TIBCO Flogo® Enterprise

Mapper Functions Guide

Version 2.25.2 | January 2025

Contents

Overview of Mapper Functions

Flogo® provides a graphical data mapper to map data between the activities within a flow, and between the trigger and the flows attached to the trigger within an app. The mapper functions can be used within a mapping expression to enable a more powerful experience. For example, when mapping data, you can coerce the data to a specific time or CONCAT two strings.

Important: For general information about the data mapper and how to use it, see the "Data Mappings" section in *TIBCO Flogo® Enterprise User Guide*.

The mapper provides commonly used functions that you can use along with the data to be mapped. The functions are grouped into the following logical categories:

Category	Description
Array	The Array functions perform operations on collections of similar types of data.
Boolean	The Boolean functions return only `true` or `false`. You can use these functions to perform operations based on a condition.
Coerce	The Coerce functions coerce data from one type to another.
Compression Functions	The Compression functions compress and uncompress strings using the GZip compression algorithm.
Data	The Data functions perform operations on mapping references.
Datetime	The Datetime functions retrieve, parse, and manage date and time values.

Category	Description
Float	The Float functions convert an integer or string to a floating-point value.
JSON	The JSON functions perform operations on JSON objects or arrays.
Math	The Math functions perform common mathematical operations on values.
Number	The Number functions perform various operations on numbers.
String	The String functions perform various string manipulations. They operate on character strings.
URL	The URL functions return parts of a specific URL, such as the host name used in the URL.
Utility	The Utility functions include general functions such as utility.renderJSON, which converts JSON data to a string.
Utils	The Utils functions include general functions for encoding and decoding strings, generating a random UUID, and so on.

Using Functions

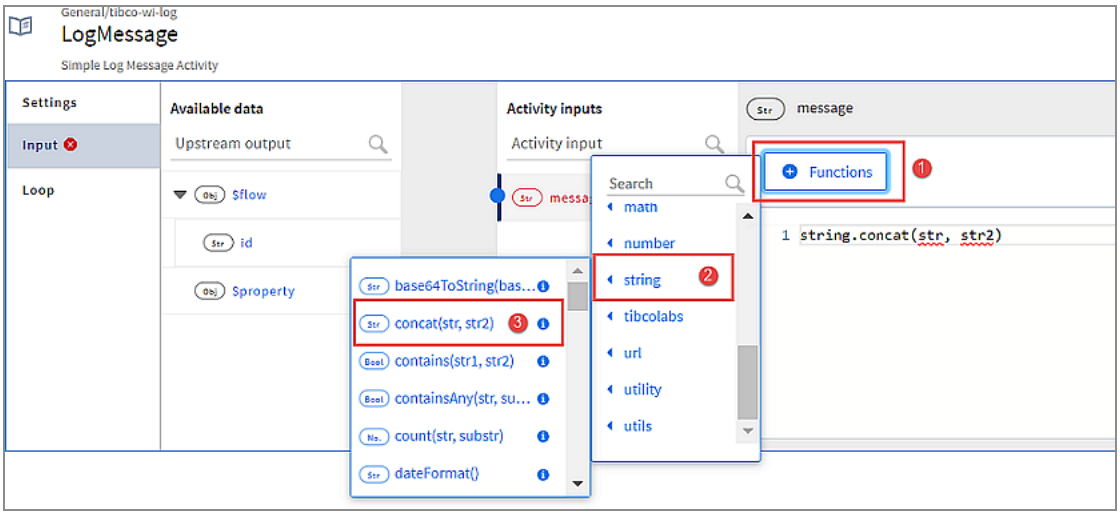
You can use a function from the list of functions available under **Functions** in the mapper. Input parameters to the function can either be mapped from an element under **Available data**, a literal value, or an expression that evaluates to the appropriate data type or any combination of them.

The following procedure illustrates an example that concatenates two strings and assigns the concatenated value to the **message**. We manually enter a value for the first string

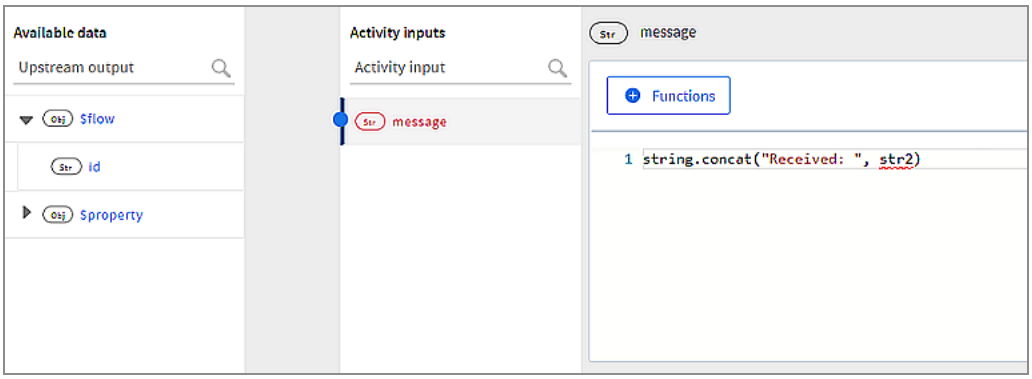
(**str1**) and map the second string to **id** under **$flow**. The value for **id** comes from the flow input.

Procedure

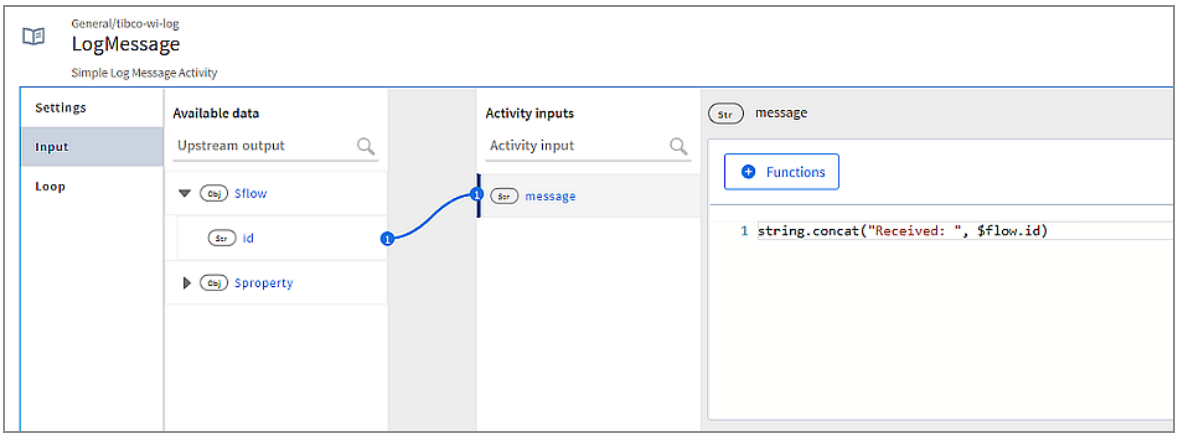
1. Click the **message** to open the text editor to the right.
2. Click **Functions**. Expand the **string** function group and click **concat(str1, str2)**.



3. Select **str1** in the function and type `"Received: "` (be sure to include the double quotes as shown) to replace **str1** with it.



4. Drag **id** from **$flow** and drop it in place of **str2**.



At run time, the output from the `concat` function is mapped to the **message**.

Array Functions

This section provides a quick reference to the Flogo array functions.

array.append

This function appends an item to an existing array.

Syntax

```
array.append(items, item)
```

Arguments

Argument	Type	Description
items	array of type any	Existing array.
item	any	Item to be appended to the existing array.

Returns

Type	Description
array of type any	Existing array with new item appended to it.

Examples

- The function `array.append($Activity[xxx].array, "new Item")` returns

`[$Activity.array, "new Item"]`.

- The function `array.append($flow.body.colors, "Red")` returns `($flow.body.colors, "Red")`.

array.contains

This function checks whether the specified item is found in the specified array.

Syntax

```
array.contains(array, item)
```

Arguments

Argument	Type	Description
array	array of type any	The array in which the specified item needs to be searched.
item	any	The item to be searched for. NOTE: The type of the item must match the type of the array.

Returns

Type	Description
boolean	• True: if the item is found in the specified array • False: if the item is not found in the specified array

Examples

The function `array.contains(array.create("A","B","C"), "B")` returns `true`.

array.count

This function counts the length of an array.

Syntax

```
array.count(items)
```

Arguments

Argument	Type	Description
items	array of type any	Array whose length needs to be counted.

Returns

Type	Description
int	Number that indicates the length of the array.

Examples

- The function `array.count($Activity.array)` returns `2`.
- The function `array.count($flow.body.colors)` returns `3`.

array.create

This function creates an array of primitive types from the specified items. All items must be of the same primitive types and must match the field type where this function is used. For example, string, integer.

Syntax

```
array.create(item1, item2)
```

Arguments

Argument	Type	Description
item1, item2	any	Items from which the array needs to be created

Returns

Type	Description
array of type any	An array of primitive types created from the specified entry.

Examples

The function `array.create("A","B","C")` returns `["A","B","C"]`.

array.delete

This function deletes a specific index of an item in the array.

Syntax

```
array.delete(items, index)
```

Arguments

Argument	Type	Description
items	array of type any	Array from which the specified index of an item needs to be deleted
index	int	Index of an item that needs to be deleted

Returns

Type	Description
array of type any	An array with the specified item deleted from the array.

Examples

The function `array.delete(array.create("item1","item2"), 1)` returns `[item1]`.

array.flatten

This function creates an array with all the subarray elements concatenated into it recursively up to the specified depth.

Syntax

```
array.flatten(items, -1)
```

Arguments

Argument	Type	Description
items	array of type any	The array that you want to flatten.
depth	int	The default depth is -1, which indicates no depth limit.

Returns

Type	Description
array of type any	The flattened array

Examples

The function `array.flatten(array.create(1, 2, array.create(3,4, array.create(5,6))), -1)` returns `[1,2,3,4,5,6]`.

array.forEach

Using this function, you can iterate over a source array and filter the array elements based on a condition. If you do not provide any arguments, you can also create a single-item array.

Important: See the Mapping Complex Arrays for a detailed explanation of using the `array.forEach()` function.

Syntax

```
array.forEach(items, scopeName, filter)
```

Arguments

If you do not provide any arguments, you can create a single-item array.

Argument	Type	Description
items	array	Source array to iterate over.
scopeName		Scope for your mapping. Each scope has a name to it. By default, the scope name is the same as the name of the source array. To use data: • To use parent data in the child array mappings, provide a scope name as a second argument in the corresponding parent `array.forEach` mapping and then use `$loop[PARENT_SCOPE_NAME].<DATA_ATTRIBUTE>` to access the corresponding data. • To use data from the current array, use `$loop.<DATA_ATTRIBUTE>`.
filter		Condition to filter the array elements. To use a filter, provide a scope name as the second argument and a boolean condition as a third argument. In the filter condition, you can use data from: • The parent array using the scope variable (for example, `$loop[PARENT_SCOPE_NAME].<DATA_ATTRIBUTE>`). • The current array (for example `$loop.<DATA_ATTRIBUTE>`).

Returns

Type	Description
array	The output array with filtered elements

Examples

- `array.forEach($flow.parameters.headers)` returns array
- `array.forEach()` returns array
- `array.forEach($flow.parameters.headers, "headerScope")` returns array
- `array.forEach($flow.parameters.headers, "headerScope", $loop.Accept == "application/json")` returns array

array.get

This function gets an item, specified by its index, from an array.

Syntax

```
array.get(items, index)
```

Arguments

Argument	Type	Description
items	array of type any	Input array of any type
index	int	Array index/position you want to get the value of

Returns

Type	Description
any	The array item present at the specified index/position

Examples

The function `array.get(array.create("item1","item2"), 1)` returns `item2`.

array.merge

This function merges multiple arrays into one.

Syntax

```
array.merge(items1, items2…)
```

Arguments

Argument	Type	Description
items1	array of type any	Input array of type any
items2	array of type any	Input array of type any

Returns

Type	Description
array	Merged array of type any.

Examples

The function `array.merge(array.create(1,2), array.create(3,4))` returns `[1,2,3,4]`.

array.reverse

This function reverses array elements.

Syntax

```
array.reverse(items)
```

Arguments

Argument	Type	Description
items	array of type any	Input array of type any

Returns

Type	Description
array of type any	Reversed array

Examples

The function `array.reverse(array.create(1,2))` returns `[2,1]`.

array.slice

This function extracts a part an array by specifying a half-open range with start index and an end index (the element at the end index is excluded).

Syntax

```
array.slice(items, start, end)
```

Arguments

Argument	Type	Description
items	array of type any	Input array.
start	int	The start index/position from which you want to extract the items from the input array.
end	int	The end index/position up to which you want the items to be extracted from the input array.

Returns

Type	Description
array of type any	The extracted array with elements between the start and end position

Examples

The function `array.slice(array.create(1,2,3,4,5), 1, 3)` returns `[2,3]`.

array.sum

This function sums all element of a number array.

Syntax

```
array.sum(items)
```

Arguments

Argument	Type	Description
items	Array of type number	Input array with elements of number type.

Returns

Type	Description
float64	Sum of all the elements in the array.

Examples

The function `array.sum(array.create(1,2))` returns `3`.

Boolean Functions

This section provides a quick reference to the Flogo boolean functions.

boolean.false

This function always returns false.

Syntax

```
boolean.false()
```

Arguments

None

Returns

Type	Description
boolean	Always returns false

Examples

The function `boolean.false()` returns `false`.

boolean.not

This function returns the reverse value of a boolean.

Syntax

```
boolean.not(bool)
```

Arguments

Argument	Type	Description
Bool	boolean	Boolean for which the reverse value needs to be returned.

Returns

Type	Description
boolean	Returns the reverse value of a boolean.

Examples

The function `boolean.not(boolean.true())` returns `false`.

boolean.true

This function always returns true.

Syntax

```
boolean.true()
```

Arguments

None

Returns

Type	Description
boolean	Always returns true

Examples

The function `boolean.true()` returns `true`.

Coerce Functions

This section provides a quick reference to the Flogo coerce functions.

coerce.toArray

This function converts the specified value to an array.

Syntax

```
coerce.toArray(value)
```

Arguments

Argument	Type	Description
value	any	The value to be coerced.

Returns

Type	Description
array	The coerced array

Examples

The function `coerce.toArray($Activity[xxx].xxx)` returns a JSON array.

coerce.toBool

This function converts the specified value to a boolean.

Syntax

```
coerce.toBool(value)
```

Arguments

Argument	Type	Description
value	any	The value to be coerced.

Returns

Type	Description
bool	The coerced bool value

Examples

The function `coerce.toBool("true")` returns `true`.

coerce.toBytes

This function converts the specified value to bytes.

Syntax

```
coerce.toBytes(value)
```

Arguments

Argument	Type	Description
value	any	The value to be coerced.

Returns

Type	Description
bytes	The coerced byte value.

Examples

The function `coerce.toBytes("hello")` returns byte data.

Coerce.toFloat32

This function converts the specified value to float32.

Syntax

```
coerce.toFloat32(value)
```

Arguments

Argument	Type	Description
value	any	The value to be coerced.

Returns

Type	Description
float32	The coerced float32 value.

Examples

The function `coerce.toFloat32("3.3")` returns `3.3`.

Coerce.toFloat64

This function converts the specified value to float64.

Syntax

```
coerce.toFloat64(value)
```

Arguments

Argument	Type	Description
value	any	The value to be coerced.

Returns

Type	Description
float64	The coerced float64 value.

Examples

The function `coerce.toFloat64("3.3")` returns `3.3`.

coerce.toInt

This function converts the specified value to an int.

Syntax

```
coerce.toInt(value)
```

Arguments

Argument	Type	Description
value	any	The value to be coerced.

Returns

Type	Description
int	The coerced int value

Examples

The function `coerce.toInt("333")` returns `333`.

Coerce.toInt32

This function converts the specified value to an int32.

Syntax

```
coerce.toInt32(value)
```

Arguments

Argument	Type	Description
value	any	The value to be coerced.

Returns

Type	Description
int32	The coerced int32 value.

Examples

The function `coerce.toInt32("333")` returns `333`.

Coerce.toInt64

This function converts the specified value to int64.

Syntax

```
coerce.toInt64(value)
```

Arguments

Argument	Type	Description
value	any	The value to be coerced.

Returns

Type	Description
int64	The coerced int64 value.

Examples

The function `coerce.toInt64("333")` returns `333`.

coerce.toObject

This function converts the specified value to an object type.

Syntax

```
coerce.toObject(value)
```

Arguments

Argument	Type	Description
value	any	The value to be coerced.

Returns

Type	Description
object	The coerced object.

Examples

The function `coerce.toObject($Activity[xxx].xxx)` returns a JSON object.

coerce.toParams

This function converts the specified value to params type.

Syntax

```
coerce.toParams(value)
```

Arguments

Argument	Type	Description
value	any	The value to be coerced.

Returns

Type	Description
params	The coerced params value.

Examples

The function `coerce.toParams($Activity[xxx].xxx)` returns a string with the name-value mapping.

coerce.toString

This function converts the specified value to a string.

Syntax

```
coerce.toString(value)
```

Arguments

Argument	Type	Description
value	Any	The value to be coerced.

Returns

Type	Description
string	The coerced string value.

Examples

The function `coerce.toString(123)` returns "`123`".

coerce.toType

This function converts a value to a specified type.

Syntax

```
coerce.toType(value, type)
```

Arguments

Argument	Type	Description
value	Any	The value to be coerced.
type	string	The data type to coerce to.

Returns

Type	Description
Any	The coerced value.

Examples

The function `coerce.toType("123","int")` returns `123`.

Compression Functions

This section provides a quick reference to the following compression functions:

- compression.gzipCompress
- compression.gzipUncompress

compression.gzipCompress

This function takes a string (stringified JSON/XML, base64 encoded binary) as an input and returns a compressed string using the GZip compression algorithm.

Syntax

```
compression.gzipCompress (str)
```

Arguments

Argument	Type	Description
str	string	A string that should be compressed.

Returns

Type	Description
string	Compressed string.

Examples

```
compression.gzipCompress("Hello World").
```

The output is compressed data in an unreadable format.

compression.gzipUncompress

This function takes a GZip compressed string as input and returns a decompressed string using the GZip decompression algorithm.

Syntax

```
compression.gzipUncompress (compressed Str)
```

Arguments

Argument	Type	Description
compressed Str	string	Compressed string that should be decompressed.

Returns

Type	Description
string	Decompressed string.

Examples

`compression.gzipUncompress("GZipCompressedString")`

The output is an uncompressed string in a readable format.

Data Functions

This section provides a quick reference to the Flogo data functions.

data.GetValue

This function returns the value from the mapping reference. If it is not defined, it returns the default value specified by the user.

Syntax

```
data.GetValue(mappingReference, default)
```

Arguments

Argument	Type	Description
mappingReference	Any	Variable name or path. For example: `$Activity[ActivityName].propertyName`
default	Any	The default value specified by the user. If the mapping reference is not defined, this default value is returned.

Returns

Type	Description
Any	The value of the mapping reference or default value

Examples

- If `$Activity[foo].book[2].price` is equal to `30`, the function `GetValue($Activity[foo].book[2].price, 90)` returns `30`.
- If `$.foo.store.addr` is not defined, the function `GetValue($.foo.store.addr, "Some address")` returns `"Some address"`.

data.isDefined

This function checks whether the specified mapping reference is defined or not.

Syntax

```
data.isDefined(mappingReference)
```

Arguments

Argument	Type	Description
mappingReference	any	Path to Activity output or app property. For example: `$Activity[ActivityName].propertyName`

Returns

Type	Description
boolean	• True: if mapping reference is defined • False: if mapping reference is not defined

Examples

- If `$Activity[foo].book[2].price` is equal to `30`, the function `isDefined($Activity`

`[foo].book[2].price)` returns `true`.

- If `$.foo.store.addr` is not defined, the function `isDefined($.foo.store.addr)` returns `false`.

Datetime Functions

This section provides a quick reference to the Flogo datetime functions.

datetime.add

This function adds the specified number of years, months, and days to the specified datetime.

Syntax

```
datetime.add(datetime, 0, 0, 0)
```

Arguments

Argument	Type	Description
datetime	datetime	Specify the date and time.
years	int	Years to be added. Default is 0.
months	int	Months to be added. Default is 0.
days	int	Days to be added. Default is 0.

Returns

Type	Description
datetime	Date and time after adding the specified number of years, months, and days.

Examples

- The function `datetime.add(datetime.current(),0,0,5)` returns the date and time of 5 days later.
- The function `datetime.add(datetime.current(), 1, 0, 0)` returns the date and time 1 year from the current date and time such as `2023-05-11T15:38:55.34586567Z`.

datetime.addHours

This function adds the specified number of hours to the mentioned datetime.

Syntax

```
datetime.addHours(datetime, 0)
```

Arguments

Argument	Type	Description
datetime	datetime	Date and time to which hours need to be added.
hours	int	Number of hours to be added. The default is 0.

Returns

Type	Description
datetime	Date and time after adding the specified number of hours.

Examples

The function `datetime.addHours(datetime.current(), 3)` returns the date and time of 3 hours later.

datetime.addMins

This function adds the specified number of minutes to the mentioned datetime.

Syntax

```
datetime.addMins(datetime, 0)
```

Arguments

Argument	Type	Description
datetime	datetime	Date and time to which minutes need to be added.
mins	int	Number of minutes to be added. The default is 0.

Returns

Type	Description
datetime	Returns the date and time after adding 30 minutes.

Examples

The function `datetime.addMins(datetime.current(), 30)` returns the date and time of 30 minutes later.

datetime.addSeconds

This function adds the specified number of seconds to the mentioned datetime.

Syntax

```
datetime.addSeconds(datetime, 0)
```

Arguments

Argument	Type	Description
datetime	datetime	Date and time to which the seconds need to be added.
seconds	int	Number of seconds to be added. The default is 0.

Returns

Type	Description
datetime	Date and time with the 30 seconds added.

Examples

The function `datetime.addSeconds(datetime.current(), 30)` returns the date and time of 30 seconds later.

datetime.create

This function creates a date by using the specified details. The details such as year, months, days, hours, minutes, seconds, nanoseconds, and timezone. The timezone follows the IANA time zone.

Syntax

```
datetime.create(0, 0, 0, 0, 0, 0, 0, UTC)
```

Arguments

Argument	Type	Description
years	int	The default is 0.
months	int	The default is 0.
days	int	The default is 0.
hours	int	The default is 0.
mins	int	The default is 0.
secs	int	The default is 0.
nsecs	int	The default is 0.
location	string	The default is UTC.

Returns

Type	Description
datetime	A date created by using the specified details. The details such as year, months, days, hours, mins, seconds, nanoseconds, and timezone.

Examples

The function `datetime.create(2020,1,3,2,22,0,0, "America/Los_Angeles")` returns `2020-01-03T02:22:00-08:00`.

datetime.current

This function returns the current datetime in the UTC timezone.

Syntax

```
datetime.current()
```

Arguments

None

Returns

Type	Description
datetime	Current datetime in the UTC timezone.

Examples

The function `datetime.current()` returns the current date and time.

datetime.currentDate

This function returns the current date in the UTC time zone.

Syntax

```
datetime.currentDate()
```

Arguments

None.

Returns

Type	Description
string	Current date in the UTC time zone.

Examples

The function `datetime.currentDate()` returns the current date in the UTC time zone.

datetime.currentDatetime

This function returns the current datetime in the UTC timezone.

Syntax

```
datetime.currentDatetime()
```

Arguments

None

Returns

Type	Description
string	Current datetime in the UTC timezone.

Examples

The function `datetime.currentDatetime()` returns the current datetime in the UTC timezone.

datetime.currentTime

This function returns the current time in the UTC timezone.

Syntax

```
datetime.currentTime()
```

Arguments

None

Returns

Type	Description
string	Current time in the UTC timezone.

Examples

The function `datetime.currentTime()` returns the current time in the UTC timezone.

datetime.diff

This function returns the difference between two specified datetime values.

Syntax

```
datetime.diff(start, end, type)
```

Arguments

Argument	Type	Description
start	datetime	Start date and time.
end	datetime	End date and time.
type	string	The difference is controlled by the argument type, in (days, hours, mins, and seconds).

Returns

Type	Description
float64	Difference between the two specified datetime values.

Examples

The function `datetime.diff(datetime.current(), datetime.addHours (datetime.current(), 2), "hours")` returns `2`.

datetime.format

This function displays the date according to the specified format. The format uses MM (month), DD(day), YYYY(year), hh(hour), mm(minute), and ss(second). Except MM, the rests are not case-sensitive. The datetime format can also be set with a following predefined layout:

- ANSIC
- Unix Date
- RubyDate
- RFC822
- RFC822Z

- RFC850
- RFC1123
- RFC1123Z
- RFC3339,
- RFC3339Nano

Syntax

```
datetime.format(datetime, format)
```

Arguments

Argument	Type	Description
datetime	datetime	Date and time to be formatted.
format	string	Format to be used for the date and time.

Returns

Type	Description
string	Date and time in the specified format.

Examples

The function `datetime.format(datetime.current(), "RFC3339")` returns the current date and time in RFC3339 format.

datetime.formatDate

This function displays the date according to the specified format. The format uses MM (month), DD(day), and YYYY(year) and they are not case-sensitive.

Syntax

```
datetime.formatDate(date, format)
```

Arguments

Argument	Type	Description
date	datetime	Date and time to be formatted.
format	string	Format to be used for the date and time.

Returns

Type	Description
string	Date and time in the specified format.

Examples

The function `datetime.formatDate("02/08/2017", "dd-MM-yyyy")` returns `08-02-2017`.

datetime.formatDatetime

This function displays the date and time according to the specified format. The format uses MM(month), DD(day), YYYY(year), hh(hour), mm(minute), and ss(second). Except for MM, the others are not case-sensitive.

Syntax

```
datetime.formatDatetime(datetime, format)
```

Arguments

Argument	Type	Description
datetime	datetime	Date and time to be formatted.
format	string	Format to be used for the date and time.

Returns

Type	Description
string	Datetime in the specified format.

Examples

The function `datetime.formatDatetime("2017-04-10T22:17:32.000+0700", "dd/MM/yyyy T hh:mm:ss")` returns `10/04/2017 T 22:17:32`.

datetime.formatTime

This function formats the time according to the specified format. The format uses hh(hour), mm(minute), ss(second), and is not case-sensitive.

Syntax

```
datetime.formatTime(datetime, format)
```

Arguments

Argument	Type	Description
datetime	datetime	Time that needs to be formatted.
format	string	Format to be used for the time.

Returns

Type	Description
string	Time in the specified format.

Examples

The function `datetime.formatTime("10:11:05.00000", "hh-mm-ss")` returns `10-11-05`.

datetime.now

This function returns the current date and time in the UTC timezone.

Syntax

```
datetime.now()
```

Arguments

None

Returns

Type	Description
string	Current date and time in the UTC timezone.

Examples

The function `datetime.now()` returns something like `2020-03-19T15:02:03+06:00`.

datetime.parse

This function parses the given datetime to the DateTime with time zone, default base on UTC. The timezone follows the IANA timezone.

Syntax

```
datetime.parse(str, UTC)
```

Arguments

Argument	Type	Description
str	any	Date and time with time zone.
timezone	string	The timezone in which the string needs to be parsed. The default is UTC.

Returns

Type	Description
datetime	Datetime converted to the specified timezone.

Examples

The function `datetime.parse("2020-03-19T15:02:03+06:00", "America/Los_Angeles")` returns `2020-03-19T02:02:03-07:00`.

Datetime.sub

This function subtracts the given number of years, months, or days from the datetime.

Syntax

```
datetime.sub(datetime, 0, 0, 0)
```

Arguments

Argument	Type	Description
datetime	datetime	Specify date and time.
years	int	Years to be subtracted. The default is 0.
months	int	Months to be subtracted. The default is 0.
days	int	Days to be subtracted. The default is 0.

Returns

Type	Description
datetime	Date and time after subtracting the specified years, months, and days.

Examples

The function `datetime.sub(datetime.current(), 1,1,1)` returns the date and time of 1 year, 1 month, and 1 day ago.

datetime.subHours

This function subtracts the given number of hours from the DateTime.

Syntax

```
datetime.subHours(datetime, hours)
```

Arguments

Argument	Type	Description
datetime	datetime	Specify the date and time.
hours	int	Hours to be subtracted.

Returns

Type	Description
datetime	Date and time after subtracting the specified number of hours.

Examples

The function `datetime.subHours(datetime.current(),1)` returns the date and time of 1 hour ago.

datetime.subMins

This function subtracts the given number of minutes from the datetime.

Syntax

```
datetime.subMins(datetime, mins)
```

Arguments

Argument	Type	Description
datetime	datetime	Specify the date and time.
mins	int	Minutes to be subtracted. Default is 0.

Returns

Type	Description
datetime	Date and time after subtracting the specified minutes.

Examples

The function `datetime.subMins(datetime.current(),10)` returns the date and time 10 minutes ago.

Datetime.subSeconds

This function subtracts the given number of seconds from the datetime.

Syntax

```
datetime.subSeconds(datetime, seconds)
```

Arguments

Argument	Type	Description
datetime	datetime	Specify the date and time.
seconds	int	Seconds to be subtracted.

Returns

Type	Description
datetime	Date and time after subtracting the specified seconds.

Examples

The function `datetime.subSeconds(datetime.current(),10)` returns what the date and time was 10 seconds ago.

Float Functions

This section provides a quick reference to the Flogo float functions.

float.float64

This function converts the input to a float64 with the specified (optional) floating point precision.

Syntax

```
float.float64(input, 16)
```

Arguments

Argument	Type	Description
input		Input to be converted to float64.
precision	int	(Optional) The default precision value is 16. Any precision value greater than 16, defaults to 16.

Returns

Type	Description
float64	Input converted to float64.

Examples

The function `float.float64("2.77876542316664548335",14)` returns `2.77876542316664`.

JSON Functions

This section provides a quick reference to the Flogo JSON functions.

json.exists

This function checks whether the key or JSONPath is present in the JSON object. For the expression format, see https://github.com/oliveagle/jsonpath.

Syntax

```
json.exists(jsonObject, key)
```

Arguments

Argument	Type	Description
jsonObject	any	The JSON object.
key	string	The key or JSONPath to check

Returns

Type	Description
boolean	True, if the value is associated with the key or JSONPath. False otherwise.

Examples

Let us say the jsonObject includes the following.

```
jsonObject =
{
 "colors":
  [
     { "id": 1, "value": "red" },
     { "id": 2, "value": "green" },
     { "id": 3, "value": "blue" }
  ]
}
```

In this case:

- The function `json.exists(jsonObject, "$loop.colors[?(@.value == 'red')].value[0]")` returns `true`.
- The function `json.exists(jsonObject, "$loop.colors[?(@.value == 'yellow')].value[0]")` returns `false`.

json.get

This function gets the value of the associated key from the JSON object.

Syntax

```
json.get(jsonObject, key)
```

Arguments

Argument	Type	Description
jsonObject	any	The JSON object.
key	string	The key for which the value to get

Returns

Type	Description
any	The value associated with the key

Examples

Let us say the jsonObject includes the following.

```
jsonObject = {
"id": 1,
"name": "bob"
}
```

The function `json.get(jsonObject, "name")` returns "bob".

json.length

This function gets the number of top-level elements in the JSON object or array.

Syntax

```
json.length(jsonObjectOrArray)
```

Arguments

Argument	Type	Description
jsonObjectorArray	any	The JSON object or array.

Returns

Type	Description
int	The top-level items in the JSON object

Examples

Let us say the jsonObject includes the following.

```
jsonObject = {
"id": 1,
"name": "bob"
}
```

The function `json.length(jsonObject)` returns 2.

json.numbersToString

This function converts every number type to string in a JSON object or array.

Syntax

```
json.numbersToString(jsonObjectOrArray)
```

Arguments

Argument	Type	Description
jsonObjectorArray	any	The JSON object or array.

Returns

Type	Description
any	The JSON object or array with numbers encoded as strings.

Examples

Let us say the jsonObject includes the following:

```
jsonObject = {
"id": 1,
"name": "bob"
}
```

The `json.numbersToString(jsonObject)` returns the following:

```
{
"id": "1",
"name": "bob"
}
```

The number 1 is returned as a string.

json.objKeys

This function gets the list of all top-level keys of a JSON object.

Syntax

```
json.objKeys(jsonObject)
```

Arguments

Argument	Type	Description
jsonObject	any	The JSON object.

Returns

Type	Description
array	The list of top-level keys.

Examples

Let us say the jsonObject includes the following.

```
jsonObject = {
"id": 1,
"name": "bob"
}
```

The function `json.objKeys(jsonObject)` returns `["id", "name"]`.

json.objValues

This function gets the list of all top-level values of a JSON object.

Syntax

```
json.objValues(jsonObject)
```

Arguments

Argument	Type	Description
jsonObject	any	The JSON object.

Returns

Type	Description
array	The list of all top-level values in the JSON object.

Examples

Let us say the jsonObject includes the following:

```
jsonObject = {
"id": 1,
"name": "bob"
}
```

The function `json.objValues(jsonObject)` returns `[1, "bob"]`.

json.path

Using this function you can query an element within JSON data. To reach the desired node or a specific field in the node in the JSON data, you must follow a specific notation defined in the JsonPath specification. See https://jqlang.github.io/jq/manual/ for details on the notation to be used and specific examples of using the notation.

Important: See the Mapping JSON Data with the json.path() Function for a detailed explanation on using the `json.path()` function.

Syntax

```
json.path(path, object)
```

Arguments

Argument	Type	Description
path	string	The search path to the element within the JSON data.
object	any	The JSON object that contains the JSON data you are searching.

Returns

Type	Description
any	The result of the JSON path.

Examples

Let us say the jsonObject includes the following:

```
jsonObject = {
"colors": [
{ "id": 1, "value": "red" },
{ "id": 2, "value": "green" },
{ "id": 3, "value": "blue" }
]
}
```

The function `json.path("$loop.colors[?(@.id == 2)].value[0]", jsonObject)` returns `green`.

json.jq

This function applies a query to the parsed JSON object and returns a filtered array. For more information, see https://jqlang.github.io/jq/manual/.

Syntax

```
json.jq(JSONobject, "query")
```

Arguments

Argument	Type	Description
`jsonObject`	any	The JSON object.
`query`	string	The query using which the JSON object is filtered.

Returns

Type	Description
`array`	The filtered JSON array is the result.

Examples

Let us say the `jsonObject` includes the following:

```
jsonObject =
{
 "colors":
  [
     { "id": 1, "value": "red" },
     { "id": 2, "value": "green" },
     { "id": 3, "value": "blue" }
  ]
}
```

In this case, the `json.jq(jsonObject, ".colours| .[0] |.value")` function returns `"red"`.

json.set

This function sets the value of the existing key or add new key and set its value in a JSON object.

Syntax

```
json.set(jsonObject, key, value)
```

Arguments

Argument	Type	Description
jsonObject	any	The JSON object.
key	string	The key for value
value	any	The value for the key

Returns

Type	Description
any	The updated JSON object.

Examples

Let us say the jsonObject includes the following:

```
jsonObject = {
"colors": [
```

```
{ "id": 1, "value": "red" },
{ "id": 2, "value": "green" },
{ "id": 3, "value": "blue" }
]
}
```

The function `json.set(jsonObject, "timestamp", 1652187627)` returns the following:

```
{
"colors": [
{ "id": 1, "value": "red" },
{ "id": 2, "value": "green" },
{ "id": 3, "value": "blue" }
],
"timestamp": 1652187627
}
```

Math Functions

This section provides a quick reference to the Flogo math functions.

math.ceil

This function returns the least integer value greater than or equal to the input.

Syntax

```
math.ceil(inputNumber)
```

Arguments

Argument	Type	Description
inputNumber	number	The input number

Returns

Type	Description
number	The ceil value of the input number

Examples

The function `math.ceil(1.49)` returns `2`.

math.floor

This function returns the greatest integer value less than or equal to the input.

Syntax

```
math.floor(inputNumber)
```

Arguments

Argument	Type	Description
inputNumber	number	The input number

Returns

Type	Description
number	The floor value of the input number

Examples

The function `math.floor(1.51)` returns 1.

math.isNaN

This function reports whether the input is an IEEE 754 "not-a-number" value.

Syntax

```
math.isNaN(input)
```

Arguments

Argument	Type	Description
input	any	The input to test

Returns

Type	Description
boolean	True, if the input is IEEE 754 "not-a-number".

Examples

The function `math.isNaN(1.0)` returns `false`.

math.mod

This function returns the floating-point remainder of x/y. The magnitude of the result is less than y and its sign agrees with that of x.

Syntax

```
math.mod(x, y)
```

Arguments

Argument	Type	Description
x	number	The dividend or first input
y	number	The divisor or second input

Returns

Type	Description
number	The remainder of the Euclidean division of x by y

Examples

The function `math.mod(7, 4)` returns `3.0`.

math.round

This function returns the nearest integer, rounding half away from zero.

Syntax

```
math.round(inputNumber)
```

Arguments

Argument	Type	Description
inputNumber	number	The input number

Returns

Type	Description
number	The nearest integer, rounding half away from zero

Examples

The function `math.round(1.50)` returns `2.0`.

math.roundToEven

This function returns the nearest integer, rounding ties to even.

Syntax

```
math.roundToEven(inputNumber)
```

Arguments

Argument	Type	Description
inputNumber	number	The input number

Returns

Type	Description
number	The nearest integer, rounding ties to even

Examples

The function `math.roundtoeven(12.5)` returns `12.0`.

math.trunc

This function returns the integer value of the input.

Syntax

```
math.trunc(inputNumber)
```

Arguments

Argument	Type	Description
inputNumber	number	The input number

Returns

Type	Description
number	The truncated integer value of the input

Examples

The function `math.trunc(3.142)` returns `3.0`.

Number Functions

This section provides a quick reference to the Flogo number functions.

number.int64

This function converts the input to an integer.

Syntax

```
number.int64(input)
```

Arguments

Argument	Type	Description
input		Input to be converted to an integer.

Returns

Type	Description
int	Input converted to an integer.

Examples

The function `number.int64("123")` returns `123`.

number.len

This function returns the length of a string.

Syntax

```
number.len(input)
```

Arguments

Argument	Type	Description
input	string	String whose length needs to be counted.

Returns

Type	Description
int	Number indicating the length of the string.

Examples

The function `number.len("123")` returns `3`.

number.random

This function generates a random number.

Syntax

```
number.random(limit)
```

Arguments

Argument	Type	Description
limit	int	The maximum number value.

Returns

Type	Description
int	A non-negative pseudo-random number.

Examples

The function `number.random(100)` returns `90`.

String Functions

This section provides a quick reference to the Flogo string functions.

String.base64ToString

This function decodes a base64-encoded string and returns the decoded string.

Syntax

```
string.base64ToString(base64str)
```

Arguments

Argument	Type	Description
base64str	string	Base64-encoded string to be decoded.

Returns

Type	Description
string	Decoded, human-readable version of the base64-encoded string.

Examples

The function `string.base64ToString("SGVsbG8sIFdvcmxk")` returns a string like `Hello, World`.

string.concat

This function concatenates a set of strings.

Syntax

```
string.concat(str1, str2)
```

Arguments

Argument	Type	Description
str1, str2	string	Strings to be concatenated.

Returns

Type	Description
string	A concatenated string.

Examples

The function `string.concat("Hello",' ', "World")` returns `Hello World`.

string.contains

This function checks whether a specified string exists within another string.

Syntax

```
string.contains(str1, str2)
```

Arguments

Argument	Type	Description
str1	string	Source string
str2	string	String to search for in the source string (str1)

Returns

Type	Description
boolean	• True: if str2 is found within str1 • False: if str2 is not found within str1

Examples

The function `string.contains("foobar","foo")` returns `true`.

string.containsAny

This function reports whether any Unicode code points in characters exist within the input settings.

Syntax

```
string.containsAny(str, substr)
```

Arguments

Argument	Type	Description
str	string	Source string
substr	string	Substring to be searched for in the source string

Returns

Type	Description
boolean	• True: if the substring exists in the source string • False: if the substring does not exist in the source string

Examples

The function `string.containsAny("Seafood", "food")` returns `true`.

string.count

This function counts the number of non-overlapping instances of a substring in the input string.

Syntax

```
string.count(str, substr)
```

Arguments

Argument	Type	Description
str	string	Source string
substr	string	Substring whose instances must be searched for in the source string

Returns

Type	Description
int	Number of instances of the substring in the string. If a substring is an empty string, the function returns 1 + the number of Unicode points in the input string.

Examples

The function `string.count("hello flogo", "o")` returns `3`.

string.dateFormat

This function returns a default date format.

Syntax

```
string.dateFormat()
```

Arguments

None

Returns

Type	Description
string	"YYYY-MM-DD"

Examples

The function `string.dateFormat()` returns `2006-01-02-07:00`.

string.lastIndex

This function returns the index of the last instance of substring in the specified input string.

Syntax

```
string.lastIndex(str, substr)
```

Arguments

Argument	Type	Description
str	string	Source string.
substr	string	Substring for which the index of the last instance needs to be searched.

Returns

Type	Description
int	Index of the last instance of substring in the source string. -1 if the substring is not present in the source string.

Examples

The function `string.lastIndex("hello flogo", "flogo")` returns `6`.

string.datetimeFormat

This function returns a default datetime format.

Syntax

```
string.datetimeFormat()
```

Arguments

None

Returns

Type	Description
string	“YYYY-MM-DDTHH:mm:ss”

Examples

The function `string.datetimeFormat()` returns `2006-01-02T15:04:05-07:00`.

string.endsWith

This function indicates whether a string ends with another string.

Syntax

```
string.endsWith(str, substr)
```

Arguments

Argument	Type	Description
str	string	Source string
substr	string	Ending string that needs to be checked for

Returns

Type	Description
boolean	• True: if the source string ends with the substring • False: if the source string does not end with the substring

Examples

The function `string.endsWith("This is a project created in Flogo", "Flogo")` returns `true`.

string.equals

This function checks whether two strings are equal.

Syntax

```
string.equals(str1, str2)
```

Arguments

Argument	Type	Description
str1, str2	string	Strings to be compared

Returns

Type	Description
boolean	True: if the strings are equal False: if the strings are not equal

Examples

The function `string.equals("Hello","Hello2")` returns `false`.

string.equalsIgnoreCase

This function checks whether two strings are equal, ignoring the case.

Syntax

```
string.equalsIgnoreCase(str1, str2)
```

Arguments

Argument	Type	Description
str1, str2	string	Strings to be compared.

Returns

Type	Description
boolean	True: if the strings are equal False: if the strings are not equal

Examples

The function `string.equalsIgnoreCase("Hello","hello")` returns `true`.

string.float

This function converts the string to a float.

Syntax

```
string.float(str1, precision)
```

Arguments

Argument	Type	Description
str1	string	Source string
precision	number	(Optional) The default is 16.

Returns

Type	Description
float64	The float value of str1

Examples

The function `string.float("2.77876542316664548335",14)` returns `2.77876542316665`.

string.index

This function returns the index of the first instance of a substring in the input string.

Syntax

```
string.index(str, substr)
```

Arguments

Argument	Type	Description
str	string	Source string
substr	string	Substring to be searched for in the source string

Returns

Type	Description
int	• Number indicating the index of the first instance of substr in str. • -1 if substr is not present in str

Examples

The function `string.index("hello flogo", "flogo")` returns `6`.

string.indexAny

In the input string, this function returns the index of the first instance of any Unicode code point form characters.

Syntax

```
string.indexAny(str, substr)
```

Arguments

Argument	Type	Description
str	string	Source string
substr	string	Substring to be searched for in the source string

Returns

Type	Description
int	-1: if no Unicode code point form characters are present in the input string

Examples

The function `string.indexAny("hello flogo", "flogo")` returns `2`.

string.integer

This function converts the string to an integer.

Syntax

```
string.integer(str1)
```

Arguments

Argument	Type	Description
Str1	string	Source string

Returns

Type	Description
int	The integer value of the source string, str1

Examples

The function `string.integer("1001")` returns `1001`.

string.join

This function converts an array to its string representation and concatenates its elements using the specified separator between each element.

Syntax

```
string.join(items, separator)
```

Arguments

Argument	Type	Description
items	array	Elements of array
separator	string	Separator to be used between each array element

Returns

Type	Description
string	Concatenated string representation of the array with the specified separator between each array element.

Examples

The function `string.join(array.create("a", "b"), "-")` returns `a-b`.

string.len

This function gets the length of a string.

Syntax

```
string.len(str1)
```

Arguments

Argument	Type	Description
Str1	string	Source string

Returns

Type	Description
int	The length of the source string, str1

Examples

The function `string.len("hello")` returns `5`.

string.length

This function returns the length of a string.

Syntax

```
string.length("str")
```

Arguments

Argument	Type	Description
Str	string	Source string

Returns

Type	Description
int	An integer that indicates the length of the string.

Examples

The function `string.length("FLOGO")` returns `5`.

string.lowerCase

This function returns the string in lowercase.

Syntax

```
string.lowerCase(str)
```

Arguments

Argument	Type	Description
Str	string	Source string

Returns

Type	Description
string	String in lower case.

Examples

The function `string.lowerCase("FLOGO")` returns `flogo`.

string.matchRegEx

This function matches the input against a regular expression.

Syntax

```
string.matchRegEx(expression, str)
```

Arguments

Argument	Type	Description
expression	string	Regular expression against which the source string needs to be matched.
str	string	Source string

Returns

Type	Description
boolean	• True: if the source string matches the regular expression • False: if the source string does not match the regular expression

Examples

The function `string.matchRegEx("foo.*", "seafood")` returns `true`.

string.regex

This function checks whether a regular expression pattern matches a string.

Syntax

```
string.regex(pattern, str)
```

Arguments

Argument	Type	Description
pattern	string	Regular expression pattern against which the source string needs to be matched.
str	string	Source string

Returns

Type	Description
boolean	• True: if the source string matches the regular expression pattern • False: if the source string does not match the regular expression pattern

Examples

The function `string.regex("foo.*", "seafood")` returns `true`.

string.repeat

This function returns a new string consisting of `<count>` copies of the specified string.

Syntax

```
string.repeat(str, count)
```

Arguments

Argument	Type	Description
str	string	String to be repeated
count	int	Number that indicates how many times the string needs to be repeated

Returns

Type	Description
string	New string consisting of `<count>` copies of the specified string

Examples

The function `string.repeat("hello flogo", 2)` returns `hello flogo hello flogo`.

string.replace

This function returns a copy of the input string where `<count>` non-overlapping instances of `<old>` are replaced by <new>.

Syntax

```
string.replace(str, old, new, count)
```

Arguments

Argument	Type	Description
str	string	Source string
old	string	Old string that needs to be replaced
new	string	New string that replaces the old string
count	int	Number of replacements to be done If the count is less than 0, there is no limit on the number of replacements that can be done.

Returns

Type	Description
string	String with <count> occurrences of <old> replaced by <new>

Examples

The function `string.replace("hello flogo", "flogo", "world", -1)` returns `hello world`.

string.replaceAll

This function returns a copy of the input string with all non-overlapping <old> instances are replaced by <new>.

Syntax

```
string.replaceAll(str, old, new)
```

Arguments

Argument	Type	Description
str	string	Source string
old	string	Old string that needs to be replaced
new	string	New string that replaces the old string

Returns

Type	Description
string	String with all occurrences of <old> replaced by <new>

Examples

The function `string.replaceAll("hello flogo", "flogo", "world")` returns `hello world`.

string.replaceRegEx

This function replaces data in a string based on a regular expression match.

Syntax

```
string.replaceRegEx(expression, str, replace)
```

Arguments

Argument	Type	Description
expression	string	Characters from the source string are replaced based on the regular expression match.
str	string	Source string
replace	string	String to be replaced with

Returns

Type	Description
string	New string, where characters matching the regular expression are replaced with the <replace> string

Examples

The function `string.replaceRegEx("foo.*", "seafood", "People")` returns `seaPeople`.

string.split

This function slices a string into all substrings separated by <sep> and returns a slice of the substrings between those separators.

Syntax

```
string.split(str, separator)
```

Arguments

Argument	Type	Description
str	string	Source string
separator	string	Separator

Returns

Type	Description
array	Slice of substrings between the specified separator

Examples

The function `string.split("Hello,World", ",")` returns `[Hello,World]`.

string.startsWith

This function returns whether a string begins with another string.

Syntax

```
string.startsWith(str, substr)
```

Arguments

Argument	Type	Description
str	string	Source string

Argument	Type	Description
substr	string	Substring to be searched for at the beginning of the source string

Returns

Type	Description
boolean	• True: if the substring is found at the beginning of the source string • False: if the substring is not found at the beginning of the source string

Examples

The function `string.startsWith("Project Flogo™", "Project")` returns `true`.

String.stringToBase64

This function returns a base64-encoded version of the input string.

Syntax

```
string.stringToBase64(str)
```

Arguments

Argument	Type	Description
str	string	Input string to be converted to a base64-encoded version.

Returns

Type	Description
string	Base64-encoded version of the input string.

Examples

The function `string.stringToBase64("Hello, World")` returns `SGVsbG8sIFdvcmxk`.

string.substring

This function gets a substring from a string.

Syntax

```
string.substring(str, start, end)
```

Arguments

Argument	Type	Description
str	string	Source string
start	string	The starting position or character
end	string	The ending position or character

Returns

Type	Description
string	The substring

Examples

The function `string.substring("abc", 1,1)` returns `b`.

string.substringAfter

This function returns the string that follows the first occurrence of the second argument.

Syntax

```
string.substringAfter(str, afterstr)
```

Arguments

Argument	Type	Description
str	string	Source string
afterstr	string	Characters after the first occurrence of this string need to be returned

Returns

Type	Description
string	• String that follows after the first occurrence of the second string. • Zero length string if the first string does not contain the second string.

Examples

The function `string.substringAfter("1999/04/01", "/")` returns `04/01`.

string.substringBefore

This function returns the string that comes before the first occurrence of the second string.

Syntax

```
string.substringBefore(str, beforestr)
```

Arguments

Argument	Type	Description
str	string	Source string
beforestr	string	Characters before the first occurrence of this string need to be returned

Returns

Type	Description
string	• String that comes before the first occurrence of the second string. • Zero length string if the first string does not contain the second string.

Examples

The function `string.substringBefore("1999/04/01", "/")` returns `1999`.

string.timeFormat

This function returns a default time format.

Syntax

```
string.timeFormat()
```

Arguments

None

Returns

Type	Description
string	Default time format

Examples

The function `string.timeFormat()` returns `15:04:05-07:00`.

string.toLower

This function returns a copy of input string with all Unicode letters mapped to their lower case.

Syntax

```
string.toLower(str)
```

Arguments

Argument	Type	Description
str	string	Source string

Returns

Type	Description
string	String converted to lower case

Examples

The function `string.toLower("Hello World")` returns `hello world`.

string.toTitleCase

This function returns a string after capitalizing the first letter of every word.

Syntax

```
string.toTitleCase(str)
```

Arguments

Argument	Type	Description
str	string	Source string

Returns

Type	Description
string	String after capitalizing the first letter of every word

Examples

The function `string.toTitleCase("hello world")` returns `Hello World`.

string.toUpper

This function returns a copy of input string with all Unicode letters mapped to their upper case.

Syntax

```
string.toUpper(str)
```

Arguments

Argument	Type	Description
str	string	Source string

Returns

Type	Description
string	String converted to upper case

Examples

The function `string.toUpper("Hello World")` returns `HELLO WORLD`.

string.tostring

This function converts an object to a string.

Syntax

```
string.tostring(input)
```

Arguments

Argument	Type	Description
input	object	Object to be converted to a string

Returns

Type	Description
string	String version of the object

Examples

The function `string.tostring(123)` returns "`123`".

string.trim

This function returns a part of the input string with all leading and trailing Unicode code points contained in the cutset removed.

Syntax

```
string.trim(str)
```

Arguments

Argument	Type	Description
str	string	Source string

Returns

Type	Description
string	String with all leading and trailing Unicode code points

Examples

- The function `tring.trim(" Hello World ", " ")` returns `Hello World`.
- The function `string.trim("¡¡¡Hello, Gophers!!!", "!¡")` returns `Hello Gophers`.

string.trimLeft

This function returns a slice of the input string with all the leading Unicode code points contained in cutset removed from the left.

Syntax

```
string.trimLeft(str, cutset)
```

Arguments

Argument	Type	Description
str	string	Source string
cutset	string	Set of characters that need to be removed from the left

Returns

Type	Description
string	Slice of the input string with all the leading Unicode code points contained in cutset removed from the left.

Examples

- The function `string.trimLeft(" Hello World ", " ")` returns `Hello World`.
- The function `string.trimLeft("¡¡¡Hello, Gophers!!!", "!¡")` returns `Hello, Gophers!!!`.

string.trimPrefix

This function returns the input string without the provided leading prefix string. If the input string does not start with the prefix, it is returned unchanged.

Syntax

```
string.trimPrefix(str, prefix)
```

Arguments

Argument	Type	Description
str	string	Source string
prefix	string	Leading prefix string

Returns

Type	Description
string	• String without the specified leading prefix string. • If the input string does not start with the prefix, it is returned unchanged.

Examples

- The function `string.trimPrefix("Hello World ", "Hello")` returns `World`.
- The function `string.trimPrefix("¡¡¡Hello, Gophers!!!", "¡¡¡Hello,")"` returns `Gophers!!!`.

string.trimRight

This function returns a slice of the input string with all the trailing Unicode code points contained in the cutset removed from the right.

Syntax

```
string.trimRight(str, cutset)
```

Arguments

Argument	Type	Description
str	string	Source string
cutset	string	Set of characters to be removed from the right

Returns

Type	Description
string	Slice of the source string with all trailing Unicode code points contained in the cutset removed from the right.

Examples

- The function `string.trimRight(" Hello World ", " ")` returns `Hello World`.
- The function `string.trimRight("¡¡¡Hello, Gophers!!!", "!¡")` returns `¡¡¡Hello, Gophers`.

string.trimSuffix

This function returns the input string without the specified trailing suffix string. If it does not end with a suffix, it is returned unchanged.

Syntax

```
string.trimSuffix(str, suffix)
```

Arguments

Argument	Type	Description
str	string	Source string
suffix	string	Trailing suffix string

Returns

Type	Description
string	• Input string without the specified trailing suffix string. • If the input string does not end with the specified suffix, it is returned unchanged.

Examples

- The function `string.trimSuffix("Hello World", "World")` returns `Hello`.
- The function `string.trimSuffix("¡¡¡Hello, Gophers!!!", ", Gophers!!!")` returns `¡¡¡Hello`.

string.upperCase

This function returns the string in uppercase.

Syntax

```
string.upperCase(str)
```

Arguments

Argument	Type	Description
str	string	Source string

Returns

Type	Description
string	String converted to upper case.

Examples

The function `string.upperCase("Flogo")` returns `FLOGO`.

URL Functions

This section provides a quick reference to the Flogo URL functions.

url.encode

This function returns the URL encoded form of the input string.

Syntax

```
url.encode(rawURLString)
```

Arguments

Argument	Type	Description
rawURLString	string	Raw URL string

Returns

Type	Description
string	The URL encoded string

Examples

The function `url.encode("https://subdomain.example.com/path?q=hello world#fragment with space")` returns `https://subdomain.example.com/path?q=hello+world#fragment%20with%20space`.

url.escapedPath

This function returns the escaped path part of the URL, removing everything except the PATH after the hostname.

Syntax

```
url.escapedPath(rawURLString)
```

Arguments

Argument	Type	Description
rawURLString	string	Raw URL string

Returns

Type	Description
string	Escaped path part of the URL

Examples

The function `url.escapedPath("https://example.com:8080/root-path/sub%2Fpath?query=example+query+%2F+question#fragment")` returns `"/root-path/sub%2Fpath"`.

url.hostname

This function returns the hostname for the URL, removing any valid port number if present. If the input is enclosed in square brackets, as literal IPv6 addresses are, the square brackets are removed from the output.

Syntax

```
url.hostname(rawURLString)
```

Arguments

Argument	Type	Description
rawURLString	string	Raw URL string

Returns

Type	Description
string	The hostname of the rawURLString

Examples

The function `url.hostname("https://example.com:8080/root-path/sub-path?query=example+query+%2F+question#fragment")` returns `example.com`.

url.path

This function returns the path part of the URL.

Syntax

```
url.path(rawURLString)
```

Arguments

Argument	Type	Description
rawURLString	string	Raw URL string

Returns

Type	Description
string	The path part of the URL

Examples

The function `url.path("https://example.com:8080/root-path/sub-path?query=example+query+%2F+question#fragment")` returns "`/root-path/sub-path`".

url.pathEscape

This function returns the escaped string so it can be safely placed inside a URL path segment, replacing special characters (including `/`) with `%XX` sequences as needed.

Syntax

```
url.pathEscape(pathString)
```

Arguments

Argument	Type	Description
pathString	string	The path string

Returns

Type	Description
string	The escaped PATH string

Examples

The function `url.pathEscape("/some-path with ([brackets])")` returns "`%2Fsome-path%20with%20%28%5Bbrackets%5D%29`".

url.port

This function returns the port part of the URL, without the leading colon. If the URL does not contain a valid numeric port, it returns an empty string.

Syntax

```
url.port(rawURLString)
```

Arguments

Argument	Type	Description
rawURLString	string	Raw URL string

Returns

Type	Description
string	The port part of the URL

Examples

The function `url.port("https://example.com:8080/root-path/sub-path?query=example+query+%2F+question#fragment")` returns `"8080"`.

url.query

This function returns the encoded query string if the second parameter is set to `true`. It returns an object with a key value pair of query and value, if the second parameter is set to `false`.

Syntax

```
url.query(rawURLString, encode)
```

Arguments

Argument	Type	Description
rawURLString	string	Raw URL string
encode	boolean	Set this argument to `true` to encode the query string. Set it to `false` to get the object.

Returns

Type	Description
Any	The encoded query string or object

Examples

The function `url.query("https://example.com:8080/root-path/sub-path?query=example+query+%2F+question#fragment", true)` returns `"query=example+query+%2F+question"`.

url.queryEscape

This function encodes the input string so it can be safely placed inside a URL query. Note that this function does not create the full query string.

Syntax

```
url.queryEscape(queryValue)
```

Arguments

Argument	Type	Description
queryValue	string	URL query value

Returns

Type	Description
string	The escaped value of the input

Examples

The function `url.queryEscape("hello world")` returns `hello+world`.

url.scheme

This function returns the URL scheme.

Syntax

```
url.scheme(rawURLString)
```

Arguments

Argument	Type	Description
rawURLString	string	Raw URL string

Returns

Type	Description
string	The URL scheme

Examples

The function `url.scheme("https://example.com:8080/root-path/sub-path?query=example+query+%2F+question#fragment")` returns `https`.

Utility Functions

This section provides a quick reference to the Flogo utility functions.

utility.renderJSON

This function converts the provided JSON data to a string. The false argument specifies whether the JSON data should be converted to an easily readable and neat format.

Syntax

```
utility.renderJSON(<JSON data>, false)
```

Arguments

Argument	Type	Description
JSON data		JSON data to be converted to string.
false		Indicates whether the JSON data must be converted to an easily readable and neat format.

Returns

Type	Description
string	JSON data in a string format.

Examples

The function `utility.renderJSON($RestInvoke.responseBody, false)` returns `{"FirstName":"ABC","LastName":"XYZ"}`.

Utils Functions

This section provides a quick reference to the Flogo utils functions.

Utils.decodeBase64

This function converts a base64 encoded string into a human-readable format.

Syntax

```
utils.decodeBase64(<input string>)
```

Arguments

Argument	Type	Description
string	string	The base64-encoded string, which needs to be decoded to a human-readable format.

Returns

Type	Description
bytes	Human-readable string represented by the base64 encoded input string.

Examples

The function `utils.decodeBase64("SGVsbG8sIFdvcmxk")` returns `Hello, World`.

Utils.encodeBase64

This function converts a string to a Base64 encoded string.

Syntax

```
utils.encodeBase64(<input string>)
```

Arguments

Argument	Type	Description
string	string	String to be converted to a Base64 encoded version.

Returns

Type	Description
string	A Base64 encoded version of the input string.

Examples

The function `utils.encodeBase64(coerce.ToBytes("Hello, World"))` returns `SGVsbG8sIFdvcmxk`.

utils.uuid

This function generates a random UUID according to the RFC 4122 standard.

Syntax

```
utils.uuid()
```

Arguments

None

Returns

Type	Description
string	UUID of 36 characters displayed in the form of 8-4-4-4-12.

Examples

The function `utils.uuid()` returns `245056b5-0d11-47aa-b564-fbbf3c6e044a`.

TIBCO Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the Product Documentation website, mainly in HTML and PDF formats.

The Product Documentation website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

The following documentation for this product is available on the TIBCO Flogo® Enterprise Product Documentation page.

How to Contact Support for TIBCO Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our product Support website.
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the product Support website. If you do not have a username, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature

requests from within the TIBCO Ideas Portal. For a free registration, go to TIBCO Community.

Legal and Third-Party Notices

SOME CLOUD SOFTWARE GROUP, INC. (“CLOUD SG”) SOFTWARE AND CLOUD SERVICES EMBED, BUNDLE, OR OTHERWISE INCLUDE OTHER SOFTWARE, INCLUDING OTHER CLOUD SG SOFTWARE (COLLECTIVELY, “INCLUDED SOFTWARE”). USE OF INCLUDED SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED CLOUD SG SOFTWARE AND/OR CLOUD SERVICES. THE INCLUDED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER CLOUD SG SOFTWARE AND/OR CLOUD SERVICES OR FOR ANY OTHER PURPOSE.

USE OF CLOUD SG SOFTWARE AND CLOUD SERVICES IS SUBJECT TO THE TERMS AND CONDITIONS OF AN AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER AGREEMENT WHICH IS DISPLAYED WHEN ACCESSING, DOWNLOADING, OR INSTALLING THE SOFTWARE OR CLOUD SERVICES (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH LICENSE AGREEMENT OR CLICKWRAP END USER AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE SAME TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

TIBCO, the TIBCO logo, the TIBCO O logo, and Flogo are either registered trademarks or trademarks of Cloud Software Group, Inc. in the United States and/or other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only. You acknowledge that all rights to these third party marks are the exclusive property of their respective owners. Please refer to Cloud SG’s Third Party Trademark Notices (https://www.cloud.com/legal) for more information.

This document includes fonts that are licensed under the SIL Open Font License, Version 1.1, which is available at: https://scripts.sil.org/OFL

Copyright (c) Paul D. Hunt, with Reserved Font Name Source Sans Pro and Source Code Pro.

Cloud SG software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the “readme” file for the availability of a specific version of Cloud SG software on a specific operating system platform.