames

The following feature is new in Ignition version **7.8.0**
Click here to check out the other new features

Description

Returns a sequence of strings representing the names of all of the schedules available.

Syntax

- Parameters

  none

- Returns

  List - A List of Strings that holds the names of all the available schedules.

- Scope

  All

Code Examples

Code Snippet

```
#This example will print a list of all available schedules:

schedules = system.user.getScheduleNames()
for schedule in schedules:
        print schedule
```

Output

```
A
Always
B
C
Example
MyComposite
MySchedule
```

system.user.getSchedules

The following feature is new in Ignition version **7.8.0**
Click here to check out the other new features

Description

Returns a sequence of all of the schedules available.

Syntax

- Parameters

  none

- Returns

  List - A list of schedule names. Schedules can be a Basic Schedule, Composite Schedule (composed of exactly two other schedules), or another type registered by a module.

- Scope

  All

Code Examples

Code Snippet

```
# This example will print a list of all available schedules:

# This function handles recursive printing of the different schedule types. Modules can register more
types than listed here.
def printScheduleInfo(aSchedule):
        if aSchedule.getType() == "basic schedule":
                print "Basic schedule type: ",aSchedule.getName(), aSchedule.getDescription(), aSchedule.
isAllDays(), aSchedule.getAllDayTime()
        elif aSchedule.getType() == "composite schedule":
                compositePieces = aSchedule.getModels()
                print "Composite schedule type:",aSchedule.getName(), aSchedule.getDescription(), " which
is made up of..."
                for piece in compositePieces:
                        printScheduleInfo(piece)
        else:
                print "Other schedule type: ", aSchedule.getName(), aSchedule.getDescription(), aSchedule.
getType(), aSchedule.isObserveHolidays()


# The main function
schedules = system.user.getSchedules()
for schedule in schedules:
        printScheduleInfo(schedule)
```

Output

```
Basic schedule type:  A None True 0:00-24:00
Basic schedule type:  Always Built-in schedule that is always available: 24x7x365 True 0:00-24:00
Basic schedule type:  B None True 0:00-24:00
Basic schedule type:  C None True 0:00-24:00
Basic schedule type:  Example An example of a M-F 8am-5pm schedule with a lunch break False 0:00-24:00
Composite schedule type: MyComposite a composite schedule  which is made up of...
Basic schedule type:  A None True 0:00-24:00
Basic schedule type:  B None True 0:00-24:00
Basic schedule type:  MySchedule A description False 0:00-24:00
```

system.user.getUser

Description

Looks up a specific user in a user source, by username. The full User object is returned except for the user's password.

Syntax

system.user.getUser(userSource, username)

- Parameters

  String userSource - The name of the user source to search for the user in.

  String username - The username of the user to search for.

- Returns

  User - A User object.

- Scope

  All

User Object

The "User" object that is returned contains all of the information about that user, except for the user's password. You can access most of the basic user properties via a call to "get" or "getOrDefault" which returns a default value if the requested item is not present. For example:

user.getOrDefault(User.Schedule)

...will return that user's schedule, or the value of "Always" if no schedule has been set as that is the default schedule. The following are the various values you may use in this manner:

- User.Username
- User.FirstName
- User.LastName
- User.Notes
- User.Schedule
- User.Language

In addition to these properties, the user object has other methods on it to retrieve more information:

- **User.getId()** - returns the internal identifier object that the backing user source needs to identify this user
- **User.getRoles()** - returns a sequence of strings representing the roles that this user belongs to
- **User.getContactInfo()** - returns a sequence of ContactInfo objects. Each of these objects will have a contactType and valueproperty representing the contact information, both strings.
- **User.getScheduleAdjustments()** - returns a sequence of ScheduleAdjustment objects. Each of these objects will have two date properties, "start" and "end", a boolean property, "available", and a string property called "note".
- **User.getPath()** - returns a QualifiedPath object that represents this user in a deterministic manner.

Code Examples

Code Snippet

```
#This example will print the first and last name of the current user using the default datasource:
userName = system.security.getUsername()
user = system.user.getUser("", userName)
print user.get(user.FirstName) + " " + user.get(user.LastName)
```

system.user.getUsers

Description

Retrieves the list of users in a specific user source. The "User" objects that are returned contain all of the information about that user, except for the user's password.

Syntax

system.user.getUsers(userSource)

- Parameters

  String userSource - The name of the user source to find the users in.

- Returns

  List - A List of User objects.

- Scope

  All

User Object

You can access most of the basic user properties via a call to "get" or "getOrDefault" which returns a default value if the requested item is not present. For example:

user.getOrDefault(User.Schedule)

...will return that user's schedule, or the value of "Always" if no schedule has been set as that is the default schedule. The following are the various values you may use in this manner:

- User.Username
- User.FirstName
- User.LastName
- User.Notes
- User.Schedule
- User.Language

In addition to these properties, the user object has other methods on it to retrieve more information:

- **User.getId()** - returns the internal identifier object that the backing user source needs to identify this user
- **User.getRoles()** - returns a sequence of strings representing the roles that this user belongs to
- **User.getContactInfo()** - returns a sequence of ContactInfo objects. Each of these objects will have a contactType and valueproperty representing the contact information, both strings.
- **User.getScheduleAdjustments()** - returns a sequence of ScheduleAdjustment objects. Each of these objects will have two date properties, "start" and "end", a boolean property, "available", and a string property called "note".
- **User.getPath()** - returns a QualifiedPath object that represents this user in a deterministic manner.

Code Examples

Code Snippet

```
#This example will print the first and last name of all users, using the default datasource:

users = system.user.getUsers("")
for user in users:
    print user.get(user.FirstName) + " " + user.get(user.LastName)
```

system.user.removeHoliday

The following feature is new in Ignition version **7.8.0**
Click here to check out the other new features

Description

Allows a holiday to be deleted.

Syntax

- Parameters

  String holidayName - The name of the holiday to delete. Name is case-sensitive.

- Returns

  UIResponse - an object with lists of warnings, errors, and info about the success or failure of the deletion

- Scope

  All

Code Examples

Code Snippet

```
def printResponse(responseList):
        if len(responseList) > 0:
                for response in responseList:
                        print "", response
        else:
                print " None"

holidayName = "Labor Day"
response = system.user.removeHoliday(holidayName)

warnings = response.getWarns()
print "Warnings are:"
printResponse(warnings)

errors = response.getErrors()
print "Errors are:"
printResponse(errors)

infos = response.getInfos()
print "Infos are:"
printResponse(infos)
```

Output

```
Warnings are:
 None
Errors are:
 None
Infos are:
 Holiday "Labor Day" deleted.
```

system.user.removeSchedule

The following feature is new in Ignition version **7.8.0**
Click here to check out the other new features

Description

Allows a schedule to be deleted. Note that schedules which are used in Composite Schedules can not be deleted until they are removed from the Composite Schedule.

Syntax

- Parameters

  String scheduleName - The name of the schedule to delete. Name is case-sensitive.

- Returns

  UIResponse - an object with lists of warnings, errors, and info about the success or failure of the deletion

- Scope

  All

Code Examples

Code Snippet

```
 # This example tries to delete the schedule MySchedule, and prints the results of the action.

def printResponse(responseList):
        if len(responseList) > 0:
                for response in responseList:
                        print "", response
        else:
                print " None"

scheduleName = "MySchedule"
response = system.user.removeSchedule(scheduleName)

warnings = response.getWarns()
print "Warnings are:"
printResponse(warnings)

errors = response.getErrors()
print "Errors are:"
printResponse(errors)

infos = response.getInfos()
print "Infos are:"
printResponse(infos)
```

Output

```
Warnings are:
 None
Errors are:
 None
Infos are:
 Schedule "MySchedule" deleted.
```

system.util

system.util.beep

Description

Tells the computer to make a "beep" sound.

Syntax

system.util.beep()

- Parameters

  Nothing

- Returns

  Nothing

- Scope

  Vision Client

Code Examples

There are no examples associated with this scripting function.

system.util.execute

Description

Executes the given commands via the operating system, in a separate process. The commands argument is an array of strings. The first string is the program to execute, with subsequent strings being the arguments to that command.

Syntax

system.util.execute(commands)

- Parameters

  String[] commands - A list containing the command (1st entry) and associated arguments (remaining entries) to execute.

- Returns

  nothing

- Scope

  All

Code Examples

Code Snippet

```
# This code would work on a Windows system to play a sound file.
system.util.execute(["sndrec32", "/play", "/close", "/embedding", "C:\\somethingwrong.wav"])
```

system.util.exit

Description

Exits the running client, as long as the shutdown intercept script doesn't cancel the shutdown event. Set force to true to not give the shutdown intercept script a chance to cancel the exit. Note that this will quit the Client completely. you can use system.security.logout() to return to the login screen.

Syntax

system.util.exit([force])

- Parameters

  boolean force - If true (1), the shutdown-intercept script will be skipped. Default is false (0). [optional]

- Returns

  nothing

- Scope

  Client

Code Examples

Code Snippet

```
# This code would exit the client after confirming with the user.
if system.gui.confirm("Are you sure you want to exit?"):
   system.util.exit()
```

system.util.getAvailableLocales

Description

Returns a collection of strings representing the Locales added to the Translation Manager, such as 'en' for English.

Syntax

system.util.getAvailableLocales()

- Parameters

  none

- Returns

  Collection

- Scope

  Client

Code Examples

There are no examples associated with this scripting function.

system.util.getAvailableTerms

Description

Returns a collection of available terms.

Syntax

system.util.getAvailableTerms()

- Parameters

  none

- Returns

  Collection

- Scope

  Client

Code Examples

There are no examples associated with this scripting function.

system.util.getClientId

Description

Returns a hex-string that represents a number unique to the running client's session. You are guaranteed that this number is unique between all running clients.

Syntax

system.util.getClientId()

- Parameters

  none

- Returns

  String - A special code representing the client's session in a unique way.

- Scope

  Client

Code Examples

Code Snippet

```
# This code would print the current client's id to the debug console.
id = system.util.getClientId()
print id
```

system.util.getConnectionMode

Description

Retrieves this client session's current connection mode. 3 is read/write, 2 is read-only, and 1 is disconnected.

Syntax

system.util.getConnectionMode()

- Parameters

  none

- Returns

  int - The current connection mode for the client.

- Scope

  Client

Code Examples

There are no examples associated with this scripting function.

system.util.getConnectTimeout

Description

Returns the connect timeout in milliseconds for all client-to-gateway communication. This is the maximum amount of time that communication operations to the Gateway will be given to connect. The default is 10,000ms (10 seconds).

Syntax

system.util.getConnectTimeout()

- Parameters

  none

- Returns

  int - The current connect timeout, in milliseconds. Default is 10,000 (ten seconds)

- Scope

  Client

Code Examples

Code Snippet

```
# This code would print out the current connect timeout
print system.util.getConnectTimeout()
```

system.util.getEdition

Description

Returns the "edition" of the Vision client - "standard", "limited", or "panel".

Syntax

system.util.getEdition()

- Parameters

  none

- Returns

  String - The edition of the Vision module that is running the client.

- Scope

  Client

Code Examples

There are no examples associated with this scripting function.

system.util.getGatewayAddress

Description

Returns the address of the gateway that the client is currently communicating with.

Syntax

system.util.getGatewayAddress()

- Parameters

  none

- Returns

  String - the address of the Gateway that the client is communicating with.

- Scope

  Client

Code Examples

Code Snippet

```
# This code would open up the gateway config page.
address = system.util.getGatewayAddress()
system.net.openURL("%s/web/config/" % address)
```

system.util.getGatewayStatus

Description

Returns a string that indicates the status of the Gateway. A status of RUNNING means that the Gateway is fully functional. Thrown exceptions return "ERROR" with the error message appended to the string. This function can be used to test all 7.7 and later Gateways. The function can also be used to test 7.6 (7.6.4 and later) and 7.5 (7.5.11 and later) Gateways. Attempting to test Gateways older than these versions will return errors.

Syntax

system.util.getGatewayStatus(gatewayAddress, connectTimeoutMillis, socketTimeoutMillis)

- Parameters

  String gatewayAddress - The gateway address to ping, in the form of ADDR:PORT/main.

  Integer connectTimeoutMillis - Optional. The maximum time in milliseconds to attempt to initially contact a Gateway.

  Integer socketTimeoutMillis - Optional. The maximum time in milliseconds to wait for a response from a Gateway after initial connection has been established.

- Returns

  String - A string that indicates the status of the Gateway. A status of RUNNING means that the Gateway is fully functional.

- Scope

  Client

Code Examples

Code Snippet

```
def checkRemoteGateway():
 import system

 status = system.util.getGatewayStatus("10.20.6.253:8088/main")

 if status == "RUNNING":
        print "Central Gateway is available!"
 else:
        print "Error: " + status

# It's important to do this as an asynchronous operation, as the method
# may block for some time.
system.util.invokeAsynchronous(checkRemoteGateway)
```

system.util.getGlobals

Description

This method returns a dictionary that provides access to the legacy global namespace. As of version 7.7.0, most new scripts use the modern style of scoping, which makes the 'global' keyword act very differently. Most importantly, the modern scoping rules mean that variables declared as 'global' are only global within that one module. The system.util.getGlobals() method can be used to interact with older scripts that used the old meaning of the 'global' keyword.

Syntax

system.util.getGlobals()

- Parameters

  none

- Returns

  PyStringMap - The global namespace, as a dictionary.

- Scope

  All

Code Examples

Code Snippet

```
# Read and print out global variable 'foo'
print system.util.getGlobals()['foo']
```

Code Snippet

```
 # Write value 'hello' to global variable 'foo'
system.util.getGlobals()['foo'] = 'hello'
```

system.util.getInactivitySeconds

Description

Returns the number of seconds since any keyboard or mouse activity. Note - this function will always return zero in the Designer.

Syntax

system.util.getInactivitySeconds()

- Parameters

  none

- Returns

  long - The number of seconds the mouse and keyboard have been inactive for this client.

- Scope

  Client

Code Examples

Code Snippet

```
# This code could run in a global timer script.
# After a 5-minute timeout, navigate back to the home screen
if system.util.getInactivitySeconds()>300 and system.nav.getCurrentWindow()!="Home":
   system.nav.swapTo("Home")
```

system.util.getLocale

Description

Returns the current string representing the user's Locale, such as 'en' for English.

Syntax

system.util.getLocale()

- Parameters

  none

- Returns

  String

- Scope

  All

Code Examples

There are no examples associated with this scripting function.

system.util.getLogger

Description

Returns a Logger object that can be used to log messages to the console. Each Logger has a name, which is typically structured hierarchically using periods, indicating where in the project the Logger is used. You can use any naming scheme you like, however a well-planned naming scheme makes finding log entries and setting log levels much easier. Loggers can be shared between scripts simply by giving them the same name. Six levels of logging are available:

- Fatal - A severe error that will cause termination of the script.
- Error - A runtime error or other unexpected condition.
- Warn - An undesired condition, but one that does not interfere with execution.
- Info - An event that should be noted on the console, but is not an error.
- Debug - Detailed information useful in debugging.
- Trace - Highly detailed information.

To view log messages from Gateway scripts, in the Gateway go to Configure > System > Console > Logs. To view log messages from Client scripts, including scripts in components, in the Client go to Help > Diagnostics > Log Viewer, or in the Designer go to Tools > Console. The default logging level is info, meaning that all messages with level info or higher are logged, and messages with a level or debug or trace are discarded. To change the logging level for a Logger in a Gateway script, go to Configure > System > Console > Levels. The new logging level will remain until it is changed or the Gateway is restarted. To change the logging level in a Client script, go to Help > Diagnostics > Logging Levels. Logging levels can not be changed in the Designer. The following methods are available to a Logger:

- Logger.fatal(String) - Logs a message with level fatal.
- Logger.fatalf(String, Args...) - Logs a formatted message with level fatal, using Java's Formatter syntax.
- Logger.error(String) - Logs a message with level error.
- Logger.errorf(String, Args...) - Logs a formatted message with level error, using Java's Formatter syntax.
- Logger.warn(String) - Logs a message with level warn.
- Logger.warnf(String, Args...) - Logs a formatted message with level warn, using Java's Formatter syntax.
- Logger.info(String) - Logs a message with level info.
- Logger.infof(String, Args...) - Logs a formatted message with level info, using Java's Formatter syntax.
- Logger.debug(String) - Logs a message with level debug.
- Logger.debugf(String, Args...) - Logs a formatted message with level debug, using Java's Formatter syntax.
- Logger.trace(String) - Logs a message with level trace.
- Logger.tracef(String, Args...) - Logs a formatted message with level trace, using Java's Formatter syntax.
- Logger.isTraceEnabled() - Returns True is the current log level is at least trace.
- Logger.isDebugEnabled() - Returns True is the current log level is at least debug.
- Logger.isInfoEnabled() - Returns True is the current log level is at least info.

Syntax

system.util.getLogger(name)

- Parameters

  String name - The name of a logger to create.

- Returns

  LoggerEx - A new Logger object used to log informational and error messages.

- Scope

  All

Code Examples

Code Snippet

```
# This code would log a message with level info
logger = system.util.getLogger("myLogger")
logger.info("Hello, world.")
```

Code Snippet

```
# This code would log a formatted message with level info.
# Note the 'f' at the end of the method name.
who = 'Bob Jones'
num = 5
logger = system.util.getLogger("myLogger")
logger.infof("Machine started by %s, employee ID %d", who, num)
```

Code Snippet

```
# This code would check if the debug level is enabled for this logger before
# executing the remaining code. Although not needed for a simple log entry like
# in this example, it can eliminate expensive function calls in a more complex
# log entry.
logger = system.util.getLogger("myLogger")
if logger.isDebugEnabled():
 logger.debug("Hello, world!")
```

system.util.getProjectName

Description

Returns the name of the project that is currently being run.

Syntax

system.util.getProjectName()

- Parameters

  none

- Returns

  String - The name of the currently running project.

- Scope

  Client

Syntax

system.util.getProjectName()

- Parameters

  none

- Returns

  String - The name of the currently running project.

- Scope

  Gateway

Code Examples

Code Snippet

```
# This code would display the name of the currently running project
system.gui.messageBox("You are running project: %s" % system.util.getProjectName())
```

system.util.getProperty

Description

Retrieves the value of a named system property. Some of the available properties are:

- file.separator. The system file separator character. (for example, "/" (unix) or "\" (windows))
- line.separator. The system line separator string. (for example, "\r\n" (carriage return, newline))
- os.arch. Operating system architecture. (for example, "x86")
- os.name. Operating system name. (for example, "Windows XP")
- os.version. Operating system version. (for example, "5.1")
- user.home. User's home directory.
- user.name. User's account name.

Syntax

system.util.getProperty(propertyName)

- Parameters

  String propertyName - The name of the system property to get.

- Returns

  String - The value for the named property.

- Scope

  All

Code Examples

Code Snippet

```
#This script would store the contents of the Text Area component in the users home directory.
homeDir = system.util.getProperty("user.home")
sep = system.util.getProperty("file.separator")
path = "%s%smyfile.txt" %(homeDir, sep)
system.file.writeFile(path, event.source.parent.getComponent("Text Area").text)
```

system.util.getReadTimeout

Description

Returns the read timeout in milliseconds for all client-to-gateway communication. This is the maximum amount of time allowed for a communication operation to complete. The default is 60,000ms (1 minute).

Syntax

system.util.getReadTimeout()

- Parameters

  none

- Returns

  int - The current read timeout, in milliseconds. Default is 60,000 (one minute)

- Scope

  Client

Code Examples

There are no examples associated with this scripting function.

system.util.getSessionInfo

Description

Returns a PyDataSet holding information about all of the sessions (logged-in users) on the Gateway. Optional regular-expression based filters can be provided to filter the username or the username and the project returned.

The PyDataSet returned has these columns:

- username (String)
- project (String)
- address (String)
- isDesigner (Boolean)
- clientId (String)
- creationTime (Date)

Note that this function will not return all sessions across a cluster - only the cluster node that is being communicated with by the client who makes the call.

Syntax

system.util.getSessionInfo([usernameFilter] [, projectFilter])

- Parameters

  String usernameFilter - A regular-expression based filter string to restrict the list by username. [optional]

  String projectFilter - A regular-expression based filter string to restrict the list by project [optional]

- Returns

  PyDataSet - A dataset representing the Gateway's current sessions.

- Scope

  All

Code Examples

Code Snippet

```
# This code would get the entire table of sessions and put it in an adjacent table
table = event.source.parent.getComponent("Table")
sessions = system.util.getSessionInfo()
table.data = system.db.toDataSet(sessions)
```

Code Snippet

```
# This code would count the number of times a user named "billy" is logged in
sessions = system.util.getSessionInfo("billy")
system.gui.messageBox("Billy has %d sessions" % len(sessions))
```

Code Snippet

```
# This code would return session info on all users starting with the letters "bi"
sessions = system.util.getSessionInfo("bi.*")
```

Code Snippet

```
# This code uses a single character wildcard in the username
sessions = system.util.getSessionInfo("bi.ly")
```

Code Snippet

```
# This code would return session info on a user named "bill.smith"
sessions = system.util.getSessionInfo("bill\.smith")
```

system.util.getSystemFlags

Description

Returns an integer that represents a bit field containing information about the currently running system. Each bit corresponds to a public bitmask as defined below. See the examples for tips on how to extract the information in this bit field are in the examples. Note that the tag[System]Client /System/SystemFlags contains the same value.

- system.util.DESIGNER_FLAG. Set if running in the Designer. (1)
- system.util.PREVIEW_FLAG. Set if running in the Designer, and the Designer is in preview mode. (2)
- system.util.CLIENT_FLAG. Set if running as a Client. (4)
- system.util.WEBSTART_FLAG. Set if running as a Client in Web Start mode. (8)
- system.util.APPLET_FLAG. Set if running as a Client in Applet mode. (16)
- system.util.FULLSCREEN_FLAG. Set if running as a Client in full-screen mode. (32)
- system.util.SSL_FLAG. Set if communication to the Gateway is encrypted with SSL. (64)
- system.util.MOBILE_FLAG. Set if currently running a mobile-launched client. (128)
- system.util.STAGING_FLAG. Set if running a staging client. (256)

Syntax

system.util.getSystemFlags()

- Parameters

  none

- Returns

  int - The system flags integer.

- Scope

  All

Code Examples

There are no examples associated with this scripting function.

system.util.invokeAsynchronous

Description

This is an advanced scripting function. Invokes (calls) the given Python function on a different thread. This means that calls to invokeAsynchronous will return immediately, and then the given function will start executing asynchronously on a different thread. This is useful for long-running data intensive functions, where running them synchronously (in the GUI thread) would make the GUI non-responsive for an unacceptable amount of time.

> **Warning!**
>
> Under no circumstances should you ever do anything in the function that is invoked asynchronously that interacts with the GUI. This means things like window navigation, setting and getting component properties, showing error/message popups, etc. If you need to do something with the GUI in this function, this must be achieved through a call to system.util.invokeLater.

Syntax

system.util.invokeAsynchronous(function)

- Parameters

  PyObject function - A Python function object that will get invoked with no arguments in a separate thread.

- Returns

  Thread thread - the executing thread.

- Scope

  All

Code Examples

Code Snippet

```
# This code would do some data-intensive processing, and then call
# back to the GUI to let it know that it is finished.
# We use default function parameters to pass the root container into these
# functions. (See a Python reference if you don't understand this)

def longProcess(rootContainer = event.source.parent):
   import system
   # Do something here with the database that takes a long time
   results = ...( something )
   # Now we'll send our results back to the UI
   def sendBack(results = results, rootContainer = rootContainer):
      rootContainer.resultsProperty = results
   system.util.invokeLater(sendBack)

system.util.invokeAsynchronous(longProcess) #Note that this is 'longProcess' instead of 'longProcess()'
```

system.util.invokeLater

Description

This is an advanced scripting function. Invokes (calls) the given Python function object after all of the currently processing and pending events are done being processed, or after a specified delay. The function will be executed on the GUI, or event dispatch, thread. This is useful for events likepr opertyChange events, where the script is called before any bindings are evaluated.

If you specify an optional time argument (number of milliseconds), the function will be invoked after all currently processing and pending events are processed plus the duration of that time.

Syntax

system.util.invokeLater(function [, delay])

- Parameters

  PyObject function - A Python function object that will be invoked later, on the GUI, or event-dispatch, thread with no arguments.

  int delay - A delay, in milliseconds, to wait before the function is invoked. The default is 0, which means it will be invoked after all currently pending events are processed. [optional]

- Returns

  nothing

- Scope

  Client

Code Examples

Code Snippet

```
# The code in the update/refresh button uses the 'date' property on the two
# calendar components, which are bound to the current_timestamp property on their
# parent. We want to simulate a button press when the window opens, but only
# after the date properties' bindings have been evaluated.

if event.propertyName == 'current_timestamp':
   # Define a function to click the button
   def clickButton(button = event.source.parent.getComponent('Refresh')):
      import system
      button.doClick()
      system.gui.messageBox("Button has been clicked!")

   # Tell the system to invoke the function after
   # the current event has been processed
   system.util.invokeLater(clickButton)
```

system.util.jsonDecode

Description

Takes a json String and converts it into a Python object such as a list or a dict. If the input is not valid json, a string is returned.

Syntax

system.util.jsonDecode(jsonString)

- Parameters

  String jsonString - The JSON string to decode into a Python object.

- Returns

  PyObject - The decoded Python object.

- Scope

  All

Code Examples

```
#The following example reads in a JSON string, and converts the string to a Python object.
#The example attempts to read the JSON string from a text file, but this could easily be modified to read
data from a web server.

#Read the JSON string
jsonString = system.file.readFileAsString("C:\tmp\\json.txt")

#Decode the JSON string and store the results into a variable
obj = system.util.jsonDecode(jsonString)

#Do something with the results. The code below prints the datatype of the results to the console.
print type(obj)
```

system.util.jsonEncode

Description

Takes a Python object such as a list or dict and converts into a json string.

Syntax

system.util.jsonEncode(pyObj)

- Parameters

  PyObject pyObj - The Python object to encode into JSON such as a Python list or dictionary.

- Returns

  String - The encoded JSON string.

- Scope

  All

Syntax

system.util.jsonEncode(pyObj, indentFactor)

- Parameters

  PyObject pyObj - The Python object to encode into JSON such as a Python list or dictionary.

  int indentFactor - The number of spaces to add to each level of indentation for prettyprinting

- Returns

  String - The encoded JSON string.

- Scope

  All

Code Examples

Code Snippet

```
#The following example builds a Python dictionary, and converts it to a JSON string

#Build the Python dictionary
employeeDict = {"employees":[{"firstName":"John", "lastName":"Doe"},{"firstName":"Anna", "lastName":"
Smith"},{"firstName":"Peter", "lastName":"Jones"}]}

#Convert the dictionary and store the resulting JSON string in a variable.
jsonString = system.util.jsonEncode(employeeDict)
```

system.util.modifyTranslation

Description

This function allows you to add or modify a global translation.

Syntax

system.util.modifyTranslation(term, translation [, locale])

- Parameters

  String term - The key term to translate.

  String translation - The translated value to store.

  String locale - If specified, the locale code (such as "es") identifying the language of the translation. Otherwise, the currently set language is used.[optional]

- Returns

  nothing

- Scope

  Client

Code Examples

Code Snippet

```
# This code adds or updates a translation into French
# for the world Hello. Note the u in front "Allô!", which
# is needed for Python strings outside of the 7-bit ASCII
# range.
system.util.modifyTranslation("Hello", u"Allô!", "fr")
```

system.util.playSoundClip

Description

Plays a sound clip from a wav file to the system's default audio device. The wav file can be specified as a filepath, a URL, or directly as a raw byte[].

Syntax

system.util.playSoundClip(wavFile)

- Parameters

  String wavFile - A filepath or URL that represents a wav file

- Returns

  nothing

- Scope

  Vision Client

Syntax

system.util.playSoundClip(wavBytes [, volume] [, wait])

- Parameters

  byte[] wavBytes

  double volume - The clip's volume, represented as a floating point number between 0.0 and 1.0 [optional]

  boolean wait - A boolean flag indicating whether or not the call to playSoundClip should wait for the clip to finish before it returns [optional]

- Returns

  nothing

- Scope

  Vision Client

Syntax

system.util.playSoundClip(wavFile [, volume] [, wait])

- Parameters

  String wavFile - A filepath or URL that represents a wav file

  double volume - The clip's volume, represented as a floating point number between 0.0 and 1.0 [optional]

  boolean wait - A boolean flag indicating whether or not the call to playSoundClip should wait for the clip to finish before it returns [optional]

- Returns

  nothing

- Scope

  Vision Client

Code Examples

Code Snippet

```
# This code would play a sound clip at full volume that was located on the current
# host's filesystem. It will not return until the clip in finished playing.
system.util.playSoundClip("C:\\sounds\\siren.wav")
```

Code Snippet

```
# This code would pull a sound clip out of a BLOB field from a database,
#playing it asynchronously at half volume.

query = "SELECT wavBlob FROM sounds WHERE type='alert_high'"
soundData = system.db.runScalarQuery(query)

system.util.playSoundClip(soundData, 0.5, 0)
```

system.util.queryAuditLog

Description

Queries an audit profile for audit history. Returns the results as a dataset.

Syntax

system.util.queryAuditLog(auditProfileName, startDate, endDate, actorFilter, actionFilter, targetFilter, valueFilter, systemFilter, contextFilter)

- Parameters

  String auditProfileName - The name of the audit profile to pull the history from.

  Date startDate - The earliest audit event to return. If omitted, the current time - 8 hours will be used.

  Date endDate - The latest audit event to return. If omitted, the current time will be used.

  String actorFilter - A filter string used to restrict the results by actor.

  String actionFilter - A filter string used to restrict the results by action.

  String targetFilter - A filter string used to restrict the results by target.

  String valueFilter - A filter string used to restrict the results by value.

  String systemFilter - A filter string used to restrict the results by system.

  Integer contextFilter - A bitmask used to restrict the results by context. 0x01 = Gateway, 0x02 = Designer, 0x04 = Client.

- Returns

  Dataset - A dataset with the audit events from the specified profile that match the filter arguments.

- Scope

  Client

Code Examples

There are no examples associated with this scripting function.

system.util.retarget

Description

This function allows you to programmatically 'retarget' the Client to a different project and/or different Gateway. You can have it switch to another project on the same Gateway, or another gateway entirely, even across a WAN. This feature makes the vision of a seamless, enterprise-wide SCADA application a reality.

The retarget feature will attempt to transfer the current user credentials over to the new project / Gateway. If the credentials fail on that project, the user will be prompted for a valid username and password. Once valid authentication has been achieved, the currently running project is shut down, and the new project is loaded.

You can pass any information to the other project through the parameters dictionary. All entries in this dictionary will be set in the global scripting namespace in the other project. Even if you don't specify any parameters, the system will set the variable _RETARGET_FROM_PROJECT to the name of the current project and _RETARGET_FROM_GATEWAY to the address of the current Gateway.

Syntax

system.util.retarget(projectName [, gatewayAddress] [, params] [, startupWindows])

- Parameters

  String projectName - The name of the project to retarget to.

  String gatewayAddress - The address of the Gateway that the project resides on. If omitted, the current Gateway will be used. Format is: "host:httpPort:sslPort/main" [optional]

  PyDictionary params - A dictionary of parameters that will be passed to the new project. They will be set as global variables in the new project's Python scripting environment. [optional]

  String[] startupWindows - A list of window names to use as the startup windows. If omitted, the project's normal startup windows will be opened. If specified, the project's normal startup windows will be ignored, and this list will be used instead. [optional]

- Returns

  nothing

- Scope

  Client

Code Examples

Code Snippet

```
# This code would switch to a project named 'TankControl' on the same Gateway
# as the currently running project
system.util.retarget("TankControl")
```

Code Snippet

```
# This code would switch to a project named 'TankControl' on a
# Gateway located at a different IP address running on port 8080, and
# would open the window named "Graph", and set a global jython variable in the
# new project named "retargetOccured" to the value 1 (one).
system.util.retarget("TankControl", "10.30.2.33:8088/main", {"retargetOccured":1}, ["Graph"])
```

Code Snippet

```
# This code would switch to a project named 'TankControl' on a
# Gateway located at a different IP address using SSL on port 8043
system.util.retarget("TankControl", "10.30.2.34:8088:8043/main")
```

Code Snippet

```
# This code would be put in a button in the target that was retargetted to,
# and act as a 'back' button, that would retarget back to the original project.

# fetch the global values that are automatically created when you retarget
project = system.util.getGlobals()['_RETARGET_FROM_PROJECT']
gateway = system.util.getGlobals()['_RETARGET_FROM_GATEWAY']

# retarget
system.util.retarget(project, gateway)
```

system.util.sendMessage

Description

This function sends a message to clients running under the Gateway, or to a project within the Gateway itself. To handle received messages, you must set up event script message handlers within a project. These message handlers run Jython code when a message is received. You can add message handlers under the "Message" section of the client/Gateway event script configuration dialogs.

Note that messages cannot received within a Designer. However, messages can be sent within the Designer in a script (assuming that read/write comm is enabled).

Syntax

system.util.sendMessage(project, messageHandler, payload, scope, clientSessionId, user, hasRole, hostName, remoteServers)

- Parameters

  String project - The name of the project containing the message handler.

  String messageHandler - The name of the message handler that will fire upon receiving a message.

  PyDictionary payload - Optional. A PyDictionary which will get passed to the message handler. Use "messagePayload" in the message handler to access dictionary variables.

  String scope - Optional. Limits the scope of the message delivery to "C" (clients), "G" (Gateway), or "CG" for clients and the Gateway. Defaults to "C" if the user name, role or host name parameters are set, and to "CG" if none of these parameters are set.

  String clientSessionId - Optional. Limits the message delivery to a client with the specified session ID.

  String user - Optional. Limits the message delivery to clients where the specified user has logged in.

  String hasRole - Optional. Limits the message delivery to any client where the logged in user has the specified user role.

  String hostName - Optional. Limits the message delivery to the client that has the specified network host name.

  List remoteServers (since 7.8.2) - Optional. A list of Strings representing Gateway Server names. The message will be delivered to each server in the list. Upon delivery, the message is distributed to the local Gateway and clients as per the other parameters.

- Returns

  List - A List of Strings containing information about each system that was selected for delivery, where each List item is comma-delimited.

- Scope

  All

Code Examples

Code Snippet

```
#Simple message to both Client and Gateway handlers
project="X"
# It's important that both Gateway and Client versions of this message handler have been created
messageHandler="myMessageHandler"
scope="CG"
myDict = {'first': "Hello", 'second': "World"}
results=system.util.sendMessage(project,messageHandler,myDict)

#Assuming that there is one local client running project X, the results List will contain these Strings:
type=Gateway,project=X,messageHandler=testHandler,filterParams={hostName=, clientSessionId=, scope=CG,
user=, hasRole=},sendStatus=SENT

type=Client,sessionId=65F7A472,clientAddress=127.0.0.1,clientHostName=127.0.0.1,project=X,
messageHandler=testHandler,filterParams={hostName=, clientSessionId=, scope=CG, user=, hasRole=},
sendStatus=SENT
```

Code Snippet

```
#Message to client handlers only where a specified user is logged in)
system.util.sendMessage(project="X",messageHandler="myMessageHandler",scope="C",user="Bob")
```

Code Snippet

```
#Message to remote servers over the Gateway Network (since 7.8.2)
servers = ["agent-8088", "agent-9000"]
system.util.sendMessage(project="X",messageHandler="myMessageHandler",remoteServers=servers)
```

system.util.setConnectionMode

Description

Sets the connection mode for the client session. Normally a client runs in mode 3, which is read-write. You may wish to change this to mode 2, which is read-only, which will only allow reading and subscribing to tags, and running SELECT queries. Tag writes and INSERT / UPDATE / DELETE queries will not function. You can also set the connection mode to mode 1, which is disconnected, all tag and query features will not work.

Syntax

system.util.setConnectionMode(mode)

- Parameters

  int mode - The new connection mode. 1 = Disconnected, 2 = Read-only, 3 = Read/Write.

- Returns

  nothing

- Scope

  All

Code Examples

Code Snippet

```
#This example, which could go in a project's startup script, would check the current username and set the
connection mode to read-only if it is the "guest" user.

username = system.security.getUsername()
if "guest" == username.lower():
   # Set "guest" user to read-only mode
   system.util.setConnectionMode(2)
else:
   system.util.setConnectionMode(3)
```

system.util.setConnectTimeout

Description

Sets the connect timeout for client-to-gateway communication. Specified in milliseconds.

Syntax

system.util.setConnectTimeout(connectTimeout)

- Parameters

  int connectTimeout - The new connect timeout, specified in milliseconds.

- Returns

  nothing

- Scope

  All

Code Examples

Code Snippet

```
# This code would set the current connect timeout to 30 seconds
system.util.setConnectTimeout(30000)
```

system.util.setLocale

Description

Sets the user's current Locale. Any valid Java locale code (case-insensitive) can be used as a parameter, including ones that have not yet been added to the Translation Manager. An invalid locale code will cause an Illegal Argument Exception.

Syntax

system.util.setLocale(locale)

- Parameters

  String locale - A locale code, such as 'en_US' for US English.

- Returns

  nothing

- Scope

  Client

Code Examples

There are no examples associated with this scripting function.

system.util.setReadTimeout

Description

Sets the read timeout for client-to-gateway communication. Specified in milliseconds.

Syntax

system.util.setReadTimeout(readTimeout)

- Parameters

  int readTimeout - The new read timeout, specified in milliseconds.

- Returns

  nothing

- Scope

  Client

Code Examples

There are no examples associated with this scripting function.

system.util.translate

Description

This function allows you to retrieve the global translation of a term from the translation database using the current locale.

Syntax

system.util.translate(term)

- Parameters

  String term - The term to look up.

- Returns

  String - The translated term.

- Scope

  Client

Syntax

system.util.translate(term, locale, strict)

- Parameters

  String term - The term to look up.

  String locale

  Boolean strict

- Returns

  String - The translated term.

- Scope

  Client

Code Examples

There are no examples associated with this scripting function