

Ignition 8.3 User Manual — Expression Function Reference

Generated 2026-10-12 from
<https://www.docs.inductiveautomation.com/docs/8.3/>

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](#).

 **JSON**

 **Advanced**

 **Aggregates**

 **Alarming**

 **Colors**

 **Date and Time**

 **Identity Provider**

 **Logic**

 **Math**

 **MongoDB - Expression Binding**

 **String**

 **Translation**

 **Type Casting**

 **Users**

JSON

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

jsonFormat

Takes a string, and returns a prettyprints string

jsonGet

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

jsonSet

Takes a JSON friendly string, a path string, and value, and will return a new JSON friendly string with the provide...

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 respect 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.

Initial String	
{ 'employees': { 'firstName': 'John', 'lastName': 'Doe' } }	
Binding with jsonFormat	Binding without jsonFormat
{ "employees": { "firstName": "John", "lastName": "Doe" } }	{ 'employees': { 'firstName': 'John', 'lastName': 'Doe' } }

Syntax

```
jsonFormat(string)
```

Parameters

Type	Parameter	Description
String	<pre>string</pre>	The string to format. The string must be in a JSON-friendly format; see examples below for possibilities.

Results

String - A prettyprint 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 data type, 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

Type	Parameter	Description
String	<code>json</code>	The JSON string. The string must be in a JSON-friendly format.
String	<code>path</code>	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

Type	Parameter	Description
String	<code>json</code>	The JSON string. The string must be in a JSON friendly format.
String	<code>path</code>	The path string.
Object	<code>value</code>	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)
```

Advanced

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

columnRearrange

Returns a view of the given dataset with the given columns in the order specified.

columnRename

Returns a view of the given dataset with the columns renamed.

forceQuality

Returns the given value, but overwrites the quality of that value.

property

Returns an object representing the value of the property at the path specified. It takes a single string as an argument.

qualifiedValue

Returns the given value, but overwrites the quality of that value.

qualityOf

Returns the QualityCode of a qualified value.

runScript

Runs a single line of Python code as an expression.

sortDataset

Returns a new dataset based on the rows in the given dataset.

tag

Returns an object representing the value of the Tag at the path specified.

timestampOf

Returns the timestamp of a qualified value.

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

Type	Parameter	Description
Dataset	<code>dataset</code>	The starting dataset.
String	<code>col</code>	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

```
// Returns a 3 column Dataset from a Table component called "fiveColDataset" in
the same Root Container, where the columns are in the given order.
columnRearrange({Root Container.fiveColDataset.data}, "secondCol", "thirdCol",
"firstCol")
```


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

Type	Parameter	Description
Dataset	<code>dataset</code>	The starting dataset.
String	<code>newName</code>	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.

 CAUTION

Be aware the quality codes linked in the description above access the 7.9 quality codes, which are the codes the forceQuality function uses. For an option using the 8.1 quality codes, see the [qualifiedValue](#) function page.

Syntax

```
forceQuality(value[, qualityCode])
```

Parameters

Type	Parameter	Description
Object	<code>value</code>	The value to force a quality on.
Integer	<code>qualityCode</code>	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

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

Type	Parameter	Description
String	<code>propertyPath</code>	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

Type	Parameter	Description
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	The subcode to include with the quality level. See Quality Codes and Overlays for more information on possible subcodes. [optional]
Object	diagnosticMessage	The diagnostic message to add to the quality. [optional]

Results

QualifiedValue - The value with a forced quality, formatted as a QualifiedValue.

Examples

Code Snippet

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

qualityOf

This function is used by Ignition's Expression language.

Description

Returns the [QualityCode](#) of a qualified value.

Syntax

```
qualityOf(value)
```

Parameters

Type	Parameter	Description
Object	<code>value</code>	The value for which you want to find the quality.

Results

QualityCode - Returns the code associated with the quality. For more information on quality codes, including a list of available codes, see [Quality Codes and Overlays](#).

Examples

Code Snippet

```
qualityOf({[default]Sensor 1/Reading}) //Returns the quality code of the tag's value
```

Code Snippet

//Use "=" to compare quality codes against constant names *or* integer codes

```
(qualityOf({[default]Sensor 1/Reading}) = {../Label.props.text}) //Returns "true"  
if the quality code matches a property
```

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 zipcode, and then bind the value of a label to the return value of that function.

The **scriptFunction** is entered as a **string** and the **pollRate** is in **milliseconds**. You can optionally add any function arguments after the poll rate.

NOTE

Normally expressions execute fairly quickly when compared to a script. However, calling `runScript` will mitigate the speed advantage of an expression. In most cases this is unnoticeable, but calling long running scripts with `runScript` can negatively impact performance.

Calling runScript 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

Type	Parameter	Description
String	<code>scriptFunction</code>	A single line of Python code. Typically the path to a script module.
Integer	<code>pollRate</code>	The poll rate of the script in milliseconds. [optional]
Object	<code>arg</code>	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 <code>runScript</code> represents how the parameters will be passed to the underlying Python function. [optional]

Results

Object - The return value of the specified function.

Syntax - Legacy

```
runScript(scriptFunction, [pollRate])
```

Parameters

Type	Parameter	Description
String	<code>scriptFunction</code>	A string representing a single line of code, including any arguments that will be passed to the function.
Integer	<code>pollRate</code>	The poll rate of the script in milliseconds. [optional]

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
A1	Z3	A1
a4	a22	a22
a7z9	z3	a1
a22	A1	z3
a77z4	a77z4	a7z9
a77z99	a77z99	a4
Z3	a4	a77z99
z3	a7z9	a77z4

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 (index)

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

Parameters

Type	Parameter	Description
Dataset	<code>dataset</code>	The starting dataset.
Integer	<code>colIndex</code>	The index of the column to sort on.
Boolean	<code>ascending</code>	A flag indicating whether or not to sort ascending. Defaults to true. [optional]
Boolean	<code>naturalOrdering</code>	A flag indicating the ordering method. True for natural, false for alphabetical. Defaults to true. [optional]

Results

DataSet - A sorted dataset.

Syntax (name)

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

Parameters

Type	Parameter	Description
Dataset	<code>dataset</code>	The starting dataset.
String	<code>colName</code>	The name of the column to sort on.
Boolean	<code>ascending</code>	A flag indicating whether or not to sort ascending. Defaults to true. [optional]
Boolean	<code>naturalOrdering</code>	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.

NOTE

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

Type	Parameter	Description
String	<code>tagPath</code>	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.
```

timestampOf

This function is used by Ignition's Expression language.

Description

Returns the timestamp of a qualified value.

Syntax

```
timestampOf(value)
```

Parameters

Type	Parameter	Description
Object	<code>value</code>	The value for which you want to find the timestamp.

Results

Date - Returns the timestamp of the value's last update.

Examples

Code Snippet

```
timestampOf({[default]Sensor 1/Reading}) //Returns the timestamp of the tag's last value update
```

Code Snippet

```
timestampOf(tag("[default]Sensor 1/Reading")) //Returns the timestamp of the tag's value at the time the expression was evaluated on the Gateway
```

Code Snippet

```
// Using tag() with a string tagPath
timestampOf(tag("[default]Sensor 1/Reading")) //Example return: 2025-01-23
08:30:00 (evaluated at the Gateway)

// Using a bound value
timestampOf({[default]Sensor 1/Reading}) //Example return: 2025-01-23 08:30:02
(timestamp from the OPC source)
```

Code Snippet

```
{[default]Sensor 1/Reading.timestamp} //Directly accesses the tag's timestamp
property, returning the timestamp from the OPC source
```

typeof

This function is used by Ignition's Expression language.

Description

Returns the simple name of the Java type.

Syntax

```
typeof(value)
```

Parameters

Type	Parameter	Description
Any	<code>value</code>	The object for which you want to find the Java type.

Results

String - Returns the simple name of the Java type.

Examples

Code Snippet

```
// Takes in a string value, returns "String".
typeof("My String")
```

Code Snippet

```
// Takes in a Tag's value using the Tag expression function. Returns a string
representing the datatype of the Tag.
typeof(tag("[default]WritableInteger1"))
```

Code Snippet

```
// Takes in a color type property from a Vision component. Returns  
"ColorUIResource".  
typeof({Root Container.Label.foreground})
```

Aggregates

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

groupConcat

Concatenates all of the values in a specified column or collection into a string, with each value separated by the s...

max

Finds and returns the maximum value in the given column of the given dataset, or the max value in a series of nu...

maxDate

Finds and returns the maximum date in the given column of the given dataset, or the max value in a series of date...

mean

Calculates the mean (a.k.a average) for the numbers in the given column of the given dataset or the mean of a ser...

median

Calculates the median for the numbers in the given column of the given dataset or the median of a series of numb...

min

Finds and returns the minimum value in the given column of the given dataset, or the min value in a series of nu...

minDate

Finds and returns the minimum date in the given column of the given dataset, or the min value in a series of date...

stdDev

Calculates the sample standard deviation of the values in the given column of the given dataset, or the standard ...

sum

Calculates the sum of the values in the given column of the given dataset, or the sum for a series of numbers spe...

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. Concatenation for collections is also supported with the [collection syntax](#).

Syntax (index)

```
groupConcat(dataset, columnIndex, separator)
```

Parameters

Type	Parameter	Description
Dataset	<code>dataset</code>	The starting dataset.
Integer	<code>columnIndex</code>	The index of the column to concatenate.
String	<code>separator</code>	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 (name)

```
groupConcat(dataset, columnName, separator)
```

Parameters

Type	Parameter	Description
Dataset	<code>dataset</code>	The starting dataset.
String	<code>columnName</code>	The name of the column to concatenate.
String	<code>separator</code>	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 (collection)

```
groupConcat(collection, separator)
```

Parameters

Type	Parameter	Description
Collection	<code>collection</code>	The starting list, tuple, or set to use.
String	<code>separator</code>	What will be used to separate each of the values.

Returns

String - A string with every value in the specified collection separated by the separator value.

Examples

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

Syntax (index) Code Snippet

```
groupConcat({Root Container.Table.data}, 1, " / ")  
//returns the string: "380 / 120 / 125 / 322"
```

Syntax (name) Code Snippet

```
groupConcat({Root Container.Table.data}, "ProductCode", ", ")  
//returns 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. If there are no rows in the dataset, null is returned.

Syntax (index)

```
max(dataset, columnIndex)
```

Parameters

Type	Parameter	Description
Dataset	<code>dataset</code>	The dataset to search through.
Integer	<code>columnIndex</code>	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

Integer - The maximum value in that column.

Syntax (name)

```
max(dataset, columnName)
```

Parameters

Type	Parameter	Description
Dataset	<code>dataset</code>	The dataset to search through.
String	<code>columnName</code>	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

Integer - The maximum value in that column.

Syntax

```
max(value[, value...])
```

Parameters

Type	Parameter	Description
Integer/Float	<code>value</code>	A number. Can be as many values as needed. Can be either a float or an integer.

Returns

Integer - The maximum value in the list of values.

Syntax

```
max(sequence)
```

Parameters

Type	Parameter	Description
Sequence	<code>sequence</code>	A list, tuple, array, or set of numerical values.

Returns

Integer - 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 (index)

```
maxDate(dataset, columnIndex)
```

Parameters

Type	Parameter	Description
Dataset	<code>dataset</code>	The starting dataset to search.
Integer	<code>columnIndex</code>	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 (name)

```
maxDate(dataset, columnName)
```

Parameters

Type	Parameter	Description
Dataset	<code>dataset</code>	The starting dataset to search.
String	<code>columnName</code>	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 (date)

```
maxDate(date[, date])
```

Parameters

Type	Parameter	Description
Date	<code>date</code>	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 (index)

```
mean(dataset, columnIndex)
```

Parameters

Type	Parameter	Description
Dataset	<code>dataset</code>	The dataset to use.
Integer	<code>columnIndex</code>	The index of the column to use. Must be a column index of the provided dataset.

Returns

Integer/Float - The mean of the values in that column.

Syntax (name)

```
mean(dataset, columnName)
```

Parameters

Type	Parameter	Description
Dataset	<code>dataset</code>	The dataset to use.
String	<code>columnName</code>	The name of the column to search through. Must match a column name in the provided dataset.

Returns

Integer/Float - The mean of the values in that column.

Syntax (value)

```
mean(value[, value...])
```

Parameters

Type	Parameter	Description
Integer/Float	<code>value</code>	A number. Can be as many values as needed. Can be either a float or an integer.

Returns

Integer/Float - The mean of the values.

Syntax (sequence)

```
mean(sequence)
```

Parameters

Type	Parameter	Description
Sequence	<code>sequence</code>	A list, tuple, array, or set of numerical values.

Returns

Integer/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 (index)

```
median(dataset, columnIndex)
```

Parameters

Type	Parameter	Description
Dataset	<code>dataset</code>	The dataset to search through.
Integer	<code>columnIndex</code>	The index of the column to search through. Must be a column index of the provided dataset.

Returns

Integer/Float - The median value in that column.

Syntax (name)

```
median(dataset, columnName)
```

Parameters

Type	Parameter	Description
Dataset	<code>dataset</code>	The dataset to search through.
String	<code>columnName</code>	The name of the column to search through. Must match a column name in the provided dataset.

Returns

Integer/Float - The median value in that column.

Syntax (value)

```
median(value[, value...])
```

Parameters

Type	Parameter	Description
Integer/Float	<code>value</code>	A number. Can be as many values as needed. Can be either a float or an integer.

Returns

Integer/Float - The median value in the list of values.

Syntax (sequence)

```
median(sequence)
```

Parameters

Type	Parameter	Description
Sequence	<code>sequence</code>	A list, tuple, array, or set of numerical values.

Returns

Integer/Float - The median 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

```
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 (index)

```
min(dataset, columnIndex)
```

Parameters

Type	Parameter	Description
Dataset	<code>dataset</code>	The dataset to search through.
Integer	<code>columnIndex</code>	The index of the column to search through. Must be a column index of the provided dataset.

Returns

Integer - The minimum value in that column.

Syntax (name)

```
min(dataset, columnName)
```

Parameters

Type	Parameter	Description
Dataset	<code>dataset</code>	The dataset to search through.
String	<code>columnName</code>	The name of the column to search through. Must match a column name in the provided dataset.

Returns

Integer - The minimum value in that column.

Syntax (value)

```
min(value[, value...])
```

Parameters

Type	Parameter	Description
Integer/Float	<code>value</code>	A number. Can be as many values as needed. Can be either a float or an integer.

Returns

Integer - The minimum value in the list of values.

Syntax (sequence)

```
min(sequence)
```

Parameters

Type	Parameter	Description
Sequence	<code>sequence</code>	A list, tuple, array, or set of numerical values.

Returns

Integer - 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 (index)

```
minDate(dataset, columnIndex)
```

Parameters

Type	Parameter	Description
Dataset	<code>dataset</code>	The starting dataset to search.
Integer	<code>columnIndex</code>	The index of the column to search for the min 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 (name)

```
minDate(dataset, columnName)
```

Parameters

Type	Parameter	Description
Dataset	<code>dataset</code>	The starting dataset to search.
String	<code>columnName</code>	The name of the column to search for the min 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 (date)

```
minDate(date[, date])
```

Parameter

Type	Parameter	Description
Date	<code>date</code>	A date. Can be as many dates as needed.

Returns

Date - The minimum date of the given dates.

Examples

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

Code Snippet

```
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 (index)

```
stdDev(dataset, columnIndex)
```

Parameters

Type	Parameter	Description
Dataset	<code>dataset</code>	The starting dataset to search.
Integer	<code>columnIndex</code>	The index of the column to search through. Must be a column index of the provided dataset.

Returns

Integer/Float - The standard deviation of the values in that column.

Syntax (name)

```
stdDev(dataset, columnName)
```

Parameters

Type	Parameter	Description
Dataset	<code>dataset</code>	The starting dataset to search.
String	<code>columnName</code>	The name of the column to search through. Must match a column name in the provided dataset.

Returns

Integer/Float - The standard deviation of the values in that column.

Syntax (value)

```
stdDev(value, [value, ...])
```

Parameters

Type	Parameter	Description
Integer/Float	<code>value</code>	A number. Can be as many values as needed. Can be either a float or an integer.

Returns

Integer/Float - The standard deviation of the values in the list of values.

Syntax (sequence)

```
stdDev(sequence)
```

Parameters

Type	Parameter	Description
Sequence	<code>sequence</code>	A list, tuple, array, or set of numerical values.

Returns

Integer/Float - The standard deviation of the values in the list of values.

Examples

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 4.00532
```

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 (index)

```
sum(dataset, columnIndex)
```

Parameters

Type	Parameter	Description
Dataset	<code>dataset</code>	The dataset to use.
Integer	<code>columnIndex</code>	The index of the column to use. Must be a column index of the provided dataset.

Returns

Integer/Float - The sum of the values in that column.

Syntax (name)

```
sum(dataset, columnName)
```

Parameters

Type	Parameter	Description
Dataset	<code>dataset</code>	The dataset to use.
String	<code>columnName</code>	The name of the column to search through. Must match a column name in the provided dataset.

Returns

Integer/Float - The sum of the values in that column.

Syntax (value)

```
sum(value, [value, ...])
```

Parameters

Type	Parameter	Description
Integer/Float	<code>value</code>	A number. Can be as many values as needed. Can be either a float or an integer.

Returns

Integer/Float - The sum of the values.

Syntax (sequence)

```
sum(sequence)
```

Parameters

Type	Parameter	Description
Sequence	<code>sequence</code>	A list, tuple, array, or set of numerical values.

Returns

Integer/Float - The sum of the values.

Examples

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

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

isAlarmActive

Returns whether there are active alarms that match the provided criteria.

isAlarmActiveFiltered

Returns whether there are active alarms that match the provided criteria.

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 Vision Client scope.

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

Syntax

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

Parameters

Type	Parameter	Description
String	tagPath	The Tag path to search for active alarms. Supports the wildcard '*'.
String	alarmName	The name of the alarm to search for. Supports the wildcard '*'. [optional]
Integer	pollRate	The poll rate in milliseconds. Only applicable in Vision Clients. [optional]

Returns

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

Examples

Code Snippet

```
isAlarmActive("[default]Tanks/Temp", "[default]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, allowAked, allowShelved, [pollRate])
```

Parameters

Type	Parameter	Description
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 '*'.
Integer	minPriority	The minimum priority of alarms to accept. 0 is Diagnostic, 1 is Low, 2 is Medium, 3 is High, 4 is Critical.
Integer	maxPriority	The maximum priority of alarms to accept. 0 is Diagnostic, 1 is Low, 2 is Medium, 3 is High, 4 is Critical.
Boolean	allowCleared	A flag that indicates whether to accept cleared alarms.
Boolean	allowAked	A flag that indicates whether to accept acknowledged alarms.
Boolean	allowShelved	A flag that indicates whether to accept shelved alarms.
Integer	pollRate	The poll rate of the function in milliseconds. Only applicable in the Client scope. [optional]

Results

Boolean - 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

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

brighter

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

color

Creates a color using the given red, green, and blue amounts, which are integers between 0-255.

darker

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

gradient

Calculates a percentage given the three numeric arguments number, low, and high.

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)
```

Parameters

Type	Parameter	Description
Color	<code>color</code>	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, for example: "#00FF00".

Syntax

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

Parameters

Type	Parameter	Description
Integer	<code>red</code>	The intensity of red, between 0 - 255.
Integer	<code>green</code>	The intensity of green, between 0 - 255.
Integer	<code>blue</code>	The intensity of blue, between 0 - 255.
Integer	<code>alpha</code>	The amount of transparency, between 0 - 255. [optional]

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)
```

Parameters

Type	Parameter	Description
Color	<code>color</code>	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

Code Snippet

```
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)
```

Parameters

Type	Parameter	Description
Integer	<code>value</code>	The value used to determine the percentage between the low and high values.
Integer	<code>low</code>	The low value to use to calculate the percentage.
Integer	<code>high</code>	The high value to use to calculate the percentage.
Color	<code>lowColor</code>	The color that will match 0%.
Color	<code>highColor</code>	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 Snippet

```
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

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

add*

Add or subtract an amount to a given date and time

dateArithmetic

Adds or subtracts some amount of time from a date, returning the resulting date.

dateDiff

Calculates the difference between the two dates, returning the result as a floating point value in the units specifie...

dateExtract

Returns an integer value that is the value of the specified date field within the given date.

dateFormat

Returns the given date as a string, formatted according to a pattern.

dateIsAfter

Compares two dates to see if date1 is after date2.

dateIsBefore

Compares two dates to see if date1 is before date2

dateIsBetween

Compares two dates to see if a target date is between two other dates.

dateIsDaylight

Checks to see if the current timezone is using daylight savings time during the date specified.

***Between**

Calculates the amount of time between two dates

fromMillis

Creates a date object given a millisecond value.

 get*

Extracts a unit of time from a date

 getDate

Creates a new Date object given a year, month and a day.

 getTimezone

Returns the ID of the current timezone depending on the scope in which it is called.

 getTimezoneOffset

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

 getTimezoneRawOffset

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

 midnight

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

 now

Returns the current time.

 setTime

Takes in a date, and returns a copy of it with the time fields set as specified.

 timeBetween

Checks to see if the given time is between the start and end times.

 toMillis

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

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

Type	Parameter	Description
Date	<code>date</code>	The starting date.
Integer	<code>value</code>	The amount of units to change the date by, where the units depend on the specific function used.

Returns

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

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)
```

Parameters

Type	Parameter	Description
Date	<code>date</code>	The starting date.
Integer	<code>value</code>	The amount to add or subtract from the given date.
String	<code>field</code>	The units of the value.

Returns

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

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)
```

Parameters

Type	Parameter	Description
Date	<code>date1</code>	The first date.
Date	<code>date2</code>	The second date.
String	<code>field</code>	The units for the difference.

Returns

Float - The difference between the 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 1.0
```

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

Type	Parameter	Description
Date	<code>date</code>	The given date to extract the field from.
String	<code>field</code>	The field to extract.

Returns

Integer - The value of the specified field within the given date. Note that months are zero-indexed, meaning January is 0, February is 1, and so on.

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.

Refer to [Data Type Formatting Reference](#) for a table of the placeholders.

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)
```

Parameters

Type	Parameter	Description
Date	<code>date</code>	The starting date to format.
String	<code>pattern</code>	The pattern to format the given date.

Returns

String - The given date formatted according to the specified 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. This is exclusive, meaning if the dates are identical the result is always false.

Syntax

```
dateIsAfter(date1, date2)
```

Parameters

Type	Parameter	Description
Date	<code>date1</code>	The first date to compare.
Date	<code>date2</code>	The second date to compare.

Returns

Boolean - Returns true if date1 is after date2. Returns false if date1 is the same as or before date2.

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. This is exclusive, meaning if the dates are identical the result is always false.

Syntax

```
dateIsBefore(date1, date2)
```

Parameters

Type	Parameter	Description
Date	<code>date1</code>	The first date to compare.
Date	<code>date2</code>	The second date to compare.

Returns

Boolean - Returns true if date1 is before date2. Returns false if date1 is the same as or after date2.

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. 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

Type	Parameter	Description
Date	<code>targetDate</code>	The date to compare.
Date	<code>startDate</code>	The start of the date range.
Date	<code>endDate</code>	The end of the date range. This date must be after the start date.

Returns

Boolean - Returns true if the targetDate is between (or equal to) the startDate and endDate. Returns false otherwise.

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

Type	Parameter	Description
Date	<code>date</code>	The date to use for the check. If omitted, the current date will be used. [optional]

Returns

Boolean - Returns true if the current time zone is using Daylight Saving Time during the specified date, or false otherwise.

Examples

Code Snippet

```
dateIsDaylight(toDate("2007-06-28 00:00:00")) // Will return True in the  
US/Pacific Timezone, due to that time zone's observation of Daylight Saving 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)
```

Parameters

Type	Parameter	Description
Date	<code>date1</code>	The first date to compare.
Date	<code>date2</code>	The second date to compare.

Returns

Integer - The number of units between the two dates. The units are determined by the specific function used.

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

Type	Parameter	Description
Integer	<code>millis</code>	The number of milliseconds since Unix epoch time (1 January 1970 at midnight UTC).

Returns

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

Examples

Code Snippet

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



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

Type	Parameter	Description
Date	<code>date</code>	The date to extract from.

Returns

Float - The value of the specific unit of time extracted from the date. The unit is determined by the specific function used.

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.

Syntax

```
getDate(year, month, day)
```

Parameters

Type	Parameter	Description
Integer	<code>year</code>	The year to set for the date.
Integer	<code>month</code>	The month to set for the date. The month is zero-based, so January is 0 and December is 11.
Integer	<code>day</code>	The day to set for the date. The day is one-based, so the first day of the month is 1.

Returns

Date - A date object created from the specified year, month, and day, with the time set to midnight of that day.

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.

▼ Details

List of java timezone IDs

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/Fakaofu
- 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

Syntax

```
getTimezone()
```

Parameters

This function does not take any parameters.

Returns

String - The ID of the current timezone.

getTimezoneOffset

This function is used by Ignition's Expression language.

Description

Returns the current time zone's offset versus UTC for a given instant, taking Daylight Saving Time into account.

Syntax

```
getTimezoneOffset([date])
```

Parameters

Type	Parameter	Description
Date	<code>date</code>	A specified date to compare the current timezone to UTC. If omitted, the current time will be used. [optional]

Returns

Float - The offset of the current timezone from UTC, considering Daylight Saving Time if applicable.

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 Saving Time would be in effect.
```

getTimezoneRawOffset

This function is used by Ignition's Expression language.

Description

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

Syntax

```
getTimezoneRawOffset()
```

Parameters

This function does not take any parameters.

Returns

Float - The offset of the current timezone from UTC, without considering Daylight Saving Time.

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

Type	Parameter	Description
Date	<code>date</code>	The date to set to midnight.

Returns

Date - A new date with the hour, minute, second, and millisecond fields set to zero.

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.

This function is one of the few expression functions that will poll. If you do not specify a pollRate, it will default to 1,000ms. If you do not want this function to poll, use a poll rate of zero.

Syntax

```
now([pollRate])
```

Parameters

Type	Parameter	Description
Integer	<code>pollRate</code>	The poll rate in milliseconds to update the time. Defaults to 1,000 ms. [optional]

Returns

Date - The current time, based on the host computer's system clock. If a poll rate is specified, the returned time will update at the given interval.

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

Type	Parameter	Description
Date	<code>date</code>	A starting date.
Integer	<code>hour</code>	The value to set the hour field to.
Integer	<code>minute</code>	The value to set the minute field to.
Integer	<code>second</code>	The value to set the second field to.

Returns

Date - A new date with the time fields set as specified. The millisecond field is not preserved.

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, endDate)
```

Parameters

Type	Parameter	Description
Date/String	date	The date or time to compare. Can be a Date object or a string.
Date/String	startDate	The start date or time. Can be a Date object or a string.
Date/String	endDate	The end date or time. Can be a Date object or a string.

Returns

Boolean - Returns true if the given date or time is between the startDate and endDate, or false if it is 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

Type	Parameter	Description
Date	<code>date</code>	The date to convert to milliseconds since epoch time.

Returns

Integer - The number of milliseconds since January 1, 1970, 00:00:00 UTC (GMT) for the given date.

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

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.

containsAll

This function checks to see if all of the listed elements are present in the collection object. The function requires ...

containsAny

This function checks to see if any of the listed elements are present in the collection object. The function requires...

containsAll

This function is used by Ignition's Expression language.

Description

 NOTE

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

Type	Parameter	Description
Object	<code>collection</code>	A collection of values. Typically from the <code>{security-zone}</code> object or the <code>{idp-attribute:X}</code> object.
String	<code>element</code>	One or more comma-separated elements to look for in the collection.

Returns

Boolean - Returns true if the collection contains all of the listed elements, or false 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

NOTE

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

Type	Parameter	Description
Object	<code>collection</code>	A collection of values. Typically from the <code>{security-zone}</code> object or the <code>{idp-attribute:X}</code> object.
String	<code>element</code>	One or more comma-separated elements to look for in the collection.

Returns

Boolean - Returns true if the collection contains any of the listed elements, or false 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')
```

Logic

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

binEnc

Takes a list of booleans and treats them like the bits in a binary number.

binEnum

Takes a list of booleans, and returns the index (starting at 1) of the first parameter that evaluates to true.

case

This function acts like the switch statement in C-like programming languages.

coalesce

This function, which accepts any number of arguments, evaluates each in order, and returns the first non-null arg...

getBit

This function returns the bit value (an integer, 0 or 1) in the number at position position, according to its binary re...

hasChanged

This function returns true if the given value has changed since the last time the Expression Item was run.

if

This function evaluates the expression condition, and returns the value of trueReturn or falseReturn depending o...

indexOf

Searches for the first occurrence of the substring inside of string. Returns the index of where substring was foun...

isBad

Tests to see whether or not the given value's quality is bad.

isBadOrError

Tests to see whether or not the given value's quality is either bad or error.

isError

Tests to see whether or not the given value's quality is either error.

isGood

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

isNull

Tests to see whether or not the argument value is null

isUncertain

Tests to see whether or not the given value's quality is Uncertain.

lastIndexOf

Searches for the last occurrence of the substring inside of string.

len

Returns the length of the argument, which may be a string or a dataset.

lookup

This looks for lookupValue in the lookupColumn of dataset.

switch

This function acts like the switch statement in C-like programming languages.

try

This expression is used to swallow errors caused by other expressions.

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.

This function enforces up to 64 max arguments. Note that 32 arguments or fewer will return an Integer and 33 arguments to 64 arguments will return a Long. More than 64 arguments will return an error.

Syntax

```
binEnc(value, [value, ...])
```

Parameters

Type	Parameter	Description
Boolean	<code>value</code>	A value that represents a bit. Each argument can be either 0 or 1. Up to 64 arguments are allowed.

Returns

- **Integer** - The integer representation of the binary value when 32 or fewer arguments are provided.
- **Long** - The integer representation of the binary value when 33 to 64 arguments are provided.

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

Type	Parameter	Description
Boolean	<code>value</code>	A value that represents a bit. Each argument can be either <code>0</code> (false) or a non-zero value (true).

Returns

- Integer** - The index (starting at 1) of the first value that evaluates to true. If no value evaluates to true, the result is 0.

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 case1..N, then returnDefault is returned.

Note that case() is similar in functionality to the [switch\(\)](#) expression function. The difference between the two is the order in which the parameters are passed.

Syntax

```
case(value, case, return[, case, return...], returnDefault)
```

Parameters

Type	Parameter	Description
Object	<code>value</code>	The value to compare against each case.
Object	<code>case</code>	A case to match the <code>value</code> to.
Object	<code>return</code>	The value to return if the corresponding <code>case</code> is matched.
Object	<code>returnDefault</code>	The default return value if no <code>case</code> arguments are matched.

Returns

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 to inspect  
    1, 44,       // case 1, return 1  
    24, 45,      // case 2, return 2  
    15, 46,      // case 3, return 3  
    -1          // default return  
)
```

Code Snippet

//The following would *return* "Running".

```
case(  
    1,          // value to inspect  
    0, "Off",    // case 1, return 1  
    1, "Running", // case 2, return 2  
    2, "Fault",  // case 3, return 3  
    "BAD STATE!" // default return  
)
```

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. A typical use case involves 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

Type	Parameter	Description
Object	<code>value</code>	Any number of values to evaluate in order.

Returns

Object - The first non-null argument. If all arguments are null, the result is null.

Examples

Code Snippet

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

Code Snippet

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

Code Snippet

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

getBit

This function is used by Ignition's Expression language.

Description

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

Syntax

```
getBit(number, position)
```

Parameters

Type	Parameter	Description
Integer	<code>number</code>	The number whose binary representation will be checked.
Integer	<code>position</code>	The bit position to evaluate, where 0 is the least significant bit.

Returns

Integer - Returns 0 or 1, depending on the bit value at the specified position.

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

NOTE

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 `includeQuality` to true means a quality change will make this function return true.

Syntax

```
hasChanged(value, [includeQuality], [pollRate])
```

Parameters

Type	Parameter	Description
Object	<code>value</code>	The value to check for changes.
Boolean	<code>includeQuality</code>	A flag that indicates if a quality change will also trigger this function. [optional]
Integer	<code>pollRate</code>	The poll rate in milliseconds. Only applicable on Expression Tags when the Execution Mode is set to Event Driven. [optional]

Returns

Boolean - Returns true if the value has changed since the last evaluation, or false otherwise.

Examples

Code Snippet

//Would return true if the referenced Tag has changed in value or quality since the last group execution.

```
hasChanged({[default]Station 1/Status},True)
```

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

Type	Parameter	Description
Boolean	<code>condition</code>	A boolean expression that determines which return value is used.
Object	<code>trueReturn</code>	The value returned if the condition evaluates to true.
Object	<code>falseReturn</code>	The value returned if the condition evaluates to false.

Returns

Object - Returns the trueReturn value if the condition is true, or the falseReturn value if the condition 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."))
```

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

Type	Parameter	Description
String or List	<code>string</code>	The string to search through. This parameter also accepts tuples and lists.
String	<code>substring</code>	The string to search for.

Returns

Integer - The index where the substring was first found in the string. Returns -1 if the substring is not found.

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
```

isBad

This function is used by Ignition's Expression language.

Description

Tests to see whether or not the given value's **quality** is bad.

Syntax

```
isBad(value)
```

Parameters

Type	Parameter	Description
Object	<code>value</code>	A value to check if its quality is bad.

Returns

Boolean - Returns true if the value's quality is bad, or false if it is not.

Examples

Code Snippet

```
isBad({path/to/myTag}) //Returns True if the value's quality is bad, False otherwise.
```

isBadOrError

This function is used by Ignition's Expression language.

Description

Tests to see whether or not the given value's **quality** is either bad or error.

Syntax

```
isBadOrError(value)
```

Parameters

Type	Parameter	Description
Object	<code>value</code>	A value to check if its quality is either bad or error.

Returns

Boolean - Returns true if the value's quality is either bad or error, or false if it is not.

Examples

Code Snippet

```
isBadOrError({path/to/myTag}) //Returns 1 if the value's quality is bad or error,  
0 otherwise.
```

isError

This function is used by Ignition's Expression language.

Description

Tests to see whether or not the given value's **quality** is error.

Syntax

```
isError(value)
```

Parameters

Type	Parameter	Description
Object	<code>value</code>	A value to check if its quality is error.

Returns

Boolean - Returns `true` if the value's quality is error, or `false` if it is not.

Examples

Code Snippet

```
isError({path/to/myTag}) //Returns True if the value's quality is error, False otherwise.
```

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

Type	Parameter	Description
Object	<code>value</code>	A value to check if its quality is good.

Returns

Boolean - Returns true if the value's quality is good, or false if it is not.

Examples

Code Snippet

```
isGood({path/to/myTag}) //Returns True if the value's quality is good, False otherwise.
```

isNull

This function is used by Ignition's Expression language.

Description

Tests to see whether the argument value is null or not. You can also check for null by simply comparing the value to the null keyword, where isNull(x) is the same as x = null.

Syntax

```
isNull(value)
```

Parameters

Type	Parameter	Description
Object	<code>value</code>	A value to check if it is null.

Returns

Boolean - Returns true if the value is null, or 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})
```

isUncertain

This function is used by Ignition's Expression language.

Description

Tests to see whether or not the given value's **quality** is Uncertain

Syntax

```
isUncertain(value)
```

Parameters

Type	Parameter	Description
Object	<code>value</code>	A value to check if its quality is uncertain.

Returns

Boolean - Returns true if the value's quality is uncertain, or false if it is not.

Examples

Code Snippet

```
isUncertain({path/to/myTag}) //Returns True if the value's quality is uncertain,  
False otherwise.
```

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

Type	Parameter	Description
String or List	<code>string</code>	The string to search through. This parameter also accepts tuples and lists.
String	<code>substring</code>	The string to search for.

Returns

Integer - The index where the substring was last found in the string. Returns -1 if the substring is not found.

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.
```

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.

The len function can be used with lists, sets, and tuples.

Syntax

```
len(value)
```

Parameters

Type	Parameter	Description
Object	<code>value</code>	The object whose length is being evaluated.

Returns

Integer - The length of the provided object (number of characters, rows, or elements depending on the type).

Examples

Code Snippet

```
len("Hello World") //Returns 11.
```

Code Snippet

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

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

Type	Parameter	Description
Dataset	<code>dataset</code>	A dataset to search through.
Object	<code>lookupValue</code>	The value to look for in the dataset.
Object	<code>noMatchValue</code>	The value to return if no match is found.
Object	<code>lookupColumn</code>	The column to search for the lookup value. Can be the column index or name. Defaults to column 0. [optional]
Object	<code>resultColumn</code>	The column to retrieve the result value from. Can be the column index or name. Defaults to column 1. [optional]

Returns

Object - The value in the result column of the same row where the lookupValue was found, or the noMatchValue if no match was found. The data type of the result will always match the type of the noMatchValue.

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)
```

Parameters

Type	Parameter	Description
Object	value	The value to check against the case values.
Object	case	A value to check against. Can include any number of case values.
Object	return	A value to return for the matching case. Must match the number of case values.
Object	returnDefault	The default value to return if no case is matched.

Returns

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 evaluated, and if it executes successfully, its value will be used. However, if an error occurs while evaluating the expression, the value of the failover parameter will be used. When the failover is returned, its data quality will be applied.

Syntax

```
try(expression, failover)
```

Parameters

Type	Parameter	Description
Object	<code>expression</code>	An expression that can evaluate to any supported data type.
Object	<code>failover</code>	The value to return if an error occurs while evaluating expression .

Returns

Object - The result of the evaluated expression, or the failover value if an error occurs.

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

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

abs

Returns the absolute value of number.

acos

Returns the arc cosine of number

asin

Returns the arc sine of number

atan

Returns the arc tangent of number

ceil

Returns the smallest floating point value that is greater than or equal to the argument and is equal to a mathematical integer.

cos

Returns the trigonometric cosine of number

exp

Returns Euler's number e raised to the power of the argument number, or enumber

floor

Returns the largest floating point value that is less than or equal to the argument and is equal to a mathematical integer.

log

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

log10

Returns the logarithm (base 10) of a number.

pow

Returns a number raised to a power.

 round

Rounds a floating point number.

 sin

Returns the trigonometric sine of a number

 sqrt

Returns the square root of the argument number

 tan

Returns the trigonometric tangent of a number

 todegrees

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

 toradians

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

abs

This function is used by Ignition's Expression language.

Description

Returns the absolute value of number.

Syntax

```
abs(number)
```

Parameters

Type	Parameter	Description
Integer/Float	<pre>number</pre>	The number to get the absolute value of.

Results

Integer/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

Type	Parameter	Description
Float	<code>number</code>	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 a 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

Type	Parameter	Description
Float	<code>number</code>	The number to get the arc sine of. Must be a value between -1 and 1.

Results

Float - The arc sine of the value 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 a number, which must be a number. The results will be an angle expressed in radians in the range of $-\pi/2$ through $\pi/2$

Syntax

```
atan(number)
```

Parameters

Type	Parameter	Description
Float	<code>number</code>	The number to get the arc tangent of.

Results

Float - The arc tangent of the value 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

Type	Parameter	Description
Float	<code>number</code>	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

Type	Parameter	Description
Integer/Float	<code>number</code>	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

Type	Parameter	Description
Integer/Float	<code>number</code>	The exponent value to raise e to the power of.

Results

Integer/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

Type	Parameter	Description
Float	<code>number</code>	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

Type	Parameter	Description
Integer/Float	<code>number</code>	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

Type	Parameter	Description
Integer/Float	<code>number</code>	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

Type	Parameter	Description
Integer/Float	<code>number</code>	The number to raise to the provided power.
Integer/Float	<code>power</code>	The power value to raise the number value to.

Results

Integer/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

Type	Parameter	Description
Float	<code>number</code>	The number to round.
Integer	<code>decimals</code>	The number of decimal places to round to. Defaults to 0. [optional]

Results

Integer/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

Type	Parameter	Description
Integer/Float	<code>number</code>	The number to get the sine of.

Results

Integer/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

Type	Parameter	Description
Integer/Float	<code>number</code>	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 a number, which is interpreted as an angle expressed in radians. The results will be a floating point value.

Syntax

```
tan(number)
```

Parameters

Type	Parameter	Description
Integer/Float	<pre>number</pre>	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

Type	Parameter	Description
Integer/Float	<code>number</code>	The number of radians.

Results

Integer/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

Type	Parameter	Description
Integer/Float	<code>number</code>	The number of degrees.

Results

Integer/Float - The radian equivalent of the degrees provided.

Examples

Code Snippet

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

MongoDB - Expression Binding

The following functions allow you to interact with MongoDB databases in expression bindings.

maxKey

Returns `org.bson.types.MaxKey`, which is used in filters for the MongoDB Perspective binding expression.

minKey

Returns `org.bson.types.MinKey`, used in filters for the MongoDB Perspective binding expression.

toObjectId

Converts String to `org.bson.types.ObjectId`.

maxKey

This function is used by Ignition's Expression language.

Description

Returns `org.bson.types.MaxKey`, which is used in filters for the MongoDB Perspective binding expression.

Syntax

```
maxKey()
```

Parameters

None

Results

MaxKey - A special data type that will match with document fields of the same MaxKey type.

Example

Code Snippet

```
// In this example, we are querying for weather data documents using the field  
"elevation"  
// In a Perspective MongoDB Binding, the expression below is applied to "value" of  
// filter { elevation: "value" }  
  
maxKey() // binding will return entire document(s) where maxKey is found
```

minKey

This function is used by Ignition's Expression language.

Description

Returns `org.bson.types.MinKey`, used in filters for the MongoDB Perspective binding expression.

Syntax

```
minKey()
```

Parameters

None

Results

MinKey - A special data type that will match with document fields of the same MinKey type.

Example

Code Snippet

```
// In this example, we are querying for weather data documents using the field  
"elevation"  
// In a Perspective MongoDB Binding, the expression below is applied to "value" of  
// filter { _id: "value" }  
  
minKey() // binding will return entire document(s) where minKey is found
```

toObjectId

This function is used by Ignition's Expression language.

Description

Converts String to org.bson.types.ObjectId.

Syntax

```
toObjectId(stringId)
```

Type	Parameter	Description
String	<code>stringId</code>	A unique, 24 character string identifier that matches with an id of an existing document of a collection.

Results

ObjectId - A unique, 12 byte identifier that matches with an id of an existing document of a collection in both value and data type.

Example

Code Snippet

```
// In a Perspective MongoDB Binding, the expression below is applied to "value" of
// filter { _id: "value" }

toObjectId("5553a998e4b02cf7151190b8") // binding will return entire document of
the matching _id
```

String

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

char

Takes a Unicode character code (as an integer), and returns the Unicode character as a string.

concat

Concatenates all of the strings passed in as arguments together.

escapeSQL

Returns the given string with special SQL characters escaped.

escapeXML

Returns the given string after being escaped to be valid for inclusion in XML.

fromBinary

Returns an integer value of the binary formatted string argument.

fromHex

Returns an integer value of the hex formatted string argument.

fromOctal

Returns an integer value of the octal formatted string argument.

left

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

lower

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

numberFormat

Returns a string version of the number argument, formatted as specified by the pattern string.

ordinal

Takes a Unicode character (as a string), and returns the corresponding character code, as an integer.

repeat

Repeats the given string some number of times.

replace

Finds all occurrences of a substring inside of a source string, and replaces them with the replacement string.

right

Returns count number of characters starting from the right side of string, where count and string are the argume...

split

Takes the starting string and splits it into substrings

stringFormat

This expression returns a formatted string using the specified format string and arguments. Mainly, this expressi...

substring

Will return the portion of the string from the startIndex to the endIndex, or end of the string if endIndex is not spe...

toBinary

Returns a binary formatted string representing the unsigned integer argument.

toHex

Returns a hex formatted string representing the unsigned integer argument.

toOctal

Returns an octal formatted string representing the unsigned integer argument.

trim

Takes the argument string and trims of any leading and/or trailing whitespace

upper

Takes a string and returns an uppercase version of it.

urlencode

Enables users to create an HTTP binding's URL on the fly.

char

This function is used by Ignition's Expression language.

Description

Takes a Unicode character code (as an integer), and returns the Unicode character as a string.

NOTE

This function can work for ASCII conversions as well, since Unicode and ASCII character codes match for all ASCII characters.

Syntax

```
char(code)
```

Parameters

Type	Parameter	Description
Integer	<code>code</code>	The character code for a Unicode character.

Results

String - The corresponding Unicode character, as a one-character string.

Examples

Code Snippet

```
char(88) //Returns "X".
```

Code Snippet

```
char(425) //Returns "?".
```

Code Snippet

```
char(-1) //Returns Error_Expression Eval.
```

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

Type	Parameter	Description
String	<code>string</code>	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 function just replaces single quotes with two single quotes, and backslashes with two backslashes. See [system.db.runPrepUpdate](#) for a safer way to sanitize user input.

Syntax

```
escapeSQL(string)
```

Parameters

Type	Parameter	Description
String	<code>string</code>	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.TextField.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

Type	Parameter	Description
String	<code>string</code>	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

Type	Parameter	Description
String	<code>string</code>	A string representation of a binary.

Results

Integer - 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

Type	Parameter	Description
String	<code>string</code>	A string representation of a hex value.

Results

Integer - 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

Type	Parameter	Description
String	<code>string</code>	A string representation of an octal.

Results

Integer - The integer of the octal value.

Examples

Code Snippet

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

Code Snippet

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

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

Type	Parameter	Description
String	<code>string</code>	The starting string.
Integer	<code>charCount</code>	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"
```

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

Type	Parameter	Description
String	<code>string</code>	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.

Syntax

```
numberFormat(number, pattern)
```

Parameters

Type	Parameter	Description
Float	<code>number</code>	The number to format.
String	<code>pattern</code>	The format pattern. Refer to the Number Format Reference page for more information about pattern strings.

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%"
```

ordinal

This function is used by Ignition's Expression language.

Description

Takes a Unicode character (as a string), and returns the corresponding character code, as an integer.

NOTE

This function can work for ASCII conversions as well, since Unicode and ASCII character codes match for all ASCII characters.

Syntax

```
ordinal(string)
```

Parameters

Type	Parameter	Description
String	<code>string</code>	A string containing a single Unicode character. If the string does not contain exactly one character, an ExpressionEval error is thrown.

Results

Integer - The character code associated with the Unicode character.

Examples

Code Snippet

```
ordinal("a") //Returns 97.
```

Code Snippet

```
ordinal("AA") //Returns Error_ExpressionEval.
```

Code Snippet

```
ordinal("ñ") //Returns 241
```

repeat

This function is used by Ignition's Expression language.

Description

Repeats the given string some number of times.

Syntax

```
repeat(string, count)
```

Parameters

Type	Parameter	Description
String	<code>string</code>	The string to repeat.
Integer	<code>count</code>	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

Type	Parameter	Description
String	<code>string</code>	The starting string.
String	<code>substring</code>	The string to search for.
String	<code>replacementString</code>	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

Type	Parameter	Description
String	<code>string</code>	The starting string.
String	<code>charCount</code>	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 returned 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

Type	Parameter	Description
String	<code>string</code>	The starting string.
String	<code>regex</code>	The string to split on.
Integer	<code>limit</code>	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"]
```

Code Snippet

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

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.

Refer to [Data Type Formatting Reference](#) for formatting elements.

Syntax

```
stringFormat(format, [args, ...])
```

Parameters

Type	Parameter	Description
String	<code>format</code>	A string that contains formatting elements in it (%s, %d, %i).
String	<code>args</code>	The arguments to use in the format. Must match the number of formatting elements in the string. [optional]

Results

String - The new formatted string.

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.
```

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

Type	Parameter	Description
String	<code>string</code>	The starting string.
Integer	<code>startIndex</code>	The index to start the substring at.
Integer	<code>endIndex</code>	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" - note 8 is the index of "r", but  
the end index is exclusive.
```

toBinary

This function is used by Ignition's Expression language.

Description

Returns a 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

Type	Parameter	Description
Integer	<code>number</code>	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 (number)

```
toHex(number)
```

Parameters

Type	Parameter	Description
Integer	<code>number</code>	The number to convert to hex.

Results

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

```
toHex(color)
```

Parameters

Type	Parameter	Description
Color	<code>color</code>	An RGBA set 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"
```

Code Snippet

```
toHex(color(192,26,68,241)) //Returns "C01A44F1"
```

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

Type	Parameter	Description
Integer	<code>number</code>	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

Type	Parameter	Description
String	<code>string</code>	The starting string.

Results

String - The starting string with all whitespace removed.

Examples

Code Snippet

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

Code Snippet

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

upper

This function is used by Ignition's Expression language.

Description

Takes a string and returns an uppercase version of it.

Syntax

```
upper(string)
```

Parameters

Type	Parameter	Description
String	<code>string</code>	The string to make uppercase.

Results

String - The starting string with all characters uppercase.

Examples

Code Snippet

```
upper("Hello World") //returns "HELLO WORLD"
```

urlEncode

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

Type	Parameter	Description
String	<code>url</code>	The URL to encode.
Boolean	<code>usePercentEscape</code>	False or blank indicates to use query parameter style escaping. True indicates to use URI fragment style escaping. [optional]

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

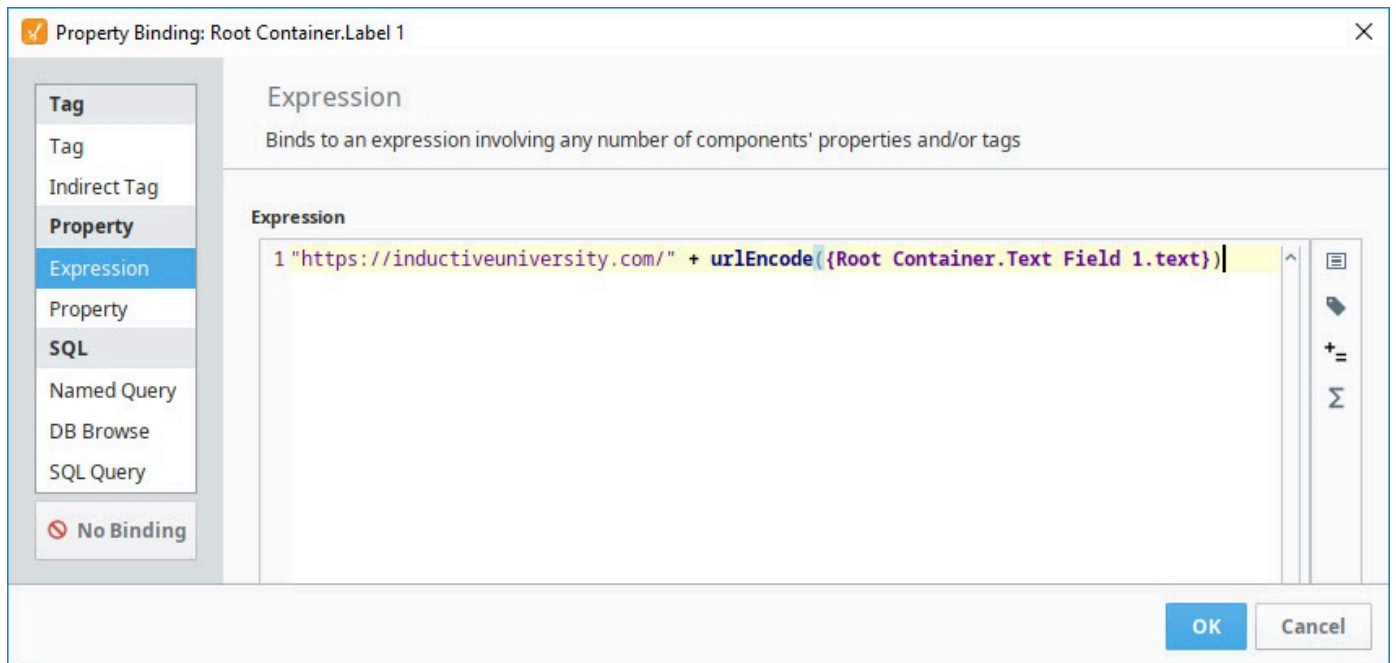
```
urlEncode("Hello World", True) //yields "Hello%20World".
```

Vision Example

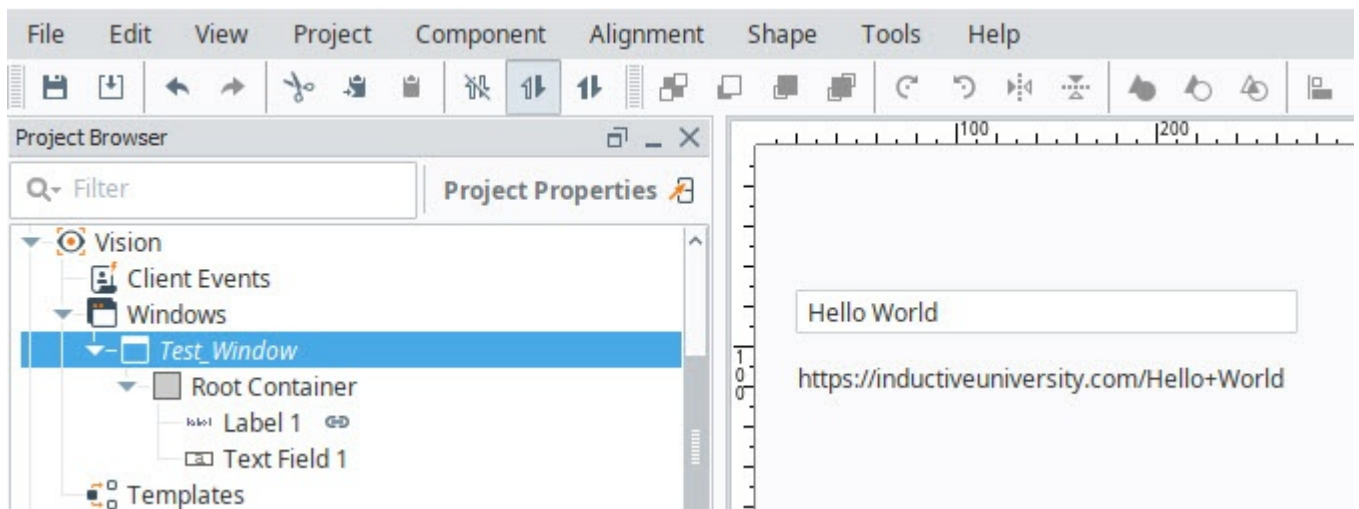
1. Drag a Text Field component and a Label component onto a window.
2. In the Vision Property Editor, enter a parameter for the URL you'll use. For this example, we used 'Hello World.'
3. Select the Label component, then click on the **Binding**  icon for the label's Text property.
4. On the Property 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.



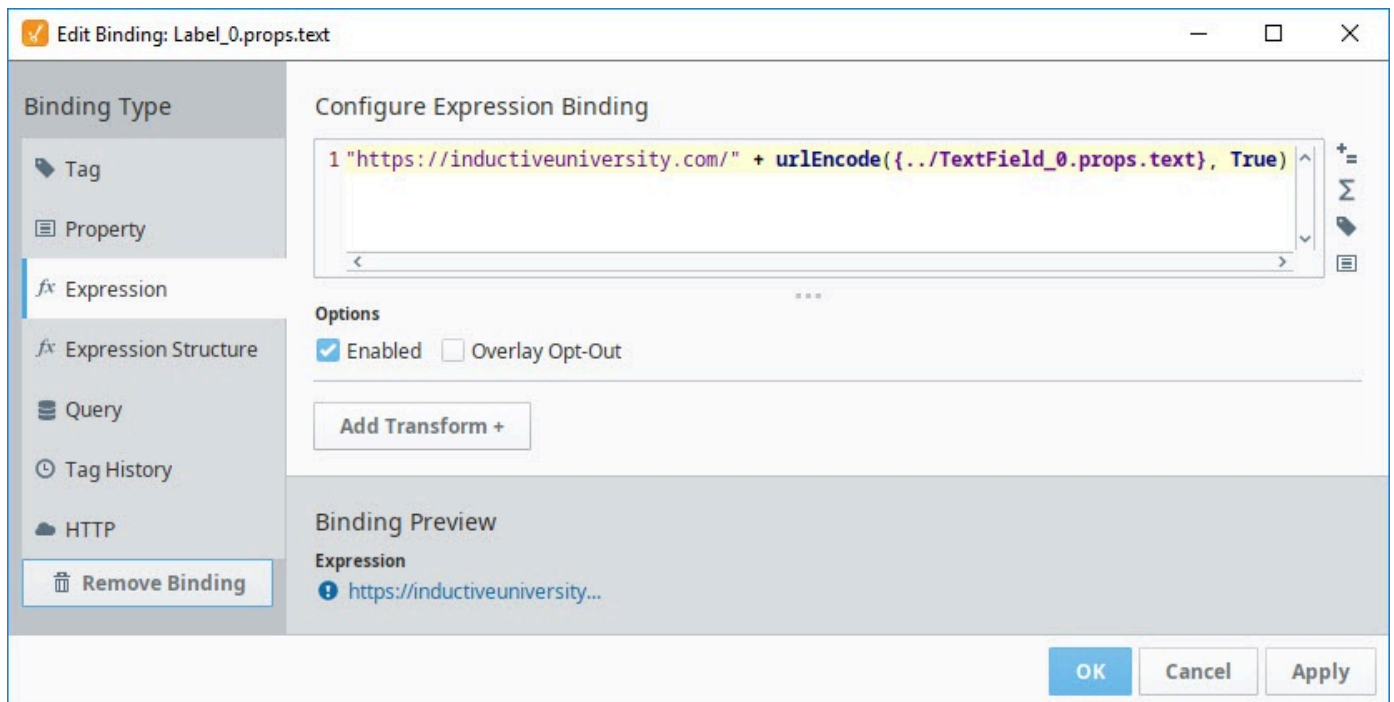
Perspective Example

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**.

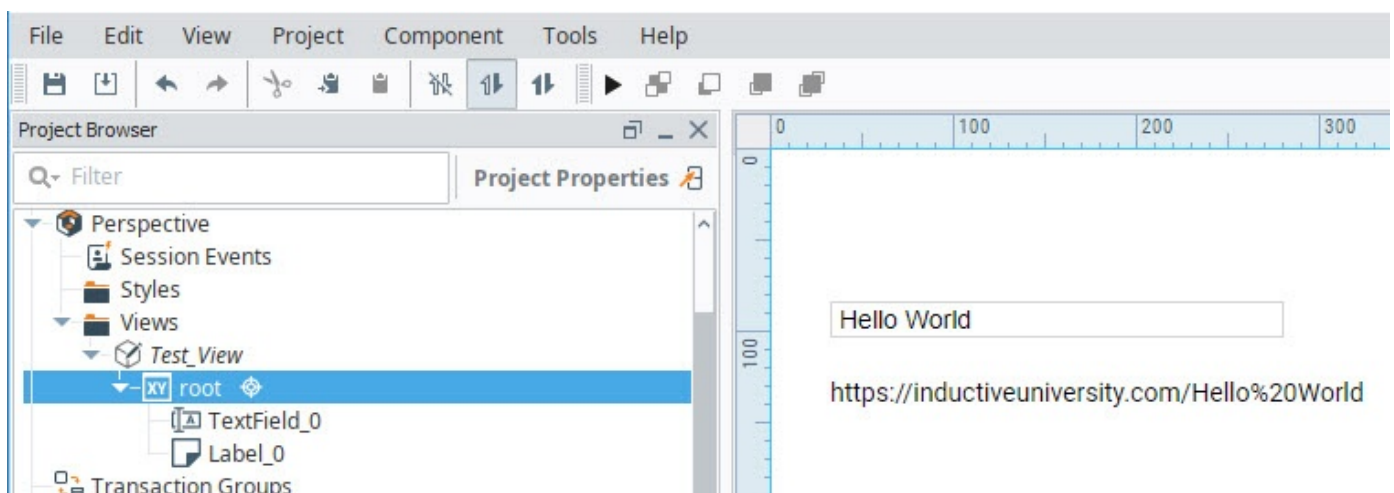
5. Enter the expression in the Configure Expression Binding section. In the example, we entered the URL plus the expression function, "urlEncode ()". We entered the oath to the Text Field's text property as the string and the parameter "True" to use URI fragment style escaping.

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

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

 **translate**

Returns a translated string, based on the current locale.

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, [languageString])
```

Parameters

Type	Parameter	Description
String	stringKey	The starting string to translate.
String	languageString	The language or locale to translate against, such as "es" or "es-ES". Accepts both the BCP-47 language tag (recommended) and the legacy underscore formats. Make sure to use locale codes for either format, and not language names, such as "Spanish". If omitted, the current locale is used. [optional]

Results

String - The starting string translated based on the specified locale, or the current locale if none was given. If the translation does not exist, the original `stringKey` is returned.

Type Casting

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

toBoolean

Tries to convert value to a boolean

toBorder

This function is used specifically when binding a Border property on a component. Typically, this is used with a C...

toColor

This function tries to convert value to a color.

toDataSet

Tries to coerce value into a dataset.

toDate

Tries to coerce value into a Date.

toDouble

Tries to coerce value into a double (64-bit floating point value).

toFloat

Tries to coerce value into a float (32-bit floating point vaule).

toFont

Coerces a string into a font.

toInt

Tries to coerce value into an integer (32-bit integer).

toInteger

Identical to the toInt expression function.

toLong

Tries to coerce value into a long (64-bit integer).

 toStr

Identical to the toString expression function.

 toString

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

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

Type	Parameter	Description
object	value	The value to type cast.
object	failover	The failover value if type casting fails. [optional]

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; fontColor; font			1 = Left 2 = Center 3 = Right 4 = Leading 5 = Trailing

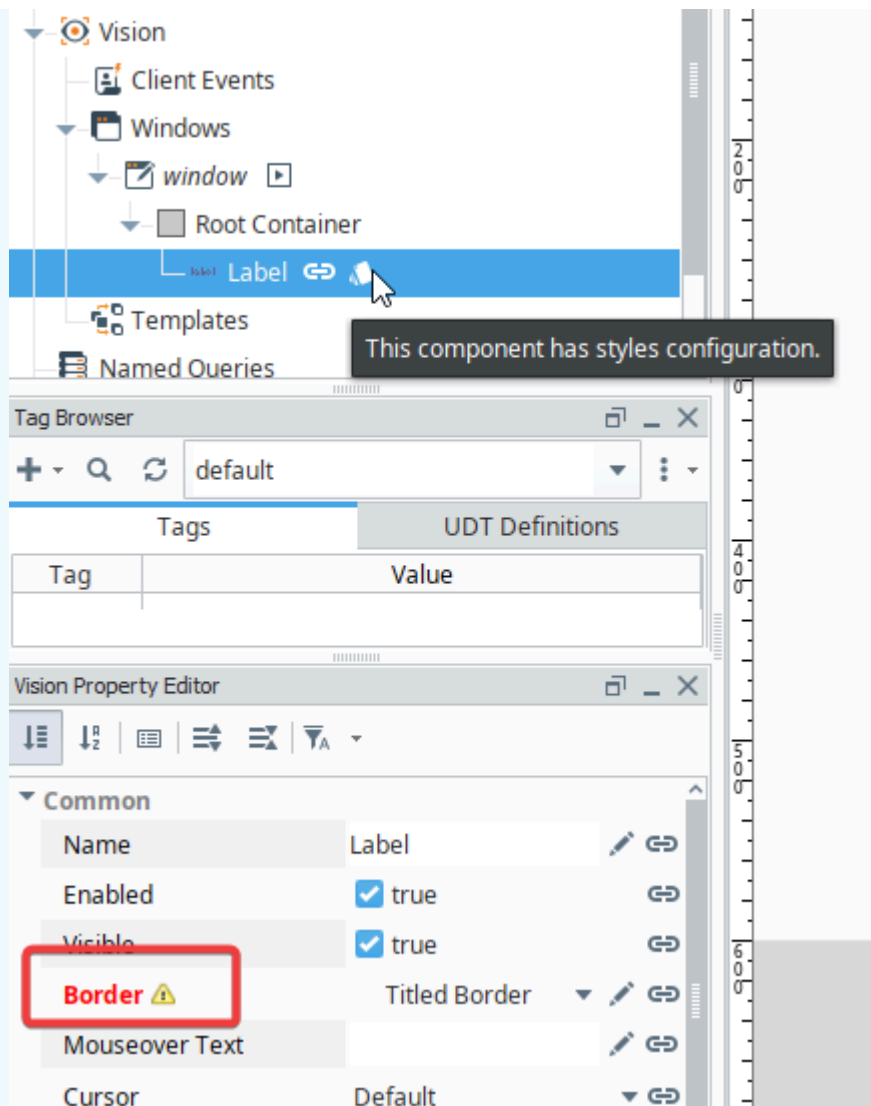
Border Type	Options	Type	Style	Font Justification
matte	color; topWidth, leftWidth; bottomWidth; rightWidth			
paneltitled	title; style; mainColor; bgColor, shadowSize, fontJustification; fontPosition; fontColor;font		1=Gradient / West-to-East 2=Gradient / North-to- South 3=Gradient / East-to-West 4=Solid	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, for example:

```
toBorder("paneltitled; title; style; mainColor; bgColor; shadowSize;  
fontJustification; fontPosition; fontColor;font")
```

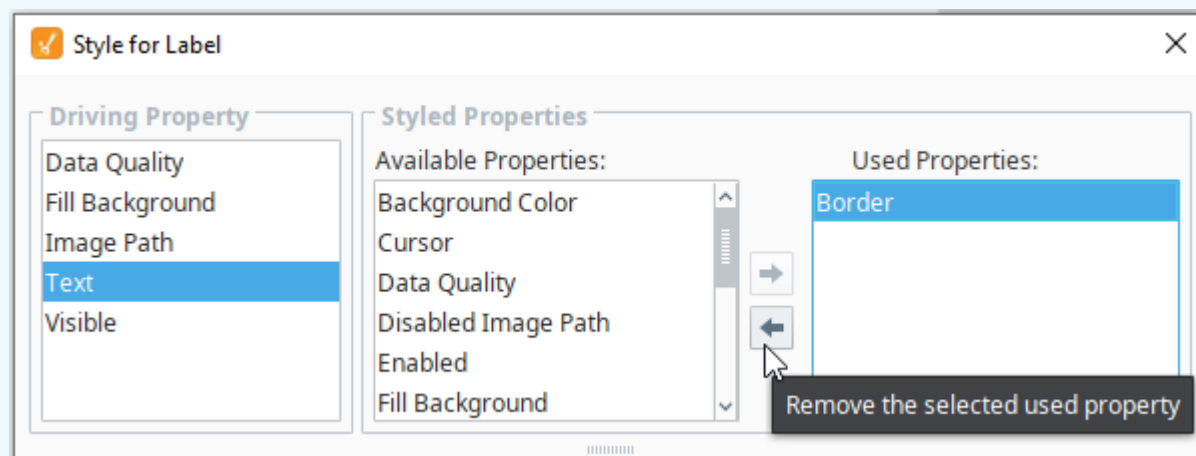
 BE MINDFUL OF STYLE CONFIGURATIONS

The use of toBorder can conflict with other configurations, typically [Styles](#). If you notice red text on the border property after applying toBorder in an expression binding, it's likely due to a Style being applied to the border property.



In this case you would need to remove the style configuration:

1. From the Designer, right-click the component and select **Style Customizer**
2. Under the **Used Properties** list, select **Border**
3. Click left arrow to remove Border from the list.
4. Click OK to close the Style Customizer.



Syntax

```
toBorder(value, [failover])
```

Parameters

Type	Parameter	Description
String	<code>value</code>	The value to type cast.
Object	<code>failover</code>	The failover value if type casting fails. [optional]

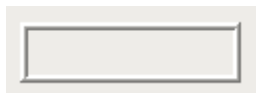
Results

Border - The value type cast as a border object.

Examples

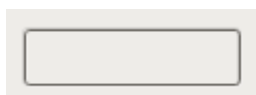
Code Snippet

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



Code Snippet

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



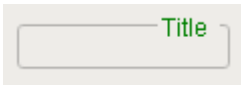
Code Snippet

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



Code Snippet

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



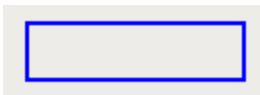
Code Snippet

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



Code Snippet

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



Code Snippet

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



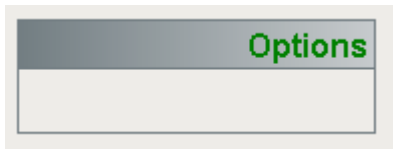
Code Snippet

```
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...
```



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, for example: "#00FF00".

Syntax

toColor(value, [failover])

Parameters

Type	Parameter	Description
String	value	The color value as a string.
Object	failover	The failover value if type casting fails. [optional]

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")
```

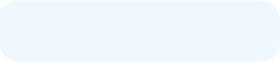
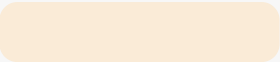
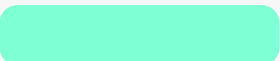
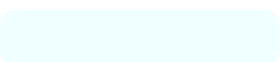
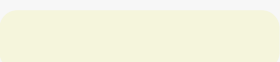
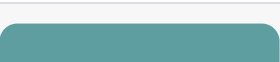
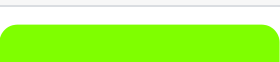
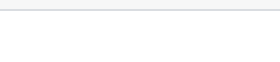
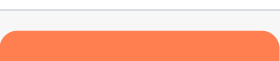
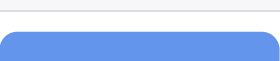
Code Snippet

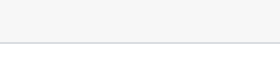
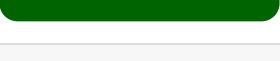
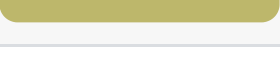
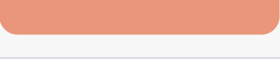
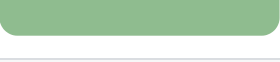
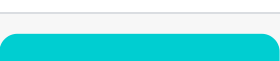
```
//You can use the failover parameter to ensure that this expression returns  
something even if the input string may be bad:  
toColor({UserOptions/CustomColor}, "black")
```

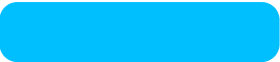
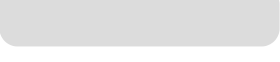
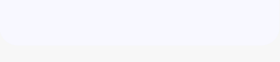
Color Options

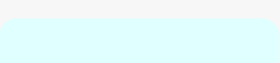
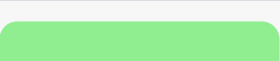
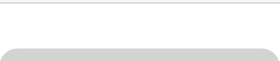
NOTE

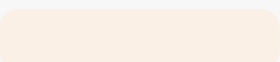
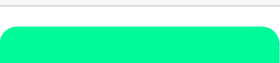
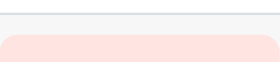
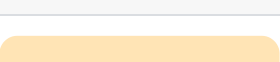
Both 'Grey' and 'Gray' are accepted as valid colors, as well as the iterations of that color such as 'DarkGrey' and 'DarkGray'.

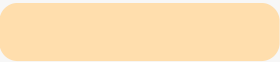
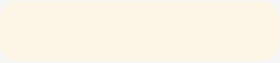
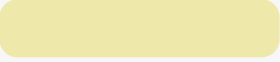
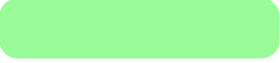
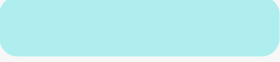
Color	Hex Color String	Example
AliceBlue	#F0F8FF	
AntiqueWhite	#FAEBD7	
Aqua	#00FFFF	
Aquamarine	#7FFFD4	
Azure	#F0FFFF	
Beige	#F5F5DC	
Bisque	#FFE4C4	
Black	#000000	
BlanchedAlmond	#FFEBCD	
Blue	#0000FF	
BlueViolet	#8A2BE2	
Brown	#A52A2A	
BurlyWood	#DEB887	
CadetBlue	#5F9EA0	
Chartreuse	#7FFF00	
Chocolate	#D2691E	
Clear	see Transparent	
Coral	#FF7F50	
CornflowerBlue	#6495ED	
Cornsilk	#FFF8DC	

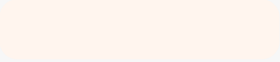
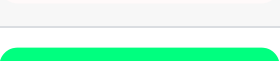
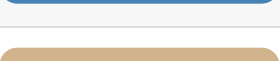
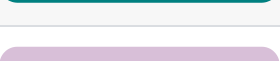
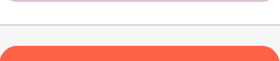
Color	Hex Color String	Example
Crimson	#DC143C	
Cyan	#00FFFF	
DarkBlue	#00008B	
DarkCyan	#008B8B	
DarkGoldenRod	#B8860B	
DarkGray DarkGrey	#A9A9A9	
DarkGreen	#006400	
DarkKhaki	#BDB76B	
DarkMagenta	#8B008B	
DarkOliveGreen	#556B2F	
Darkorange	#FF8C00	
DarkOrchid	#9932CC	
DarkRed	#8B0000	
DarkSalmon	#E9967A	
DarkSeaGreen	#8FBC8F	
DarkSlateBlue	#483D8B	
DarkSlateGray DarkSlateGrey	#2F4F4F	
DarkTurquoise	#00CED1	
DarkViolet	#9400D3	

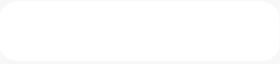
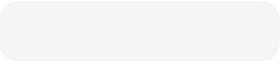
Color	Hex Color String	Example
DeepPink	#FF1493	
DeepSkyBlue	#00BFFF	
DimGray DimGrey	#696969	
DodgerBlue	#1E90FF	
Feldspar	#D19275	
FireBrick	#B22222	
FloralWhite	#FFFAF0	
ForestGreen	#228B22	
Fuchsia	#FF00FF	
Gainsboro	#DCDCDC	
GhostWhite	#F8F8FF	
Gold	#FFD700	
GoldenRod	#DAA520	
Gray Grey	#808080	
Green	#008000	
GreenYellow	#ADFF2F	
HoneyDew	#F0FFF0	
HotPink	#FF69B4	
IndianRed	#CD5C5C	

Color	Hex Color String	Example
Indigo	#4B0082	
Ivory	#FFFFFF0	
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	

Color	Hex Color String	Example
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	

Color	Hex Color String	Example
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	

Color	Hex Color String	Example
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	

Color	Hex Color String	Example
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

Type	Parameter	Description
Object	<code>value</code>	The value to type cast, typically a DataSet or PyDataSet.
Object	<code>failover</code>	The failover value if type casting fails. [optional]

Results

Dataset - The value type cast as a dataset.

Examples

Code Snippet

```
toDataSet(runScript("app.funcs.runSomeFunction()")) //coerces the value returned  
by 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

Type	Parameter	Description
Object	value	The value to type cast into a date.
Object	failover	The failover value if type casting fails. [optional]

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

Type	Parameter	Description
Object	<code>value</code>	The value to type cast.
Object	<code>failover</code>	The failover value if type casting fails. [optional]

Results

Double - The value type cast as a double.

Examples

Code Snippet

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

Code Snippet

```
toDouble({Root Container.TextField.text}, 0.0) //returns the value in the text box  
as a double, or 0.0 if the value doesn't represent a 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

Type	Parameter	Description
Object	<code>value</code>	The value to type cast.
Object	<code>failover</code>	The failover value if type casting fails. [optional]

Results

Float - The value type cast as a float.

Examples

Code Snippet

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

Code Snippet

```
toFloat({Root Container.TextField.text}, 0.0) //returns the value in the text box  
as a float, or 0.0 if the value doesn't represent a 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 represents the font's point size.

Syntax

```
toFont(value, [failover])
```

Parameters

Type	Parameter	Description
String	value	The value to type cast to a font.
Object	failover	The failover value if type casting fails. [optional]

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

Type	Parameter	Description
Object	<code>value</code>	The value to type cast.
Object	<code>failover</code>	The failover value if type casting fails. [optional]

Results

Integer - 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.TextField.text}, -1) //returns the value in the text box as  
an int, or -1 if the value doesn't represent a number.
```

toInteger

This function is used by Ignition's Expression language.

Description

Identical to the [toInt](#) expression function.

Syntax

```
toInteger(value, [failover])
```

Parameters

Type	Parameter	Description
Object	<code>value</code>	The value to type cast.
Object	<code>failover</code>	The failover value if type casting fails. [optional]

Results

Integer - The value type cast as an int.

Examples

Code Snippet

```
toInteger("38") //returns 38
```

Code Snippet

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

Code Snippet

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

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

Type	Parameter	Description
Object	<code>value</code>	The value to type cast.
Object	<code>failover</code>	The failover value if type casting fails. [optional]

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.TextField.text}, -1) //returns the value in the text box as  
an long, or -1 if the value doesn't represent a number.
```

toStr

This function is used by Ignition's Expression language.

Description

Identical to the [toString](#) expression function.

Syntax

```
toStr(value, [failover])
```

Parameters

Type	Parameter	Description
Object	<code>value</code>	The value to type cast, typically a DataSet or PyDataSet.
Object	<code>failover</code>	The failover value if type casting fails. [optional]

Results

String - The value type cast as a string.

Examples

Code Snippet

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

Code Snippet

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

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

Type	Parameter	Description
Object	<code>value</code>	The value to type cast.
Object	<code>failover</code>	The failover value if type casting fails. [optional]

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

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

hasRole

Returns true if the user has the given role. The username and usersource parameters are optional in the client sc...

isAuthorized

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

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

Type	Parameter	Description
String	<code>role</code>	The name of a role.
String	<code>username</code>	A username. Defaults to the current user. [optional]
String	<code>usersource</code>	The usersource of the username. Defaults to the usersource of the current user. [optional]

NOTE

When using Perspective, all parameters are required since it's in the Gateway scope. For certain Perspective applications, using [isAuthorized](#) is recommended instead.

Results

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

Examples

Code Snippet

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

Code Snippet

```
// This is an example using a username and userSource, as required for  
Perspective.  
hasRole("Administrator", "bob", "default")
```

Code Snippet

```
// This example shows how to handle when the Identity Provider and Users, Roles  
names are not identical.  
// If the Identity Provider name is "CompanyIdentity", but the Users, Roles the  
IdP points to is "CompanyUsers",  
// include the Users, Roles name as the userSource to avoid receiving an error.  
hasRole("Operator", {session.props.auth.user.userName}, "CompanyUsers")
```

isAuthorized

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. Scope is Perspective Sessions only.

Syntax

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

Parameters

Type	Parameter	Description
Boolean	<code>isAllOf</code>	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	<code>securityLevel</code>	One or more String paths, represented by <code>securityLevelN</code> , 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

Boolean - 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')
```