Scripting and Syntax

This page covers DGLux5 script.

Topics on this page are:

- When to use DGLux5 script
- The differences between DGLux5 script and JavaScript
- How to access the page model
- How to use script for debugging
- How to reuse script, or how to call a script function from another block

This page also contains a reference of some available operators, formats, and functions in the following categories:

- String, number, logical, and bit-wise operators
- String usage and functions
- Number formats, number functions, math functions, and constants
- DateTime formats and functions
- List functions
- JSON functions
- XML functions
- DGLux5 table functions

Finally, this page contains examples for:

- Custom latitude and longitude operations
- Custom easing functions

See also:

- Custom JS Components
- Using Custom div Elements

When to Use DGLux5 Script

DGLux5 features a visual programming environment, the dataflow. However, in some situations, script is necessary or is more efficient than the dataflow.

You can use script in these dataflow blocks:

- Script
- JavaScript
- Filter (**condition** property)

- Column Mapping (**from *n*** property)

You can also use the page model syntax, which is described below, in binding paths. See Bindings.

Although DGLux5 script has some differences from JavaScript, JavaScript can be used in DGLux5 to accomplish many custom calculation tasks. The Internet is a good resource for finding JavaScript functions and excerpts.

Differences between DGLux5 Script and JavaScript

Read this section if you are already familiar with JavaScript, ActionScript, or another ECMA262-based scripting language.

The differences between JavaScript and DGLux5 script are as follows:

- In DGLux5 script, `list` does not support dynamic properties. For example, the following code does not work in DGLux5:

```
list = [];
list.name = 'mylist'; // Error: this does not work in DGLux5.
```

- DGLux5 script uses the `DateTime` type instead of the JavaScript `Date` type. This is because January in `DateTime` corresponds to one, while January in `Date` corresponds to zero. For example, the following code demonstrates `DateTime` functioning correctly and `Date` functioning incorrectly in DGLux5.

```
//Returns 2014-01-01.
d = new DateTime(2014,1,1);

//Does not work in DGLux5. In JavaScript this would mean 2014-02-01.
incorrect = new Date(2014,1,1);
```

Page Model

The script execution context and the block location both have an important relationship with the DGLux5 page model.

Use the at sign (@) to access the virtual model of the page and interact with other elements in the model. For example:

- @ refers to the current block.
- `@parent` refers to the logical parent of the current block or element. You can use this string more than once to refer to further ancestors, such as `@parent.@parent.@parent`.
- `@params` refers to the parameters of the symbol being referred to. For example, when used inside a symbol, in the symbol's root dataflow model, `@parent.@params` refers to the parameters of this symbol.
- `@stage` refers to the first immediate ancestor stage of the the current block or element.
    - If used in a page, `@stage` refers to the root for the page.
    - If used in a symbol, `@stage` refers to the root of the symbol.
- `@lib` refers to the root of the Project Dataflow.
- In a binding, `@parent.@table.0_v1` refers to the cell in row zero, column v1, for a table that shares the same parent as the current block.
- `@<pageModelName>` refers to the page model name of the component at the specified path. For example, to refer to the **width** property of a shape component that is in a group, you might use `@stage.@group1.@shape1.width`.

How to View Element Types and Page Model Names

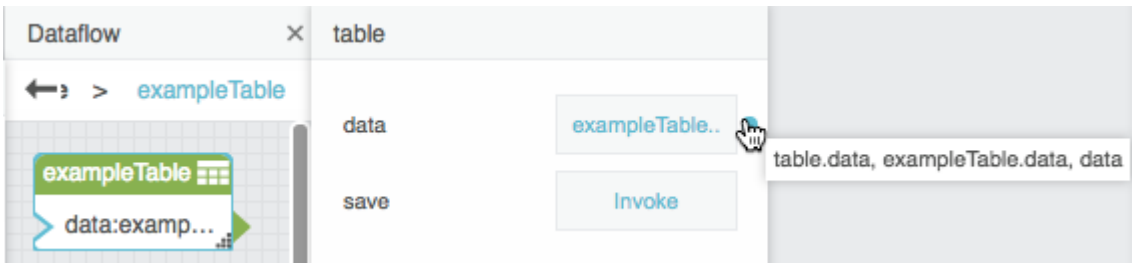
To view the page model name of any property and the property's parent:

1. Hover over the property until a blue dot appears.
2. Hold Option and right-click the blue dot.

   A tooltip displays all of the following:

    - Parent element type and property name
    - Parent element name in page model syntax and property name
    - Property name

   The following images demonstrate this tooltip.



How to Change Page Model Names

To change an element name in the page model syntax:

1. Double-click the element name in the Outline or the dataflow block header.
2. Type the new name.
3. Press Command + Enter on Mac or Ctrl + Enter on Windows.

If you press only Enter, the name is changed in the DGLux5 user interface but not in the page model.

Important

Changing an element's page model name breaks any scripts or bindings that already refer to that element. You must also change the name in these scripts or bindings.

How to Access and Set Values via Script

The following example demonstrates how to use script to access and set a value in the page model.

```
//Sets the "a" property of the current block to
//the value of the string block located on
//the parent dataflow.
@.a = @parent.string.value;
```

Use = to set a runtime value for an object property, and use ~= to set a value that will be serialized as part of a subsequent save. The following examples demonstrate how to set both kinds of values.

```
//Sets the runtime "value" property of the
//parent object's "string" block to
//"stringValue".
@parent.string.value = "stringValue";

//Sets the serialized "value" property of the
//parent object's "string" block to
//"stringValue".
@parent.string.value ~= "stringValue";
```

How to Create Bindings via Script

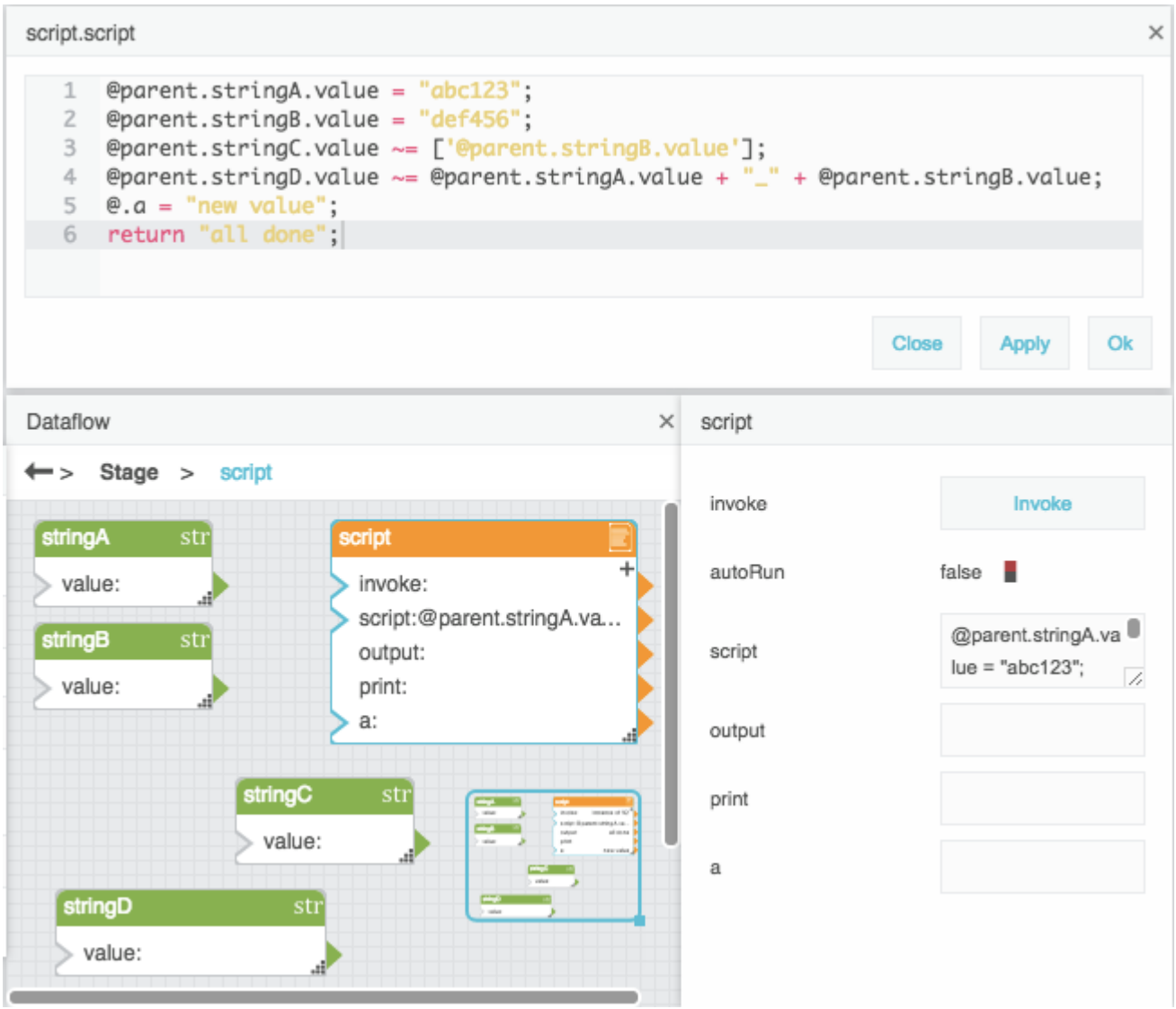
To create a binding via script, use `<targetPropertyPath> ~= ['<sourcePropertyPath>']`. The following examples demonstrate how to create bindings via script.

```
//Creates a serialized binding of "height" to
//"width" of the parent component.
@parent.width ~= ['height'];

//Another binding example.
@parent.string.value ~= ['@parent.input.value'];
```

Example

The following image shows the dataflow model before invoking a script. This script sets values, creates a binding, and returns an output value.

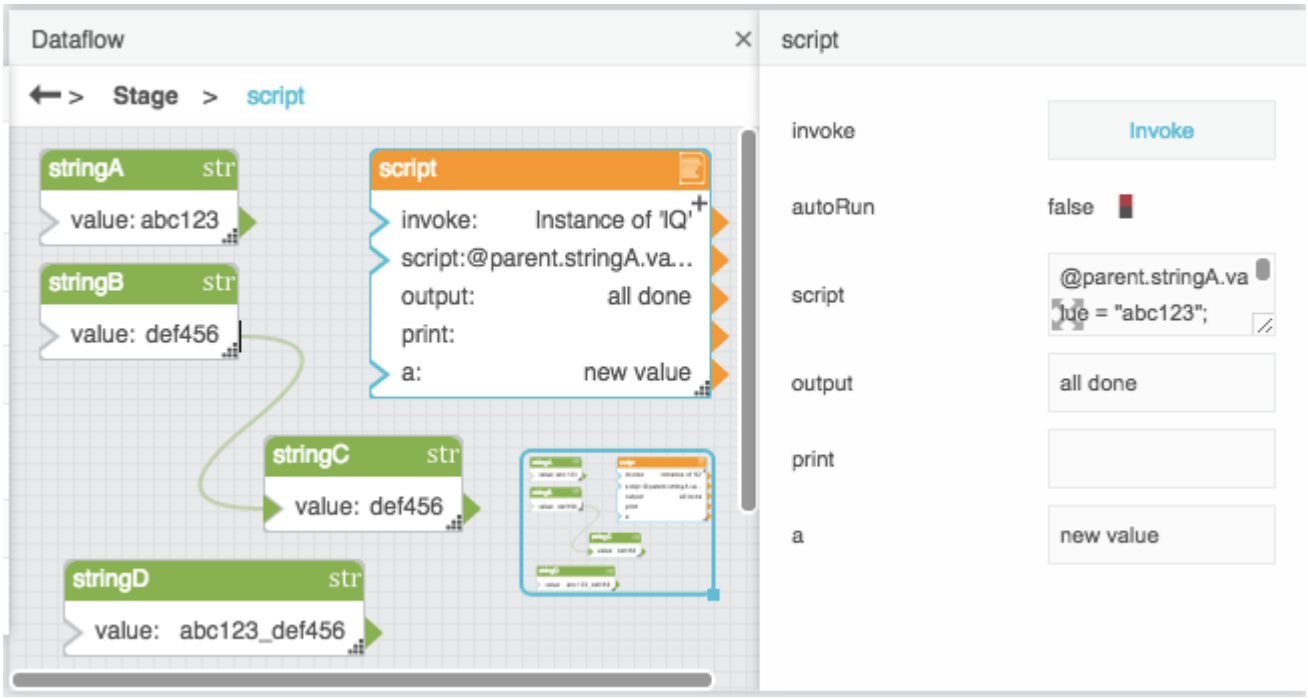


The following is the script in the example.

```
@parent.stringA.value = "abc123";
@parent.stringB.value = "def456";
@parent.stringC.value ~= ['@parent.stringB.value'];
```

```
@parent.stringD.value ~= @parent.stringA.value + "_" + @parent.stringB.value;
return "all done";
```

The following image shows the same dataflow model after invoking the script.



How to Traverse the Page Model

You can use `while` loops to traverse the page model. The following example shows how to search the parent object, then the parent's parent, and so on, until a symbol is found.

```
searchObject = @parent;
while(searchObject != null && typeof(searchObject) != 'symbol'){
  searchObject = searchObject.@parent;
}

if (searchObject != null){
  // set the value
}
```

Debugging

The following examples print script output to the block's **print** property or to the browser console.

To print the results of an expression to the current block's **print** property:

```
//Prints 2 + 2 = 4
print("2 + 2 = ", 2 + 2);
```

To print results of expressions to the browser console:

```
/*Prints the following output to the browser's console:
1
2
3
4
5
*/

json = JSON.parse("[1,2,3,4,5]");
for (var i = 0; i < json.length; i++) {
    printConsole(json[i]);
}
```

How to Reuse Script

You can refer to a script from within another Script, Column Mapping, or Filter block. When you do this, you can use `$params`, which is a special list that is only available when a script is called as a function from another script.

The following example creates a script and then calls it from a Script block and a Column Mapping block:

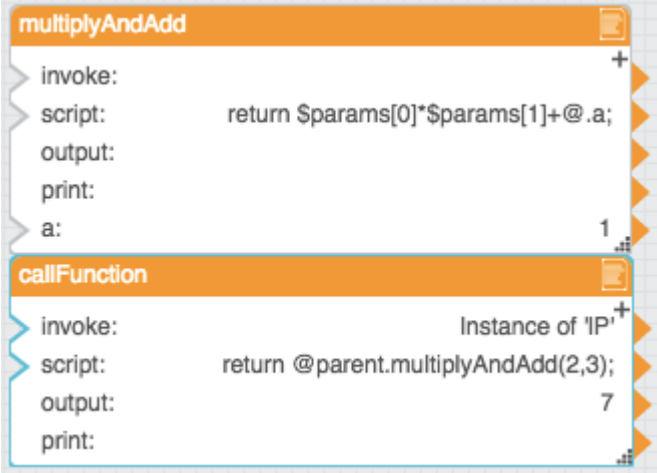
1. Create a Script block, and change its page model name to `multiplyAndAdd`.
2. For the **script** property of `multiplyAndAdd`, add a script that multiplies two `$params` values and adds some other variable to the product:

   ```
   return $params[0] * $params[1] + @.a;
   ```

   For this example, you must also add an **a** property to `multiplyAndAdd` by clicking the plus sign, and set the value of **a** as 1.
3. Add a second Script block, and change its name to `callFunction`.
4. For the **script** property of `callFunction`, add a script that calls `multiplyAndAdd` and provides some parameter values:

   ```
   return @parent.multiplyAndAdd(2,3);
   ```

   In this example, `callFunction` returns 7, as shown in the following image.

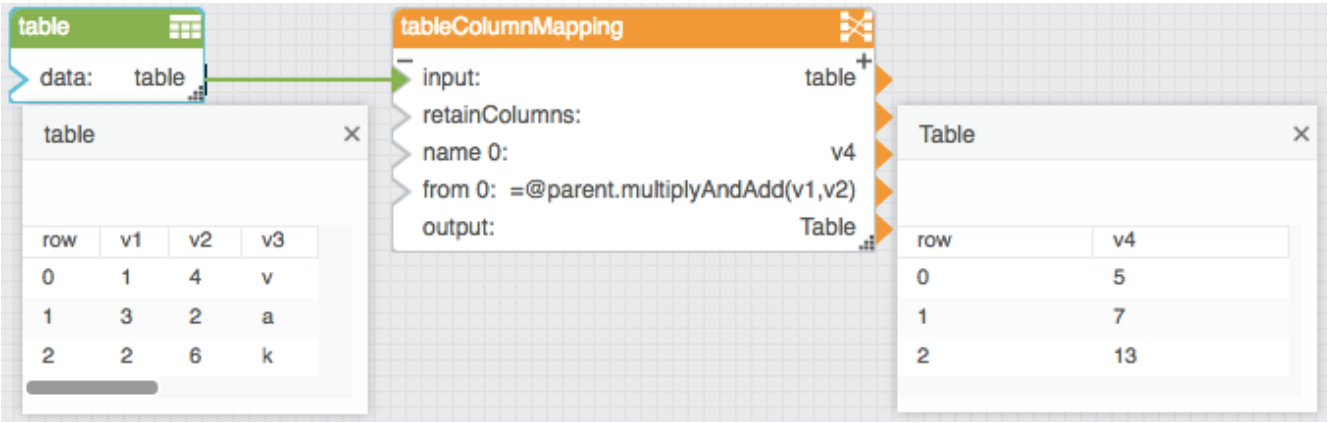


5. Add a Table block and Column Mapping block.
6. Bind the Table block value to the Column Mapping block input.
7. For **name 0**, enter v4.

8. For **from 0**, enter script that calls `multiplyAndAdd`:

   `=@parent.multiplyAndAdd(v1,v2)`

   In each row, the new column contains the product of **v1** and **v2** in this row, plus the value of **@.a** in `multiplyAndAdd`. The following image demonstrates the input and output tables.



Operators, Formats, and Functions

This section demonstrates some operators, formats, and functions that can be used in DGLux5. For example, the Column Mapping block, Filter block, and Date Time Operations blocks use some of these items.

Operators

These operators can be used in DGLux5.

String Operators

Operator	Description	Example	Result
Plus sign (+)	Concatenates two strings.	`'a'+'b';`	ab

Number Operators

Operator	Description	Example	Result
Plus sign (+)	Adds two numbers.	`5+5;`	`10`
Hyphen (-)	Subtracts two numbers.	`10-5;`	5
Asterisk (*)	Multiplies two numbers.	`2*2;`	4
Forward slash (/)	Divides two numbers.	`9/2;`	`4.5`
Percent symbol (%)	Performs a modulo operation. This operation divides two numbers and returns the remainder.	`10%3;`	1

Logical Operators

Logical operators appear in expressions that can be evaluated to TRUE or FALSE.

Operator	Example	Result
Less Than (<)	`5<10;`	`true`
Greater Than (>)	`10>5;`	`true`
Less Than or Equal To (<=)	`5<=5;`	`true`
Greater Than or Equal To (>=)	`5>=5;`	`true`
Equal To (==)	`'5'==5`	`true`
Strictly Equal To (===)	`5==='5';`	`false`
Not Equal To (!=)	`'1'!=1;`	`false`
Not Strictly Equal To (!==)	`'1'!=='1'`	`true`
And (&&)	`true && false;`	`false`
Or (||)	`true || false;`	`true`

Bit-Wise Operators

You can use the following bitwise operators:

Operator	Description
<<	Shift Left
>>	Shift Right
&	Bit-wise And

Operator	Description
|	Bit-wise Or
^	Bit-wise Xor

String Usage and Functions

String Literals

The following example demonstrates string literals in DGLux5:

```
str1 = 'Single quotes';
str2 = "Double quotes";
str3 = 'Double quotes in "single" quotes';
str4 = "Single quotes in 'double' quotes";
str5 = "escape quote \"";
```

Reserved Characters

In strings, you can use a backslash (\) to escape the following reserved characters:

- Single quotation mark (')
- Double quotation mark (")
- Backslash (\)

For example, the following code sets the string block's value to `C:\`:

```
@parent.string.value = "C:\\";
```

Special Sequences

In strings, you can use the following special character sequences to create these effects:

- `\n` creates a new line.
- `\r` creates a carriage return.
- `\t` creates a tab.
- `\uXXXX` uses a Unicode code point.

String Functions

The following functions perform an operation on an input string.

Function	Description and Example
`String(obj)`	Takes an input object, casts the object as a string, and returns the string. The following example returns 123 as a string. `String(123);`
`String(num,16)`	Takes an input base-10 number, converts the number to base 16, and returns the base-16 number as a string. The following example returns the string 1b. `String(27,16);`
`str.length`	Counts the number of characters in an input string and returns the number. The following example returns the number 4. `"acbd".length;`
`str.charAt(pos)`	Returns the character at the specified position in an input string. The first position is represented as 0. The following example returns the character b. `"abc".charAt(1);`
`str.charCodeAt(pos)`	Returns the character code for the character at the specified position in an input string. The first position is represented as 0. The following example returns 98, the character code for the letter "b". `"abc".charCodeAt(1);`
`str.indexOf(searchStr)`	Returns the position of the first character of the first occurrence of the search string, or returns -1 if the search string is not found. The first position is represented as 0. The following example returns the number 1. `"abc".indexOf("bc");`
`str.lastIndexOf(searchStr)`	Returns the position of the first character of the last occurrence of the search string, or returns -1 if the search string is not found. The first position is represented as 0. The following example returns the number 2. `"abb".lastIndexOf("b");`
`str.split(delimiter)`	Takes an input string that contains delimiters and returns an array of strings. The following example returns the array `["1","2","3"]`. `"1,2,3".split(",");`
`str = list.join(char)`	Takes an input list and converts it to an output string by joining items in the list with a provided delimiting character. The following example returns the string 1/2/3. `[1,2,3].join("/");`
`str.subStr(start, length)`	Extracts the specified number of characters starting from the specified position, and returns the substring. The first position is represented as 0. The following example returns the string bc. `"abcd".subStr(1,2);`

Function	Description and Example
`str.subString(start,end)`	Extracts the substring beginning with the specified start position and ending one position before the specified end value. The first position in a string is represented as 0. The following example returns the string `b`. `"abcd".subString(1,2);`
`str.replace(regexPattern, replace)`	Replaces all instances of the specified regular expression pattern with the replacement string. The following example returns the string `d d`. `"DabcG DdefG".replace(/D(\w+)/g, "d");`
`str.replaceAll(find, replace)`	Replaces all instances of the specified `find` string with the specified `replace` string. The following example returns the string `1_2_3`. `"1#2#3".replaceAll("#","_");`
`str.toLowerCase()`	Converts all letters in a string to lowercase letters. The following example returns the string `dg`. `"DG".toLowerCase();`
`str.toUpperCase()`	Converts all letters in a string to uppercase letters. The following example returns the string `DG`. `"dg".toUpperCase();`
`str.encodeUriComponent()`	Encodes an input URI component. The following example returns the string `a%20b`. `"a b".encodeUriComponent();`
`str.decodeUriComponent()`	Decodes an encoded URI component string. The following example returns the string `a b`. `"a%20b".decodeUriComponent();`
`str.splitQuery()`	Splits the input query string and decodes the string into an Object. The following example returns the Object `{"a b": "1", "c": "d"}`. `"a%20b=1&c=d".splitQuery();`
`str.encodeBase64();`	Encodes the input string, using base-64 encoding. The following example returns an encoded string. `"abc123".encodeBase64();`
`str.decodeBase64();`	Decodes a string that was encoded using `str.encodeBase64()`. The following example returns a decoded string. `"YWJjMTQ0".decodeBase64();`
`str.md5();`	Generates a 16-byte hash value that identifies the input string. The following example returns a 16-byte hash value. `"abc123".md5();`
`str.sha1();`	Generates a 20-byte hash value that identifies the input string. The following example returns a 20-byte hash value. `"abc123".sha1();`
`str.sha256();`	Generates an almost-unique 256-bit hash value that identifies the input string. The following example returns a 256-byte hash value. `"abc123".sha256();`

Number Formats, Functions, and Constants

Tip

To quickly check whether a value is number, add zero to the value, and then test whether the value has changed. If the value is a number, it does not change; if the value is a string, the zero is concatenated.

```
val = "123";
if (0+val == val) return "It's Numeric!";
```

Number Format Patterns

You can specify a number format using the following subset of ICU formatting patterns:

Formatting Pattern	Description
Zero (0)	Represents a single digit and is included even if the digit is a non-significant zero.
Number sign (#)	Represents a single digit and is omitted if the digit is a non-significant zero.
Decimal (.)	Represents the decimal separator.
Hyphen (-)	Represents a minus sign.
Comma (,)	Represents the grouping separator.
The letter E	Separates a number and that number’s exponent.
Plus sign (+)	When used before an exponent, indicates to prefix a positive exponent with a plus sign.
Percent symbol (%)	When used in a prefix or suffix, indicates to multiply the number by 100 and show the number as a percentage.
Per mile sign (‰ or \u2030)	Indicates to multiply the number by 1000 and show the number as per mile.
Currency sign (¤ or \u00A4)	Gets replaced by the currency name.
Single quotes (')	Used to quote special characters.
Semicolon (;)	Separates the positive and negative patterns if both are present.

Number Functions

Function	Description and Example
`Number(str)`	Takes an input string, converts the string to a number, and returns the number. The following example returns the number 12.12. `Number("12.12");`
`isNaN(str)`	Returns TRUE if the string is not a number. The following example returns the boolean value TRUE. `isNaN(Number("abc"));`

Function	Description and Example
num = parseNumber(str)	Parses a number from a string and returns the number. The following example returns the number 12.11. `parseNumber('$12.11');`
str = numberFormat(num, pattern)	Formats the input number using the input number format pattern. The following example returns the string $123.46. `numberFormat(123.456789,"$#.00");`

Math Functions

Function	Description and Example
Math.abs(num)	Returns the absolute value of num. The following example returns 47. `Math.abs(-47);`
Math.min(num1,num2,…)	Returns the smallest input number. The following example returns -2. `Math.min(-2,-1,0,1,2);`
Math.max(num1,num2,…)	Returns the greatest input number. The following example returns 2. `Math.max(-2,-1,0,1,2);`
Math.sin(num)	Returns the sine of num, where num is given in radians. The following example returns 1. `Math.sin(Math.PI / 2);`
Math.cos(num)	Returns the cosine of num, where num is given in radians. The following example returns -1. `Math.cos(Math.PI);`
Math.tan(num)	Returns the tangent of num, where num is given in radians. The following example returns 1.5574077246549023. `Math.tan(1);`
Math.asin(num)	Returns the arcsine in radians of num. The following example returns TRUE. `Math.asin(1) == Math.PI / 2;`
Math.acos(num)	Returns the arccosine of num. The following example returns TRUE. `Math.acos(-1) == Math.PI;`
Math.atan(num)	Returns the arctangent of num. The following example returns 1. `Math.atan(1.5574077246549023);`
Math.atan2(y, x)	Returns the arctangent of the quotient of y divided by x. The following example returns 1.4056476493802699. `Math.atan2(90,15);`
Math.ceil(num)	Rounds up num to the nearest integer. The following example returns 2. `Math.ceil(1.2);`
Math.floor(num)	Rounds down num to the nearest integer. The following example returns 1. `Math.floor(1.7);`

Function	Description and Example
`Math.round(num)`	Rounds num up or down to the nearest integer. The following example returns 2. `Math.round(1.7);`
`Math.exp(num)`	Returns the result of raising the mathematical constant *e* to the value of num. The following example returns TRUE. `Math.exp(2) == Math.pow(Math.E,2);`
`Math.log(num)`	Returns the natural (base-*e*) logarithm of num. The following example returns 2.302585092994046 . `Math.log(10);`
`Math.sqrt(num)`	Returns the square root of num. The following example returns 3. `Math.sqrt(9);`
`Math.pow(num, exponent)`	Returns the result of raising num to the value of exponent. The following example returns 8. `Math.pow(2,3);`
`Math.random()`	Returns a random number between zero and one. The following example returns a random number. `Math.random();`

Number Constants

Constant	Value
Math.PI	3.141592653589793, the constant pi
Math.E	2.718281828459045, the constant *e*
Math.LN2	0.6931471805599453, the natural (base-*e*) logarithm of two
Math.LN10	2.302585092994046, the natural (base-*e*) logarithm of ten
Math.LOG2E	1.4426950408889634, the base-2 logarithm of the constant *e*
Math.LOG10E	0.4342944819032518, the base-10 logarithm of the constant *e*
Math.SQRT2	1.4142135623730951, the square root of two
Math.SQRT1_2	0.7071067811865476, the square root of one-half

Tip

The following example gets a random value within a certain range:

```
function getRandom(min, max) {
    return Math.random() * (max - min) + min;
}

getRandom(50,75);
```

DateTime Formats and Functions

Examples of supported new DateTime (dateString) formats

The following are examples of date and time strings that have been formatted using date and time formatting strings:

- 2012-02-27 13:27:00
- 2012-02-27 13:27:00.123456z
- 20120227 13:27:00
- 20120227T132700
- 20120227
- +20120227
- 2012-02-27T14Z
- 2012-02-27T14+00:00

Note

The string 2002-02-27T14:00:00-0500 represents the same date and time as the string 2002-02-27T19:00:00Z.

Supported dateFormat patterns

The following table lists date and time pattern components. These components can be included within a date and time format string.

Notes

- Date and time strings are affected by locale.
- In some languages, the month of a year, or the day of a week, is a different word or declension from a standalone month or day.

Symbol	Meaning	Example
G	Era designator	AD
y	Year	2017
M	Month in year, number	8
MM	Month in year, number as two digits	08
MMM	Month in year, abbreviated name	Aug
MMMM	Month in year, full name	August

Symbol	Meaning	Example
L	Standalone month, number	0
LL	Standalone month, number as two digits	08
LLL	Standalone month, abbreviated name	Aug
LLLL	Standalone month, full name	August
d	Day in month	10
c	Standalone day	10
h	Hour in 12-hour time (1–12)	12
H	Hour in 24-hour time (0–23)	0
m	Minute in hour	30
s	Second in minute	55
S	Fractional second	978
E	Day of week, abbreviated name	Fri
EEEE	Day of week, full name	Friday
D	Day of year	189
a	AM or PM designator	PM
k	Hour of day, in 24-hour time (0–23)	24
K	Hour of day, in 12-hour time (0–11)	0
z	General time zone	-8
Z	Time zone, RFC 822	-800
v	Time zone, generic name	PDT
Q	Quarter	Q3
Single quotation mark (')	Escapes a character or encloses a string	'o''clock', 'Date='

Supported ICU/dateFormat skeletons

The following table lists ICU date and time skeleton components. These components can be combined to create an ICU dateFormat skeleton, which can format DateTime values.

Note

- Date and time strings are affected by locale.
- You can use two vertical bar characters separated by a space (| |) to join two dateFormat skeletons. For example, the string `yMd| |Hms` produces the string `8/24/2015 15:38:00`.

String	Meaning	Example (en_US locale)
d	Day	24
E	Day of week	Thu
EEEE	Day of week	Thursday
LLL	Standalone month	Aug

String	Meaning	Example (en_US locale)
LLLL	Standalone month	August
M	Month	8
Md	Month and day	8/24
MEd	Month, day, and day of week	Thu, 8/24
MMM	Month	Aug
MMMd	Month and day	Aug 24
MMMEd	Month, day, and day of week	Thu, Aug 24
MMMM	Month	August
MMMMd	Month and day	August 24
MMMMEEEEd	Month, day, and day of week	Thursday, August 24
QQQ	Quarter	Q3
QQQQ	Quarter	3rd quarter
y	Year	2015
yM	Year and month	8/2017
yMd	Year, month, and day	8/1/2017
yMEd	Year, month, day, and day of week	Thu, 8/24/2017
yMMM	Year and month	Aug 2017
yMMMd	Year, month, and day	August 1, 2017
yMMMEd	Year, month, day, and day of week	Thu, Aug 24, 2017
yMMMM	Year and month	August 2017
yMMMMd	Year, month, and day	August 24, 2017
yMMMMEEEEd	Year, month, day, and day of week	Thursday, August 24, 2017
yQQQ	Year and quarter	Q3 2017
yQQQQ	Year and quarter	3rd quarter 2017
H	Hour in day (24-hour time)	15
Hm	Hour and minute (24-hour time)	15:38
Hms	Hour, minute, and second (24-hour time)	15:38:00
j	Hour (12-hour time)	3 PM
jm	Hour and minute (12-hour time)	3:38 PM
jms	Hour, minute, and second (12-hour time)	3:38:00 PM
m	Minute	38
ms	Minute and second	38:00
s	Second	0

DateTime Functions

Note

If a `DateTime` object is sent to an output property of the Script dataflow block, the object is converted to a string, and the block property holds a string.

Function	Description and Example
`new DateTime()`	Returns a `DateTime` object that is initialized with the current date and time. The following function returns a year, such as 2017. `new DateTime().year;`
`new DateTime(millisecondsSinceEpoch)`	Returns a `DateTime` object that is initialized with the specified value. The following function returns 1969-12-31T16:00:00.000. `new DateTime(0);`
`new DateTime(dateString)`	Returns a `DateTime` object that is initialized with the parsed date and time string. The following function returns 2017-02-27T13:27:00.000. `new DateTime('2017-02-27 13:27:00');`
`new DateTime(y,m,d,h,m,s,ms)`	Returns a `DateTime` object that is initialized with the specified values. The following function returns 2017-01-01 00:00:00.000. `new DateTime(2017,1,1,0,0,0,0);`
`day`	Returns a day of the month. The following function returns 1. `new DateTime(2017,1,1).day;`
`month`	Returns a month of the year. January is represented as 1. The following function returns 2. `new DateTime('2017-02-27 13:27:00').month;`
`year`	Returns a year. The following function returns a year, such as 2016. `new DateTime().year;`
`hour`	Returns an hour. The following function returns 13. `new DateTime('2017-02-27 13:27:00').hour;`
`minute`	Returns a minute. The following function returns 27. `new DateTime('2017-02-27 13:27:00').minute;`
`second`	Returns a second. The following function returns 0. `new DateTime('2017-02-27 13:27:00').second;`
`millisecond`	Returns a millisecond. The following function returns 798. `new DateTime('2017-01-01T13:27:00.798').millisecond;`
`millisecondsSinceEpoch`	Returns the milliseconds since the start of the Unix epoch. The following function returns the millisecondsSinceEpoch, such as 1466113382710. `new DateTime().millisecondsSinceEpoch;`
`weekday`	Returns a day of the week. Monday is represented as 1, and Sunday is represented as 7. The following function returns the weekday, such as 4. `new DateTime().weekday;`
`isUtc`	Returns whether the date is UTC. The following function returns false. `new DateTime(2017,1,1).isUtc;`

Function	Description and Example
`dateNow()`	Returns the millisecondsSinceEpoch of the current date and time. The following function returns the millisecondsSinceEpoch, such as 1466113699717. `dateNow();`
`dateParse(str)`	Parses a date from a string and then returns `int(millisecondsSinceEpoch)` of that date. `dateParse` supports a super-set of formats supported by `DateTime(dateString)`. The following function returns 1293868800000. `dateParse('1/1/2011');`
`dateFormat(date, dateFormat)`	Parses date string or uses millisecondsSinceEpoch and returns the string in the specified dateFormat. The following function returns a string, such as 06/2016. `dateFormat(dateNow(),"MM/yyyy");`

DateTime Function Examples

The following example quickly finds the number of days in the previous month.

```
dt = new DateTime();
lastMonth = new DateTime(dt.year,dt.month, 0, dt.hour, dt.minute, dt.second,
dt.millisecond);
print (lastMonth.day);
```

The following example returns the most recent Sunday before the current date.

```
dt = new DateTime();
lastSunday = new DateTime(dt.year, dt.month, dt.day - dt.weekday);
```

List Functions

Function	Description and Example
`length`	Returns the number of items in a list. The following example returns the number 3. `[1,2,3].length;`

`push`	Adds an item to the end of a list.
	The following example returns the list `[1,2,3]`.
	`row = [1,2];`
	`row.push(3);`
	`return row;`

Function	Description and Example

`pop`	Removes an item from the end of a list and returns the item.
	In the following example, the value of a is 3, and the value of b is `[1,2]`.
	`row = [1,2,3];`
	`@.a = row.pop();`
	`@.b = row;`

`indexOf(searchObject)`	Returns the position of the first character of the first occurrence of the search string. Returns –1 if the search string is not found. The first position is represented as 0. The following example returns the number 1. `["a","b","c"].indexOf("b");`
`join(delimiter)`	Returns a delimited string of items in the list. The following example returns the string `a and b and c`. `["a","b","c"].join(" and ");`

`splice(start,len,newItem1,newItem2,…)`	Removes *n* items beginning at the start position, where *n* = `len`. Then, inserts new items into the same position and returns the items that are removed. The first position is represented with 0.
	In the following example, the value of a is `[2,3]`, and the value of b is `[1,0,0,4]`.
	`row = [1,2,3,4];`
	`@.a = row.splice(1,2,0,0);`
	`@.b = row;`

JSON Functions

Function	Description and Example
`JSON.parse(str)`	Converts a string to a JSON object. `JSON.parse('{"a":1}');` The above example returns the JSON object `{"a":1}`. If this object is sent to an output property of the Script dataflow block, the object is converted to a string, and the block property holds a string.

`JSON.stringify(obj)`	Converts a JSON object to a string.
	The following example returns the string `{"a":1}`.
	`var json = JSON.parse('{"a":1}');`
	`JSON.stringify(json);`

JSON Function Example

The following example uses a `for` loop to print the `id` values for each item in a JSON object and to print rows of dashes to represent null items.

```
json = {"menu": {
    "header": "SVG Viewer",
    "items": [
```

```
        {"id": "Open"},
        {"id": "OpenNew", "label": "Open New"},
        null,
        {"id": "ZoomIn", "label": "Zoom In"},
        {"id": "ZoomOut", "label": "Zoom Out"},
        {"id": "OriginalView", "label": "Original View"},
        null,
        {"id": "Quality"},
        {"id": "Pause"},
        {"id": "Mute"},
        null,
        {"id": "Find", "label": "Find..."},
        {"id": "FindAgain", "label": "Find Again"},
        {"id": "Copy"},
        {"id": "CopyAgain", "label": "Copy Again"},
        {"id": "CopySVG", "label": "Copy SVG"},
        {"id": "ViewSVG", "label": "View SVG"},
        {"id": "ViewSource", "label": "View Source"},
        {"id": "SaveAs", "label": "Save As"},
        null,
        {"id": "Help"},
        {"id": "About", "label": "About SVG Viewer..."}
    ]
}};

print(json.menu.header);

for (var i = 0; i < json.menu.items.length; i++) {
    if (json.menu.items[i] == null)
    {
        print("----");
    }
    else
    {
        print(json.menu.items[i].id);
    }
}
```

The output for this example is the following:

```
SVG ViewerOpenOpenNew----ZoomInZoomOutOriginalView----QualityPauseMute----
FindFindAgainCopyCopyAgainCopySVGViewSVGViewSourceSaveAs----HelpAbout
```

XML Functions

Function	Description and Example
XML.parse(str)	Converts an XML string into XML elements. `var xml = XML.parse('<p id=\"a\">b<\/p>');` In the above example, the value of `var` is the XML object b . If this object is sent to an output property of the Script dataflow block, the object is converted to a string, and the block property holds a string.

XML.stringify(xmlElement)	Converts XML elements into an XML string
	The following example returns the string b .
	`var xml = XML.parse('<p id=\"a\">b<\/p>');`
	`@.a = XML.stringify(xml);`

query(str)	Returns the first matching element.
	The following example returns <item>ITEM 1</item>.
	`var xml = XML.parse('\<items\>\<item\>ITEM 1\</item\>\<item\>ITEM 2\</item\>\</items\>');`
	`xml.query('item');`

queryAll(str)	Returns a List of matching elements.
	The following example returns <item>ITEM 1</item>, <item>ITEM 2</item>.
	`var xml = XML.parse('\<items\>\<item\>ITEM 1\</item\>\<item\>ITEM 2\</item\>\</items\>');`
	`var items = xml.queryAll('item');`
	`@.a = items.join(', ');`

children	Returns a List of all children.
	The following example returns B 1,<c>C 1</c>.
	`var xml = XML.parse('\<a\>\<b\>B 1\<c\>C 1\</c\>\</b\>\<b\>B 2\</b\>\</a\>');`
	`var ch = xml.query('b').children;`
	`return ch.join(',');`

elements	Returns a list of children elements.
	The following example returns <c>C 1</c>.
	`var xml = XML.parse('\<a\>\<b\>B 1\<c\>C 1\</c\>\</b\>\<b\>B 2\</b\>\</a\>');`
	`var el = xml.query('b').elements;`
	`return el.join(',');`

name	Returns the name of an element.
	The following example returns a.
	`var xml = XML.parse('\<a b=\"1\"\>A\</a\>');`
	`return xml.name;`

data	Returns the data of an element, when the element is a data node. The `text` function is strongly recommended instead of the `data` function.
	The following example returns A.
	`var xml = XML.parse('\<a b=\"1\"\>A\</a\>');`
	`return xml.children[0].data;`

text	Returns the text of an element. Checks whether the node is a data node or has a data node as a child.
	The following example returns A.
	`var xml = XML.parse('\<a b=\"1\"\>A\</a\>');`
	`return xml.text;`

Function	Description and Example

`getAttribute(str)`	Returns an attribute value.
	The following example returns 1.
	`var xml = XML.parse('\<items\>\<item att=\"1\"\>A\</item\>\</items\>');`
	`var item = xml.query('item');`
	`var att = item.getAttribute('att');`
	`return att;`

`remove`	

Table Functions

Function	Description and Example
`tableGetRows(table)`	Gets a list of rows from a table object. The following example returns a list, such as `[[0,1,4,"v"],[1,3,2,"a"],[2,2,6,"k"]]`. `tableGetRows(@parent.table);`
`tableGetColumns(table)`	Gets a list of information about columns in a table object. The following function returns a list, such as `[…{"name":"v1","type":"string","width":100,"formula":null,"meta":{"name":"v1"}}…]`. `tableGetColumns(@parent.table);`
`tableSet(table,rows,[columns], serialize=true)`	Sets data in the specified table. The `row` column is added automatically and contains automatically generated row IDs. The following function erases all data in the table, adds two new rows, and adds the specified column headers. A `row` column is also automatically created. `tableSet(@parent.table,[["a","b"],["c","d"]],["col1","col2"]);`
`tableAddRows(table, rows)`	Adds rows to a table object. A dummy value must be passed as the first item in any row. This dummy value will always be overriden by the automatically generated row ID in the `row` column. The following function adds two rows to the table. `tableAddRows(@parent.table,[["","a","b"],["","c","d"]]);`
`tableRemoveRows(table, start, [count=1])`	Removes a number of rows from table object. The following function removes two rows from a table, beginning with the row at index 0. `tableRemoveRows(@parent.table, 0, 2);`
`tableClear(table)`	Clears the table. The following function removes all rows from a table. Columns remain. `tableClear(@parent.table);`
`$thisRow['column Name']`	Selects the column value from the current row being iterated on by a Filter or Column Mapping block. If the column name contains no spaces, the column name alone can be used in place of `$thisRow['column Name']`. For more examples, see Column Mapping and Filter. The following example demonstrates how to use `$thisRow`: `=$thisRow['Energy Usage'].indexOf("History") > -1` The new column contains a TRUE value in each row where the string "History" appears in the Energy Usage column and a FALSE value in each row where the string "History" does not appear.

Examples

This section contains some custom functions that can be used in script.

Lat-Long Operations

The following example contains some custom latitude and longitude functions.

```
function toRad(num) {return num * Math.PI / 180;}
function toDeg(num) {return num * 180 / Math.PI;}
```

```
function calcDistance(lat1, lon1, lat2, lon2) {
  var R = 6371; // Radius of the earth in km
  var dLat = (lat2 - lat1) * Math.PI / 180;  // deg2rad below
  var dLon = (lon2 - lon1) * Math.PI / 180;
  var a =
     0.5 - Math.cos(dLat)/2 +
     Math.cos(lat1 * Math.PI / 180) * Math.cos(lat2 * Math.PI / 180) *
     (1 - Math.cos(dLon))/2;

  return R * 2 * Math.asin(Math.sqrt(a));
}

function calcBearing (lat1,lng1,lat2,lng2)
{
    var dLon = (lng2-lng1);
    var y = Math.sin(dLon) * Math.cos(lat2);
    var x = Math.cos(lat1)*Math.sin(lat2) -
Math.sin(lat1)*Math.cos(lat2)*Math.cos(dLon);
    var brng = toDeg(Math.atan2(y, x));
    return 360 - ((brng + 360) % 360);
}

function calcSpeed(lat1,lon1,lat2,lon2,time1,time2)
{
    var d = calcDistance(lat1,lon1,lat2,lon2);
    var time = dateParse(time2)-dateParse(time1);
    return d/(time/1000/60/60);
}
```

Custom Easing Functions

The following example contains some custom easing functions.

```
easing =
{
    // t: current time, b: begInnIng value, c: change In value, d: duration

    swing: function (t, b, c, d) {
        return easing[easeOutQuad](t, b, c, d);
    },
    easeInQuad: function (t, b, c, d) {
        return c*(t/=d)*t + b;
    },
    easeOutQuad: function (t, b, c, d) {
        return -c *(t/=d)*(t-2) + b;
    },
```

```
    easeInOutQuad: function (t, b, c, d) {
        if ((t/=d/2) < 1) return c/2*t*t + b;
        return -c/2 * ((--t)*(t-2) - 1) + b;
    },
    easeInCubic: function (t, b, c, d) {
        return c*(t/=d)*t*t + b;
    },
    easeOutCubic: function (t, b, c, d) {
        return c*((t=t/d-1)*t*t + 1) + b;
    },
    easeInOutCubic: function (t, b, c, d) {
        if ((t/=d/2) < 1) return c/2*t*t*t + b;
        return c/2*((t-=2)*t*t + 2) + b;
    },
    easeInQuart: function (t, b, c, d) {
        return c*(t/=d)*t*t*t + b;
    },
    easeOutQuart: function (t, b, c, d) {
        return -c * ((t=t/d-1)*t*t*t - 1) + b;
    },
    easeInOutQuart: function (t, b, c, d) {
        if ((t/=d/2) < 1) return c/2*t*t*t*t + b;
        return -c/2 * ((t-=2)*t*t*t - 2) + b;
    },
    easeInQuint: function (t, b, c, d) {
        return c*(t/=d)*t*t*t*t + b;
    },
    easeOutQuint: function (t, b, c, d) {
        return c*((t=t/d-1)*t*t*t*t + 1) + b;
    },
    easeInOutQuint: function (t, b, c, d) {
        if ((t/=d/2) < 1) return c/2*t*t*t*t*t + b;
        return c/2*((t-=2)*t*t*t*t + 2) + b;
    },
    easeInSine: function (t, b, c, d) {
        return -c * Math.cos(t/d * (Math.PI/2)) + c + b;
    },
    easeOutSine: function (t, b, c, d) {
        return c * Math.sin(t/d * (Math.PI/2)) + b;
    },
    easeInOutSine: function (t, b, c, d) {
        return -c/2 * (Math.cos(Math.PI*t/d) - 1) + b;
    },
    easeInExpo: function (t, b, c, d) {
        return (t==0) ? b : c * Math.pow(2, 10 * (t/d - 1)) + b;
    },
    easeOutExpo: function (t, b, c, d) {
        return (t==d) ? b+c : c * (-Math.pow(2, -10 * t/d) + 1) + b;
```

```
    },
    easeInOutExpo: function (t, b, c, d) {
        if (t==0) return b;
        if (t==d) return b+c;
        if ((t/=d/2) < 1) return c/2 * Math.pow(2, 10 * (t - 1)) + b;
        return c/2 * (-Math.pow(2, -10 * --t) + 2) + b;
    },
    easeInCirc: function (t, b, c, d) {
        return -c * (Math.sqrt(1 - (t/=d)*t) - 1) + b;
    },
    easeOutCirc: function (t, b, c, d) {
        return c * Math.sqrt(1 - (t=t/d-1)*t) + b;
    },
    easeInOutCirc: function (t, b, c, d) {
        if ((t/=d/2) < 1) return -c/2 * (Math.sqrt(1 - t*t) - 1) + b;
        return c/2 * (Math.sqrt(1 - (t-=2)*t) + 1) + b;
    },
    easeInElastic: function (t, b, c, d) {
        var s=1.70158;var p=0;var a=c;
        if (t==0) return b;  if ((t/=d)==1) return b+c;  if (!p) p=d*.3;
        if (a < Math.abs(c)) { a=c; var s=p/4; }
        else var s = p/(2*Math.PI) * Math.asin (c/a);
        return -(a*Math.pow(2,10*(t-=1)) * Math.sin( (t*d-s)*(2*Math.PI)/p ))
+ b;
    },
    easeOutElastic: function (t, b, c, d) {
        var s=1.70158;var p=0;var a=c;
        if (t==0) return b;  if ((t/=d)==1) return b+c;  if (!p) p=d*.3;
        if (a < Math.abs(c)) { a=c; var s=p/4; }
        else var s = p/(2*Math.PI) * Math.asin (c/a);
        return a*Math.pow(2,-10*t) * Math.sin( (t*d-s)*(2*Math.PI)/p ) + c +
b;
    },
    easeInOutElastic: function (t, b, c, d) {
        var s=1.70158;var p=0;var a=c;
        if (t==0) return b;  if ((t/=d/2)==2) return b+c;  if (!p)
p=d*(.3*1.5);
        if (a < Math.abs(c)) { a=c; var s=p/4; }
        else var s = p/(2*Math.PI) * Math.asin (c/a);
        if (t < 1) return -.5*(a*Math.pow(2,10*(t-=1)) * Math.sin( (t*d-
s)*(2*Math.PI)/p )) + b;
        return a*Math.pow(2,-10*(t-=1)) * Math.sin( (t*d-s)*(2*Math.PI)/p )*.5
+ c + b;
    },
    easeInBack: function (t, b, c, d, s) {
        if (s == undefined) s = 1.70158;
        return c*(t/=d)*t*((s+1)*t - s) + b;
    },
```

```
    easeOutBack: function (t, b, c, d, s) {
        if (s == undefined) s = 1.70158;
        return c*((t=t/d-1)*t*((s+1)*t + s) + 1) + b;
    },
    easeInOutBack: function (t, b, c, d, s) {
        if (s == undefined) s = 1.70158;
        if ((t/=d/2) < 1) return c/2*(t*t*(((s*=(1.525))+1)*t - s)) + b;
        return c/2*((t-=2)*t*(((s*=(1.525))+1)*t + s) + 2) + b;
    },
    easeInBounce: function (t, b, c, d) {
        return c - easing.easeOutBounce (d-t, 0, c, d) + b;
    },
    easeOutBounce: function (t, b, c, d) {
        if ((t/=d) < (1/2.75)) {
            return c*(7.5625*t*t) + b;
        } else if (t < (2/2.75)) {
            return c*(7.5625*(t-=(1.5/2.75))*t + .75) + b;
        } else if (t < (2.5/2.75)) {
            return c*(7.5625*(t-=(2.25/2.75))*t + .9375) + b;
        } else {
            return c*(7.5625*(t-=(2.625/2.75))*t + .984375) + b;
        }
    },
    easeInOutBounce: function (t, b, c, d) {
        if (t < d/2) return easing.easeInBounce (t*2, 0, c, d) * .5 + b;
        return easing.easeOutBounce (t*2-d, 0, c, d) * .5 + c*.5 + b;
    }
};
return easing[@.a](0+@.b,0,1,1,0+@.e)*@.c+@.d;
```

Previous: LOG10E

Next: Custom JS Components