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.

In this section, we cover all of the built in expression functions available inside Ignition. Each page will have a banner at the top that looks like this:

> This function is used by **Ignition's Expression** language.

This lets you know that you are looking at a function for the expression language.

Advanced

Aggregates

Alarming

Colors

Date and Time

JSON

Logic

Math

String

Translation

Type Casting

Users

Advanced

Advanced Functions

The following functions allow you to interact with Ignition in more advanced ways through expression bindings.

In This Section ...

columnRearrange

This function is used by **Ignition's Expression** language.

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.

Syntax

columnRearrange(dataset[, col...])

- Parameters

  DataSet dataset - The starting dataset.

  string col - *Optional*. Any number of column names, in the order that they should appear. The columns specified must match columns in the original dataset.

- Results

  DataSet - A new dataset with columns in the order specified.

Examples

Code Snippet

```
columnRearrange(fiveColDataset, "secondCol", "thirdCol", "firstCol") // returns a 3 column
Dataset, where the columns are in the given order.
```

columnRename

This function is used by **Ignition's Expression** language.

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.

Syntax

columnRename(dataset[, newName...])

- Parameters

  DataSet dataset - The starting dataset.

  string newName - *Optional*. Any number of new column names. The columns specified must match the number of columns in the original dataset.

- Results

  DataSet - A new dataset with new column names.

Examples

Code Snippet

```
columnRename(twoColDataset, "colOne", "colTwo") // returns a Dataset with columns ["colOne",
"colTwo"]
```

forceQuality

This function is used by **Ignition's Expression** language.

Description

Returns the given value, but overwrites the quality of that value. See the quality codes for a list of potential quality codes. 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])

- Parameters

  object value - The value to force a quality on.

  int qualityCode - *Optional*. The qualityCode to force on the value.

- Results

  object - The value with a forced quality.

Examples

Code Snippet

```
forceQuality({Tanks/Tank15})   //returns the value of the Tank15 tag, but always with a good
quality code.
```

Code Snippet

```
forceQuality({Tanks/Tank15}, 410) //returns the value of the Tank15 tag, but always with a
TAG_DISABLED quality.
```

property

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

This function is used by **Ignition's Expression** language.

Description

Returns an object representing the value of the property at the path specified. It takes a single string as an argument and attempts to lookup the property value at the specified path. What makes this function useful is that the path itself can be the result of an expression, meaning it can be dynamic. Normally, you'd use the expression language's built-in bound-value syntax to use a property value in an expression.

Perspective only

The property() expression function is only accessible in Perspective.

Syntax

property(propertyPath)

- Parameters

  string propertyPath - The property path to the property.

- Results

  object - The value of the property.

Examples

Code Snippet

```
property("this.custom." + {view.params.ControlType})
```

qualifiedValue

This function is used by **Ignition's Expression** language.

Description

Returns the given value, but overwrites the quality of that value. Provides more control over the value than the forceQuality() expression.

Syntax

qualifiedValue(value, level[, subcode, diagnosticMessage])

- Parameters

  object value - The value to force a quality on.

  object level - The level to force on the value. Possible levels are Good or 0, Uncertain or 1, Bad or 2, Error or 3.

  object subcode - *Optional*. The subcode to include with the quality level. See Tag Quality and Overlays for more information on possible subcodes.

  object diagnosticMessage - *Optional*. The a diagnostic message to add to the quality.

- Results

  object - The value with a forced quality.

Examples

Code Snippet

```
qualifiedValue(1, 'bad', 515, 'New Quality')   //returns the value 1 but with a quality of
Bad_Disabled("New Quality")
```

runScript

This function is used by **Ignition's Expression** language.

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 zip code, 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 - Preferred

runScript(scriptFunction, [pollRate], [arg1], [arg2], [arg...])

- Parameters

  string scriptFunction - A single line of python code. Typically the path to a script module.

  int pollRate - *Optional*. The poll rate of the script in milliseconds.

  object arg - *Optional*. Any number of argument objects that will be passed into the given script. This expression function can't make use of keyword invocation, so the order of the arguments passed to runScript represents how the parameters will be passed to the underlying Python function.

- Results

  object - The return value of the specified function.

Syntax - Legacy

runScript(scriptFunction, [pollRate])

- Parameters

  string scriptFunction - A string representing a single line of code, including any arguments that will be passed to the function.

  int pollRate - *Optional*. The poll rate of the script in milliseconds.

- Results

  object - The return value of the specified function.

Examples

Here is our scripting function we are going to run that is located in a Project Library script called **textScript**. The project the script was in was also set as the Gateway Scripting Project.

Code Snippet - Python Function

```
def myFunc(text="Hello World!", moreText="Good bye"):
        return text
```

General Usage

```
// This code block shows how to use runScript without additional parameters

// Preferred syntax
runScript("textScript.myFunc")

// Legacy syntax
runScript("textScript.myFunc()")
```

Passing Arguments

```
// Preferred syntax
runScript("textScript.myFunc", 0, "Hello Again", "See ya later")

// Legacy syntax
runScript("textScript.myFunc('Hello Again', 'See ya later')", 0)
```

Example - Legacy Usage

```
// Legacy syntax using string concatenation
runScript("textScript.myFunc('" +{_gensim_/Writeable/WriteableString1} + "')") // This would run
the function and pass in the value of the WriteableString1 tag.
```

sortDataset

This function is used by **Ignition's Expression** language.

Description

Takes a dataset and returns a sorted version of dataset. The sort order is determined by a single column. This works on numeric, as well as alphanumeric columns. When sorting alphanumerically, contiguous numbers are treated as a single number: you may recognize this as a "natural sort".

Sort Order

The table below represents an example of how alphanumeric values are sorted by the function (assuming a natural sort). Where **Raw Column Values** represents an initial set of values, and the Sorted columns show how the function sorts in **Ascending** and **Descending** order.

Raw Column Values	Sorted - Ascending	Sorted - Descending
a1	a1	Z3
a22	A1	z3
Z3	a4	a77z99
z3	a7z9	a77z4
a4	a22	a22
a77z4	a77z4	a7z9
a77z99	a77z99	a4
a7z9	z3	a1
A1	Z3	A1

Some caveats to be aware of:

- Null values for string columns are sorted first
- Null values for numeric columns are sorted last
- Casing is not used as a method of sorting. If the only difference between two cells is the casing, then the resulting order depends largely on where the cells were in the raw column.

Syntax

sortDataset(dataset, colIndex, [ascending], [naturalOrdering])

- Parameters

  Dataset dataset - The starting dataset.

  int colIndex - The index of the column to sort on.

  bool ascending - A flag indicating whether or not to sort ascending. Defaults to true. [Optional]

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

  bool naturalOrdering - A flag indicating the ordering method. True for natural, false for alphabetical. Defaults to true. [Optional]

- Results

  Dataset - A sorted dataset

Syntax

sortDataset(dataset, colName, [ascending], [naturalOrdering])

- Parameters

  Dataset dataset - The starting dataset.

  string colName- The name of the column to sort on.

  bool ascending - A flag indicating whether or not to sort ascending. Defaults to true. [Optional]

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

  bool naturalOrdering - A flag indicating the ordering method. True for natural, false for alphabetical. Defaults to true. [Optional]

- Results

  Dataset - A sorted dataset

Examples

Code Snippet

```
sortDataset(dataset, 0, true) // returns a Dataset sorted ascending on column 0.
```

Code Snippet

```
sortDataset(dataset, "Column 1", false) // returns a Dataset sorted descending on the column
named "Column 1".
```

tag

This function is used by **Ignition's Expression** language.

Description

Returns an object representing 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. The object returned by the function may need to be converted to a standard data type. Check out the Type Casting functions for more information.

When using the tag() function in a logic function, the tag value will remain subscribed to, even if the logic function chooses a different outcome. This can affect tags that are on a leased scan class.

Syntax

tag(tagPath)

- Parameters

  string tagPath - The tag path to the tag.

- Results

  object - The value of the tag. The object returned by the function may need to be converted to a standard data type using one of the various Type Casting functions.

Examples

Code Snippet

```
tag("Tanks/Tank5") //returns Tank5's value.
```

Code Snippet

```
tag("Tanks/Tank" + {Root Container.TankNum}) //returns the value for the tank represented by the
dynamic property TankNum on the Root Container.
```

Aggregates

Aggregate Functions

The following functions allow you to fetch aggregated values from datasets in expression bindings.

In This Section ...

groupConcat

This function is used by **Ignition's Expression** language.

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, columnIndex, separator)

- Parameters

  DataSet dataset - The starting dataset.

  int columnIndex - The index of the column to concatenate.

  string separator - What will be used to separate each of the values.

- Returns

  string - A string with every value in the specified column of the specified dataset separated by the separator value.

Syntax

groupConcat(dataset, columnName, separator)

- Parameters

  DataSet dataset - The starting dataset.

  string columnname - The name of the column to concatenate.

  string separator - What will be used to separate each of the values.

- Returns

  string - A string with every value in the specified column of the specified dataset separated by the separator value.

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

This function is used by **Ignition's Expression** language.

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.

This function expects the datatype of the column to be **numeric**: other datatypes, such as strings, will throw an exception.

Any null values in the column are ignored.

Syntax

max(dataset, columnIndex)

- Parameters

  DataSet dataset - The dataset to search through.

  int columnIndex - The index of the column to search through. Must be a column index of the provided dataset. Additionally, the datatype of the column must be numeric.

- Returns

  int - The maximum value in that column.

Syntax

max(dataset, columnName)

- Parameters

  DataSet dataset - The dataset to search through.

  string columnName - The name of the column to search through. Must match a column name in the provided dataset. Additionally, the datatype of the column must be numeric.

- Returns

  int - The maximum value in that column.

Syntax

max(value[, value...])

- Parameters

  int/float value - A number. Can be as many values as needed. Can be either a float or an integer.

- Returns

  int - The maximum value in the list of values.

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

Code Snippet

```
max({Root Container.Table.data}, 1) //would return 380
```

Code Snippet

```
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
```

Code Snippet

```
max({SomeValue}, 0) //The following example is a great way to make sure a value never goes below zero:
```

maxDate

This function is used by **Ignition's Expression** language.

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.

Syntax

maxDate(dataset, columnIndex)

- Parameter

  DataSet dataset - The starting dataset to search.

  int columnIndex - The index of the column to search for the max date. Must be a column index of the provided dataset.

- Returns

  Date - The maximum date of the given date column in the given dataset.

Syntax

maxDate(dataset, columnName)

- Parameter

  DataSet dataset - The starting dataset to search.

  string columnName - The name of the column to search for the max date. Must match a column name in the provided dataset.

- Returns

  Date - The maximum date of the given date column in the given dataset.

Syntax

maxDate(date[, date])

- Parameter

  Date date - A date. Can be as many dates as needed.

- Returns

  Date - The maximum date of the given dates.

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

Code Snippet

```
maxDate({Root Container.Table.data}, "AlarmTime") //You could use this expression to get the
date and time for the most recent alarm.
```

Code Snippet

```
maxDate(now(0), addMinutes(now(0), 5)) // This would return the Date that is 5 minutes from now.
```

mean

This function is used by **Ignition's Expression** language.

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, null is returned.

Syntax

mean(dataset, columnIndex)

- Parameters

  DataSet dataset - The dataset to use.

  int columnIndex - The index of the column to use. Must be a column index of the provided dataset.

- Returns

  int/float - The mean of the values in that column.

Syntax

mean(dataset, columnName)

- Parameters

  DataSet dataset - The dataset to use.

  string columnName - The name of the column to search through. Must match a column name in the provided dataset.

- Returns

  int/float - The mean of the values in that column.

Syntax

mean(value[, value...])

- Parameters

  int/float value - A number. Can be as many values as needed. Can be either a float or an integer.

- Returns

  int/float - The mean of the values.

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

Code Snippet

```
mean({Root Container.Table.data}, "Weight") //... would return 5.58675
```

Code Snippet

```
mean(1,2,3) //... would return 2
```

median

This function is used by **Ignition's Expression** language.

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.

Syntax

median(dataset, columnIndex)

- Parameters

  DataSet dataset - The dataset to search through.

  int columnIndex - The index of the column to search through. Must be a column index of the provided dataset.

- Returns

  int/float - The median value in that column.

Syntax

median(dataset, columnName)

- Parameters

  DataSet dataset - The dataset to search through.

  string columnName - The name of the column to search through. Must match a column name in the provided dataset.

- Returns

  int/float - The median value in that column.

Syntax

median(value[, value...])

- Parameters

  int/float value - A number. Can be as many values as needed. Can be either a float or an integer.

- Returns

  int/float - The median value in the list of values.

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

Code Snippet

```
median({Root Container.Table.data}, "Weight") //... would return 5.566
```

```
median(1,2,3,3,10) //... would return 3
```

min

This function is used by **Ignition's Expression** language.

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.

Syntax
min(dataset, columnIndex) • Parameters DataSet dataset - The dataset to search through. int columnIndex - The index of the column to search through. Must be a column index of the provided dataset. • Returns int - The minimum value in that column.

Syntax
min(dataset, columnName) • Parameters DataSet dataset - The dataset to search through. string columnName - The name of the column to search through. Must match a column name in the provided dataset. • Returns int - The minimum value in that column.

Syntax
min(value[, value...]) • Parameters int/float value - A number. Can be as many values as needed. Can be either a float or an integer. • Returns int - The minimum value in the list of values.

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

Code Snippet

```
min({Root Container.Table.data}, 1) //... would return 120
```

Code Snippet

```
min(0, 10/2, 3.14) //... would return 0
```

Code Snippet

```
min({SomeValue}, 180) //This example is a great way to make sure a value never goes above 180
```

minDate

This function is used by **Ignition's Expression** language.

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.

Syntax

minDate(dataset, columnIndex)

- Parameter

  DataSet dataset - The starting dataset to search.

  int columnIndex - The index of the column to search for the max date. Must be a column index of the provided dataset.

- Returns

  Date - The minimum date of the given date column in the given dataset.

Syntax

minDate(dataset, columnName)

- Parameter

  DataSet dataset - The starting dataset to search.

  string columnName - The name of the column to search for the max date. Must match a column name in the provided dataset.

- Returns

  Date - The minimum date of the given date column in the given dataset.

Syntax

minDate(date[, date])

- Parameter

  Date date - A date. Can be as many dates as needed.

- Returns

  Date - The minimum date of the given dates.

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

This function is used by **Ignition's Expression** language.

Description

Calculates the simple 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.

Syntax

stdDev(dataset, columnIndex)

- Parameters

  DataSet dataset - The dataset to search through.

  int columnIndex - The index of the column to search through. Must be a column index of the provided dataset.

- Returns

  int/float - The standard deviation of the values in that column.

Syntax

stdDev(dataset, columnName)

- Parameters

  DataSet dataset - The dataset to search through.

  string columnName - The name of the column to search through. Must match a column name in the provided dataset.

- Returns

  int/float - The standard deviation of the values in that column.

Syntax

stdDev(value[, value...])

- Parameters

  int/float value - A number. Can be as many values as needed. Can be either a float or an integer.

- Returns

  int/float - The standard deviation of the values in the list of values.

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

Code Snippet

```
stdDev({Root Container.Table.data}, "Weight") //... would return 3.4687
```

sum

This function is used by **Ignition's Expression** language.

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.

Syntax

sum(dataset, columnIndex)

- Parameters

  DataSet dataset - The dataset to use.

  int columnIndex - The index of the column to use. Must be a column index of the provided dataset.

- Returns

  int/float - The sum of the values in that column.

Syntax

sum(dataset, columnName)

- Parameters

  DataSet dataset - The dataset to use.

  string columnName - The name of the column to search through. Must match a column name in the provided dataset.

- Returns

  int/float - The sum of the values in that column.

Syntax

sum(value[, value...])

- Parameters

  int/float value - A number. Can be as many values as needed. Can be either a float or an integer.

- Returns

  int/float - The sum of the values.

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

Code Snippet

```
sum({Root Container.Table.data}, 1) //... would return 947
```

Code Snippet

```
sum(1,2,3) //... would return 6
```

Alarming Expressions

Alarming Functions

The following functions allow you to view alarm status in expression bindings.

In This Section ...

isAlarmActive

This function is used by **Ignition's Expression** language.

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.

When calling this from the Gateway scope, the **Tag Provider** must be included in the path.

Syntax

isAlarmActive(tagPath[, alarmName][, pollRate])

- Parameters

  string tagPath - The tagpath to search for active alarms. Supports the wildcard '*'.

  string alarmName - *Optional*. The name of the alarm to search for. Supports the wildcard '*'.

  int pollRate - *Optional*. The poll rate in milliseconds. Only applicable in Vision Clients.

- Returns

  Boolean - True if an alarm is active, False if no active alarms were found.

Examples

Code Snippet

```
isAlarmActive("[default]Tanks/Temp", "Tank_Temp_High")   //when the Tank_Temp_High alarm is
active then this expression returns True.
```

isAlarmActiveFiltered

This function is used by **Ignition's Expression** language.

Description

Returns whether there are active alarms that match the provided criteria. It is more granular than isAlarmActive. The tag path, alarm name, and display path all support wildcards ('*'). The min and max priority expect a number between 0 (diagnostic) and 4 (critical). The pollRate parameter is only applicable in the client scope and is optional.

When calling this from the Gateway scope, the **Tag Provider** must be included in the path.

Syntax

isAlarmActiveFiltered(tagPath, alarmName, displayPath, minPriority, maxPriority, allowCleared, allowAcked, allowShelved [, pollRate])

- Parameters

  string tagPath - The tag path to search for active alarms. Accepts the wildcard '*'.

  string alarmName - The alarm name to search for active alarms. Accepts the wildcard '*'.

  string displayPath - The display path to search for active alarms. Accepts the wildcard '*'.

  int minPriority - The minimum priority of alarms to accept. 0 is Diagnostic, 1 is Low, 2 is Medium, 3 is High, 4 is Critical.

  int maxPriority - The maximum priority of alarms to accept. 0 is Diagnostic, 1 is Low, 2 is Medium, 3 is High, 4 is Critical.

  Bool allowCleared - A flag that indicates whether to accept cleared alarms.

  Bool allowAcked - A flag that indicates whether to accept acknowledged alarms.

  Bool allowShelved - A flag that indicates whether to accept shelved alarms.

  int pollRate - *Optional*. The poll rate of the function in milliseconds. Only applicable in the Client scope.

- Results

  Bool - True if there are active alarms, False if there are not.

Examples

Code Snippet

```
isAlarmActiveFiltered("*", "*", "*", 4, 4, 0, 1, 0)  //when any critical alarm is active, even
if acknowledged, then this expression returns True.
```

Colors

Color Functions

The following functions allow you to modify or set color values in expression bindings.

In This Section ...

brighter

This function is used by **Ignition's Expression** language.

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)

- Parameter

  Color color - A color to make brighter. Can use the color function to create a color value.

- Results

  Color - A color that is one shade brighter than the color passed in.

Examples

Code Snippet

```
brighter(color(100,150,250)) //returns the color (142,214,255)
```

color

This function is used by **Ignition's Expression** language.

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.

Note: This function was designed to return color objects to Vision bindings, and will not work with Perspective bindings. Instead, Perspective color properties can simply use string hex codes to derive a color from a binding. Example: "#00FF00".

Syntax

color(red, green, blue[, alpha])

- Parameter

  int red - The intensity of Red, between 0 - 255.

  int green - The intensity of Green, between 0 - 255.

  int blue - The intensity of Blue, between 0 - 255.

  int alpha - *Optional*. The amount of transparency, between 0 - 255.

- Results

  Color - Returns a color with the given RGB value.

Examples

There are no expression function examples associated with this expression function.

darker

This function is used by **Ignition's Expression** language.

Description

Returns a color that is one shade darker than the color given as an argument.

Syntax

darker(color)

- Parameter

  Color color - A color to make darker. Can use the color function to create a color value.

- Results

  Color - A color that is one shade darker than the color passed in.

Examples

```
darker(color(100,150,250)) //returns the color (70,105,175)
```

gradient

This function is used by **Ignition's Expression** language.

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(value, low, high, lowColor, highColor)

- Parameter

  int value - The value used to determine the percentage between the low and high values.

  int low - The low value to use to calculate the percentage.

  int high - The high value to use to calculate the percentage.

  Color lowColor - The color that will match 0%.

  Color highColor - The color that will match 100%.

- Results

  Color - A color that is a mix of the two given colors based on the percentage.

Examples

code

```
gradient(0, 0, 100, toColor("red"), toColor("blue")) //returns red.
```

Code Snippet

```
gradient(100, 0, 100, toColor("red"), toColor("blue")) //returns blue.
```

Code Snippet

```
gradient(60, 0, 100, toColor("red"), toColor("blue")) //returns a shade of purple.
```

Code Snippet

```
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

Date and Time Functions

The following functions allow you to check or modify time values in expression bindings.

In This Section ...

add*

This function is used by **Ignition's Expression** language.

Description

This function is a set of functions that include:

Function	Description
addMillis	Add or subtract an amount of milliseconds to a given date and time.
addSeconds	Add or subtract an amount of seconds to a given date and time.
addMinutes	Add or subtract an amount of minutes to a given date and time.
addHours	Add or subtract an amount of hours to a given date and time.
addDays	Add or subtract an amount of days to a given date and time.
addWeeks	Add or subtract an amount of weeks to a given date and time.
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.
addYears	Add or subtract an amount of years to a given date and time.

Syntax

add*(date, value)

- Parameters

  Date date - The starting date.

  int value - The amount of units to change the date by, where the units is dependent on the function used.

- Results

  Date - A new date that has been changed by the amount specified.

Code Examples

Code Snippet

```
addWeeks(now(), 2) //Adds 2 weeks to the current time
```

Code Snippet

```
addDays(now(), -5) //Subtracts 5 days from the current time
```

Code Snippet

```
addHours({Root Container.Calendar.date}, 5) //This example would add 5 hours to the date passed
in from a calendar component
```

dateArithmetic

This function is used by **Ignition's Expression** language.

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, value, field)

- Parameter

  Date date - The starting date.

  int value - The value to add or subtract from the given date.

  string field - The units of the value.

- Results

  Date - A new date, that has been altered by the amount of units specified.

Examples

Code Snippet

```
dateArithmetic(toDate("2010-01-04 8:00:00"), 5, "hour") //returns the date '2010-01-04 13:00:00'
```

Code Snippet

```
dateArithmetic({Root Container.DatePicker.date}, -8, "days") //returns a date eight days before
the date in a Popup Calendar component.
```

dateDiff

This function is used by **Ignition's Expression** language.

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(date1, date2, field)

- Parameter

  Date date1 - The first date.

  Date date2 - The second date.

  string field - The units that the difference will be specified in.

- Results

  float - The difference between two dates in the units specified.

Examples

Code Snippet

```
dateDiff(toDate("2008-2-24 8:00:00"), toDate("2008-2-24 8:15:30"), "minute") //returns 15.5
```

Code Snippet

```
dateDiff(toDate("2008-2-24 8:00:00"), toDate("2008-3-12 9:28:00"), "month") //returns 1
```

Code Snippet

```
dateDiff(toDate("2008-2-24 8:00:00"), toDate("2008-3-12 9:28:00"), "day") //returns 17.02
```

dateExtract

This function is used by **Ignition's Expression** language.

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. To get a month index starting at 1, simply add 1 to the function result.

Syntax

dateExtract(date, field)

- Parameters

  Date date - The given date.

  string field - The field to extract the value from.

- Results

  int - The value of the specified field within the given date.

Examples

Code Snippet

```
dateExtract(toDate("2003-9-14 8:00:00"), "year") //returns 2003
```

Code Snippet

```
dateExtract(toDate("2009-1-15 8:00:00"), "month") //returns 0
```

Code Snippet

```
dateExtract(toDate("2008-1-24 8:00:00"), "month") + 1 //returns 1
```

dateFormat

This function is used by **Ignition's Expression** language.

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)

- Parameter

  Date date - The starting date.

  string pattern - The pattern to format the given date to.

- Results

  string - The given date formatted based on the given format pattern.

Examples

Code Snippet

```
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
```

Code Snippet

```
dateFormat(toDate("2003-9-14 8:00:00"), "yyyy-MM-dd h a") //returns the string "2003-09-14 8 AM"
```

Code Snippet

```
dateFormat(toDate("2003-9-14 8:00:00"), "MMM d, yyyy") //returns the string "Sep 14, 2003"
```

Code Snippet

```
dateFormat(now(), 'yyyy-MM-dd 00:00:00') //returns the current date, but forces the time to 00:
00:00
```

dateIsAfter

This function is used by **Ignition's Expression** language.

Description
Compares two dates to see if date1 is after date2. Note: This is exclusive, meaning if the dates are identical the result is always False.

Syntax

dateIsAfter(date1, date2)

- Parameters

  Date date1 - The first date to compare.

  Date date2 - The second date to compare

- Results

  Bool - True if date1 is at or after date2, False if not.

Code Examples

Code Snippet

```
dateIsAfter(now(), toDate("2016-04-12 00:00:00"))
// Will be true if the current time is after April 12th, 2016 at midnight
```

dateIsBefore

This function is used by **Ignition's Expression** language.

Description
Compares two dates to see if date1 is before date2. Note: This is exclusive, meaning if the dates are identical the result is always False.

Syntax

dateIsBefore(date1, date2)

- Parameter

  Date date1 - The first date to compare.

  Date date2 - The second date to compare.

- Results

  Bool - True if date1 is at or before date2, False if not.

Code Examples

Code Snippet

```
dateIsBefore(now(), toDate("2016-04-12 00:00:00"))
// Will be true if the current time is before April 12th, 2016 at midnight
```

dateIsBetween

This function is used by **Ignition's Expression** language.

Description
Compares two dates to see if a target date is between two other dates. Note: This is inclusive, meaning if the targetDate is the same as the start or end date the result is True.

Syntax

dateIsBetween(targetDate, startDate, endDate)

- Parameters

  Date targetDate - The date to compare.

  Date startDate - The start of a date range.

  Date endDate - The end of a date range. This date must be after the start date.

- Results

  Bool - True if the targetDate is at or between the startDate and endDate, False if not.

Code Examples

Code Snippet

```
dateIsBetween(now(), toDate("2016-06-12 00:00:00"), toDate("2016-06-19 00:00:00"))
// Will be true if the current time is between the 12th and 19th of June 2016
```

dateIsDaylight

This function is used by **Ignition's Expression** language.

Description
Checks to see if the current timezone is using daylight savings time during the date specified. Will use the current date if no date is specified.

Syntax
dateIsDaylight([date]) • Parameters Date date - *Optional*. The date to use. Will use the current date if omitted. • Results Bool - True if the current timezone is using daylight savings time during the specified date.

Code Examples
Code Snippet

```
dateIsDaylight(toDate("2007-06-28 00:00:00")) // Will return True in the US/Pacific Timezone,
due to that timezone's observation of Daylight Savings time on the specified date.
```

*Between

This function is used by **Ignition's Expression** language.

Description

This function is a set of functions that include:

Function	Description
millisBetween	Calculates the number of whole milliseconds between two dates.
secondsBetween	Calculates the number of whole seconds between two dates.
minutesBetween	Calculates the number of whole minutes between two dates.
hoursBetween	Calculates the number of whole hours between two dates.
daysBetween	Calculates the number of whole days between two dates. Daylight savings changes are taken into account.
weeksBetween	Calculates the number of whole weeks between two dates.
monthsBetween	Calculates the number of whole months between two dates. Daylight savings changes are taken into account.
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

***between(date1, date2)**

- Parameter

  Date date1 - The first date to compare.

  Date date2 - The second date to compare.

- Results

  int - The number of units between the two dates. The units is specified by the function used.

Code Examples

Code Snippet

```
daysBetween(toDate("2017-04-28 00:00:00"), toDate("2017-03-22 00:00:00")) //This will print -37
```

Code Snippet

```
weeksBetween({Root Container.Calendar1.date}, {Root Container.Calendar2.date}) //Will grab the
number of weeks between two dates of calendar components.
```

fromMillis

This function is used by **Ignition's Expression** language.

Description

Creates a date object given a time, in milliseconds, past Unix epoch (1 January 1970 at midnight UTC).

Syntax

fromMillis(millis)

- Parameters

  int millis - The number of milliseconds since epoch time.

- Results

  Date - The date representing the given number of milliseconds since epoch time.

Code Examples

Code Snippet

```
fromMillis(1503092125000)//This example will print out the date "Fri Aug 18 14:35:25 PDT 2017"
```

get*

This function is used by **Ignition's Expression** language.

Description

This function is a set of functions that include:

Function	Description
getMillis	Extracts the milliseconds from a date, ranging from 0-999.
getSecond	Extracts the second from a date, ranging from 0-59.
getMinute	Extracts the minutes from a date, ranging from 0-59.
getHour12	Extracts the hour from a date. Uses a 12 hour clock, so noon and midnight are returned as 0.
getHour24	Extracts the hour from a date. Uses a 24 hour clock, so midnight is zero.
getDayOfWeek	Extracts the day of the week from a date. Sunday is day 1, Saturday is day 7.
getDayOfMonth	Extracts the day of the month from a date. The first day of the month is day 1.
getDayOfYear	Extracts the day of the year from a date. The first day of the year is day 1.
getMonth	Extracts the month from a date, where January is month 0.
getQuarter	Extracts the quarter from a date, ranging from 1-4.
getYear	Extracts the year from a date.
getAMorPM	Returns a 0 if the time is before noon, and a 1 if the time is equal to or after noon.

Syntax

get*(date)

- Parameters

  Date date - The date to extract from.

- Results

  int - The value of the units of the date specified. The units are determined by the function used.

Code Examples

Code Snippet

```
getMonth(now()) //This returns the current month.
```

Code Snippet

```
getQuarter(getDate(2017, 3, 15)) //The date, April 15th, is in the second quarter, so this
returns 2.
```

Code Snippet

```
getDayOfWeek({Root Container.Calendar.date}) //Will return the day of the week of the selected
date of the calendar component
```

getDate

This function is used by **Ignition's Expression** language.

Description
Creates a new Date object given a year, month and a day. The time will be set to midnight of that day. January is 0 and December is 11. The first day of the month is 1.

Syntax

getDate(year, month, day)

- Parameters

  int year - The year that the date will be set to.

  int month - The month that the date will be set to.

  int day - The day that the date will be set to.

- Results

  Date - The date created from the specified integers.

Code Examples

Code Snippet

```
getDate(2016, 11, 1) //This example will create a new date object set to December 1st, 2016.
```

getTimezone

This function is used by **Ignition's Expression** language.

Description

Returns the ID of the current timezone depending on the scope in which it is called. If run in a Vision client or Perspective session scope, the function will display the timezone of the client or session. If run in an expression in a global scope (e.g. on an expression tag), it will return the timezone of the gateway.

*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

getTimezone( )

- Parameters

  None

- Results

  string - The current timezone ID.

Code Examples

There are no examples associated with this expression function.

getTimezoneOffset

This function is used by **Ignition's Expression** language.

Description

Returns the current timezone's offset versus UTC for a given instant, taking Daylight Savings Time into account.

Syntax

getTimezoneOffset([date])

- Parameters

  Date date - *Optional*. A specified date to compare the current timezone to UTC. Will use the current time if left blank.

- Results

  float - The offset of the current time from UTC.

Code Examples

Code Snippet

```
getTimezoneOffset(getDate(2017, 1, 22)) //returns -8.0, if you are in Pacific Time.
```

Code Snippet

```
getTimezoneOffset(getDate(2017, 6, 22)) //returns -7.0, if you are in Pacific Time, since
Daylight Savings Time would be in effect.
```

getTimezoneRawOffset

This function is used by **Ignition's Expression** language.

Description
Returns the current timezone's offset versus UTC, not taking daylight savings into account.

Syntax
getTimezoneRawOffset() • Parameters None • Results float - The offset of the current time from UTC.

Code Examples
Code Snippet `getTimezoneRawOffset() //returns -8.0 if you are in the Pacific Timezone, regardless of time of year`

midnight

This function is used by **Ignition's Expression** language.

Description

Returns a copy of a date with the hour, minute, second, and millisecond fields set to zero.

Syntax

midnight(date)

- Parameters

  Date date - The date to set to midnight.

- Results

  Date - The new date set to midnight.

Code Examples

Code Snippet

```
midnight(now()) //This will take the current date and set the time to midnight
```

now

This function is used by **Ignition's Expression** language.

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 p oll. 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])

- Parameters

  int pollRate - *Optional*. The poll rate in milliseconds to update the time at. Default is 1000 ms.

- Results

  Date - The current time.

Examples

Code Snippet

```
now() //returns the current time, updates every second.
```

Code Snippet

```
now(13000) //returns the current time, updates every 13 seconds.
```

Code Snippet

```
dateFormat(now(0), "MMM d, h:mm a") //returns a string representing the current time, formatted
like "Feb 12, 9:54 AM". Does not update.
```

setTime

This function is used by **Ignition's Expression** language.

Description
Takes in a date, and returns a copy of it with the time fields set as specified. Note that the millisecond field is not preserved.

Syntax

setTime(date, hour, minute, second)

- Parameters

  Date date - A starting date.

  int hour - The value to set the hour field to.

  int minute - The value to set the minute field to.

  int second - The value to set the second field to.

- Results

  Date - The new date with the time set as specified.

Code Examples

Code Snippet

```
setTime({Root Container.Calendar.date}, 1, 37, 44) //This example will set the date object to
the current date with the time set to 01:37:44.
```

timeBetween

This function is used by **Ignition's Expression** language.

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,startDate,ednDate)

- Parameters

  Date/string date - The date to compare. Can be either a date or a string.

  Date/string startDate - The start date to compare to. Can be either a date or a string.

  Date/string endDate - The end date to compare to. Can be either a date or a string.

- Results

  Bool - True if the date is between the start and end date, False if not.

Examples

Code Snippet

```
timeBetween(toDate("2003-9-14 12:00:00"), toDate("2003-9-14 8:00:00"),toDate("2003-9-14 18:00:
00")) //returns true
```

Code Snippet

```
timeBetween("2:00:00 pm", "9:00:00 am", "5:00:00 pm") //returns true
```

Code Snippet

```
timeBetween(toDate("2003-9-14 20:00:00"), toDate("2003-9-14 18:00:00"), toDate("2003-9-15 2:00:
00")) //returns true
```

toMillis

This function is used by **Ignition's Expression** language.

Description
Converts a Date object to its millisecond value elapsed since January 1, 1970, 00:00:00 UTC (GMT)

Syntax
toMillis(date) • Parameters Date date - The date to convert to epoch time. • Results int - The number of milliseconds from epoch time of the give date.

Code Examples

Code Snippet

```
// This will take the date Aug 22, 2017 at 14:35:25 PST and convert it to milliseconds from
epoch time which is 1,500,767,134,000
toMillis(setTime(getDate(2017, 6, 22), 16, 45, 34))
```

Identity Provider

Identity Provider Functions

The following functions allow you to test whether specified elements are present in an IdP collection object. They can be used only in the Security Level Rules and User Attribute Mapping sections of the Gateway webpage.

In This Section ...

containsAll

This function is used by **Ignition's Expression** language.

Description

This function is only available for Security Level Rules and User Attribute Mapping.

This function checks to see if all of the listed elements are present in the collection object. The function requires at least two arguments, a `collection` and an `element`.

Syntax

containsAll(collection, element0[, elementN])

- Parameters

  object collection - A collection of values. Typically from the {security-zone} object or the {idp-attribute:X} object.

  string element - One or more comma separated elements to look for.

- Results

  Boolean - True if the collection object contained all of the listed elements. False if otherwise.

Examples

Code Snippet

```
// returns true for a login attempt against an Ignition IdP, if the user has both Administrator
and Operator roles
containsAll({attribute-source:idTokenClaims:roles}, 'Administrator', 'Operator')
```

Code Snippet

```
// returns true for a login attempt if the login location is in all three of the specified
security zones
containsAll({security-zones}, 'site 1', 'mill', 'offshore')
```

containsAny

This function is used by **Ignition's Expression** language.

Description

This function is only available for Security Level Rules and User Attribute Mapping.

This function checks to see if any of the listed elements are present in the collection object. The function requires at least two arguments, a `collection` and an `element`.

Syntax

containsAny(collection, element0[, elementN])

- Parameters

  object collection - A collection of values. Typically from the {security-zone} object or the {idp-attribute:X} object.

  string element - One or more comma separated elements to look for.

- Results

  Boolean - True if the collection object contained any of the listed elements. False if otherwise.

Examples

Code Snippet

```
// returns true for a login attempt against an Ignition IdP, if the user has either the
Administrator or Operator roles
containsAny({attribute-source:idTokenClaims:roles}, 'Administrator', 'Operator')
```

Code Snippet

```
// returns true for a login attempt if the login location is in at least one of the specified
security zones
containsAny({security-zones}, 'site 1', 'mill', 'offshore')
```

JSON

JSON Functions

The following functions allow you to manipulate JSON strings in expression bindings.

In This Section ...

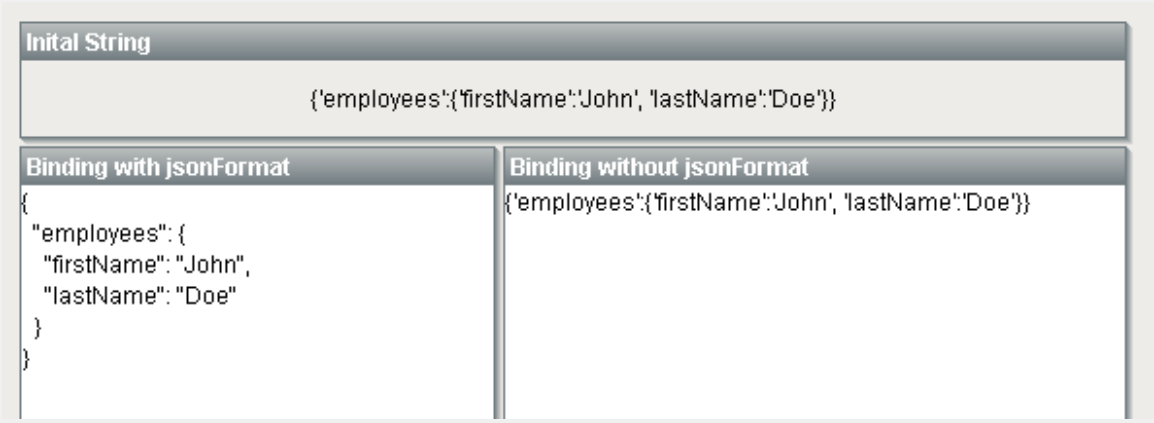
jsonFormat

This function is used by **Ignition's Expression** language.

Description

Takes a string, and returns a prettyprints string: making the string easier to read by humans. Especially useful in cases where the string is displayed on components that can demonstrate carriage returns.

The image below shows a Label component with a JSON-friendly string. Below that are two Text Area components that are bound to the Label's text: one using the jsonFormat() function and the other without.



Syntax

jsonFormat(string)

- Parameters

  String string - The string to format. The string must be in a JSON-friendly format.

- Results

  string - A prettyprints string of the specified string.

Examples

Code Snippet

```
// This example builds a JSON friendly string, and returns a prettyprint version of the string.
// Since the parameter passed is a string datatype, the whole parameter must be wrapped in
quotes.
jsonFormat("{item1:10,item2:20}")
```

Code Snippet

```
// Another valid JSON format - you may optionally wrap inner strings in quotation marks.
jsonFormat("[{'item1':'apples','item2':'bananas'},{'item1':'oranges','item2':'carrots'}]")
```

jsonGet

This function is used by **Ignition's Expression** language.

Description

Takes a JSON friendly string and a path string, and returns the value of that path.

Syntax

jsonGet(json, path)

- Parameters

  string json - The JSON string. The string must be in a JSON-friendly format.

  string path - The path to look for in the JSON string.

- Results

  object - The value at the path.

Examples

Code Snippet

```
// This example takes a JSON friendly string and finds the value located at the path item.
secondThing, which is 2.

jsonGet("{'item':{'firstThing':1, 'secondThing':2}}", "item.secondThing")
```

Code Snippet

```
// This example takes a JSON friendly string and finds the value located at the path item, which
is {"firstThing":1,"secondThing":2}.

jsonGet("{'item':{'firstThing':1, 'secondThing':2}}", "item")
```

jsonSet

This function is used by **Ignition's Expression** language.

Description

Takes a JSON friendly string, a path string, and value, and will return a new JSON friendly string with the provided path set to the provided value. This is best used in conjunction with the Derived Tags writeback value.

Syntax

jsonSet(json, path, value)

- Parameters

  string json - The JSON string. The string must be in a JSON friendly format.

  string path - The path string.

  object value - The replacement for the value at the path.

- Results

  string - A JSON friendly string with a new value set at the specified path.

Examples

Code Snippet

```
// This example takes a JSON friendly string and sets the provided path to the given value. This
would then return the string {'item':{'firstThing':1, 'secondThing':5}}

jsonSet("{'item':{'firstThing':1, 'secondThing':2}}", "item.secondThing", 5)
```

Logic

Logic Functions

The following functions allow you to preform logic and evaluate values in expression bindings.

In This Section ...

binEnc

This function is used by **Ignition's Expression** language.

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.

Syntax

binEnc(value[, value...])

- Parameters

  Bool value - A value that represents a bit. Can be either 0 or 1, and can enter in as many values as necessary.

- Results

  int - The integer representation of the binary value entered.

Examples

Code Snippet

```
binEnc(0,0,1,0) //returns 4 (the value of 0100)
```

Code Snippet

```
binEnc(true,0,1,1,0) //returns 13 (the value of 01101)
```

binEnum

This function is used by **Ignition's Expression** language.

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.

Syntax

binEnum(value[, value...])

- Parameters

  int value - Any number of values.

- Results

  int - The index of the first true value.

Examples

Code Snippet

```
binEnum(0, 1, 0) //returns 2
```

Code Snippet

```
binEnum(0, false, 15, 0, 23) //returns 3 (the index of the 15 - any non-zero number is "true")
```

case

This function is used by **Ignition's Expression** language.

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 cas e1..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, case, return[, case, return...], returnDefault)

- Parameters

  object value - A value of any type.

  object case - A case to match the value to.

  object return - The return if its pair case has been matched.

  object returnDefault - The default return if none of the case arguments were matched.

- Results

  object - The return value for the matched case, or the returnDefault value if no case was matched.

Examples

Code Snippet

```
//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
```

Code Snippet

```
//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
"BAD STATE!") // default
```

coalesce

This function is used by **Ignition's Expression** language.

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(value[, value...])

- Parameters

  object value - Any number of values.

- Returns

  object - The first non null argument.

Examples

Code Snippet

```
coalesce(null, "abc") //would return "abc"
```

Code Snippet

```
coalesce("xyz", "abc") //would return "xyz"
```

Code Snippet

```
coalesce({Root Container.MyDataSet}[0,"ColumnName"], 0) //would return the value in the dataset
if it isn't null, but 0 if it is null.
```

getBit

This function is used by **Ignition's Expression** language.

Description

This function returns the bit value (an integer, 0 or 1) in a number at a given position, according to its binary representation. The least significant bit in a number is position 0.

Syntax

getBit(number, position)

- Parameters

  int number - The number value to start with.

  int position - The bit position to check.

- Results

  int - A 0 or 1, depending on the bit at the position of the integer specified.

Examples

Code Snippet

```
getBit(0,0) //would return 0
```

Code Snippet

```
getBit(1,0) //would return 1
```

Code Snippet

```
getBit(8,2)  //would return 0
```

hasChanged

This function is used by **Ignition's Expression** language.

Description

This function is only available in Transaction Group Expression Items and Expression Tags.

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[, includeQuality][, pollRate])

- Parameters

  object value - The number value to check for change

  bool includeQuality - *Optional*. A flag that indicates if a quality change will also trigger this tag.

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

  int pollRate - *Optional*. The poll rate in milliseconds. Only applicable on Expression Tags when the Execution Mode is set to Event Driven. All other Execution Modes will ignore this parameter.

  The pollRate argument was not present in Ignition 7, but is required in Ignition 8.0.4 for Event Driven Expression Tags. This means that any Expression Tags that are in the Event Driven Execution Mode will either need to have the Execution Mode changed or the poll rate added in on upgrade to 8.0.4, or the function will run once when the value changes and then will not work properly again.

- Results

  bool - True if the value has changed since the last time the expression was evaluated, False if not.

Examples

Code Snippet

```
hasChanged({[default]Station 1/Status},True) //would return true if the referenced tag has
changed in value or quality since the last group execution
```

if

This function is used by **Ignition's Expression** language.

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)

- Parameters

  object condition - The condition to evaluate.

  object trueReturn - The true return value.

  object falseReturn - The false return value.

- Results

  object - Returns the trueReturn if the condition is true, falseReturn if it is False.

Examples

Code Snippet

```
if(1, "Yes", "No") //would return "Yes"
```

Code Snippet

```
if(0, "Yes", "No") //would return "No"
```

Code Snippet

```
if({Root Container.CheckBox.selected}, "Selected", "Not Selected") //would return with a
description of the state of the checkbox
```

Code Snippet

```
//nests if functions to check the value of 2 different Tags, and return a message based on which
ones are greater than 0.
if({tag1} > 0, if({tag2} > 0, "Both Tags are positive.", "Tag 1 is positive."), if({tag2} > 0,
"Tag 2 is positive.", "Neither Tag is positive."))
```

isGood

This function is used by **Ignition's Expression** language.

Description
Tests to see whether or not the given value is good quality.

Syntax
isGood(value) • Parameters object value - A value to check if it is good • Results Bool - True if the value is good, False if it is not.

Examples
Code Snippet

```
isGood({path/to/myTag}) //returns 1 if the value is good, 0 otherwise.
```

isNull

This function is used by **Ignition's Expression** language.

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. isNull(x) is the same as x = null.

Syntax

isNull(value)

- Parameters

  object value - A value to check if it is null.

- Results

  Bool - True if the value is null, False if it is not.

Examples

Code Snippet

```
//returns "Value is Null" if the property is null, and the value otherwise.
if(isNull({Root Container.MyProperty}), "Value is Null", {Root Container.MyProperty})
```

lookup

This function is used by **Ignition's Expression** language.

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 noMatchValue.

Syntax

lookup(dataset, lookupValue, noMatchValue[, lookupColumn][, resultColumn])

- Parameters

  DataSet dataset - A dataset to search through.

  object lookupValue - The value to look for.

  object noMatchValue - The result value if no match.

  object lookupColumn - *Optional*. The column to lookup. Can either be the column index or the name of the column. Defaults to 0.

  object resultColumn - *Optional*. The column to pull the result value from. Can either be the column index or the name of the column. Defaults to 1.

- Results

  object - The value in the result column of the same row that the lookupValue was found, or the noMatchValue if a match was not found. The data type of this object will always be coerced to match the type of the noMatchValue parameter.

Examples

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"

Code Snippet

```
lookup({Root Container.Table.data}, "Carrots", -1.0) //returns 3.50
```

Code Snippet

```
lookup({Root Container.Table.data}, "Grapefruit", -1) //returns -1, the noMatchValue
```

Code Snippet

```
lookup({Root Container.Table.data}, "Walnuts", "Unknown", 0, "Category") //returns "Nut"
```

Code Snippet

```
lookup({Root Container.Table.data}, "Pecans", "Unknown", 0, 2) //returns "Unknown", the
noMatchValue
```

switch

This function is used by **Ignition's Expression** language.

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, case[, caseN...], return[, returnN...], returnDefault)

- Parameter

  object value - The value to check against the case values.

  object case - A value to check against. Can be any number of case values.

  object return - A value to return for the matching case. Must be the same number of return values as case values.

  object returnDefault - The default return if no case is matched.

- Results

  object - The return value for the case that matched the value, or the returnDefault value if no matches were found.

Examples

Code Snippet

```
//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
```

Code Snippet

```
//The following would return "Running".
switch(
1, // value
0, 1, 2, // cases 1-3
"Off", // return 1
"Running", // return 2
"Fault", // return 3
"BAD STATE!") // default
```

try

This function is used by **Ignition's Expression** language.

Description

This expression is used to suppress 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.

Syntax

try(expression, failover)

- Parameters

  object expression - An arbitrary expression.

  object failover - The value to use if there is an error in the expression parameter.

- Results

  object - The result of the expression or the failover value if there is an error.

Examples

Code Snippet

```
try(toInteger("boom"), -1) // returns -1 with a quality code of 192 (good)
```

Code Snippet

```
// fetch an integer value from the first row of a table. Return -1 if there are no rows
try({Root Container.Power Table.data}[0, 'Integer Column'], -1)
```

Math

Math Functions

The following functions allow you to preform math functions on values in expression bindings.

In This Section ...

abs

This function is used by **Ignition's Expression** language.

Description
Returns the absolute value of number.

Syntax
abs(number) • Parameters int/float number - The number to get the absolute value of. • Results int/float - The absolute value of the number provided.

Examples
Code Snippet

```
abs(-4) //returns 4
```

acos

This function is used by **Ignition's Expression** language.

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) • Parameters float number - The number to get the arc cosine of. Must be a value between -1 and 1. • Results float - The arc cosine of the value provided.

Examples
Code Snippet

```
acos(.38) //returns 1.181
```

asin

This function is used by **Ignition's Expression** language.

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)

- Parameters

  float number - The number to get the arc sine of. Must be between -1 and 1.

- Results

  float - The arc sine of the number provided.

Examples

Code Snippet

```
asin(.38) //returns 0.3898
```

atan

This function is used by **Ignition's Expression** language.

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 t hrough pi/2

Syntax

atan(number)

- Parameters

  float number - The number to get the arc tangent of.

- Results

  float - The arc tangent of the number provided.

Examples

Code Snippet

```
atan(.38) //returns 0.3631
```

ceil

This function is used by **Ignition's Expression** language.

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)

- Parameters

  float number - The number to get the ceiling of.

- Results

  float - The ceiling of the value provided..

Examples

Code Snippet
`ceil(2.38) //returns 3.0`

cos

This function is used by **Ignition's Expression** language.

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)

- Parameters

  int/float number - The number to get the cosine of.

- Results

  float - The cosine of the number provided.

Examples

Code Snippet

```
cos(1.89) //returns -0.31381
```

exp

This function is used by **Ignition's Expression** language.

Description

Returns Euler's number e raised to the power of the argument number, or e^{number}

Syntax

exp(number)

- Parameters

  int/float number - The exponent value to raise e to the power of.

- Results

  int/float - The value of e to the power of the value provided.

Examples

Code Snippet

```
exp(5) //returns 148.4
```

floor

This function is used by **Ignition's Expression** language.

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)

- Parameters

  float number - The number to get the floor of.

- Results

  float - The floor of the number provided.

Examples

Code Snippet

```
floor(2.72) //returns 2.0
```

log

This function is used by **Ignition's Expression** language.

Description
Returns the natural logarithm (base e) of a number.

Syntax
log(number) • Parameters int/float number - The number to get the log of. • Results float - The log of the number provided.

Examples
Code Snippet

```
log(28) //returns 3.332
```

log10

This function is used by **Ignition's Expression** language.

Description

Returns the logarithm (base 10) of a number.

Syntax

log10(number)

- Parameters

  int/float number - The number to get the log base 10 of.

- Results

  float - The log base 10 of the number provided.

Examples

Code Snippet

```
log10(28) // returns 1.447
```

pow

This function is used by **Ignition's Expression** language.

Description
Returns a number raised to a power.

Syntax

pow(number, power)

- Parameters

  int/float number - The number to raise to the provided power.

  int/float power - The power value to raise the number value to.

- Results

  int/float - The result of the number provided raised to the power provided.

Examples

Code Snippet

```
pow(2,3) //returns 8
```

round

This function is used by **Ignition's Expression** language.

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])

- Parameters

  float number - The number to round.

  int decimals - *Optional*. The number of decimal places to round to. Defaults to 0.

- Results

  int/float - The value provided rounded to the specified decimal places.

Examples

Code Snippet

```
round(3.829839, 2) //returns 3.83
```

sin

This function is used by **Ignition's Expression** language.

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)

- Parameters

  int/float number - The number to get the sine of.

- Results

  int/float - The sine of the number provided.

Examples

Code Snippet

```
sin(1.89) //returns 0.9495
```

sqrt

This function is used by **Ignition's Expression** language.

Description

Returns the square root of the argument number.

Syntax

sqrt(number)

- Parameters

  int/float number - The number to get the square root of.

- Results

  float - The square root of the number provided.

Examples

Code Snippet

```
sqrt(64) //returns 8.0
```

tan

This function is used by **Ignition's Expression** language.

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) • Parameters int/float number - The number to get the tangent of. • Results float - The tangent of the number provided.

Examples
Code Snippet

```
tan(1.89) //returns -3.026
```

todegrees

This function is used by **Ignition's Expression** language.

Description
Converts an angle measured in radians to an equivalent angle measured in degrees.

Syntax
todegrees(number) • Parameters int/float number - The number radians • Results int/float - The degree equivalent of the radians provided.

Examples
Code Snippet

```
todegrees(3.14) //returns 179.9088
```

toradians

This function is used by **Ignition's Expression** language.

Description
Converts an angle measured in degrees to an equivalent angle measured in radians.

Syntax
toradians(number) • Parameters int/float number - The number of degrees. • Results int/float - The radian equivalent of the degrees provided.

Examples
Code Snippet

```
toradians(180) //returns 3.141592653589793
```

String

String Functions

The following functions allow you to search or modify string values in expression bindings.

In This Section ...

concat

This function is used by **Ignition's Expression** language.

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(string[, string...])

- Parameters

  string string - Any number of string values to concatenate together.

- Results

  string - A string that is all of the strings provided concatenated together.

Examples

Code Snippet

```
concat("The answer is: ", "42") //returns "The answer is: 42"
```

escapeSQL

This function is used by **Ignition's Expression** language.

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)

- Parameters

  string string - The starting string.

- Results

  string - A string that has been formatted so that single quotes are replaced with two single quotes, and backslashes are replaced with two backslashes.

Examples

Code Snippet

```
"SELECT * FROM mytable WHERE option = '" + escapeSQL("Jim's Settings") + "'" // returns SELECT *
FROM mytable WHERE option='Jim''s Settings'
```

Code Snippet

```
"SELECT * FROM mytable WHERE option = '" + escapeSQL({Root Container.Text Field.text}) + "'"
//returns a query with sanitized user input from a text field.
```

escapeXML

This function is used by **Ignition's Expression** language.

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)

- Parameters

  string string - The starting string.

- Results

  string - A string that has been escaped for XML

Examples

Code Snippet

```
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

This function is used by **Ignition's Expression** language.

Description

Returns an integer value of the binary formatted string argument. Numbers outside of the range (-2^{31}) - $(2^{31}-1)$, and strings that are not binary numbers, return null.

Syntax

fromBinary(string)

- Parameters

  string string - A string representation of a binary.

- Results

  int - The integer value of the specified binary.

Examples

Code Snippet

```
fromBinary("1111") //returns 15
```

Code Snippet

```
fromBinary("-1111") //returns -15
```

fromHex

This function is used by **Ignition's Expression** language.

Description

Returns an integer value of the hex formatted string argument. Numbers outside of the range (-2^{31}) - $(2^{31}-1)$, and strings that are not hex numbers, return null.

Syntax

fromHex(string)

- Parameters

  string string - A string representation of a hex value.

- Results

  int - The integer of the hex value.

Examples

Code Snippet

```
fromHex("ff") //returns 255
```

Code Snippet

```
fromHex("0xff") //returns 255
```

Code Snippet

```
fromHex("-ff") //returns -255
```

fromOctal

This function is used by **Ignition's Expression** language.

Description

Returns an integer value of the octal formatted string argument. Numbers outside of the range (-2^{31}) - $(2^{31}-1)$, and strings that are not octal numbers, return null.

Syntax

fromOctal(string)

- Parameters

  string string - A string representation of an octal.

- Results

  int - The integer of the octal value.

Examples

Code Snippet

```
fromOctal("77") //returns 63
```

Code Snippet

```
fromOctal("-77") //returns -63
```

indexOf

This function is used by **Ignition's Expression** language.

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)

- Parameters

  string string - The string to search through.

  string substring - The string to search for.

- Results

  string - The index where the substring was first found in the string.

Examples

Code Snippet

```
indexOf("Hamburger", "urge") //returns 4
```

Code Snippet

```
indexOf("Test", "") //returns 0
```

Code Snippet

```
indexOf("Dysfunctional", "fun") //returns 3
```

Code Snippet

```
indexOf("Dysfunctional", "marble") //returns -1
```

Code Snippet

```
indexOf("banana", "n") //returns 2
```

lastIndexOf

This function is used by **Ignition's Expression** language.

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)

- Parameters

  string string - The string to search through.

  string substring - The string to search for.

- Results

  string - The index where the substring was last found in the string.

Examples

Code Snippet

```
lastIndexOf("Hamburger", "urge") //returns 4
```

Code Snippet

```
lastIndexOf("Test", "") //returns 4
```

Code Snippet

```
lastIndexOf("Dysfunctional", "fun") //returns 3
```

Code Snippet

```
lastIndexOf("Dysfunctional", "marble") //returns -1
```

Code Snippet

```
lastIndexOf("banana", "n") //returns 4
```

left

This function is used by **Ignition's Expression** language.

Description

Returns count characters from the left side of string, where count and string are the arguments to the function.

Syntax

left(string, charCount)

- Parameters

  string string - The starting string.

  int charCount - The number of characters to return.

- Results

  string - A string that is the first charCount number of characters of the specified string.

Examples

Code Snippet

```
left("hello", 2) //returns "he"
```

Code Snippet

```
left("hello", 0) //returns ""
```

Code Snippet

```
left("hello", 5) //returns "hello"
```

len

This function is used by **Ignition's Expression** language.

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)

- Parameters

  object value- The starting object.

- Results

  int - The length of the provided object.

Examples

Code Snippet

```
len("Hello World") //returns 11
```

Code Snippet

```
len({Root Container.Table.data}) //returns the number of rows in the table.
```

lower

This function is used by **Ignition's Expression** language.

Description

Takes a string and returns a lower-case version of it.

Syntax

lower(string)

- Parameters

  string string - The string to make lowercase.

- Results

  string - The starting string with all characters lowercase.

Examples

Code Snippet

```
lower("Hello World") // returns "hello world"
```

numberFormat

This function is used by **Ignition's Expression** language.

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. The negative subpattern will only be used to specify the prefix and suffix. The number of digits, , minimal digits, and other characteristics are all the same as the positive pattern.
%	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;(#)	(1,234)
4096	0.###E0	4.096E3
.348	#.00%	34.80%
34.8	#0.00'%'	34.80%

Syntax

numberFormat(number, pattern)

- Parameters

  float number- The number to format.

  string pattern - The format pattern.

- Results

  string - The string representation of the number formatted according to the pattern provided.

Examples

Code Snippet

```
numberFormat(34.8, "#0.00'%'") //returns the string "34.80%"
```

repeat

This function is used by **Ignition's Expression** language.

Description

Repeats the given string some number of times.

Syntax

repeat(string, count)

- Parameters

  string string - The string to repeat

  int count - The number of times to repeat the string.

- Results

  string - The given string repeated the given number of times.

Examples

Code Snippet

```
repeat("hello", 2) //returns "hellohello"
```

Code Snippet

```
repeat("hello", 0) //returns ""
```

replace

This function is used by **Ignition's Expression** language.

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, substring, replacementString)

- Parameters

  string string - The starting string.

  string substring - The string to search for.

  string replacementString - The string to replace any instances of the substring with.

- Results

  string - The starting string with all instances of the substring replaced by the replacementString.

Examples

Code Snippet

```
replace("XYZ", "Y", "and") //returns "XandZ"
```

Code Snippet

```
replace("bob and mary went to bob's house", "bob", "judith") //returns "judith and mary went to
judith's house"
```

right

This function is used by **Ignition's Expression** language.

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)

- Parameters

  string string - The starting string.

  string charCount - The number of characters to return.

- Results

  string - A string of the number of characters specified in the charCount from the specified string.

Examples

Code Snippet

```
right("hello", 2) //returns "lo"
```

Code Snippet

```
right("filename.pdf", 3) //returns "pdf"
```

Code Snippet

```
right("hello", 0) //returns ""
```

split

This function is used by **Ignition's Expression** language.

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])

- Parameters

  string string - The starting string.

  string regex - The string to split on.

  int limit - *Optional*. The max number of splits to make. Default 0 which is as many as possible.

- Results

  Dataset - The split string, with a single column called parts, where each row is a new part of the string.

Examples

Code Snippet

```
split("hello,world", ",") //returns dataset [["hello"], ["world"]]
```

Code Snippet

```
split("boo:and:foo", ":") //returns dataset [["boo"], ["and"], ["foo"]]
```

Code Snippet

```
split("boo:and:foo", ":", 2) //returns dataset [["boo"], ["and:foo"]]
```

substring

This function is used by **Ignition's Expression** language.

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. Indexes outside of the range of the string throw a StringIndexOutOfBoundsException.

Syntax

substring(string, startIndex[, endIndex])

- Parameters

  string string - The starting string.

  integer startIndex - The index to start the substring at.

  integer endIndex - *Optional*. The end index of the substring.

- Results

  string - The substring from the start to end indexes of the specified string.

Examples

Code Snippet

```
substring("unhappy", 2) //returns "happy"
```

Code Snippet

```
substring("hamburger", 4, 8) //returns "urge"
```

toBinary

This function is used by **Ignition's Expression** language.

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(number)

- Parameters

  int number - The value to convert to binary.

- Results

  string - The string form of the binary representation of the specified number.

Examples

Code Snippet

```
toBinary(255) //returns "11111111"
```

Code Snippet

```
toBinary(-255) //returns "11111111111111111111111100000001"
```

toHex

This function is used by **Ignition's Expression** language.

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(number)

- Parameters

  int number - The number to convert to hex.

- Results

  string - A string that is the hex value of the specified value.

Examples

Code Snippet

```
toHex(255) //returns "FF"
```

Code Snippet

```
toHex(-255)  //returns "FFFFFF01"
```

toOctal

This function is used by **Ignition's Expression** language.

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(number)

- Parameters

  int number - The value to convert to octal.

- Results

  string - A string that is the octal of the specified value.

Examples

Code Snippet

```
toOctal(255) //returns "377"
```

Code Snippet

```
toOctal(-255) //returns "37777777401"
```

trim

This function is used by **Ignition's Expression** language.

Description

Takes the argument string and trims of any leading and/or trailing whitespace, returning the result.

Syntax

trim(string)

- Parameters

  string string - The starting string.

- Results

  string - The starting string with all whitespace removed.

Examples

Code Snippet

```
trim("Hello Carlos ") //returns "Hello Carlos"
```

Code Snippet

```
trim(" Goodbye.") //returns "Goodbye."
```

upper

This function is used by **Ignition's Expression** language.

Description
Takes a string and returns an upper-case version of it.

Syntax
upper(string) • Parameters string string - The string to make uppercase. • Results string - The starting string with all characters uppercase.

Examples
Code Snippet
`upper("Hello World") //returns "HELLO WORLD"`

stringFormat

This function is used by **Ignition's Expression** language.

Description
This expression returns a formatted string using the specified format string and arguments. Mainly, this expression is used for building dynamic string objects.

Syntax
stringFormat(format[, args...]) • Parameters string format - The a string that contains formatting elements in it (%s, %d, %i). string args - *Optional*. The arguments to use in the format. Must match the number of formatting elements in the string. • Results string - The new formatted string.

Formatting Elements

Element Character	Data Type to Substitute Element Character	Description
'b', 'B'	Boolean	If the corresponding argument *arg* is NULL, then a False is substituted into the format string. If *arg* is boolean, then the string conversion of *arg* will be substituted into the format string. For every other condition, True is substituted into the format string.
's', 'S'	String	The string value *arg* is substituted into the formatted sting.
'c', 'C'	Character	The unicode value *arg* is substituted into the formatted string.
'd'	Integral	The decimal value *arg* is substituted into the formatted string.
'f'	Floating Point	The floating point value *arg* is substituted into the formatted string.
't', 'T'	Date/Time	This is the **prefix** for date/time arg values to be used for string formatting. See the Date/Time Formatting Elements table below for additional characters.

Date/Time Formatting Elements

Date/Time Formatting Element Suffix	Description
'H'	Hour of the day for the 24–hour clock, formatted as two digits with a leading zero where necessary i.e. 00 - 23.
'I'	Hour of the 12-hour clock, formatted as two digits with a leading zero as necessary i.e. 01 - 12.
'k'	Hour of the day for the 24-hour clock, i.e. 0-24.
'k'	Hour of the 12-hour clock i.e. 1 - 12.
'M'	Minute within the hour formatted as two digits with a leading zero where necessary i.e. 00 - 59.
'S'	Seconds within a minute, formatted as two digits with a leading zero where necessary i.e. 00 - 59.
'L'	Millisecond within the second formatted as three digits with leading zeros as necessary i.e. 000-999.
'B'	Locale-specific full month name i.e. "January", "March".
'b'	Locale-specific abbreviated month name i.e. "Jan", "Mar".
'A'	Full name of the day of the week i.e. "Monday".
'a'	Abbreviated name of the day of the week i.e. "Mon".
'Y'	Year formatted as a 4 digit numeric value with leading zeros where necessary i.e. 0005 would be the year 5 in the Gregorian calendar.
'y'	Last two digits of the year formatted with leading zeros where necessary.
'j'	Day of the year formatted as three digits with leading zeros where necessary i.e. 001 - 366 for the Gregorian calendar.
'm'	Month, formatted as a two digit number with leading zeros where necessary i.e. 01-13.
'd'	Day of the month formatted as two digit number with leading zeros where necessary i.e. 00 - 31.
'e'	Day of the month formatted as two digits i.e. 0-31.

Examples

Code Snippet

```
stringFormat("The boolean value is: %b", null) //returns The boolean value is: False
```

Code Snippet

```
stringFormat("Hello %s", "world") //returns "Hello world"
```

Code Snippet

```
stringFormat("%s, %s, %s", 1, 2, 3) //returns "1, 2, 3"
```

Code Snippet

```
stringFormat("%d, %d, %d", 4, 5, 6) //returns "4, 5, 6"
```

Code Snippet

```
stringFormat("Today is: %tA", now()) //returns Today is: Tuesday
```

Code Snippet

```
stringFormat("The current month is: %tB", now()) //returns The current month is: June
```

urlEncode

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

This function is used by **Ignition's Expression** language.

Description

The expression function urlEncode() enables users to create an HTTP binding's URL on the fly. It will be possible to freely bind different URL path parameters to different component properties on a view or window.

The first argument is the string that's being encoded, second parameter is a boolean arg that controls whether you use query parameter style escaping or URI fragment style escaping.

Syntax

urlEncode(string, [usePercentEscape])

- Parameters

  string url- The URL to encode.

  bool usePercentEscape - *Optional*. False or black indicates to use query parameter style escaping. True indicates to use URI fragment style escaping.

- Results

  string- The encoded URL.

Examples

Code Snippet

```
urlEncode("Hello World")   //yields "Hello+World"
```

Code Snippet

```
urlEncode("Hello World", False)  //yields "Hello+World"
```

Code Snippet

```
urlEncode("Hello World", True)  //yields "Hello%20World"
```

Example

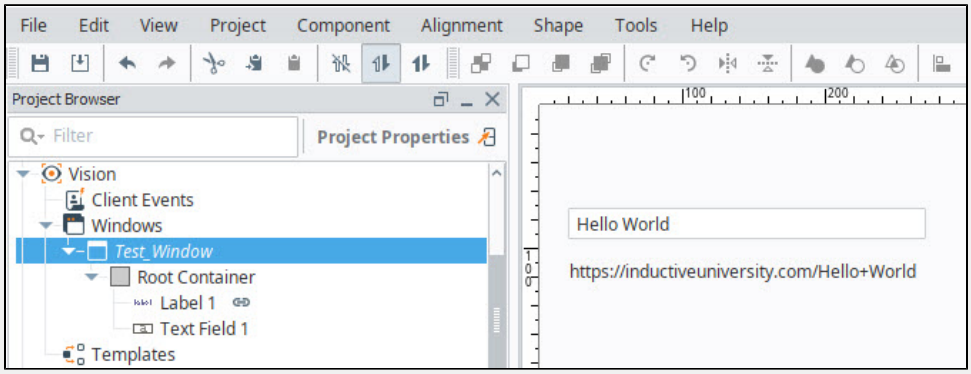
Vision Example

1. Drag a Text Field component and a Label component onto a window.
2. In the Vision Property Editor, enter a parameter for the URL you'll use. For this example, we used 'Hello World.'
3. Select the Label component, then click on the **Binding** icon for the label's Text property.
4. On the Property Binding screen, select **Expression**.
5. Enter the expression in the Configure Expression Binding section. In the example, we entered the URL plus the expression function, "urlEncode ()". We entered the oath to the Text Field's text property as the string.

Code Snippet

```
"https://inductiveuniversity.com/" + urlEncode({Root Container.Text Field 1.
text})
```

6. Alternatively, you can get the path for the text field by putting the cursor inside the parentheses, clicking the **Property Value** icon, then selecting the text property from the Text Field component.
7. Click **OK** to save the binding.The encoded URL is now displayed in the Label component.



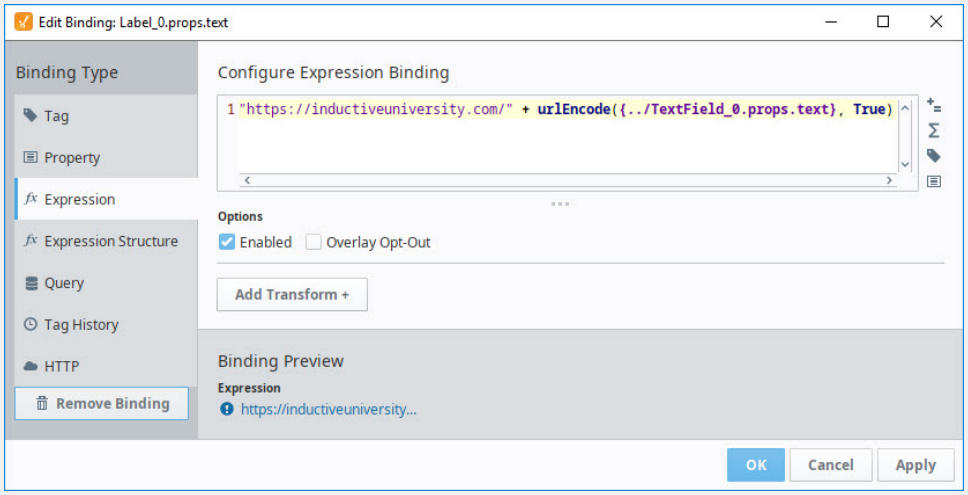
Example

Perspective Example

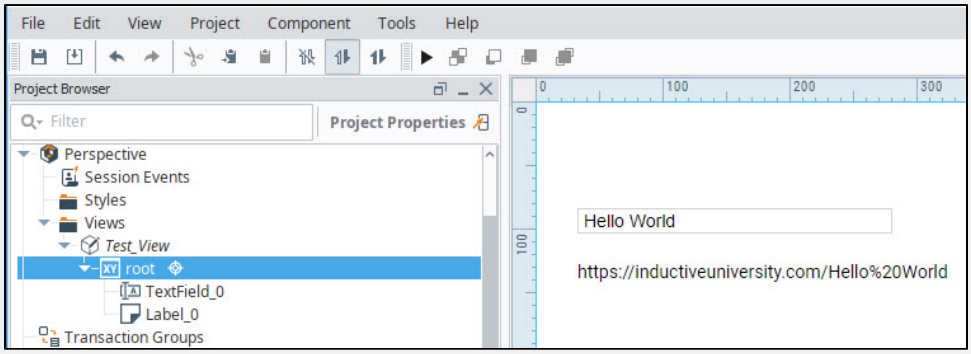
1. Drag a Text Field component and a Label component onto a view.
2. In the Perspective Property Editor, enter a parameter for the URL you'll use. For this example, we used 'Hello World.'
3. Select the Label component, then click on the **Binding** icon for the label's Text property.
4. On the Property Binding screen, select **Expression**.
5. Enter the expression in the Configure Expression Binding section. In the example, we entered the URL plus the expression function, "urlEncode ()". We entered the oath to the Text Field's text property as the string and the parameter "True" to use URI fragment style escaping.

Code Snippet

```
"https://inductiveuniversity.com/" + urlEncode({../TextField_0.props.text},
True)
```



6. Alternatively, you can get the path for the text field by putting the cursor inside the parentheses, clicking the **Property Value** icon, then selecting the text property from the Text Field component.
7. Click **OK** to save the binding.The encoded URL is now displayed in the Label component.



Translation

Translation Functions

The following functions allow you to interact with the Translation system through expression bindings.

In This Section ...

translate

This function is used by **Ignition's Expression** language.

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 exists in the client and Gateway scopes.

Syntax

translate(stringKey)

- Parameters

  string stringKey - The starting string to translate.

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

  string languageString - The language or locale to use.

- Results

  string - The starting string translated based on the current locale. If the translation does not exist, will return the specified value.

Examples

This expression function does not have any examples associated with it.

Type Casting

Type Casting Functions

The following functions allow you to change value types in expression bindings.

In This Section ...

toBoolean

This function is used by **Ignition's Expression** language.

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])

- Parameters

  object value - The value to type cast.

  object failover - *Optional*. The failover value if type casting fails.

- Results

  Bool - The value type cast as a bool.

Examples

Code Snippet

```
toBoolean(1) //returns true
```

Code Snippet

```
toBoolean("abc", false) //returns false
```

toBorder

This function is used by **Ignition's Expression** language.

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; font Color; 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])

- Parameters

  string value - The value to type cast.

  object failover - *Optional*. The failover value if type casting fails.

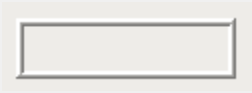
- Results

  Border - The value type cast as a border object.

Examples

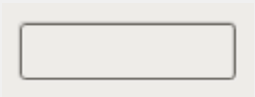
Code Snippet

```
toBorder("bevel;1010")  //returns this...
```



Code Snippet

```
toBorder("button")
```

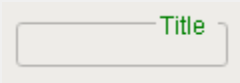


Code Snippet

```
toBorder("etched;0")
```

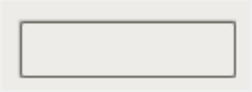
Code Snippet

```
toBorder("etchedtitled;Title;5;3;right;green;Arial")
```



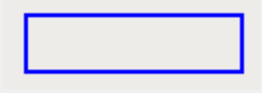
Code Snippet

```
toBorder("field")
```



Code Snippet

```
toBorder("line;blue;2")
```



Code Snippet

```
toBorder("linetitled;Title") //returns this...
```

Title

Code Snippet

```
toBorder("matte;red;10;1;1;1") //returns this...
```

Code Snippet

```
toBorder("paneltitled;Options;1;grey;white;0;3;0;green;Dialog,bold,16") //returns this...
```

Options

toColor

This function is used by **Ignition's Expression** language.

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]"

Note: This function was designed to return color objects to Vision bindings, and will not work with Perspective bindings. Instead, Perspective color properties can simply use string hex codes to derive a color from a binding. Example: "#00FF00".

Syntax

toColor(value[, failover])

- Parameters

  string value - The color value as a string.

  object failover - *Optional*. The failover value if type casting fails.

- Results

  Color - The value type cast as a color object.

Examples

Code Snippet

```
//All of these expressions return the color red.
toColor("red")
toColor("#FF0000")
toColor("255,0,0")
```

Code Snippet

```
//You can use the failover parameter to ensure that this expression returns something even if
the input string may be bad:
toColor({UserOptions/CustomColor}, "black")
```

Color Options

Both 'Grey' and 'Gray' are accepted as valid colors as well as the iterations of that color such as 'DarkGrey' and 'DarkGray'.

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	see Transparent
Coral	#FF7F50
CornflowerBlue	#6495ED
Cornsilk	#FFF8DC
Crimson	#DC143C
Cyan	#00FFFF
DarkBlue	#00008B
DarkCyan	#008B8B
DarkGoldenRod	#B8860B
DarkGray DarkGrey	#A9A9A9
DarkGreen	#006400
DarkKhaki	#BDB76B
DarkMagenta	#8B008B
DarkOliveGreen	#556B2F
Darkorange	#FF8C00
DarkOrchid	#9932CC
DarkRed	#8B0000
DarkSalmon	#E9967A
DarkSeaGreen	#8FBC8F
DarkSlateBlue	#483D8B
DarkSlateGray DarkSlateGrey	#2F4F4F
DarkTurquoise	#00CED1
DarkViolet	#9400D3
DeepPink	#FF1493
DeepSkyBlue	#00BFFF

DimGray DimGrey	#696969
DodgerBlue	#1E90FF
Feldspar	#D19275
FireBrick	#B22222
FloralWhite	#FFFAF0
ForestGreen	#228B22
Fuchsia	#FF00FF
Gainsboro	#DCDCDC
GhostWhite	#F8F8FF
Gold	#FFD700
GoldenRod	#DAA520
Gray Grey	#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
LightGray LightGrey	#D3D3D3
LightPink	#FFB6C1
LightSalmon	#FFA07A
LightSeaGreen	#20B2AA
LightSkyBlue	#87CEFA
LightSlateBlue	#8470FF

LightSlateGray LightSlateGrey	#778899
LightSteelBlue	#B0C4DE
LightYellow	#FFFFE0
Lime	#00FF00
LimeGreen	#32CD32
Linen	#FAF0E6
Magenta	#FF00FF
Maroon	#800000
MediumAquaMarine	#66CDAA
MediumBlue	#0000CD
MediumOrchid	#BA55D3
MediumPurple	#9370DB
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	#DB7093
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 SlateGrey	#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

This function is used by **Ignition's Expression** language.

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]) • Parameters object value - The value to type cast, typically a DataSet or PyDataSet. object failover - *Optional*. The failover value if type casting fails. • Results DataSet - The value type cast as a dataset.

Examples

Code Snippet

```
toDataSet(runScript("app.funcs.runSomeFunction()")) //coerces the value returned by the a
project scripting function into a dataset.
```

toDate

This function is used by **Ignition's Expression** language.

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 the supported formats:

- yyyy-MM-dd
- MM/dd/yyyy
- MM/dd/yyyy HH:mm:ss
- hh:mm:ss a
- hh:mm a
- MM/dd/yyyy hh:mm:ss a
- yyyy-MM-dd HH:mm:ss.SSS
- yyyy-MM-dd HH:mm:ss
- EEE MMM dd HH:mm:ss z yyyy
- yyyyMMdd.HHmmssSSSZ

If not, type casting fails.The failover value must be a number or string with the same restrictions.

Syntax

toDate(value[, failover])

- Parameters

  object value - The value to type cast into a date.

  object failover - *Optional*. The failover value if type casting fails.

- Results

  date - The value type cast as a date.

Examples

Code Snippet

```
toDate("2007-04-12 16:28:22") //returns April 12th, 2007, 4:28:22 PM
```

toDouble

This function is used by **Ignition's Expression** language.

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])

- Parameters

  object value - The value to type cast.

  object failover - *Optional*. The failover value if type casting fails.

- Results

  Double - The value type cast as a double.

Examples

Code Snippet

```
toDouble("38.772") //returns 38.772
```

Code Snippet

```
toDouble({Root Container.Text Field.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

This function is used by **Ignition's Expression** language.

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])

- Parameters

  object value - The value to type cast.

  object failover - *Optional*. The failover value if type casting fails.

- Results

  Float - The value type cast as a float.

Examples

Code Snippet

```
toFloat("38.772") //returns 38.772
```

Code Snippet

```
toFloat({Root Container.Text Field.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

This function is used by **Ignition's Expression** language.

Description

Coerces a string into a font. The string must be in the format:

'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])

- Parameters

  string value - The value to type cast to a font.

  object failover - *Optional*. The failover value if type casting fails.

- Results

  Font - The value type cast as a font.

Examples

Code Snippet

```
toFont("font(Dialog,Bold,12)") //returns the standard font used in most clients.
```

toInt

This function is used by **Ignition's Expression** language.

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])

- Parameters

  object value - The value to type cast.

  object failover - *Optional*. The failover value if type casting fails.

- Results

  int - The value type cast as an int.

Examples

Code Snippet

```
toInt("38") //returns 38
```

Code Snippet

```
toInt("33.9") // returns 34
```

Code Snippet

```
toInt({Root Container.Text Field.text}, -1) //returns the value in the text box as an int, or -1
if the value doesn't represent an number.
```

toInteger

This function is used by **Ignition's Expression** language.

Description
Identical to the toInt expression function.

Syntax
toInteger(value[, failover]) • Parameters object value - The value to type cast. object failover - *Optional*. The failover value if type casting fails. • Results int - The value type cast as an int.

Examples
This expression function does not have examples associated with it.

toLong

This function is used by **Ignition's Expression** language.

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])

- Parameters

  object value - The value to type cast.

  object failover - *Optional*. The failover value if type casting fails.

- Results

  Long - The value type cast as a long.

Examples

Code Snippet

```
toLong("38") //returns 38
```

Code Snippet

```
toLong("33.9") //returns 34
```

Code Snippet

```
toLong({Root Container.Text Field.text}, -1) //returns the value in the text box as an long, or
-1 if the value doesn't represent an number.
```

toStr

This function is used by **Ignition's Expression** language.

Description
Identical to the toString expression function.

Syntax
toStr(value[, failover]) • Parameters object value - The value to type cast, typically a DataSet or PyDataSet. object failover - *Optional*. The failover value if type casting fails. • Results string - The value type cast as a string.

Examples
There are no examples associated with this expression function.

toString

This function is used by **Ignition's Expression** language.

Description

Represents the value as a string. Will succeed for any type of value.

Syntax

toString(value[, failover])

- Parameters

  object value - The value to type cast.

  object failover - *Optional*. The failover value if type casting fails.

- Results

  string - The value type cast as a string.

Examples

Code Snippet

```
toString(1/3.0) // returns "0.3333333333333333"
```

Code Snippet

```
toString({Root Container.Table.data}) //returns something like: "Dataset [150R x 3C]"
```

Users

User Functions

The following functions allow you to interact with the User/Role system through expression bindings.

In This Section ...

hasRole

This function is used by **Ignition's Expression** language.

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])

- Parameters

  string role - The name of a role.

  string username - *Optional*. A username. Defaults to the current user.

  string usersource - *Optional*. The usersource of the username. Defaults to the usersource of the current user.

- Results

  Bool - True if the specified user has the specified role, False if not.

Examples

Code Snippet

```
// This is an example using a username and userSource:
hasRole("Administrator", "bob", "default")
```

Code Snippet

```
// This is an example using the current user and default userSource in the client scope:
hasRole("Administrator")
```

isAuthorized

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

This function is used by **Ignition's Expression** language.

Description
Returns a qualified value with a boolean value which is true if the user in the current session is authorized, false otherwise.

Syntax

isAuthorized(isAllOf, securityLevel[, securityLevelN...])

- Parameters

  boolean isAllOf - True if the current user must have all of the given security levels to be authorized, false if the current user must have at least one of the given security levels to be authorized

  string securityLevels - One or more String paths to a security level node in the form "Path/To/Node". Each level in the tree is delimited by a forward slash character. Additional security level paths are simply added to the end of the parameter list. The Public node is never a part of the path.

- Results

  Bool - Returns a qualified value with a boolean value which is true if the user in the current session is authorized, false otherwise. The quality of the qualified value is the worst of the qualities of all the qualified values of each argument.

Examples

Code Snippet

```
// returns true if the current user has both Administrator and Baz roles
// returns false if they have only one or if they have neither
isAuthorized(true, 'Authenticated/Roles/Administrator', 'Foo/Bar/Baz')
```