paperwork

CRIB SHEET

Expressions

'crib sheet'

A set of notes to help you remember important information, especially one taken secretly into an exam room

'expression'

In mathematics, a symbol or group of symbols that represent an amount.

Document Status	
Date Created	15 May 2025
Author	Paperwork team
Version	1.0

About

A **paperwork** template is designed to be dynamic, separating the data from the structure and the style. Databinding is the mechanism used to include that data content into the document layout as part of the generation process.

To enable this in **paperwork**, expressions are used to perform multiple calculations and evaluations on the current data 'model'.

Relevant Concepts

In order to get the best out of this document, you should be familiar with the following technologies:

- HTML (including XHTML and namespaces).
- Cascading styles sheets.
- JSON Object notation.

Table of Contents

Example Data

The following JSON data value is used throughout the example expressions as the `model`

```
"model" : {
    "number": 20.9,
    "int": 11,
    "boolean": true,
    "date": "12 June 2023 11:45:00",
    "array": [10,11,12],
    "color": "#330033",
    "bg": "silver",
    "padding": "20pt",
    "items": [
      {"name": "First Item", "index" : 1},
      {"name": "Second Item", "index": 3},
      {"name": "Third Item", "index": 2}
    ],
    "days": ["sun","mon","tues","wed","thur","fri","sat"],
    "nested" : {
      "p1" : "one",
      "p2" : "two"
    }
}
```

For Example	
{{model.number}}	20.9
{{model.int}}	11
{{model.boolean}}	True
{{model.array}}	[10,11,12]
{{model.items}}	[{"name": "First Item", "index" : 1}, {"name": "Second Item", "index": 3}, {"name": "Third Item", "index": 2}]

Using Expressions

Adding an expression in a template

In **paperwork** we use 'binding' of expressions within html content to add dynamic content. Simply wrap them in a 'handlebars' - double curly brackets e.g. {{ ... }}.
This can be to either update the textual content of the document, or to alter the layout or appearance of the document.

Binding values within the content of a page

Within the content of a tag (or element) the binding statement can appear anywhere, *as long as* it does not overlap or change tags.

```
<table>
   <tr>
      <td>Count is {{count(model.items)}}</td>
   </tr>
</table>

<!--  The following would fail as the closing 'td' is not present.
      <td>Count is {{concat(count(model.items), '</td>')}}
-->
```

To add complex html content into a document see the help sheet on Binding Complex Content

Binding values within attributes

Inside a starting tag (or element), any of the available attributes can be bound to a value using the same handlebars notation.

NOTE: For all attributes, *(except 'style')*, the entire value should either be an expression, or a simple value, not a combination of both.

```
<table hidden="{{if(count(model.items) > 0, '','hidden')}}" >
      ...

</table>

 <-- This will not work with the units outside
<img width="{{model.array[0]}}pt" >...</>  -->

 <!-- But a calculated value can be used instead -->
<img width="{{concat(model.array[0], 'pt')}}" >...</>
```

Using Expressions

Adding expressions in styles

paperwork supports CCS styles, as a tag, inline within an element tag or as a separate referenced stylesheet.
The `var(...)` and `calc(...)` functions support full calculations and expressions, either with the standard css variables, or with the current data model.

Binding values within stylesheets

As part of a stylesheet or <style> tag **paperwork** can use both relative or absolute units or a combination of both with the `calc` funtion.
It supports both standard css variables, or bound values within the current template model using `var`.

```
pre.code {
   width: calc(100% - 20pt);
   border: solid 1pt var(--border-color);
   background-color: var(model.color);
   color: calc(concat("rgb(",model.array[0], model.array[1], model.array[2], ")");
  }
```

Binding values within style attributes

The tag style attribute can also be bound to a css variable or template model value.
As the tag is part of the document generation process, it can be called multiple times for each loop itteration, and use any values within the current context.

```
<table
  style='border: solid 1pt var(model.color ?? "black"); background-color: silver;' >

  ...

  </table>
```

NOTE: Similiar to tag attributes, the value of a style component value should either be an expression, or a simple value, *not a combination of both*.

Using Expressions

Expression notation types

When binding values in **paperwork** a data model or value is added to the template with a key name e.g. `model`.
If that model is a complex object then it's object graph can continue to be traversed using the standard notations, starting with the root item.

Expression Type	Notation	Example
Property (.) dot	{{model.int}}	11
Array Indexor	{{model.array[2]}}	12
Dictionary access	{{model.nested['p1']}}	two
Array Indexor Property	{{model.items[1].name}}	Second Item
Constant	{{Pi}}	3.1415926536
Functions	{{count(model.array)}}	3
Current item (.property) *	{{eachof(model.items, .index)}}	1;3;2

* Looping over items for the accessing the current item is supported with the template component tag discussed later.

Storing expression results in variables

During the execution of a document, complex expressions may need to be calculated, or a look up set of values may need to be used multiple times.
It is often easier and more readable if the results of the expression be stored within the current document execution, rather than re-typed and re-run each time.

To support this **paperwork** has extended the use of the <var> tag to create *or update* models within the document. These can either be simple values or complex object or arrays.

Storing a value

For example, the following will store all the items with an index > 1 in "newModel"

```
<var data-id='newModel' data-value='{{selectWhere(model.items, .index > 1)}}' />
```

Using the variable

And these can then be accessed and used later in the template

```
<span >Items bigger than 1: {{join(' ;', eachOf(newModel, .name))}}</span>
```

During the document processing the value can be updated any number of times, so parent values can be passed down or calculations made with new data. See Looping over model data

Looping over model data

One of the most frequent requirements with any bound data is to enumerate over a list or array of results.
To do this in **paperwork** use the `<template>` tag, with the values to be used for each itteration set with the `data-bind` attribute.

For example:

```
<ol>
  <template data-bind'{{model.items}}' >
   <li>{{.name}}</li>
  </template>
</ol>
```

Will be output as:

1. First Item
2. Second Item
3. Third Item

The template tag can have any inner content, as long as it is one or more valid xhtml tags *(opened and closed within the template content)*.
It also has 4 more attributes specific to altering the items that are each being bound.

Attribute	Type	Description
`data-bind-start`	Integer	The zero based index at which to start outputting the binding content. e.g. setting the value to 2 would skip the first to items found and begin output of the third. *The default is 0*
`data-bind-step`	Integer	The 'step' between each item to be bound (irrespective of the starting index). e.g. setting the value to 2 would skip every other item found. *The default is 1*
`data-bind-max`	Integer	The maximum number of items to be bound. e.g. setting the value to 3 would limit the number of bound items to 3 or less. *The default is 0 which is treated as no max*
`data-content`	html	This dynamically defines the content of the template based on binding to another source of xhtml/html.

Escaping expressions

All content between 2 sets of handlebars will be treated as an expression, and an attempt made to compile the text inside.

If this is not meant to happen, then prefix and postfix the content with a backslash '\'.

For example:

```
<span>Some text that \{{will stay as unparsed content}}\.</span>
```

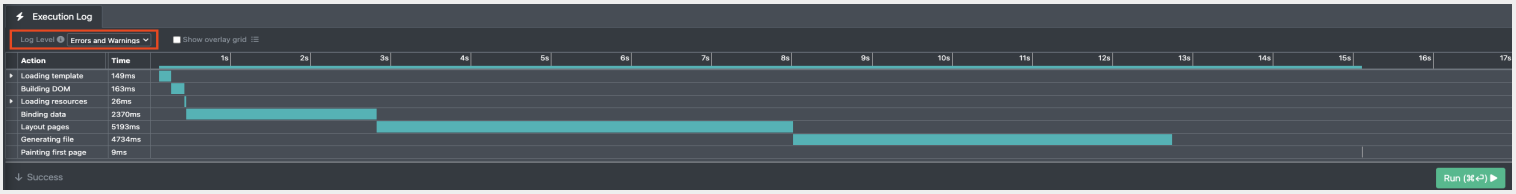
Will be output as:

Some text that {{will stay as unparsed content}}.

Checking errors in expressions

paperwork will attempt to evaluate an expression at runtime, and produce the result, however if it cannot undertand the content no result will be output.
If this is happening, and the reason is not evident then the document creation trace log should be used.

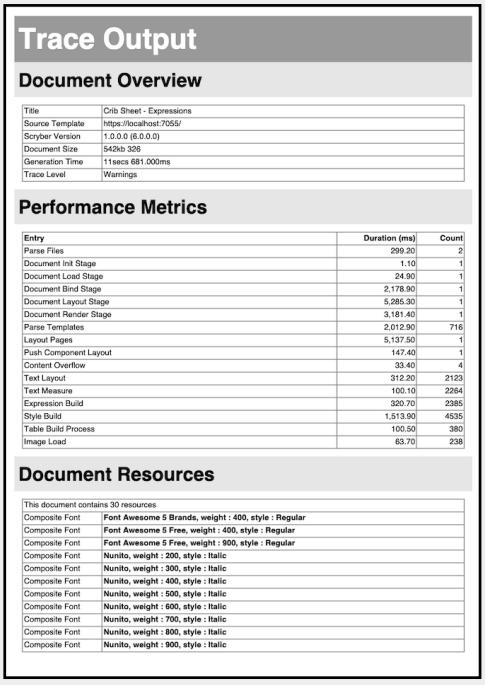
The easiest way from the **paperwork** IDE is to use the execution log, which can be shown from the bottom of the editor.



Or add the pocessing instruction to the top of the template manually:

```
<?paperwork append-log='true' log-level='Errors' ?>
```

The execution log will then be appended to the end of the document as a whole, and can be checked for expression issues.



Trace Output

Document Overview

Title	Crib Sheet - Expressions
Source Template	https://localhost:7055/
Scryber Version	1.0.0.0 (6.0.0.0)
Document Size	542kb 326
Generation Time	11secs 681.000ms
Trace Level	Warnings

Performance Metrics

Entry	Duration (ms)	Count
Parse Files	299.20	2
Document Init Stage	1.10	1
Document Load Stage	24.90	1
Document Bind Stage	2,178.90	1
Document Layout Stage	5,285.30	1
Document Render Stage	3,181.40	1
Parse Templates	2,012.90	716
Layout Pages	5,137.50	1
Push Component Layout	147.40	1
Content Overflow	33.40	4
Text Layout	312.20	2123
Text Measure	100.10	2264
Expression Build	320.70	2385
Style Build	1,513.90	4535
Table Build Process	100.50	380
Image Load	63.70	238

Document Resources

This document contains 30 resources	
Composite Font	Font Awesome 5 Brands, weight : 400, style : Regular
Composite Font	Font Awesome 5 Free, weight : 400, style : Regular
Composite Font	Font Awesome 5 Free, weight : 900, style : Regular
Composite Font	Nunito, weight : 200, style : Italic
Composite Font	Nunito, weight : 300, style : Italic
Composite Font	Nunito, weight : 400, style : Italic
Composite Font	Nunito, weight : 500, style : Italic
Composite Font	Nunito, weight : 600, style : Italic
Composite Font	Nunito, weight : 700, style : Italic
Composite Font	Nunito, weight : 800, style : Italic
Composite Font	Nunito, weight : 900, style : Italic

Finally, if you are all out for the code:

```
var doc = Scryber.Components.Document.ParseDocument('[path to source]');
doc.AppendTraceLog = true;
doc.TraceLog.SetRecordLevel(TraceRecordLevel.Errors);
doc.SaveAsPDF(ms);
```

Binary Operators

Binary operators have both a left and right value with the operator between, and will return a single result from the combined operation. The precedence order is in the standard PEDMAS - however parentheses (brackets) are always recommended to clarify and explain intent.

The following Binary Operators are supported

Binary Operators

Bitwise And (&) operator

The bitwise operators will perform bit level comparison operations on integer values. All bits within the values will be compared based on the AND truth tables.

Usage

```
{{LHS & RHS}}
```

Parameters

Name	Type	Description
LHS	`integer`	Any integer, or a constant or expression that returns an integer value.
RHS	`integer`	Any integer value, or a constant or expression that returns an integer value.

Returns

Type	Description
`integer`	The result of the bitwise operation as a numeric value.

Notes

If either the LHS, or RHS parameters are not integers, then an attempt will be made to convert the parameters string representation to an integer. If the conversion cannot be made, then the entire expression will error. For explicit conversion, use the integer() function.

Examples

11 & 7	3
string(0b1011 & 7, 'b')	11
model.int & 7	3

See Also

Bitwise Or (|) operator *Integer Function* *Double Function* *String Function*

Binary Operators

Bitwise Or (|) operator

The bitwise operators will perform bit level comparison operations on integer values. All bits within the values will be compared based on the OR truth tables.

Usage

```
{{LHS | RHS}}
```

Parameters

Name	Type	Description
LHS	`integer`	Any integer, or a constant or expression that returns an integer value.
RHS	`integer`	Any integer value, or a constant or expression that returns an integer value.

Returns

Type	Description
`integer`	The result of the bitwise operation as a numeric value.

Notes

If either the LHS, or RHS parameters are not integers, then an attempt will be made to convert the parameters string representation to an integer. If the conversion cannot be made, then the entire expression will error. For explicit conversion, use the integer() function.

Examples

11 | 7	15
string(0b1011 | 7, 'b')	1111
model.int | 7	15

See Also

Bitwise And (&) operator | *Integer Function* | *Double Function* | *String Function*

Binary Operators

Bitwise Shift Left (<<) operator

The left shift (<<) operator returns a number whose binary representation is the first operand shifted by the specified number of bits to the left. Excess bits shifted off to the left are discarded, and zero bits are shifted in from the right.

Usage

```
{{LHS << RHS}}
```

Parameters

Name	Type	Description
LHS	`integer`	Any integer, or a constant or expression that returns an integer value.
RHS	`integer`	Any integer value, or a constant or expression that returns an integer value.

Returns

Type	Description
`integer`	The result of the bitwise operation as a numeric value.

Notes

If either the LHS, or RHS parameters are not integers, then an attempt will be made to convert the parameters string representation to an integer. If the conversion cannot be made, then the entire expression will error. For explicit conversion, use the integer() function.

Examples

11 << 7	1408
string(0b1011 << 7, 'b')	10110000000
string(model.int << 7)	1408

See Also

Bitwise Shift Right (>>) operator | *Integer Function* | *Double Function* | *String Function*

Bitwise Shift Right (>>) operator

The right shift (>>) operator returns a number whose binary representation is the first operand shifted by the specified number of bits to the right. Excess bits shifted off to the right are discarded, and zero bits are shifted in from the left.

Usage

```
{{LHS >> RHS}}
```

Parameters

Name	Type	Description
LHS	`integer`	Any integer, or a constant or expression that returns an integer value.
RHS	`integer`	Any integer value, or a constant or expression that returns an integer value.

Returns

Type	Description
`integer`	The result of the bitwise operation as a numeric value.

Notes

If either the LHS, or RHS parameters are not integers, then an attempt will be made to convert the parameters string representation to an integer. If the conversion cannot be made, then the entire expression will error. For explicit conversion, use the integer() function.

Examples

11 >> 2	2
string(0b1011 >> 2, 'b')	10
string(model.int >> 2)	2

See Also

Bitwise Shift Left (<<) operator | *Integer Function* | *Double Function* | *String Function*

Binary Operators

Divide (/) operator

The divide operator will divide the LHS value by the RHS value and return the result

Usage

```
{{LHS / RHS}}
```

Parameters

Name	Type	Description
LHS	`number`	Any number, or a constant or expression that returns a number value.
RHS	`number`	Any number, or a constant or expression that returns a number value.

Returns

Type	Description
`number`	The result of the divide operation as a numeric value.

Notes

If either the LHS, or RHS parameters are not numbers then an attempt will be made to convert the parameters string representation to a number. If the conversion cannot be made, then the entire expression will error.

Examples

50 / 4	12.5
50 / model.number	2.3923444976
model.number / "12"	1.7416666667
model.number / "two"	

See Also

Multiply (*) operator *Integer Function* *Double Function* *String Function*

Binary Operators

Exponent (^) operator

The exponent operator raise the LHS number to the power of the RHS number

Usage

```
{{LHS ^ RHS}}
```

Parameters

Name	Type	Description
LHS	`number`	Any number, or a constant or expression that returns a number value.
RHS	`number`	Any number, or a constant or expression that returns a number value.

Returns

Type	Description
`number`	The result of the exponent operation as a numeric value.

Notes

If either the LHS, or RHS parameters are not numbers then an attempt will be made to convert the parameters string representation to a number. If the conversion cannot be made, then the entire expression will error.

Examples

50 ^ 4	6250000
50 ^ (model.int / 10)	73.9378818314
model.number ^ "2"	6946330348617177
"12" ^ 2	144
model.number ^ "two"	

See Also

Divide (/) operator *Multiply (*) operator* *Integer Function* *Double Function* *String Function*

Minus (-) operator

The minus operator will subtract the second (right) operand from the first (left) operand.

Usage

```
{{LHS - RHS}}
```

Parameters

Name	Type	Description
LHS	`number`	Any number, or a constant or expression that returns a number value.
RHS	`number`	Any number, or a constant or expression that returns a number value.

Returns

Type	Description
`number`	The result of the subtract operation as a numeric value.

Notes

If either the LHS, or RHS parameters are not numbers then an attempt will be made to convert the parameters string representation to a number. If the conversion cannot be made, then the entire expression will error.

Examples

50 - 4	46
50 - model.number	29.1
model.number - "12"	8.9
model.number - "two"	

See Also

Plus (+) operator *Integer Function* *Double Function* *String Function*

Modulo (%) operator

The modulo operator will divide the LHS value by the RHS value and return the remainder of the division

Usage

```
{{LHS % RHS}}
```

Parameters

Name	Type	Description
LHS	`number`	Any number, or a constant or expression that returns a number value.
RHS	`number`	Any number, or a constant or expression that returns a number value.

Returns

Type	Description
`number`	The result of the modulo operation as a numeric value.

Notes

If either the LHS, or RHS parameters are not numbers then an attempt will be made to convert the parameters string representation to a number. If the conversion cannot be made, then the entire expression will error.

Examples

50 % 4	2
50 % model.number	8.2
model.number % "12"	8.9
model.number % "two"	

See Also

Divide (/) operator *Integer Function* *Double Function* *String Function*

Multiply (*) operator

The multiply operator will return the product value of the LHS by the RHS

Usage

```
{{LHS * RHS}}
```

Parameters

Name	Type	Description
LHS	`number`	Any number, or a constant or expression that returns a number value.
RHS	`number`	Any number, or a constant or expression that returns a number value.

Returns

Type	Description
`number`	The result of the multiply operation as a numeric value.

Notes

If either the LHS, or RHS parameters are not numbers then an attempt will be made to convert the parameters string representation to a number. If the conversion cannot be made, then the entire expression will error.

Examples

50 * 4	200
50 * model.number	1045
model.number * "12"	250.8
model.number * "two"	

See Also

Divide (/) operator | *Integer Function* | *Double Function* | *String Function*

Binary Operators

Null coalescing (??) operator

The null coalesce operator (??) will provide the first (left) value if it is not null, otherwise it will provide the second (right) value.

Usage

```
{{LHS ?? RHS}}
```

Parameters

Name	Type	Description
LHS	`object`	Any object, or a constant or expression that returns an object value.
RHS	`object`	Any object, or a constant or expression that returns an object value.

Returns

Type	Description
`object`	The result of the null comparison in the type provided of either the left or right hand side arguments. No conversion is automatically done.

Examples

null ?? 'replaced'	replaced
'not-replaced' ?? 'replaced'	not-replaced
model.color ?? '#aaaaaa'	#330033
model.notset ?? '#aaaaaa'	#aaaaaa

See Also

If Function *String Function*

Binary Operators

Plus (+) operator

The plus operator will add 2 numbers together or join 2 strings, if the first operand is a string

Usage

```
{{LHS + RHS}}
```

Parameters

Name	Type	Description
LHS	`object`	Any object, constant or expression. If it is not a number then it will be converted to a string.
RHS	`object`	Any object, constant or expression. If the LHS parameter is a number then this RHS value should be a number, or be able to be converted to a number

Returns

Type	Description
`number`	If both values are convertable to numeric values.
`string`	If the first value is a string, then the concatenation of both parameters.

Notes

When the first operand is a string then the following argument will be concatenated, however if the first argument is a number then the second will attempt to be converted to a number. If in doubt use the conversion functions, or concatenate function.

Examples

12 + 4	16
12 + model.number	32.9
"12" + model.number	1220.9
model.number + "12"	32.9
model.number + " - as a string"	
"To a string - " + model.number	To a string - 20.9

See Also

Relational Operators

The following Relational Operators are supported

Relational Operators

Equal (==) operator

The equality (==) operator will return true if the left and right side are the same value. If the types of values are different, then an attempt to convert to the same value will be made.

Usage

```
{{LHS == RHS}}
```

Parameters

Name	Type	Description
LHS	`Object`	Required. Any object, constant or expression
RHS	`Object`	Required. Any object, constant or expression

Returns

Type	Description
`boolean`	true if the LHS and RHS objects are considered equal; otherwise, false

Notes

Zero is classed as false, any other non null value is true.

Examples

20.9 == 20.9	True
model.number == 20.0	False
model.number == '20.9'	True
0 == false	True
1 == false	False
model.number == null	False
model.bg == 'silver'	True
model.bg == 'SILVER'	False

See Also

Not Equal (!=) operator

if

Relational Operators

Greater than (>) operator

The more than (>) operator will return true if the left side is greater than the right side. Again, if the types of values are different, then an attempt to convert to the same type will be made. String comparison is case sensitive

Usage

```
{{LHS > RHS}}
```

Parameters

Name	Type	Description
LHS	`Object`	Required. Any object, constant or expression
RHS	`Object`	Required. Any object, constant or expression

Returns

Type	Description
`boolean`	true if the LHS object is consideredand more than the RHS object, otherwise false

Notes

More information on the 'if' function can be found later in the logical functions.

Examples

20.9 > 21.0	False
20.9 > 20.0	True
model.number > 30.0	False
model.bg > 'silver'	False
if(model.number > 20,'More', 'Less or equal')	More
if(model.bg > 'silver','More', 'Less or equal')	Less or equal
if(model.bg > 'SILVER','More', 'Less or equal')	More

Relational Operators

Greater than or equal (>=) operator

The greater than or equal (>=) operator will return true if the left side is more than or the same as the right side. If the types of values are different, then an attempt to convert to the same type will be made. String comparison is case sensitive

Usage

```
{{LHS >= RHS}}
```

Parameters

Name	Type	Description
LHS	`Object`	Required. Any object, constant or expression
RHS	`Object`	Required. Any object, constant or expression

Returns

Type	Description
`boolean`	true if the LHS object is consideredand greater than or eaqual to the RHS object, otherwise false

Notes

More information on the 'if' function can be found later in the logical functions.

Examples

20.9 >= 20.0	True
20.9 >= 30.0	False
model.number >= 30.0	False
model.bg >= 'silver'	True
if(model.number >= 20,'More or equal', 'Less')	More or equal
if(model.bg >= 'silver','More or equal', 'Less')	More or equal

Less than (<) operator

The less than (<) operator will return true if the left side is considered less than the right side. Again, if the types of values are different, then an attempt to convert to the same type will be made. String comparison is case sensitive

Usage

```
{{LHS < RHS}}
```

Parameters

Name	Type	Description
LHS	`Object`	Required. Any object, constant or expression
RHS	`Object`	Required. Any object, constant or expression

Returns

Type	Description
`boolean`	true if the LHS object is consideredand less than the RHS objects, otherwise false

Notes

More information on the 'if' function can be found later in the logical functions.

Examples

20.9 < 21.0	True
20.9 < 20.9	False
model.number < 30.0	True
model.bg < 'silver'	False
if(model.number < 21,'Less', 'More or equal')	Less
if(model.bg < 'silver','Less', 'More or equal')	More or equal
if(model.bg < 'SILVER','Less', 'More or equal')	More or equal

Less than or equal (<=) operator

The less than or equal (<=) operator will return true if the left side is less than or the same as the right side. If the types of values are different, then an attempt to convert to the same type will be made. String comparison is case sensitive

Usage

```
{{LHS <= RHS}}
```

Parameters

Name	Type	Description
LHS	`Object`	Required. Any object, constant or expression
RHS	`Object`	Required. Any object, constant or expression

Returns

Type	Description
`boolean`	true if the LHS object is consideredand less than or eaqual to the RHS object, otherwise false

Notes

More information on the 'if' function can be found later in the logical functions.

Examples

20.9 <= 20.0	False
20.9 <= 20.9	True
model.number <= 30.0	True
model.bg <= 'silver'	True
model.bg <= 'liver'	False
if(model.number <= 21,'Less or equal', 'More')	Less or equal
if(model.bg <= 'silver','Less or equal', 'More')	Less or equal
if(model.bg <= 'SILVER','Less or equal', 'More')	More

Not Equal (!=) operator

The non-equality (!=) operator will return true if the left and right side are *not* the same value. Again, if the types of values are different, then an attempt to convert to the same type will be made.

Usage

```
{{LHS != RHS}}
```

Parameters

Name	Type	Description
LHS	`Object`	Required. Any object, constant or expression
RHS	`Object`	Required. Any object, constant or expression

Returns

Type	Description
`boolean`	true if the LHS and RHS objects are *not* considered equal, otherwise false

Notes

Zero is classed as false, any other non null value is true

Examples

20.9 != 20.9	False
model.number != 20.0	True
model.number != '20.9'	False
0 != true	True
1 != true	False
model.number != null	True
model.bg != 'silver'	False
model.bg != 'SILVER'	True

Logical Operators

The following Logical Operators are supported

Logical Operators

And (&&) operator

The logical and (&&) operator will return true if the left and right side are both true, otherwise it will return false. Either side can be a constant or an expression, and if the types of values are different, then an attempt to convert to the same value will be made.

Usage

```
{{LHS && RHS}}
```

Parameters

Name	Type	Description
LHS	`Object`	Required. Any object, constant or expression
RHS	`Object`	Required. Any object, constant or expression

Returns

Type	Description
`boolean`	true if both the LHS and RHS objects are considered true otherwise, false

Notes

Using the && operator within an xhtml template requires the operand to be escaped to & otherwise the actual structure of the document is invalid. As with other operators 0 and null are false.

Examples

true && true	True
true && false	False
model.boolean && true	True
null && true	False
'true' && true	True
model.boolean && 1	True
model.boolean && 0	False
model.boolean && -1	True

Logical Operators

Not (!) operator

The not (!) unary operator will return true if the contained value results in false, or false if the contained value results in true. The contained value can be a constant or another expression, and if the type of value is not a boolean value, then an attempt to convert to boolean will be made.

Usage

```
{{!RHS}}
```

Parameters

Name	Type	Description
RHS	`Object`	Required. Any object, constant or expression

Returns

Type	Description
`boolean`	true if the RHS object is considered false otherwise, true

Notes

The !! operator must currently be separated by parentheses e.g. !(!true), directly using !!true will cause an error

Examples

!true	False
!0	True
!10	False
!(model.number + 10)	False
!(model.number - 20.9)	True
!'false'	True
!model.boolean	False

Logical Operators

Or (||) operator

The logical or (||) operator will return true if the left or right side are true, otherwise it will return false. Either side can be a constant or an expression, and if the types of values are different, then an attempt to convert to the same value will be made.

Usage

```
{{LHS || RHS}}
```

Parameters

Name	Type	Description
LHS	`Object`	Required. Any object, constant or expression
RHS	`Object`	Required. Any object, constant or expression

Returns

Type	Description
`boolean`	true if either the LHS or RHS objects are considered true otherwise, false

Notes

As with other operators 0 and null are false.

Examples

true || false	True
0 || false	False
model.boolean || false	True
null || false	False
'true' || false	True
model.boolean || 1	True
model.boolean || 0	True
model.boolean || -1	True

Conversion Functions

The following Conversion Functions are supported

Conversion Functions

Boolean Function

The boolean() function will return true if the contained expression can be converted to a true value, otherwise false, or false if the contained expression results in false. The contained expression can be a constant or another expression, and an attempt to convert to boolean will be made.

Usage

```
{{boolean(expr)}}
```

Parameters

Name	Type	Description
expr	`Object`	Required. Any object, constant or expression

Returns

Type	Description
`boolean`	If 'expr' is considered a true, non zero or null value returns true, otherwise false

Notes

A non-empty string value is considered true, unless it is 'false'

Examples

boolean(1)	True
boolean(0)	False
boolean(-1)	True
boolean(null)	False
boolean(model.number)	True
boolean('')	False
boolean('a string')	True
boolean('false')	False
boolean(model.array)	True

Conversion Functions

Date Function

The date function will return the value, of the contained expression, as a date. If no contained expression is provided, then it will return the current date and time. The contained expression can be a constant or another expression, and an attempt to convert to date will be made. If a second *format* parameter is provided, then the date will be parsed according to that format.

Usage

```
{{date()}} or {{date(expr)}} or {{date(expr, format)}}
```

Parameters

Name	Type	Description
expr	`Object`	Optional. Any value, constant or expression that is either a string value for a date, or a number (representing the number of milliseconds past since 01-01-0001 00:00:00)
format	`string`	Optional. Any value, constant or expression that returns a string. A standard (non-cultural) string representation or the format for the date.

Returns

Type	Description
`DateTime`	The 'expr' as a DateTime, or if no 'expr' is given, then the current date and time.

Notes

Once converted to a date there are many date manipulation functions available to alter the value.

Examples

date()	05/15/2025 09:35:45
date(63822124800000)	06/12/2023 00:00:00
date('12 June 2023')	06/12/2023 00:00:00
date('12 06 2023', 'dd MM yyyy')	06/12/2023 00:00:00
date('12 06 2023', 'MM dd yyyy')	12/06/2023 00:00:00
string(date('12 06 2023', 'MM dd yyyy'),'dd MMM yyyy')	06 Dec 2023

Decimal Function

The decimal function will return the value as a decimal of the contained expression. The contained expression can be a constant or another expression, and an attempt to convert to decmial will be made.

Usage

```
{{decimal(expr)}}
```

Parameters

Name	Type	Description
expr	`Object`	Required. Any object, constant or expression

Returns

Type	Description
`decimal`	If 'expr' can be converted to a decimal value, the value is returned.

Examples

decimal(0)	0
decimal(0.25)	0.25
decimal('0.25')	0.25
decimal(false)	0
decimal(true)	1
decimal(model.int + 1)	12
decimal(model.array)	

Double Function

The double function will return the value as a double of the contained expression. The contained expression can be a constant or another expression, and an attempt to convert to double will be made.

Usage

```
{{double(expr)}}
```

Parameters

Name	Type	Description
expr	`Object`	Required. Any object, constant or expression

Returns

Type	Description
`double`	If 'expr' can be converted to a double value, the value is returned

Examples

double(0)	0
double(0.25)	0.25
double('0.25')	0.25
double(false)	0
double(true)	1
double(model.int)	11
double(model.array)	

Conversion Functions

Integer Function

The integer function will return the value as an integer of the contained expression. The contained expression can be a constant or another expression, and an attempt to convert to integer will be made. Null or invalid values will cause an error

Usage

```
{{integer(expr)}}
```

Parameters

Name	Type	Description
expr	`Object`	Required. Any object, constant or expression

Returns

Type	Description
`Int34`	If 'expr' can be converted to a integer value, the value is returned

Notes

*If the value to be converted to an integer is a floating point (double, decimal) value, then it will be rounded to the nearest integer. If it is a string, then it *must* be able an integral value representation.*

Examples

integer(0)	0
integer(10.6)	11
integer('10')	10
integer('10.6')	
integer(double('10.6'))	11
integer(false)	0
integer(true)	1
integer(model.number)	21
integer(model.array)	

String Function

The string function will return the value of the contained expression as a string. Complex (object and array values will be expanded). A formatting parameter can also be specifiied to alter how the value is encoded.

Usage

```
{{string(expr)}} or {{string(expr, format)
```

Parameters

Name	Type	Description
expr	`Object`	Required. Any object, constant or expression
format	`string`	Required. Any value, constant or expression as a string

Returns

Type	Description
`string`	If 'expr' can be converted to a integer value, the value is returned

Notes

The standard .NET format conversion notations can be used, but cultures are not currently set on the expressions. This may be supported in the future, however the formatting string can be bound to a model value, as well, for flexibility.

Examples

string(10)	10
string(10 + 1)	11
string(10) + 1	101
string(false)	False
string(true)	True
string(model.int)	11
string(model.number)	20.9
string(model.array)	[10,11,12]
string(model.items[1])	{"name": "Second Item", "index": 3}
string(model.notset)	
string(model.notset) + ' - after'	
string(model.notset ?? '') + ' - after'	- after
string(10.3456,'##0.00')	10.35
string(10.3456,'000.00#')	010.346
string(10.3456,'£#,##0.00')	£10.35
string(model.number + 1,'# ##0,00 €')	022 €
string(date())	05/15/2025 09:35:45
string(date(),'D')	Thursday, 15 May 2025
string(date(),'dd MMM yyyy')	15 May 2025
string(integer(model.number))	21
string(date('12 06 2023', 'MM dd yyyy'),'D')	Wednesday, 06 December 2023

Mathematical Functions

The following Mathematical Functions are supported

Abs Function

The abs() function will return the absolute (positive) value of a provided number. If the value is negative it will return the positive value of that number, otherwise it will return the original number.

Usage

```
{{abs(expr)}}
```

Parameters

Name	Type	Description
expr	`number`	Required. Any value, constant or expression that represents a numeric value

Returns

Type	Description
`number`	If 'expr' is considered a number, the absolute value of that number in its type will be returned (e.g. integer returns integer, double returns double).

Examples

Abs(10)	10
Abs(1 - 10)	9
Abs('-5')	
Abs(double('-5'))	5
Abs(null)	
Abs(model.int)	11
Abs(model.int - model.number)	9.9

Acos Function

The acos() function will return the arccosine, or inverse cosine, of the provided number in radians.

Usage

```
{{acos(expr)}}
```

Parameters

Name	Type	Description
expr	`number`	Required. Any value, constant or expression that represents a numeric value

Returns

Type	Description
`double`	The radian value arccosine of the expr value.

Notes

If the result is needed in degrees, then multiply by 180/Pi or use the deg() function to get the angle in degrees

Examples

acos(1)	0
acos(0.3304651081)	1.234
acos(-0.5)	2.0943951024
acos(0.5)*180/Pi	60
deg(acos(0.5))	60
acos(model.number/100.0)*180/Pi	77.9362438645

Asin Function

The asin() function will return the arcsine, or inverse sine of the provided number in radians.

Usage

```
{{aSin(expr)}}
```

Parameters

Name	Type	Description
expr	`number`	Required. Any value, constant or expression that represents a numeric value

Returns

Type	Description
`double`	The radian value arcsine of the expr value.

Notes

If the result is needed in degrees, then multiply by 180/Pi or use the deg() function to get the angle in degrees

Examples

asin(1)	1.5707963268
asin(0.3304651081)	0.3367963268
asin(-0.5)	-0.5235987755982989
asin(0.5)*180/Pi	30
deg(asin(0.5))	30
asin(model.number/100.0)*180/Pi	12.0637561355

Atan Function

The atan() function will return the arctangent, or inverse tangent of the provided number in radians.

Usage

```
{{atan(expr)}}
```

Parameters

Name	Type	Description
expr	`number`	Required. Any value, constant or expression that represents a numeric value

Returns

Type	Description
`double`	The radian value arctangent of the expr value.

Notes

If the result is needed in degrees, then multiply by 180/Pi or use the deg() function to get the angle in degrees

Examples

atan(1)	0.7853981634
atan(2.8560298389)	1.234
atan(1.7320508076) * 180/Pi	60
deg(atan(1.7320508076))	60

Ceiling Function

The ceiling() function will return the lowest possible integer value above the provided value.

Usage

```
{{ceiling(expr)}}
```

Parameters

Name	Type	Description
expr	`number`	Required. Any value, constant or expression that represents a numeric value

Returns

Type	Description
`decimal or double`	If 'expr' is considered a number, the ceiling value of that number in its type will be returned as either a decimal if the expr is a decimal or as double if expr is a double or any other numeric type.

Examples

ceiling(20.3456)	21
ceiling(-20.3456)	-20
ceiling(model.number)	21
ceiling('20.9')	21
ceiling('two')	
ceiling(model.number + 0.6)	22

Cos Function

The cos() function will return the cosine of the provided angle (in radians).

Usage

```
{{cos(expr)}}
```

Parameters

Name	Type	Description
expr	`number`	Required. Any value, constant or expression that represents a numeric value

Returns

Type	Description
`double`	If 'expr' is considered a number, the cosine value of that number in radians will be returned as double, even if expr is another numeric type.

Notes

If the angle is in degrees, then multiply by Pi/180 or use the rad() function to get the angle in radians

Examples

cos(1.234)	0.3304651081
cos(2.0943951024)	-0.500000000005893
cos(60*Pi/180)	0.5
cos(rad(60))	0.5
cos(model.number*Pi/180)	0.9342044743
cos(rad(model.number))	0.9342044743

Deg Function

The deg() function will convert any numeric value (expected to be in radians), to its equivalent degree value based on the rotation around half a circle circumference.

Usage

```
{{deg(expr)}}
```

Parameters

Name	Type	Description
expr	`number`	Required. Any value, constant or expression that represents a numeric value

Returns

Type	Description
`double`	If 'expr' is considered a number, the degrees function will convert that number in radians to a double in degrees, even if expr is another numeric type.

Examples

deg(Pi/180)	1
round(deg(1.047198),4)	60
round(deg('1.047198'),4)	60
deg('two')	
deg(rad(model.number + 9.1))	30

E Function

The e() function will return the value of Eulers number (to 10 decimal places). It can also be used as a constant without the parenthese. e.g. e and E() are equivalent.

Usage

```
{{e()}} or {{e}}
```

Returns

Type	Description
`double`	Eulers constant value to 10 decimal places.

Notes

As a constant it also means that no template variables or parameters should be named 'e' as a conflicting state would arise

Examples

E()	2.7182818285
e	2.7182818285
e * model.int	29.901100113

Floor Function

The floor() function will return the highest possible integer value below the provided value.

Usage

```
{{floor(expr)}}
```

Parameters

Name	Type	Description
expr	`number`	Required. Any value, constant or expression that represents a numeric value

Returns

Type	Description
`decimal or double`	If 'expr' is considered a number, the floor value of that number in its type will be returned as either a decimal if the expr is a decimal or as double if expr is a double or any other numeric type.

Examples

floor(20.3456)	20
floor(-20.3456)	-21
floor(model.number)	20
floor('20.3456')	20
floor('two')	
floor(model.number + 0.6)	21

Log Function

The log() function will return the logarithm of the first argument in the base of the second argument.

Usage

```
{{log(expr, newbase)}}
```

Parameters

Name	Type	Description
expr	`number`	Required. Any value, constant or expression that represents a numeric value
newbase	`number`	Required. Any value, constant or expression that represents a numeric value

Returns

Type	Description
`double`	If 'expr' is considered a number, the logarithm value of that number be returned as a double in the new base.

Examples

log(10,2)	3.3219280949
log(10,e)	2.302585093
log('10',e)	2.302585093
log('10','2')	3.3219280949
log('10','e')	
log(10,10)	1
log(model.number * 1000, model.int)	4.1484315645

Log10 Function

The log10() is a standard shorthand for the log(n,10) function, and will return the base 10 logarithm of a number.

Usage

```
{{log10(expr)}}
```

Parameters

Name	Type	Description
expr	`number`	Required. Any value, constant or expression that represents a numeric value

Returns

Type	Description
`double`	If 'expr' is considered a number, the logarithm value of that number be returned as a double in the new base.

Examples

log10(1)	0
log10(10)	1
log10('10')	1
log10('two')	
log10(model.number * 1000)	4.3201462861

Pi Function

The pi() function will return the value of Pi (to 10 decimal places). It can also be used as a constant without the parenthese. e.g. pi and PI() are equivalent.

Usage

```
{{pi()}} or {{pi}}
```

Returns

Type	Description
`double`	The value of Pi to 10 decimal places.

Notes

As a constant it also means that no template variables or parameters should be named 'pi' as a conflicting state would arise

Examples

Pi()	3.1415926536
pi	3.1415926536
pi/2	1.5707963268

Pow Function

The pow() function requires 2 arguments and will return the first argument, raised to the exponent of the second argument e.g. pow(3,2) is equivalent to 3^2.

Usage

```
{{pow(expr, exponent)}}
```

Parameters

Name	Type	Description
expr	`number`	Required. Any value, constant or expression that represents a numeric value
exponent	`number`	Required. Any value, constant or expression that represents a numeric value

Returns

Type	Description
`double`	If 'expr' is considered a number, the value of that number will be raised by the exponent and returned as a double.

Examples

pow(10,2)	100
pow(1,3)	1
pow('2',3)	8
pow('2.2','3.3')	13.4894687605
pow('two','3.3')	
pow(model.number,model.int-8)	9129.329

Rad Function

The rad() function will convert a value in degrees to it's radian equivalent.

Usage

```
{{rad(expr)}}
```

Parameters

Name	Type	Description
expr	`number`	Required. Any value, constant or expression that represents a numeric value

Returns

Type	Description
`double`	If 'expr' is considered a number, the radians function will convert that number in degrees to a double in radians, even if expr is another numeric type.

Examples

rad(180)/Pi	1
round(rad(60),7)	1.0471976
round(rad('60'),4)	1.0472
rad('sixty')	
rad(model.number + 9.1)	0.5235987756

Round Function

The round() function will return the value of a provided number to the nearest integer, or rounded to any provided precision.

Usage

```
{{round(expr)}} or {{round(expr,precision)}}
```

Parameters

Name	Type	Description
expr	`number`	Required. Any value, constant or expression that represents a numeric value
precision	`integer`	Optional. Any value, constant or expression that represents an integer value, or rounded up to the nearest integer value. Will not be converted from a string value

Returns

Type	Description
`double`	If 'expr' is considered a number, the round value of that number will be returned as a double with either 0 or 'precision' number of numbers after the decimal separator.

Examples

round(20.3456)	20
round(model.number)	21
round(20.3456, 1)	20.3
round(20.3456, 3)	20.346
round('20.3456', 1)	20.3
round(20.3456, '3.6')	
round('two point three', 1)	
round('20.3456', 'three')	
round(model.number + 0.6, 0)	22

Sign Function

The sign() function will return 1 if the value of a provided number is positive. If the value is negative it will return -1, and zero will return 0.

Usage

```
{{sign(expr)}}
```

Parameters

Name	Type	Description
expr	`number`	Required. Any value, constant or expression that represents a numeric value

Returns

Type	Description
`integer`	Either 1, 0 or -1 depending on the value of the numeric expression

Examples

Sign(10.4)	1
Sign(1 - 10)	-1
Sign(model.int)	1
Sign(model.notset)	
Sign(0)	0
sign(null)	
sign('two')	

Sin Function

The sin() function will return the sine of the provided angle (in radians).

Usage

```
{{sin(expr)}}
```

Parameters

Name	Type	Description
expr	`number`	Required. Any value, constant or expression that represents a numeric value

Returns

Type	Description
`double`	The resultant sine value of the provided angle

Notes

If the angle is in degrees, then multiply by Pi/180 or use the rad() function to get the angle in radians

Examples

sin(1.234)	0.9438182094
sin(2.0943951024)	0.8660254038
sin(60*Pi/180)	0.8660254038
sin(rad(60))	0.8660254038
sin('2.0943951024')	0.8660254038
sin('two')	
sin(model.number*Pi/180)	0.3567379993
sin(rad(model.number))	0.3567379993

Sqrt Function

The sqrt() function will return the square root of a number.

Usage

```
{{sqrt(expr)}}
```

Parameters

Name	Type	Description
expr	`number`	Required. Any value, constant or expression that represents a numeric value

Returns

Type	Description
`double`	The resultant square root value of the provided number

Examples

sqrt(10)	3.1622776602
sqrt('10')	3.1622776602
sqrt('ten')	
sqrt(e)	1.6487212707
sqrt(model.number + 4.1)	5

Tan Function

The tan() function will return the tangent of the provided angle (in radians).

Usage

```
{{tan(expr)}}
```

Parameters

Name	Type	Description
expr	`number`	Required. Any value, constant or expression that represents a numeric value

Returns

Type	Description
`double`	If 'expr' is considered a number, the cosine value of that number in radians will be returned as double, even if expr is another numeric type.

Notes

If the angle is in degrees, then multiply by Pi/180 or use the rad() function to get the angle in radians

Examples

tan(1.234)	2.8560298389
tan(2.0943951024)	-1.7320508075416592
tan(60*Pi/180)	1.7320508076
tan(rad(60))	1.7320508076
tan(model.number*Pi/180)	0.3818628674
tan(rad(model.number))	0.3818628674

Truncate Function

The truncate() function will remove any floating point value. This means negative values are equivalent to ceiling(expr) and positive values are equivalent to floor(expr).

Usage

```
{{truncate(expr)}}
```

Parameters

Name	Type	Description
expr	`number`	Required. Any value, constant or expression that represents a numeric value

Returns

Type	Description
`double`	If 'expr' is considered a number, the truncated value of that number will be returned as double, even if expr is another numeric type.

Examples

truncate(20.6456)	20
truncate(-20.6456)	-20
truncate('-20.6456')	-20
truncate('two')	
truncate(model.number)	20
truncate(model.number + 0.6)	21

String Functions

The following String Functions are supported

Concat(enate) Function

The concat() function will take any length of parameters and concatenate them into a single string returning the result. If one of the parameters is a collection or array, then each of the entries in that array will be appended to the string in order.

Usage

```
{{concat(expr, expr, expr, expr ...)}}
```

Parameters

Name	Type	Description
expr	`object or array`	One or more. Any string, constant or expression (that will be evaluated and converted to a string). If the expr is an array then the contents will be expanded once and each entry concatenated.

Returns

Type	Description
`string`	The resultant concatenation of all the expression values

Notes

If one or more of the values is null, then the entry will be skipped and any further values processed

Examples

concat('hello ','world!')	hello world!
concat(model.array)	101112
concat('09',model.array, '!')	09101112!
concat('09',model.notset, '!')	09!
concat(model.number, ' + 10 = ', model.number + 10)	20.9 + 10 = 30.9
concat(model.items)	{"name": "First Item", "index" : 1}{"name": "Second Item", "index": 3}{"name": "Third Item", "index": 2}

Contains Function

The contains() function will return true if the second string parameter appears at least once somewhere in the first string parameter. If not then the function returns false. The search is case-sensitive.

Usage

```
{{contains(expr, comparison)}}
```

Parameters

Name	Type	Description
expr	`string`	Required. Any string, constant or expression (that will be evaluated and converted to a string).
comparison	`string`	Required. Any string, constant or expression (that will be evaluated and converted to a string).

Returns

Type	Description
`boolean`	True if the expr string contains the comparison string

Examples

contains('hello world!','world')	True
contains('hello world!','World')	False
contains(concat(model.array),'10')	True
contains(concat(model.array),'13')	False

See Also

EndsWith Function *StartsWith Function* *Matches Function* *IsMatch Function*

EndsWith Function

The endsWith() function will return true if the second string parameter appears at the end of the first string parameter. If not then the function returns false. The search is by case-sensitive.

Usage

```
{{endswith(expr, comparison)}}
```

Parameters

Name	Type	Description
expr	`string`	Required. Any string, constant or expression (that will be evaluated and converted to a string).
comparison	`string`	Required. Any string, constant or expression (that will be evaluated and converted to a string).

Returns

Type	Description
`boolean`	True if the expr string ends with the comparison string

Examples

endswith('hello world!','world!')	True
endswith('hello world!','world')	False
endswith('hello world!','World!')	False
endswith(concat(model.array),'10')	False
endswith(concat(model.array),'12')	True

See Also

Contains Function *StartsWith Function* *Matches Function* *IsMatch Function*

Eval Function

The eval() function accepts a single parameter and will try and evaluate and execute the result by converting the string to another expression.

> **Warning**
>
> No range checking or validation is done on the value of the expr. It is up to you to know that remote content could be loaded based on input values and limit the ability, or trust.

Usage

```
{{eval(expr)}}
```

Parameters

Name	Type	Description
expr	`string`	Required. Any string, constant or expression (that will be evaluated and converted to a string) and evaluated

Returns

Type	Description
`object`	The results of the expression

Notes

The eval function has access to all the variables and values within the document context, as it is executing as a standard expression

Examples

eval('10 + 2')	12
eval('concat(model.array)')	101112
integer(eval('concat(model.array)')) + 1000	102112

IndexOf Function

The indexof() function will return the starting (zero based) index of the first appearance of the second parameter within the first. If it is not found then the function returns -1. The search is case-sensitive.

Usage

```
{{indexof(expr, comparison)}}
```

Parameters

Name	Type	Description
expr	`string`	Required. Any string, constant or expression (that will be evaluated and converted to a string).
comparison	`string`	Required. Any string, constant or expression (that will be evaluated and converted to a string).

Returns

Type	Description
`integer`	If found in the expr string, returns the index of first appearance of the string.

Examples

indexof('hello world!','world')	6
indexof('hello world!','World')	-1
indexof('hello world!','o')	4
substring('hello world!', 0,indexof('hello world!','o') + 1)	hello
indexof(concat(model.array),'10')	0
indexof(concat(model.array),'13')	-1

See Also

EndsWith Function *StartsWith Function* *Substring Function* *Matches Function* *IsMatch Function*

IsMatch Function

The ismatch() function is a *Regular Expression* function, and will return true if the first string parameter matches the regular expression pattern string second parameter.

Usage

```
{{ismatch(expr, pattern)}}
```

Parameters

Name	Type	Description
expr	`object`	Required. Any string, constant or expression (that will be evaluated and converted to a string).
pattern	`string`	Required. Any string, constant or expression (that will be evaluated and converted to a string). This should be a valid regular .net expression. If the expression cannot be understood then no value will be returned for the entire expression.

Returns

Type	Description
`boolean`	If true then there is a matching character set within the string based on the pattern.

Notes

The Regex use is based on the dot.net parser and compiler (rather than the javascript version). More information can be found here - https://learn.microsoft.com/en-us/dotnet/standard/base-types/regular-expressions

Examples

ismatch('Hello World', '^[A-Z]'))	True
ismatch('Hello World', '^[a-z]')	False
ismatch('Hello World', '\b\w{1,5}\b')	True
ismatch('Hello World', '\b\w{1,4}\b')	False

See Also

Matches Function | *Contains Function* | *Swap Function* | *TrimStart Function*

TrimEnd Function

String Functions

Join Function

The join() function will take any length of parameters and concatenate them into a single string inserting the first parameter after each of the following parameters except the last, returning the result. If one of the parameters is a collection or array, then each of the entries in that array will be appended to the string in order.

Usage

```
{{join(separator, expr, expr, expr, expr ...)}}
```

Parameters

Name	Type	Description
spearator	`string`	Required. Any string, constant or expression (that will be evaluated and converted to a string), that will be injected between each expr value.
expr	`object or array`	One or more. Any string, constant or expression (that will be evaluated and converted to a string). If the expr is an array then the contents will be expanded once and each entry concatenated.

Returns

Type	Description
`string`	The resultant concatenation of all the expression values

Notes

If one or more of the values is null, then the entry will be skipped and any further values processed

Examples

join(' ','hello','world!')	hello world!
join(',',model.array)	10,11,12
join(',',model.notset)	
join(' + ', '09',model.array, '13')	09 + 10 + 11 + 12 + 13
join(' + ', '09',model.notset, '13')	09 + 13
eval(join(' + ', '09',model.array, '13'))	55

Length Function

The length() function will return the total number of characters within the first string parameter.

Usage

```
{{length(expr)}}
```

Parameters

Name	Type	Description
expr	`object`	Required. Any string, constant or expression (that will be evaluated and converted to a string).

Returns

Type	Description
`integer`	The total number or individual characters in the string

Examples

length('hello world!')	12
length(join(',',model.array))	8
substring(join(',',model.array),0, length(join(',',model.array)) - 3)	10,11

See Also

Count Function *Substring Function* *Matches Function* *IsMatch Function*

Matches Function

The matches() function will return an array of all the strings in the first string parameter that match the regular expression string second parameter.

Usage

```
{{matches(expr, pattern)}}
```

Parameters

Name	Type	Description
expr	`object`	Required. Any string, constant or expression (that will be evaluated and converted to a string).
pattern	`string`	Required. Any string, constant or expression (that will be evaluated and converted to a string). This should be a valid .net regular expression in quotes. If the expression cannot be understood then no value will be returned for the entire expression.

Returns

Type	Description
`arrary`	All the matching strings in the expression, based on the pattern.

Notes

The Regex use is based on the dot.net parser and compiler (rather than the javascript version). More information ca be found here - https://learn.microsoft.com/en-us/dotnet/standard/base-types/regular-expressions

Examples

join(', ',matches('Hello World', '\b([A-Z][a-z]+)\b'))	Hello, World
join(', ',matches('Hello world!', '\b([A-Z][a-z]+)\b'))	Hello

See Also

IsMatch Function *Contains Function* *Split Function* *Swap Function* *Join Function*

PadLeft Function

The padleft() function will return the first string parameter padded at the left (start) of the string to the total length specified in the second parameter with the (first) character of the third parameter.

Usage

```
{{padleft(expr, length, paddingChar)}}
```

Parameters

Name	Type	Description
expr	`string`	Required. Any string, constant or expression (that will be evaluated and converted to a string).
length	`integer`	Required. Any number, constant or expression, that will be evaluated as an integer.
paddingChar	`string`	Required. Any string, constant or expression, that will be used to pad the left of the string to ensure it is always at least the required length number of characters.

Returns

Type	Description
`string`	The original expression with a minimum length and any required space prefixed by the padding character.

Notes

If the paddingChar parameter is a string with more that one character, only the first character is significant and will be used

Examples

padleft('hello world!',15, '-')	---hello world!
padleft('hello world!',10, '-')	hello world!
padleft('hello world!',15, '-a')	---hello world!
padleft(concat(model.array),model.number,'0')	000000000000000101112

See Also

PadRight Function

The padright() function will return the first string parameter padded at the right (end) of the string to the total length specified in the second parameter with the (first) character of the third parameter.

Usage

```
{{padright(expr, length, paddingChar)}}
```

Parameters

Name	Type	Description
expr	`string`	Required. Any string, constant or expression (that will be evaluated and converted to a string).
length	`integer`	Required. Any number, constant or expression, that will be evaluated as an integer.
paddingChar	`string`	Required. Any string, constant or expression, that will be used to pad the right of the string to ensure it is always at least the required length number of characters.

Returns

Type	Description
`string`	The original expression with a minimum length and any required space prefixed by the padding character.

Notes

If the paddingChar parameter is a string with more that one character, only the first character is significant and will be used

Examples

padright('hello world!',15, '-')	hello world!---
padright('hello world!',10, '-')	hello world!
padright('hello world!',15, '-a')	hello world!---
padright(concat(model.array),model.number,'0')	101112000000000000000

See Also

Replace Function

The replace() function will return the first string parameter where all occurances of the second string parameter are replaced with the third parameter. The search is case-sensitive.

Usage

```
{{replace(expr, searchString, replacement)}}
```

Parameters

Name	Type	Description
expr	`string`	Required. Any string, constant or expression (that will be evaluated and converted to a string).
searchString	`string`	Required. Any value, constant or expression, that will be converted to a string. Any occurance of this set of characters will be replaced by the value of the replacement
replacement	`string`	Required. Any string, constant or expression, that will be converted to a string, and used to replace the search string

Returns

Type	Description
`string`	The original expression with any occurances of the specifed search string, replaced. If the search string is not found, then the original string will be returned.

Examples

replace('hello world!','hello', 'hi')	hi world!
replace('hello world!','o', '0')	hell0 w0rld!
replace(concat(model.array),'1','2')	202222
replace(concat(model.array),'3','4')	101112

See Also

IsMatch Function *Matches Function* *Substring Function*

String Functions

Split Function

The split() function will return a collection of strings that have been separated based on the index of the second parameter.

Usage

```
{{split(expr, splitString)}}
```

Parameters

Name	Type	Description
expr	`string`	Required. Any string, constant or expression (that will be evaluated and converted to a string).
splitString	`string`	Required. Any value, constant or expression, that will be converted to a string. Any occurance of this set of characters will cause a new item in the returned result.

Returns

Type	Description
`array`	The original expression converted to an array of individual values chunked into an array of strings by the splitString. If no occurances were found an array with 1 item of the original string will be returned.

Examples

split('hello world!',' ')[0]	hello
split('hello world!',' ')[1]	world!
join(', ', split('hello world!','l'))	he, , o wor, d!

See Also

Join Function | *Matches Function* | *Replace Function* | *Substring Function*

StartsWith Function

The startsWith() function will return true if the second string parameter appears at the start of the first string parameter. If not then the function returns false. The search is case-sensitive.

Usage

```
{{startsWith(expr, comparison)}}
```

Parameters

Name	Type	Description
expr	`string`	Required. Any string, constant or expression (that will be evaluated and converted to a string).
comparison	`string`	Required. Any string, constant or expression (that will be evaluated and converted to a string).

Returns

Type	Description
`boolean`	True if the expr string starts with the comparison string

Examples

startswith('hello world!','world!')	True
startswith('hello world!','world')	False
startswith('hello world!','World!')	False
startswith(concat(model.array),'10')	True
startswith(concat(model.array),'12')	False

See Also

Contains Function *EndsWith Function* *Matches Function* *IsMatch Function*

Substring Function

The substring() function will return a modified version of the first string parameter starting at the index of the second parameter and optionally limited in length to the third integer parameter.

Usage

```
{{substring(expr, startIndex, length)}}
```

Parameters

Name	Type	Description
expr	`string`	Required. Any string, constant or expression (that will be evaluated and converted to a string).
startIndex	`integer`	Required. Any number, constant or expression, that will be evaluated as an integer and used as the zero-based starting character position of a substring.
length	`integer`	Optional. Any number, constant or expression, that will be evaluated as an integer and used for the number of returned characters in the substring.

Returns

Type	Description
`string`	The original expression starting from the required startIndex offset, optionally trucated to the specified length.

Notes

If the paddingChar parameter is a string with more that one character, only the first character is significant and will be used

Examples

substring('hello world!',6)	world!
substring('hello world!',6,5)	world
substring(concat(model.array),indexof(concat(model.array), '0'))	01112

See Also

PadRight Function *Substring Function* *Length Function*

Swap Function

The swap() function will replace *all* of the values in the first string parameter that match the regular expression string second parameter, with the value passed in the third parameter.

Usage

```
{{swap(expr, pattern, replacement)}}
```

Parameters

Name	Type	Description
expr	`object`	Required. Any string, constant or expression (that will be evaluated and converted to a string).
pattern	`string`	Required. Any string, constant or expression (that will be evaluated and converted to a string). This should be a valid .net regular expression in quotes. If the expression cannot be understood then no value will be returned for the entire expression.
replacement	`string`	Required. Any string, constant or expression (that will be evaluated and converted to a string). The value can be null or an empty string, but must be provided.

Returns

Type	Description
`string`	The original string with *all* the matching strings in the expression changed to the replacement, based on the pattern.

Notes

The Regex use is based on the dot.net parser and compiler (rather than the javascript version). More information ca be found here - https://learn.microsoft.com/en-us/dotnet/standard/base-types/regular-expressions

Examples

swap('Hello World', '\b([A-Z])', 'A')	Aello Aorld
swap('Hello world', '\b([A-Z][a-z]+)\b', 'Hi')	Hi world

See Also

IsMatch Function | *Contains Function* | *Matches Function* | *Replace Function*

Join Function

ToLower Function

The tolower() function will return the string parameter converting all characters to lowercase characters.

Usage

```
{{tolower(expr)}}
```

Parameters

Name	Type	Description
expr	`object`	Required. Any string, constant or expression (that will be evaluated and converted to a string).

Returns

Type	Description
`string`	The string where all UPPER case characters are converted to their lowercase equivalent.

Examples

toLower('Hello World!')	hello world!
tolower(join(', ', eachOf(model.items, .name)))	first item, second item, third item

See Also

Join Function *EachOf Function* *Matches Function* *ToUpper Function*

ToUpper Function

The toupper() function will return the string parameter converting all characters to uppercase (capital) characters.

Usage

```
{{toupper(expr)}}
```

Parameters

Name	Type	Description
expr	`object`	Required. Any string, constant or expression (that will be evaluated and converted to a string).

Returns

Type	Description
`string`	The string where all lower case characters are converted to their UPPERCASE equivalent.

Examples

toUpper('Hello World!')	HELLO WORLD!
toupper(join(', ', eachOf(model.items, .name)))	FIRST ITEM, SECOND ITEM, THIRD ITEM

See Also

Join Function *EachOf Function* *Matches Function* *ToLower Function*

Trim Function

The trim() function will return the first string removing any white space characters from the start and end of the value when converted to a string.

Usage

```
{{trim(expr)}}
```

Parameters

Name	Type	Description
expr	`object`	Required. Any string, constant or expression (that will be evaluated and converted to a string).

Returns

Type	Description
`string`	The string where all white space characters are removed from the start and the end.

Examples

trim(' Hello World ')	Hello World
concat('#',trim(' Hello World '),'#')	#Hello World#
concat('#',trim(' - Hello World - '),'#')	#- Hello World -#

See Also

PadLeft Function *PadRight Function* *Replace Function* *Matches Function*
TrimStart Function *TrimEnd Function*

TrimEnd Function

The trimend() function will return the first string parameter removing any white space characters from the right (end) of the string.

Usage

```
{{trimend(expr)}}
```

Parameters

Name	Type	Description
expr	`object`	Required. Any string, constant or expression (that will be evaluated and converted to a string).

Returns

Type	Description
`string`	The string where all white space characters are removed from the end of the string.

Examples

trimend(' Hello World ')	Hello World
concat('#',trimend(' Hello World '),'#')	# Hello World#
concat('#',trimend(' - Hello World - '),'#')	# - Hello World -#

See Also

PadLeft Function *PadRight Function* *Replace Function* *Matches Function*
TrimStart Function *Trim Function*

TrimStart Function

The trimstart() function will return the first string parameter removing any white space characters from the left (start) of the string.

Usage

```
{{trimstart(expr)}}
```

Parameters

Name	Type	Description
expr	`object`	Required. Any string, constant or expression (that will be evaluated and converted to a string).

Returns

Type	Description
`string`	The string where all white space characters are removed from the start.

Examples

trimstart(' Hello World ')	Hello World
concat('#',trimstart(' Hello World '),'#')	#Hello World #
concat('#',trimstart(' - Hello World - '),'#')	#- Hello World - #

See Also

PadLeft Function *PadRight Function* *Replace Function* *Matches Function*
TrimEnd Function *Trim Function*

Date Functions

The date functions works on (Gregorian) DateTime values, not strings. DateTime values can be converted from strings, or simply created with the 'date()' function.

When working with the current date and tim
value in a variable, or pass it to the docum
through as time increases and the docume

The following Date Functions are supported

AddDays Function

Adds the specified number of days in the second parameter (either positive or negative), to the date value in the first parameter, returning the result

Usage

```
{{addDays(expr, count)}}
```

Parameters

Name	Type	Description
expr	`datetime`	Required. Any date value, constant or expression (that will be evaluated and converted to a DateTime).
count	`number`	The number of whole days, or partial days to add to the expr value. Where 0.5 days is 12hrs, etc.

Returns

Type	Description
`datetime`	The expr value, with the amount of days and time added to the value. This will also take into account the number of days in the month and any leap years.

Notes

If the date expression is not a DateTime, then a conversion attempt will be made. However, explicit conversion before execution is a better option.

Examples

string(date(), 'dd MMM yyyy HH:mm:ss')	15 May 2025 09:35:47
string(addDays(date(), 31), 'dd MMM yyyy')	15 Jun 2025
string(addDays(date(), 365*4), 'dd MMM yyyy')	14 May 2029
string(addDays(date(), 0.5), 'dd MMM yyyy HH:mm:ss')	15 May 2025 21:35:47
string(addDays(date(), model.int + model.number), 'dd MMM yyyy HH:mm:ss')	16 Jun 2025 07:11:47
string(addDays(date(), model.notset), 'dd MMM yyyy HH:mm:ss')	

AddHours Function

Adds the specified number of hours in the second parameter (either positive or negative), to the date value in the first parameter, returning the result. If the expr or the count are null then null will be returned.

Usage

```
{{addHours(expr, count)}}
```

Parameters

Name	Type	Description
expr	`datetime`	Required. Any date value, constant or expression (that will be evaluated and converted to a DateTime).
count	`number`	The number of whole hours, or partial hours to add to the expr value. Where 0.5 hours is 30 minutes, etc.

Returns

Type	Description
`datetime`	The expr value, with the amount of time added to the value. This will also take into account the number of days in the month and any leap years.

Notes

If the date expression is not a DateTime, then a conversion attempt will be made. However, explicit conversion before execution is a better option.

Examples

string(date(), 'dd MMM yyyy HH:mm:ss')	15 May 2025 09:35:47
string(addHours(date(), 24), 'dd MMM yyyy')	16 May 2025
string(addHours(date(), -10), 'dd MMM yyyy HH:mm:ss')	14 May 2025 23:35:47
string(addHours(date(), 31 * 24), 'dd MMM yyyy')	15 Jun 2025
string(addHours(date(), 0.5), 'dd MMM yyyy HH:mm:ss')	15 May 2025 10:05:47
string(addHours(date(), model.int * model.number), 'dd MMM yyyy HH:mm:ss')	24 May 2025 23:29:47
string(addHours(date(), model.notset), 'dd MMM yyyy HH:mm:ss')	

AddMilliseconds Function

Adds the specified number of milli-seconds in the second parameter (either positive or negative) to the date value in the first parameter, returning the result

Usage

```
{{addMilliseconds(expr, count)}}
```

Parameters

Name	Type	Description
expr	`datetime`	Required. Any date value, constant or expression (that will be evaluated and converted to a DateTime).
count	`number`	The number of whole milli-seconds, or partial milli-seconds to add to the expr value, to a double precision for .net 8 (rounded for .net Standard 2.0)

Returns

Type	Description
`datetime`	The expr value, with the amount of time added to the value.

Notes

Conversion of the current date with added milli-seconds may result in inaccuracies. Store the current sate in a variable or field to be used during execution.
If the expression is not a DateTime, then a conversion attempt will be made, however, explicit conversion before execution is a better option.

Examples

string(date(), 'dd MMM yyyy HH:mm:ss.fff')	15 May 2025 09:35:47.162
string(addMilliseconds(date(), 1), 'HH:mm:ss.fff')	09:35:47.166
string(addMilliseconds(date(), 2 * 1000), 'HH:mm:ss.fff')	09:35:49.167
string(addMilliseconds(date(), 0.5), 'HH:mm:ss.fff')	09:35:47.169
string(addMilliseconds(date(), model.int * model.number), 'HH:mm:ss.fff')	09:35:47.400
string(addMilliseconds(date(), model.notset), 'HH:mm:ss.fff')	

AddMinutes Function

Adds the specified number of minutes in the second parameter (either positive or negative) to the date value in the first parameter, returning the result

Usage

```
{{addMinutes(expr, count)}}
```

Parameters

Name	Type	Description
expr	`datetime`	Required. Any date value, constant or expression (that will be evaluated and converted to a DateTime).
count	`number`	The number of whole minutes, and/or partial minutes to add to the expr value. Where 0.5 minutes is 30 seconds, etc.

Returns

Type	Description
`datetime`	The expr value, with the amount of time added to the value. This will also take into account the number of days in the month and any leap years.

Notes

If the date expression is not a DateTime, then a conversion attempt will be made. However, explicit conversion before execution is a better option.

Examples

string(date(), 'dd MMM yyyy HH:mm:ss')	15 May 2025 09:35:47
string(addMinutes(date(), 30), 'dd MMM yyyy HH:mm:ss')	15 May 2025 10:05:47
string(addMinutes(date(), -10), 'dd MMM yyyy HH:mm:ss')	15 May 2025 09:25:47
string(addMinutes(date(), 24 * 60), 'dd MMM yyyy HH:mm:ss')	16 May 2025 09:35:47
string(addMinutes(date(), 0.5), 'dd MMM yyyy HH:mm:ss')	15 May 2025 09:36:17
string(addMinutes(date(), model.int * model.number), 'dd MMM yyyy HH:mm:ss')	15 May 2025 13:25:41
string(addMinutes(date(), model.notset), 'dd MMM yyyy HH:mm:ss')	

AddMonths Function

Adds the specified number of months in the second parameter (either positive or negative), to the date value in the first parameter, returning the result

Usage

```
{{addMonths(expr, count)}}
```

Parameters

Name	Type	Description
expr	`datetime`	Required. Any date value, constant or expression (that will be evaluated and converted to a DateTime).
count	`integer`	The number of *whole* months to add to the expr value. Partial months are not supported, and any real number will be rounded to it's nearest whole number.

Returns

Type	Description
`datetime`	The expr value, with the amount of months and time added to the value.

Notes

If the date expression is not a DateTime, then a conversion attempt will be made. However, explicit conversion before execution is a better option.

Examples

string(date(), 'dd MMM yyyy HH:mm:ss')	15 May 2025 09:35:47
string(addMonths(date(), 3), 'dd MMM yyyy')	15 Aug 2025
string(addMonths(date(), -3*4), 'dd MMM yyyy')	15 May 2024
string(addMonths(date(), 20.9), 'dd MMM yyyy HH:mm:ss')	15 Feb 2027 09:35:47
string(addMonths(date(), model.int * 5), 'dd MMM yyyy HH:mm:ss')	15 Dec 2029 09:35:47
string(addMonths(date(), model.notset), 'dd MMM yyyy HH:mm:ss')	

AddSeconds Function

Adds the specified number of seconds in the second parameter (either positive or negative to the date value in the first parameter, returning the result

Usage

```
{{addMinutes(expr, count)}}
```

Parameters

Name	Type	Description
expr	`datetime`	Required. Any date value, constant or expression (that will be evaluated and converted to a DateTime).
count	`number`	The number of whole seconds, and/or partial seconds to add to the expr value. Where 0.5 seconds is 500 milli-seconds, etc.

Returns

Type	Description
`datetime`	The expr value, with the amount of time added to the value. This will also take into account the number of days in the month and any leap years.

Notes

If the date expression is not a DateTime, then a conversion attempt will be made. However, explicit conversion before execution is a better option.

Examples

string(date(), 'dd MMM yyyy HH:mm:ss')	15 May 2025 09:35:47
string(addSeconds(date(), 30), 'HH:mm:ss')	09:36:17
string(addSeconds(date(), -10), 'HH:mm:ss')	09:35:37
string(addSeconds(date(), 30 * 60), 'HH:mm:ss')	10:05:47
string(addSeconds(date(), 24 * 60 * 60), 'dd MMM yyyy HH:mm:ss')	16 May 2025 09:35:47
string(addSeconds(date(), 0.5), 'HH:mm:ss.fff')	09:35:47.784
string(addSeconds(date(), model.int * model.number), 'HH:mm:ss')	09:39:37
string(addSeconds(date(), model.notset), 'dd MMM yyyy HH:mm:ss')	

AddYears Function

Adds the specified discreet number of years in the second parameter (either positive or negative), to the date value in the first parameter, returning the result

Usage

```
{{addDays(expr, count)}}
```

Parameters

Name	Type	Description
expr	`datetime`	Required. Any date value, constant or expression (that will be evaluated and converted to a DateTime).
count	`integer`	The number of *whole* years to add to the expr value. Partial years are not supported, and any real number will be rounded to it's nearest whole number.

Returns

Type	Description
`datetime`	The expr value, with the amount of years added to the value.

Notes

If the date expression is not a DateTime, then a conversion attempt will be made. However, explicit conversion before execution is a better option.

Examples

string(date(), 'dd MMM yyyy HH:mm:ss')	15 May 2025 09:35:47
string(addYears(date(), 1), 'dd MMM yyyy HH:mm:ss')	15 May 2026 09:35:47
string(addYears(date(), -10), 'dd MMM yyyy HH:mm:ss')	15 May 2015 09:35:47
string(addYears(date(), 20.9), 'dd MMM yyyy HH:mm:ss')	15 May 2046 09:35:47
string(addYears(date(), model.int * 5), 'dd MMM yyyy HH:mm:ss')	15 May 2080 09:35:47
string(addYears(date(), model.int + model.number), 'dd MMM yyyy HH:mm:ss')	15 May 2057 09:35:47
string(addDays(date(), model.notset), 'dd MMM yyyy HH:mm:ss')	

DayOfMonth Function

Returns the day of the month of the date expr parameter. The first day of the month is 1, not zero.

Usage

```
{{dayOfMonth(expr)}}
```

Parameters

Name	Type	Description
expr	`datetime`	Required. Any date value, constant or expression (that will be evaluated and converted to a DateTime).

Returns

Type	Description
`integer`	The day of the month. This is non-zero based, so the first day of the month will be 1, not zero (0).

Notes

If the date expression is not a DateTime, then a conversion attempt will be made. However, explicit conversion before execution is a better option.

Examples

string(date(), 'dd MMM yyyy HH:mm:ss')	15 May 2025 09:35:47
string(dayofmonth(date()))	15
dayofmonth(addDays(date(), 20))	4
dayofmonth(addDays(date(), -20))	25
dayofmonth(addDays(date(), 20)) + model.int	15

DayOfWeek Function

Returns the day of the week (0 - 6) of the date expr parameter

Usage

```
{{dayOfWeek(expr)}}
```

Parameters

Name	Type	Description
expr	`datetime`	Required. Any date value, constant or expression (that will be evaluated and converted to a DateTime).

Returns

Type	Description
`integer`	The day of the week. This is zero based from Sunday, Monday = 1, Tuesday = 2, etc.

Notes

If the date expression is not a DateTime, then a conversion attempt will be made. However, explicit conversion before execution is a better option.

Examples

string(date(), 'dd MMM yyyy HH:mm:ss')	15 May 2025 09:35:47
dayofweek(date())	4
dayofweek(date('2024-11-03', 'yyyy-MM-dd')	0
dayofweek(addDays(date(), 20))	3
model.days[dayofweek(date())]	thur
dayofweek(date(model.date, 'dd MMMM yyyy HH:mm:ss'))	1

DayOfYear Function

Returns the total number of days that have passed in the current year for the date expr parameter

Usage

```
{{dayOfYear(expr)}}
```

Parameters

Name	Type	Description
expr	`datetime`	Required. Any date value, constant or expression (that will be evaluated and converted to a DateTime).

Returns

Type	Description
`integer`	The total number of days. As this is non-zero based, the first of Jauary is day one (1).

Notes

If the date expression is not a DateTime, then a conversion attempt will be made. However, explicit conversion before execution is a better option.

Examples

string(date(), 'dd MMM yyyy HH:mm:ss')	15 May 2025 09:35:47
dayOfYear(date())	135
dayOfYear(addDays(date(),300))	70
dayofyear(date('2024-01-01', 'yyyy-MM-dd')	1
dayofyear(date('2024-12-31', 'yyyy-MM-dd')	366
dayofyear(date('2025-12-31', 'yyyy-MM-dd')	365
dayofyear(date(model.date, 'dd MMMM yyyy HH:mm:ss'))	163

DaysBetween Function

Returns the total number of days (inc partial days) that have passed between the two provided dates

Usage

```
{{daysBetween(startdate, enddate)}}
```

Parameters

Name	Type	Description
startdate	`datetime`	Required. Any date value, constant or expression (that will be evaluated and converted to a DateTime).
enddate	`datetime`	Required. Any date value, constant or expression (that will be evaluated and converted to a DateTime).

Returns

Type	Description
`double`	The total number of days and partial days between the startdate and enddate.

Notes

If the dates are not passed as a DateTime, then a conversion attempt will be made. However, explicit conversion before execution is a better option.

Examples

string(date(), 'dd MMM yyyy HH:mm:ss')	15 May 2025 09:35:47
daysBetween(date('2024-01-01', 'yyyy-MM-dd'), date())	500.3998548264
floor(daysBetween(date('2024-01-01', 'yyyy-MM-dd'), date()))	500
daysBetween(date('24-01-01', 'yy-MM-dd'), date('24-01-02', 'yy-MM-dd'))	1
daysBetween(date(model.date, 'dd MMMM yyyy hh:mm:ss'), date())	702.9102715741

HourOf Function

Returns the discreet number of hours passed for the date expr parameter

Usage

```
{{hourof(expr)}}
```

Parameters

Name	Type	Description
expr	`datetime`	Required. Any date value, constant or expression (that will be evaluated and converted to a DateTime).

Returns

Type	Description
`integer`	The total number hours passed in the date tme of day (0 to 23).

Notes

If the dates are not passed as a DateTime, then a conversion attempt will be made. However, explicit conversion before execution is a better option.

Examples

string(date(), 'dd MMM yyyy HH:mm:ss')	15 May 2025 09:35:47
hourof(date())	9
hourof(date('2024-01-01', 'yyyy-MM-dd'))	0
hourOf(addSeconds(date('24-01-01', 'yy-MM-dd'),-1))	23
hourof(date(model.date, 'dd MMMM yyyy hh:mm:ss'))	11

HoursBetween Function

Returns the total number of hours (inc partial hours) that have passed between the two provided dates

Usage

```
{{hoursBetween(startdate, enddate)}}
```

Parameters

Name	Type	Description
startdate	`datetime`	Required. Any date value, constant or expression (that will be evaluated and converted to a DateTime).
enddate	`datetime`	Required. Any date value, constant or expression (that will be evaluated and converted to a DateTime).

Returns

Type	Description
`double`	The total number of hours and partial hours between the startdate and enddate, where 30 minutes = 0.5.

Notes

If the dates are not passed as a DateTime, then a conversion attempt will be made. However, explicit conversion before execution is a better option.

Examples

string(date(), 'dd MMM yyyy HH:mm:ss')	15 May 2025 09:35:47
hoursBetween(date('2024-01-01', 'yyyy-MM-dd'), date())	12009.5965347222
floor(hoursBetween(date('2024-01-01', 'yyyy-MM-dd'), date()))	12009
hoursBetween(date('24-01-01', 'yy-MM-dd'), date('24-01-02', 'yy-MM-dd'))	24
hoursBetween(date(model.date, 'dd MMMM yyyy hh:mm:ss'), date())	16869.8465363889

MillisecondOf Function

Returns the discreet number of milliseconds passed in the current second for the date expr parameter

Usage

```
{{millisecondof(expr)}}
```

Parameters

Name	Type	Description
expr	`datetime`	Required. Any date value, constant or expression (that will be evaluated and converted to a DateTime).

Returns

Type	Description
`integer`	The number of milliseconds in the second, for the date time of day (0 to 999).

Notes

Conversion of the current date with added milli-seconds may result in inaccuracies. Store the current sate in a variable or field to be used during execution.
If the dates are not passed as a DateTime, then a conversion attempt will be made. However, explicit conversion before execution is a better option.

Examples

string(date(), 'dd MMM yyyy HH:mm:ss')	15 May 2025 09:35:47
millisecondof(date())	558
millisecondof(date('2024-01-01', 'yyyy-MM-dd'))	0
millisecondof(addSeconds(date('24-01-01', 'yy-MM-dd'),0.5))	500
millisecondof(date(model.date, 'dd MMMM yyyy hh:mm:ss'))	0
millisecondof(date())	565

MillisecondsBetween Function

Returns the total number of hours (inc partial hours) that have passed between the two provided dates

Usage

```
{{millisecondsBetween(startdate, enddate)}}
```

Parameters

Name	Type	Description
startdate	`datetime`	Required. Any date value, constant or expression (that will be evaluated and converted to a DateTime).
enddate	`datetime`	Required. Any date value, constant or expression (that will be evaluated and converted to a DateTime).

Returns

Type	Description
`double`	The total number of milli-seconds and partial milli-seconds between the startdate and enddate.

Notes

Conversion of the current date with added milli-seconds may result in inaccuracies. Store the current sate in a variable or field to be used during execution.
If the dates are not passed as a DateTime, then a conversion attempt will be made. However, explicit conversion before execution is a better option.

Examples

string(date(), 'dd MMM yyyy HH:mm:ss')	15 May 2025 09:35:47
millisecondsBetween(addHours(date(),1.5), date())	-5400000
floor(millisecondsBetween(date('2024-01-01', 'yyyy-MM-dd'), date('2024-01-02', 'yyyy-MM-dd')))	86400000

MonthOf Function

Returns the current month number passed for the date expr parameter - non-zero based

Usage

```
{{monthof(expr)}}
```

Parameters

Name	Type	Description
expr	`datetime`	Required. Any date value, constant or expression (that will be evaluated and converted to a DateTime).

Returns

Type	Description
`integer`	The current month number in the current year for the date (1 to 12).

Notes

If the dates are not passed as a DateTime, then a conversion attempt will be made. However, explicit conversion before execution is a better option.

Examples

string(date(), 'dd MMM yyyy HH:mm:ss')	15 May 2025 09:35:47
monthof(date())	
monthof(date('2024-01-01', 'yyyy-MM-dd'))	
monthof(date('2024-12-01', 'yyyy-MM-dd'))	
monthof(date(model.date, 'dd MMMM yyyy hh:mm:ss'))	

SecondOf Function

Returns the discreet number of seconds passed in the current minute for the date expr parameter

Usage

```
{{secondOf(expr)}}
```

Parameters

Name	Type	Description
expr	`datetime`	Required. Any date value, constant or expression (that will be evaluated and converted to a DateTime).

Returns

Type	Description
`integer`	The number of seconds in the minute, for the date time of day (0 to 59).

Notes

Conversion of the current date with added seconds may, potentially, result in inaccuracies. Store the current state in a variable or field to be used during execution.
If the dates are not passed as a DateTime, then a conversion attempt will be made. However, explicit conversion before execution is a better option.

Examples

string(date(), 'dd MMM yyyy HH:mm:ss')	15 May 2025 09:35:47
secondof(date())	47
secondof(date('2024-01-01', 'yyyy-MM-dd'))	0
secondof(addMinutes(date('24-01-01', 'yy-MM-dd'),0.5))	30
secondof(date(model.date, 'dd MMMM yyyy hh:mm:ss'))	0
secondof(date())	47

SecondsBetween Function

Returns the total number of seconds (inc partial seconds) that have passed between the two provided dates

Usage

```
{{secondsBetween(startdate, enddate)}}
```

Parameters

Name	Type	Description
startdate	`datetime`	Required. Any date value, constant or expression (that will be evaluated and converted to a DateTime).
enddate	`datetime`	Required. Any date value, constant or expression (that will be evaluated and converted to a DateTime).

Returns

Type	Description
`double`	The total number of seconds and partial seconds between the startdate and enddate.

Notes

Conversion of the current date with added seconds may result in inaccuracies. Store the current sate in a variable or field to be used during execution.
If the dates are not passed as a DateTime, then a conversion attempt will be made. However, explicit conversion before execution is a better option.

Examples

string(date(), 'dd MMM yyyy HH:mm:ss')	15 May 2025 09:35:47
secondsBetween(addHours(date(),1.5), date())	-5400
secondsBetween(date('2024-01-01', 'yyyy-MM-dd'), date('2024-01-02', 'yyyy-MM-dd'))	86400

YearOf Function

Returns the current year for the date expr parameter - non-zero based

Usage

```
{{monthof(expr)}}
```

Parameters

Name	Type	Description
expr	`datetime`	Required. Any date value, constant or expression (that will be evaluated and converted to a DateTime).

Returns

Type	Description
`integer`	The current year for the date (e.g. 2024).

Notes

If the dates are not passed as a DateTime, then a conversion attempt will be made. However, explicit conversion before execution is a better option.

Examples

string(date(), 'dd MMM yyyy HH:mm:ss')	15 May 2025 09:35:47
yearof(date())	2025
yearof(date('2024-01-01', 'yyyy-MM-dd'))	2024
yearof(date('2025-12-01', 'yyyy-MM-dd'))	2025
yearof(date(model.date, 'dd MMMM yyyy hh:mm:ss'))	2023

Logical Functions

The following Logical Functions are supported

If Function

Checks the first parameter and if the result is true, then returns the second parameter, otherwise the third parameter is evaluated and returned.

Usage

```
{{if(expr, truevalue, falsevalue)}}
```

Parameters

Name	Type	Description
expr	`object`	Required. Any object, constant or expression that will be evaluated and if successful will be returned.
truevalue	`object`	Required. Any object, constant or expression that will be evaluated and returned if the first parameter is equivalent to true.
falsevalue	`object`	Required. Any object, constant or expression that will be evaluated and returned if the first parameter is equivalent to false.

Returns

Type	Description
`object`	Either the truevalue, or the falsevalue. No matching type checking or validation is done.

Notes

If the first expression is, or results in null or 0, then is is considered false.

Examples

if(0, 'is true', 'is false')	is false
if(1, 'is true', 'is false')	is true
if(model.number > 20, model.number, 20)	20.9
if(count(model.array) >= 4, model.array[3], model.array[2])	12
if(model.notset, model.notset, model.number)	20.9

IfError Function

Checks the first parameter and if the evaluated result does not cause an error, then it will be returned, otherwise the second parameter is evaluated and returned.

Usage

```
{{ifError(expr, fallback)}}
```

Parameters

Name	Type	Description
expr	`object`	Required. Any object, constant or expression that will be evaluated and if successful will be returned.
fallback	`object`	Required. Any object, constant or expression that will be evaluated and returned if the first parameter fails.

Returns

Type	Description
`object`	Either the expr value, or the fallback value. No matching type checking or validation is done.

Notes

If the first expression is, or results in null, then null will be returned, no error - this includes property accessors that are not present. However index out of bounds for array and keys not present for dictionaries will result in the error fallback.
The if error function does not wrap around invalid expression structure - this is considered a FATAL error with the template. But it will wrap around any errors for the eval function execution.

Examples

iferror(date(), 'no date')	05/15/2025 09:35:47
iferror(date('32 13 2024'), 'not date')	not date
iferror(substring('32 13 2024', 5, 10), 'too short')	too short
iferror(model.notset, 'null is valid')	
iferror(model.notset, 'null is valid') ?? 'null replacement'	null replacement
iferror(model['notset'], 'key not present')	key not present
iferror(model.array[4], 'out of bounds')	out of bounds
iferror(model['notset'], model['number'])	20.9
iferror(eval('date(('), 'invalid expression')	invalid expression

Logical Functions

In Function

The in function takes the first parameter, and checks any following parameters to see if they are equal to the first. If so then it returns true, otherwise false. If one of the following parameters is an collection or array, then it will expand the contents and check each individual item.

Usage

```
{{in(compareTo, item1 [,item2,item3, ...])}}
```

Parameters

Name	Type	Description
compareTo	`object`	Required. Any object, constant or expression that will be compared to each of the following items.
item1 etc.	`object`	Required. Any object, constant or expression that will be compared against. If it is an array then each of the contents will be checked to find if the and returned if the first parameter fails.

Returns

Type	Description
`boolean`	Either true if the value of compareTo was found in the following parameters, otherwise false.

Notes

When one or more of the items is a collection, then the inner items will be checked. However, if one of the inner items is an array, this will not be expanded, only compared directly.

Individual characters will not be searched for within a string - use the conta

Examples

in(12, 10, 11, 12, 13)	True
in(14, 10, 11, 12, 13)	False
in('sun', model.days)	True
in(12, model.array)	True
in(14, model.array)	False
in('12', model.days,'other', model.array)	True

Aggregate Functions

The following Aggregate Functions are supported

Aggregate Functions

Average Function

The average function (synonyous with mean) will return the sum of the arguments passed to the function, divided by the number of arguments. If one of those arguments is a collection or array (or a function that returns an array), then each item in the array will be included in the calculation.

Usage

```
{{average(item1 [,item2,item3, ...])}}
```

Parameters

Name	Type	Description
item1 etc...	`number or array of numbers`	Required, one or more. Any object, constant or expression that will return in a number or an array of numbers.

Returns

Type	Description
`number`	The total sum of the numbers divided by the count of numbers.

Notes

When one or more of the items is a collection, then the inner items will be checked. However, if one of the inner items is an array, this will not be expanded and the expression will fail as a whole.

Examples

average(10, 11, 12, 13.4)	11.6
average(model.array)	11
average(model.days)	
average(9, model.array, 13, 101)	26

AverageOf Function

The averageOf function will return the sum of evaluation of the second argument in the context of the current item from the first array or collection argument.

Usage

```
{{averageof(collection, valueExpr)}}
```

Parameters

Name	Type	Description
collection.	`array`	Required. Any object, constant or expression that will return an array of values.
valueExpr.	`number`	Required. Any object, constant or expression that will return a number (or null) based on the current item in the collection.

Returns

Type	Description
`number`	The total sum of the extracted numbers divided by the count of items in collection.

Notes

When one or more of the items is a collection, then the inner items will be added. If the value is null (including inside the array) then it will be treated as zero.

Examples

averageof(model.items, .index)	2
averageof(model.array, .)	11
averageof(model.days, length(.))	3.2857142857
averageof(collect(9,model.array,13, 101), if(. < 100, ., null))	9.1666666667

Aggregate Functions

Count Function

The count function will return the number of non-null arguments passed to the function. If one of those arguments is a collection or array (or a function that returns an array), then it will add the non-null array length to the count.

Usage

```
{{count(item1 [,item2,item3, ...])}}
```

Parameters

Name	Type	Description
item1 etc...	`object or array`	Required, one or more. Any object, constant or expression that will return a value or an array of values.

Returns

Type	Description
`number`	The total number of values extracted in the parameters.

Notes

*When one or more of the items is a collection, then the inner items will be added. If the inner item itself is an array, then it will be treated as *one*.*

*If the value is null (including inside the array) then it will *NOT* be counted.*

Examples

count(10, 11, 12, 13.4)	4
count(10, 11, null, 12, 13.4)	0
count(model.items)	3
count('one', model.array, model.days, null)	11

Max Function

The max function will return the largest value based on the non-null arguments passed to the function. If one of those arguments is a collection or array (or a function that returns an array), then it will check the non-null array values for the maximum.

Usage

```
{{max(item1 [,item2,item3, ...])}}
```

Parameters

Name	Type	Description
item1 etc...	`object or array`	Required, one or more. Any object, constant or expression that will return a value or an array of values.

Returns

Type	Description
`object`	The maximum value of the extracted in the parameters.

Notes

If one or more of the values is a string, then the comparison will revert to checking as a group of strings. If conversion of the value to a number from a string is needed, then use the maxof function.

Examples

max(10, 11, 12, 13.4)	13.4
max(10, 13.4, null, 12, 11)	13.4
max(model.array)	12
max('one', model.array, model.days, null)	wed

MaxOf Function

The maxof function will return the largest value based on the second parameter in the context of the current item from the array of collection in the first parameter.

Usage

```
{{maxof(colleciton, valueExpr)}}
```

Parameters

Name	Type	Description
collection	`array`	Required. Any object, constant or expression that will return an array of values.
valueExpr	`expression`	Required. Any current data expression that will return the value to compare to.

Returns

Type	Description
`object`	The maximum value of all the non-null extracted values in the collection based on the expression.

Notes

The value returned from the second expression parameter should be comparable (e.g. integer or double or date or string), null will be ignored and skipped.

Examples

maxof(model.items, .index)	3
maxof(model.items, .name)	Third Item
maxof(collect('one', 'two', 'three', 'four'), length(.))	5
maxof(collect(9,model.array,13, 101), if(. < 100, ., null))	13

Aggregate Functions

Mean Function

The mean function (synonyous with average, but explicit in the mechanism is being used), will return the sum of the arguments passed to the function, divided by the number of arguments. If one of those arguments is a collection or array (or a function that returns an array), then each item in the array will be included in the calculation.

Usage

```
{{mean(item1 [,item2,item3, ...])}}
```

Parameters

Name	Type	Description
item1 etc...	`number or array of numbers`	Required, one or more. Any object, constant or expression that will return in a number or an array of numbers.

Returns

Type	Description
`number`	The total sum of the numbers divided by the count of numbers.

Notes

When one or more of the items is a collection, then the inner items will be checked. However, if one of the inner items is an array, this will not be expanded, and the expression will fail as a whole.

Examples

mean(10, 11, 12, 13.4)	11.6
mean(model.array)	11
mean(model.days)	
mean(9, model.array, 13, 101)	26

Median Function

The median function, will return the middle value of all the arguments passed to the function. If one of those arguments is a collection or array (or a function that returns an array), then each item in the array will be included in the calculation.

Usage

```
{{median(item1 [,item2,item3, ...])}}
```

Parameters

Name	Type	Description
item1 etc...	`number or array of numbers`	Required, one or more. Any object, constant or expression that will return in a number or an array of numbers.

Returns

Type	Description
`number`	The middle value out of all of numbers collected and sorted.

Notes

The values should all be numeric

Examples

median(10, 11, 12, 13.4)	11.5
median(model.array)	11
median(model.days)	
median(9, model.array, 13, 101)	11.5

Min Function

The min function will return the smallest value based on the non-null arguments passed to the function. If one of those arguments is a collection or array (or an expression that returns an array), then it will check the non-null array values for the minimum.

Usage

```
{{min(item1 [,item2,item3, ...])}}
```

Parameters

Name	Type	Description
item1 etc...	`object or array`	Required, one or more. Any object, constant or expression that will return a value or an array of values.

Returns

Type	Description
`object`	The minimum value of the extracted in the parameters.

Notes

If one or more of the values is a string, then the comparison will revert to checking as a group of strings. If conversion of the value to a number from a string is needed, then use the maxof function.

Examples

min(10, 11, 12, 13.4)	10
min(10, 13.4, null, 12, 1)	1
min(model.array)	10
min('!', model.array, model.days, null)	!

MinOf Function

The minof function will return the smallest value based on the second parameter in the context of the current item from the array of collection in the first parameter.

Usage

```
{{minof(collection, valueExpr)}}
```

Parameters

Name	Type	Description
collection	`array`	Required. Any object, constant or expression that will return an array of values.
valueExpr	`expression`	Required. Any current data expression that will return the value to compare to.

Returns

Type	Description
`object`	The minimum value of all the non-null extracted values in the collection based on the expression.

Notes

The value returned from the second expression parameter should be comparable (e.g. integer or double or date or string), null will be ignored and skipped.

Examples

minof(model.items, .index)	1
minof(model.items, .name)	First Item
minof(collect('one', 'two', 'three', 'four'), length(.))	3
minof(collect(9,model.array,13, -10), if(. > 0, ., null))	9

Mode Function

The mode function, will return the number value that occurs most often in all the arguments passed to the function. If one of those arguments is a collection or array (or a function that returns an array), then each item in the array will be included in the calculation.

Usage

```
{{mode(item1 [,item2,item3, ...])}}
```

Parameters

Name	Type	Description
item1 etc...	`number or array of numbers`	Required, one or more. Any object, constant or expression that will return in a number or an array of numbers.

Returns

Type	Description
`number`	The most occurant value out of all of numbers collected and sorted.

Notes

The values should all be numeric. When all values are unique, or there are multiple values with the same occurance the first will be returned

Examples

mode(10, 11, 12, 11)	11
mode(10, 11, 12, 11)	11
mode(10, 110, 12, 110)	110
mode(model.array)	10
mode(model.days)	
mode(model.array, 13, 11)	11

Sum function

The sum function will return the total accumulated value based on the non-null arguments passed to the function. If one of those arguments is a collection or array (or an expression that returns an array), then it will include the sum of the non-null array values.

Usage

```
{{sum(item1 [,item2,item3, ...])}}
```

Parameters

Name	Type	Description
item1 etc...	`object or array`	Required, one or more. Any object, constant or expression that will return a value or an array of values.

Returns

Type	Description
`number`	The total summed value of the extracted in the parameters.

Notes

The individual values should be numeric, i.e. indeger or real, (or null) so the total can be calculated. If the value cannot be added then the expression will fail as a whole.

Examples

sum(10, 11, 12, 13.4)	46.4
sum(10, 13.4, null, 12, 1)	36.4
sum(model.array)	33
sum(9, model.array, null, 13.6)	55.6

SumOf function

The sumof function will return the total value based on the second parameter in the context of the current item from the array of collection in the first parameter.

Usage

```
{{sumof(collection, valueExpr)}}
```

Parameters

Name	Type	Description
collection	`array`	Required. Any object, constant or expression that will return an array of values.
valueExpr	`expression`	Required. Any current data expression that will return the value to add to.

Returns

Type	Description
`number`	The minimum value of all the non-null extracted values in the collection based on the expression.

Notes

The value returned from the second expression parameter should be a number (e.g. integer or double), null will be ignored and skipped.

Examples

sumof(model.array, .)	33
sumof(model.items, .index)	6
sumof(collect('one', 'two', 'three', null, 'four'), length(.))	15
sumof(model.items, .name)	

Collection Functions

The following Collection Functions are supported

Collection Functions

Collect Function

The collect function will return all the arguments passed to the function as a single collection / array, in the order they are added. If one of those arguments is a collection or array (or a function that returns an array), then it will add the items in the array to the result.

Usage

```
{{collect(item1 [,item2,item3, ...])}}
```

Parameters

Name	Type	Description
item1 etc...	`object or array`	One or more. Any objects, constantc or expressions that will return a value, or an array of values.

Returns

Type	Description
`array`	The complete collection of the entries in the parameters as a single array.

Notes

No validation of type of structure to be added is performed, the returned array can contain any type of content, at any index

Examples

join('; ',collect(10, 11, 12, 13.4))	10; 11; 12; 13.4
join('; ',collect(model.array, 13.4))	10; 11; 12; 13.4
join('; ',collect(model.days, model.items, count(model.array)))	sun; mon; tues; wed; thur; fri; sat; {"name": "First Item", "index" : 1}; {"name": "Second Item", "index": 3}; {"name": "Third Item", "index": 2}; 3
sum(collect(model.array, eachof(model.items, .index)))	39

EachOf Function

The eachof function will return the an array of all the values based on the second parameter in the context of the current item from the array or collection in the first parameter.

Usage

```
{{eachof(collection, valueExpr)}}
```

Parameters

Name	Type	Description
collection	`array`	Required. Any objects, constant or expressions that will return an array of values.
valueExpr	`expression`	Required. Any current data expression that will return the value to add to the resultant array.

Returns

Type	Description
`array`	The complete collection of the entries returned from the valueExpre in the parameters as a single array.

Notes

nulls are ignored in this case. If the returned value is null, then it will not be added to the array.

Examples

join('; ',eachOf(collect(10, 11, 12, 13.4), if(. <= 13, ., null))	10; 11; 12
sum(eachof(model.items, .index))	6
sum(eachof(model.items, if(.index > 2, 5, .index)))	8

FirstWhere function

Returns the first entry of the collection or array where the second expression results in a value considered to be true

Usage

```
{{firstwhere(collection, selectExpr)}}
```

Parameters

Name	Type	Description
collection	`array`	Required. Any objects, constant or expressions that will return an array of values.
selectExpr	`boolean`	Required. Any current data expression that will indicate if the item value should be considered a match. If the returned value is null or 0, then it will not be considered false, otherwise it will be considered true and become the result.

Returns

Type	Description
`object`	The first item in the array that matched from the selectExpr in the parameters.

Notes

FirstWhere will return a single entry from the orginal array or collection if a a match is found, or null if none of the items match the second expression.

Examples

firstWhere(collect(10, 11, 12, 13.4), . > 10)	11
firstWhere(model.items, .index != 1)	{"name": "Second Item", "index": 3}
firstWhere(model.days, length(.) > 3)	tues

Index Function

During processing of a collection by a template, the current index (zero-based) is remembered. As a template goes through each item and processes the content it increments the current index. This value can be accessed by the index() function. Only 1 value is available, however the current value can be stored in a variable before an inner loop is executed.

Usage

```
{{index()}}
```

Returns

Type	Description
`integer`	The zero based index of the looping content template.

Notes

As the document here is created dynamically, the examples use an index value that increments each time.
The final example calls back up the hierachy for stored values on each of the parent template indexes.

Examples

index()	0
index()	1
if(index() % 2 == 1,'odd', 'even')	even
concat('This is the ', index()+ 1, 'th item')	This is the 4th item
concat(sectionIndex,'.', funcIndex, '.', index())	9.3.4

Reverse Function

The reverse function will return all the arguments passed to the function as a single collection / array, in the *reverse* order they are added. If one of those arguments is a collection or array (or a function that returns an array), then it will add the items in the array to the result (reversed).

Usage

```
{{reverse(item1 [,item2,item3, ...])}}
```

Parameters

Name	Type	Description
item1 etc...	`object or array`	One or more. Any objects, constant or expressions that will return a value, or an array of values.

Returns

Type	Description
`array`	The complete collection of the entries in the parameters as a single array in reverse order.

Notes

No validation of type of structure to be added is performed, the returned array can contain any type of content, at any index

Examples

join('; ',reverse(10, 11, 12, 13.4))	13.4;12;11;10
join('; ',reverse(model.array, 13.4))	13.4;12;11;10
join('; ',reverse(sortBy(model.days,.)))	wed;tues;thur;sun;sat;mon;fri

SelectWhere Function

The selectwhere function will return the an array of all the items from the array or collection in the first parameter that are matched against the second parameter in the context of the current item.

Usage

```
{{selectwhere(collection, selectExpr)}}
```

Parameters

Name	Type	Description
collection	`array`	Required. Any objects, constant or expressions that will return an array of values.
selectExpr	`boolean`	Required. Any current data expression that will indicate if the item value should be considered a match. If the returned value is null or 0, then it will be considered false, otherwise it will be considered true and the array item added to the result.

Returns

Type	Description
`array`	The complete collection of the entries in the source array that were matched from the selectExpr in the parameters as a single array.

Notes

SelectWhere differs from the EachOf function, in that the object that is added to the result array for EachOf is the value that is returned from the valueExpr, whereas SelectWhere will add the original item if the result of the select expression is considered true

Examples

join('; ',selectWhere(collect(10, 11, 12, 13.4), . > 10))	11; 12; 13.4
join(', ' ,selectWhere(model.items, .index != 1))	{"name": "Second Item", "index": 3}, {"name": "Third Item", "index": 2}
join(', ', selectWhere(model.days, length(.) > 3))	tues, thur

SortBy Function

The SortBy function will return the an array of all the items from the array or collection in the first parameter ordered by the compared value returned from the result of the sort expression.

Usage

```
{{sortby(collection, sortExpr)}}
```

Parameters

Name	Type	Description
collection	`array`	Required. Any objects, constant or expressions that will return an array of values.
sortExpr	`boolean`	Required. Any current data expression that will returned a comparable value.

Returns

Type	Description
`array`	The complete collection of the entries in the source array that are sorted in another single array.

Notes

*The sortby function will always return the entries of the array, sorted in *ascending* order. e.g. 1,2,3,4 or A, B, C, D; and the comparison is case sensitive.*

Examples

join('; ',sortBy(collect(13.4, 10, 12, 11), .))	10; 11; 12; 13.4
join(', ' ,sortBy(model.items, .index))	{"name": "First Item", "index" : 1}, {"name": "Third Item", "index": 2}, {"name": "Second Item", "index": 3}
join(', ' ,sortBy(model.items, .name))	{"name": "First Item", "index" : 1}, {"name": "Second Item", "index": 3}, {"name": "Third Item", "index": 2}

Index

S

T

Y