Functions for Selligent Marketing Cloud

PRODUCT MANUAL

1 Foreword

Copyright

User is aware that this manual may contain errors or inaccuracies and that it may be revised without advance notice. This manual is updated frequently.

Selligent welcomes any recommendations or suggestions regarding the manual, as it helps to continuously improve the quality of our products and manuals.

(This document was last reviewed January 2023)

Table of Contents

2 Available SELLIGENT Marketing Cloud functions

2.1 String operations

2.1.1 len

Format: len(*string* str)

Use: Returns the length of the specified string.

Parameters:
str (string): string from which the length must be returned

Return value:
Integer: length of the string

Example:

Function:	len([MASTER.NAME]) / len('Selligent')
Output:	9

2.1.2 find

Format: find(*string* haystack, *string* needle)
search(*string* haystack, *string* needle)

Use: Searches haystack for the first occurrence of needle. This is case sensitive

Parameters:
needle (string): string to find
haystack (string): string to be searched

Return value:
Integer: If needle found: offset of needle in haystack
If needle not found: -1

Examples:

Function:	find([MASTER.NAME], 'e')
Output:	5
Function:	find('Selligent', 'e')
Output:	1

2.1.3 findoneof

Format: findoneof(*string* needles, *string* haystack)

Use: Searches haystack for the first occurrence of any character contained in needles

Parameters:
Needles (string): character to search for
Haystack (string): string to be searched

Return value:
Integer: If one of the characters from the needles is found : offset of the first character found in the haystack

If none are found: -1

Examples:

Function: findoneof ('ae', 'I love cats')

Output: 5

(starting from position 0, the e is the first character found, on position 5)

Function: findoneof (',','I love them, those cats')

Output: 11

(starting from position 0, the comma is found on position 11)

Function: findoneof ('b','I love cats')

Output: -1

(the b is not found in the haystack)

2.1.4 reversefind

Format: reversefind(*string* haystack, *string* needle)

Use: Searches haystack for the last occurrence of needle

Parameters:
needle (string): character to search for
haystack (string): string to be searched

Return value:
Integer: If one of the characters from needles found: last offset of character in haystack
If none found: -1

Example:

Function:	reversefind('I love cats', 'a')
Output:	8

2.1.5 replace

Format: replace(*string* haystack, *string* originalNeedle, *string* newNeedle)

Use: Searches haystack for the every occurrence of originalNeedle, and replaces them with newNeedle.

Parameters:
haystack (string): string to be searched
originalNeedle (string): substring that should be replaced by another one
newNeedle (string): String that is used to replace any occurrence of originalNeedle

Return value:
string: returns the modified haystack string

Example:

Function:	replace('I love cats', 'cats', 'dogs')
Output:	I love dogs

2.1.6 reverse

Format: reverse(*string* str)

Use: Returns the reverse of a string value.

Parameters:
str (string): string to be reversed

Return value:
string: returns the reverse string

Example:

Function:	reverse('I love cats')
Output:	stac evol I

2.1.7 trim

Format: trim(*string* str)

Use: Trims spaces, newlines and tabs from both ends of str.

Parameters:
str (string): string to be trimmed

Return value:
string: returns the trimmed string

Example:

Function:	trim(' abc ')
Output:	abc

2.1.8 left

Format: left(*string* str; integer count)

Use: Returns the first count characters from str.

Parameters:
str (string): source string
count (integer): Number of characters to select, starting from the most left character

Return value:
string: returns the left part from str

Example:

Function:	left('I love cats',5)
Output:	I lov

2.1.9 mid

Format: mid(*string* str; integer offset, integer count)

Use: count characters from the middle of str, starting at offset.

Parameters:
str (string): source string
offset (integer): starting offset
count (integer): Number of characters to select, starting from the offset

Return value:
string: returns the mid part from str

Example:

Function:	mid('I love cats',2,4)
Output:	love

2.1.10 right

Format: right(*string* str, integer count)

Use: Returns the last count characters from str.

Parameters:
str (string): source string
count (integer): Number of characters to select from the end of str

Return value:
string: returns the right part from str

Example:

Function:	right('I love cats',4)
Output:	cats

2.1.11 upper

Format: upper(*string* str)

Use: Returns str, with all characters converted to uppercase.

Parameters:
str (string): source string

Return value:
string: returns the uppercased str

Example:

Function:	upper('I love cats')
Output:	I LOVE CATS

2.1.12 lower

Format: lower(*string* str)

Use: Returns str, with all characters converted to lowercase.

Parameters:
str (string): source string

Return value:
string: returns the lowercased str

Example:

Function:	lower('I love cats')
Output:	i love cats

2.1.13 proper

Format: proper(*string* str)

Use: str, with every first character of every contained word converted to uppercase.

Parameters:
str (string): source string

Return value:
string: returns the converted str

Example:

Function:	proper('I love cats')
Output:	I Love Cats

2.1.14 concat

Format: concat(*string* str1, *string* str2, *string* strn)

Use: Concatenates the strings.

Parameters:
str1 (string): First string
str2 (string): Second string
strn (string): Nth string

Return value:
string: returns the concatenated string

Example:

Function:	concat('I', ' Love ', 'Cats')
Output:	I Love Cats

2.1.15 contains

Format: contains(*string* haystack, *string* needle)

Use: text search within a string (appears somewhere inside the string)

Parameters:
haystack (string): string to perform the search in
needle (string): text to search for

Return value:
string: returns true if the needle is found

Examples:

Function:	contains('CAT,Frog,doggybag,hotdog','dog')
Output:	**True** , as 'dog' can be found somewhere inside the string (even multiple times in this example 'CAT,Frog,**dog**gybag,hot**dog**')

Function: contains('CAT,Frog,doggybag,hotdog','bird')

Output: **False** , as 'bird' can't be found anywhere inside the string

2.1.16 containstag

Format: contains(*string* haystack*, string* needle)

Use: search for a specific string within a string of separate values.
The 'string of values' is split into separate items by separators (a separator can be : comma, pipe or semicolon).
The search is performed on each individual item.

Parameters:
haystack (string): string to perform the search in
needle (string): string to search for

Return value:
string: returns true if the needle is found

Examples:

Function: containstag('CAT,DOG,Frog,doggybag,hotdog','dog')

Output: **True** , as 'dog' is found as a value inside the string 'CAT,**DOG**,Frog,doggybag,hotdog'. (In this example, the string values are split by commas. Note that only 'dog' matches the search, as 'doggybag' and 'hotdog' are different values, and thus do not match the searched needle.)

Function: containstag('CAT,doggy,bird','dog')

Output: **False** , as neither 'cat', 'doggy' or 'bird' match the searched string 'dog'

2.1.17 explode

Format: explode(*string* haystack, *string* needle)

Use: convert a string to an array by splitting up on needle

Parameters:
haystack (string): string to be split up
needle (string): string to split the string up by

Return value:
array: a multivalue list containing the elements of the split up haystack

Example:

Function: explode('a+b+c+d+e', '+')

Output: ['a', 'b', 'c', 'd', 'e']

Remark : The output will not be shown in the message preview in Selligent Marketing Cloud, as it's an array. You can for example add an extra function around it like join in order to view output in the message preview : join(explode('a+b+c+d+e', '+'),'-') will result in a-b-c-d-e

2.1.18 tostring

Format: tostring(*variant* value)

Use: converts the value to a string

Parameters:
value to be converted

Return value:
a string

Example:

Function:	tostring(123)
Output:	123

2.1.19 toint

Format: toint(*string* text)

Use: Convert a string to a (truncated) integer value. Only numeric values up to the first separator are included in the output result.
When the string starts with other than numeric values, the output result is 0.

Parameters:
string to be converted
Return value:
the integer representation of the string value

Examples:

Function: toint ('123.678')
Output: 123

Function: toint ('-123.678')
Output: -123

Function: toint ('2022 is the current year')
Output: 2022

Function: toint ('16-10-1981')
Output: 16

Function: toint ('text')
Output: 0

**Note: Instead of using a string value, a decimal (without quotes) also can be used.
In that case, the resulting integer is the rounded value of the decimal.
For example : toint(123.789) returns 124.**

2.1.20 todecimal

Format: todecimal(*variant* value)

Use: convert the value to its decimal value

Parameters:
value to be converted
Return value:
the decimal representation of the value, with 2 decimal digits (.00), rounded

Examples:

Function: todecimal('123')
Output: 123.00

Function: todecimal('123.3456')
Output: 123.35

Function: todecimal(123.3456)
Output: 123.35

Note: Any data type can be used as value (integer, string, etc).
Keep in mind that using a nonsensical value (such as a date) results in a decimal representation that doesn't make sense.

2.1.21 tobool

Format: tobool(*string* text)

Use: convert a value to a boolean (1 which indicates TRUE or 0 which indicates FALSE)

Parameters:

text (object): value to be converted

Return value:
boolean: the boolean value of 'text'

Examples:

Function: tobool('TRUE')

Output: 1

Function: tobool([VARIABLE.isVisible])
being a boolean variable containing the text value 'TRUE'

Output: 1

2.1.22 startswith

Format: startswith(p1, val)

Use: determines whether a string starts with the characters of a particular string

Parameters:

p1 : string/field to search in
val : value to search for

Return value:
boolean: true or false

Example:

Function: startswith([MASTER.NAME], 'Jo')

Output: true

2.1.23 endswith

Format: endswith(p1, val)

Use: determines whether a string ends with the characters of a particular string

Parameters:

p1 : string/field to search in
val : value to search for

Return value:
boolean: true or false

Example:

Function: endswith([PREFERENCES.INTERESTS], 'fashion')

Output: true

2.1.24 notcontains

Format: notcontains(p1, val)

Use: determines whether a string does not contain a particular string

Parameters:

p1 : string/field to search in
val : value to search for

Return value:
boolean: true or false

Example:

Function: notcontains([MASTER.POSTAL_CODE], '1000')

Output: true

2.1.25 getprop

Format: getprop(*str*, *idx*, *sep*)

Use: Automatically escapes single quotes and removes parentheses from the variable to be used in JavaScript.

Parameters:

str : string containing the properties
idx : 1-based property index
sep (optional) : string containing accepted separators

Return value:

string

Example:

Function:	getprop('Barcelona; Brussels; New York',3,';')
Output:	New York

2.2 Math functions

2.2.1 abs

Format: abs(*integer* number)

Use: number, without sign (basically converts negative numbers to positive ones = the 'absolute' values)

Parameters:
Number (*integer*): Number to convert

Return value:
integer: returns the converted number

Examples:

Function:	abs(5)
Output:	5
Function:	abs(-5)
Output:	5

2.2.2 average

Format: average(number_1, number_2, …., number_n)

Use: returns the average of the provided numbers

Parameters:
number_1, number_2, up to number_n (all numeric types are accepted as input values, including integer, float, decimal, …)

Return value:
integer: returns the calculated number (truncated)

Examples:

Function:	average(1,2,3,4,5)
Output:	3
Function:	average(2.25,4.83,60.88)
Output:	22

2.2.3 mul

Format: mul(*integer* number_1, *integer* number_2)

Use: Returns the integer result of multiplying number_1 with number_2

Parameters:
Number_1,number_2 (*integer*): numbers to multiply

Return value:
integer

Example:

Function:	mul(8,4)
Output:	32

2.2.4 div

Format: div(*integer* nominator, *integer* denominator)

Use: Returns the integer result of the division of nominator with denominator

Parameters:
nominator (*integer*)
denominator (*integer*)

Return value:
integer

Example:

Function:	div(6, 2)
Output:	3

2.2.5 add

Format: add(*integer* number_1, *integer* number_2)

Use: Returns the integer result of adding number_1 with number_2

Parameters:
Number_1 (*integer*)
Number_2 (*integer*)

Return value:
integer

Example:

Function:	add(12, 34)
Output:	46

2.2.6 sub

Format: sub(*integer* number_1, *integer* number_2)

Use: Returns the integer result of subtracting number_2 from number_1

Parameters:
Number_1 (*integer*)
Number_2 (*integer*)

Return value:
integer

Example:

Function:	sub(34, 12)
Output:	22

2.2.7 round

Format: round(*decimal* number, *int* digits)

Use: rounds the float number to the specified number of digits.

Parameters:
Number (decimal): number to convert

Digits (int): number of digits

Return value:
string: returns the rounded number

Examples:

Function:	round(1.23456,3)
Output:	1.235
Function:	round(1.23456, 1)
Output:	1.2

2.2.8 mod

Format: *int* number_1 % *int* number_2

Use: calculates the remainder on a division of number_1 by number_2.

Parameters:

Number_1 (integer): divided number

Number_2 (integer): Divider

Return value:

int: the remaining number

Examples:

Function:	mod(3, 2)
Output:	1
Function:	mod(12, 8)
Output:	4

2.2.9 formatnumber

Format: formatnumber(*decimal* value, *string* format)

Use: Formats a given number with the specified format.

Parameters:

value (decimal): The number to format

format (string): The format to use

Return value:

string: The formatted number

Example:

Function:	formatnumber(1234567.1234, '0,0.00')
Output:	1,234,567.12

2.3 Date functions

2.3.1 add+*datetype*

Format:

addyears(*datetime* date, *integer* amount)
addquarters(*datetime* date, *integer* amount)
addmonths(*datetime* date, *integer* amount)
addweeks(*datetime* date, *integer* amount)
adddays(*datetime* date, *integer* amount)
addhours(*datetime* date, *integer* amount)
addminutes(*datetime* date, *integer* amount)
addseconds(*datetime* date, *integer* amount)

Use: Returns date, added with an amount of years, months, days,... depending on the selected function.

Parameters:

date (datetime): date to add to

amount in years, months, days, hours, minutes or seconds to add

Return value: Datetime: Returns the resulting datetime value

Examples:

Function:	adddays(todatetime('2009-09-24','yyyy-MM-dd'), 4)
Output:	2009-09-28 00:00:00
Function:	adddays(todatetime('2009-09-24','yyyy-MM-dd'), -4)
Output:	2009-09-20 00:00:00

2.3.2 subtract+*datetype*

Format:

subtractyears(*datetime* date, *integer* amount)
subtractquarters(*datetime* date, *integer* amount)
subtractmonths(*datetime* date, *integer* amount)
subtractweeks(*datetime* date, *integer* amount)
subtractdays(*datetime* date, *integer* amount)
subtracthours(*datetime* date, *integer* amount)

subtractminutes(*datetime* date, *integer* amount)
subtractseconds(*datetime* date, *integer* amount)

Use: Returns date, subtracted with an amount of years, months, … depending on the selected function.

Parameters:

date (datetime): date to subtract from

amount in years, months, days, hours, minutes, seconds to subtract

Return value: Datetime: Returns the resulting datetime value

Examples:

Function:	subtractdays(todatetime('2019-10-19','yyyy-MM-dd'), 6)
Output:	2019-10-13 00:00:00
Function:	subtractdays(todatetime('2019-10-19','yyyy-MM-dd'), -6)
Output:	2019-10-25 00:00:00

2.3.3 year

Format: year (*datetime* dt)

Use: returns the year of a specific date/time.

Parameters:

dt (datetime)

Example:

Function:	year(todatetime('1981-07-12','yyyy-MM-dd'))
Output:	1981

2.3.4 quarter

Format: quarter (*datetime* dt)

Use: returns the quarter of a specific date/time.

Parameters:

dt (datetime)

Example:

Function:	quarter(todatetime('1981-07-12','yyyy-MM-dd'))
Output:	3

2.3.5 month

Format: month (*datetime* dt)

Use: returns the month of a specific date/time.

Parameters:

dt(datetime)

Example:

Function:	month(todatetime('1981-07-12','yyyy-MM-dd'))
Output:	7

2.3.6 day

Format: day (*datetime* dt)

Use: returns the day of a specific date/time.

Parameters:

dt(datetime)

Example:

Function:	day(todatetime('1981-07-12','yyyy-MM-dd'))
Output:	12

2.3.7 hour

Format: hour (*datetime* dt)

Use: returns the hour of a specific date/time.

Parameters:

dt (datetime)

Example:

Function:	hour(todatetime('1981-07-12 11:22:56','yyyy-MM-dd hh:mm:ss'))
Output:	11

2.3.8 minute

Format: minute (*datetime* dt)

Use: returns the minutes of a specific date/time.

Parameters:

dt (datetime)

Example:

Function:	minute(todatetime('1981-07-12 12:34:56','yyyy-MM-dd hh:mm:ss'))
Output:	34

2.3.9 second

Format: second (*datetime* dt)

Use: returns the seconds of a specific date/time.

Parameters:

dt (datetime)

Example:

Function:	second(todatetime('1981-07-12 12:34:56','yyyy-MM-dd hh:mm:ss'))
Output:	56

2.3.10 dayofyear

Format: dayofyear (*datetime* dt)

Use: returns the day of the year for a specific date/time.

Parameters:

dt (datetime)

Example:

Function:	dayofyear(todatetime('1981-07-12','yyyy-MM-dd'))
Output:	193

2.3.11 dayofweek

Format: dayofweek (*datetime* dt)

Use: Returns the day of the week of the specified datetime, with Sunday as first day of week.

Parameters:
dt (datetime):

Return value:
integer:

Example:

Function:	dayofweek(todatetime('1981-07-14','yyyy-MM-dd'))
Output:	3

2.3.12 days

Format: days (*datetime* dt1, *datetime* dt2)

Use: returns the difference in days between two date times.

Parameters:

dt1(datetime)

dt2 (datetime)

Examples:

Function: days(todatetime('2018-01-12','yyyy-MM-dd'),todatetime('2018-01-20','yyyy-MM-dd'))
Output: -8

Negative output value in case dt1 is an earlier datetime than dt2

Function: days(todatetime('2018-01-20','yyyy-MM-dd'),todatetime('2018-01-12','yyyy-MM-dd'))

Output: 8

Positive output value in case dt1 is a later datetime than dt2

2.3.13 hours

Format: hours (*datetime* dt1, *datetime* dt2)

Use: returns the difference in hours between two date times.

Parameters:

dt1(datetime)

dt2 (datetime)

Examples:

Function: hours(todatetime('2009-01-12','yyyy-MM-dd'),todatetime('2009-01-13','yyyy-MM-dd'))

Output: -24

Negative output value in case dt1 is an earlier datetime than dt2

Function: hours(todatetime('2009-01-13','yyyy-MM-dd'),todatetime('2009-01-12','yyyy-MM-dd'))

Output: 24

Positive output value in case dt1 is a later datetime than dt2

2.3.14 minutes

Format: minutes (*datetime* dt1, *datetime* dt2)

Use: returns the difference in minutes between two date times.

Parameters:

dt1(datetime)

dt2 (datetime)

Examples:

Function: minutes(todatetime('2009-01-12 00:00:00','yyyy-MM-dd hh:mm:ss'),todatetime('2009-01-13 00:00:00','yyyy-MM-dd hh:mm:ss'))

Output: -1440

Negative output value in case dt1 is an earlier datetime than dt2

Function: minutes(todatetime('2009-01-13 00:00:00','yyyy-MM-dd hh:mm:ss'),todatetime('2009-01-12 00:00:00','yyyy-MM-dd hh:mm:ss'))

Output: 1440

Positive output value in case dt1 is a later datetime than dt2

2.3.15 seconds

Format: seconds (*datetime* dt1,*datetime* dt2)

Use: returns the difference in seconds between two date times.

Parameters:

dt1(datetime)

dt2 (datetime)

Examples:

Function: seconds(todatetime('2009-01-12 00:00:00','yyyy-MM-dd hh:mm:ss'),todatetime('2009-01-13 00:00:00','yyyy-MM-dd hh:mm:ss'))

Output: -86400

Negative output value in case dt1 is an earlier datetime than dt2

Function: seconds(todatetime('2009-01-13 00:00:00','yyyy-MM-dd hh:mm:ss'),todatetime('2009-01-12 00:00:00','yyyy-MM-dd hh:mm:ss'))

Output: 86400

Positive output value in case dt1 is a later datetime than dt2

2.3.16 sysdate

Format: sysdate()

Use: Returns the current system datetime in UTC

Parameters:
none

Return value:
datetime: Returns the current system datetime

Example: On September 24th 2019 at 1PM:

Function:	sysdate()
Output:	2019-09-24 13:00:00

2.3.17 isdate

Format: isdate (*string* dt, *string* format)

Use: Returns true if the specified dt can be converted into a datetime.

Parameters:
dt (datetime): date to be checked
format (string): de format to check the date against

Return value:
boolean: If date appears to be valid, the return value is 1, otherwise 0

Examples:

Function:	isdate('2016-01-12', 'yyyy-MM-dd')
Output:	True
Function:	isdate('2016-02-31', 'yyyy-MM-dd')
Output:	False

2.3.18 currentunixtimestamp

Format: currentunixtimestamp()

Use: retrieves the unix timestamp in milliseconds

Parameters: none

Return value:

datetime

Example:

Function:	currentunixtimestamp()
Output:	1476177429.89314

2.3.19 todatetime

Format: todatetime(*string* value).

Use: converts the value to a datetime.

Parameters: value (string): value to convert

Return value: A datetime

2.3.20 tounixtimestamp

Format: tounixtimestamp (date)

Use: converts a date to UNIX timestamp in seconds

Parameters:

date (datetime): date to be converted

Return value:

datetime: returns a UNIX timestamp for a specific date/time.

Example:

Function:	tounixtimestamp(todatetime('1981-07-12 13:34:56','yyyy-MM-dd HH:mm:ss'))
Output:	363792896

2.3.21 format

Format: format(*datetime* date, *constant* format)

Use: Returns a string representation of the specified date in the supplied format.

Parameters:

Date (datetime): date to convert to string

Format (constant):

d	The day of the month, from 1 through 31.
dd	The day of the month, from 01 through 31.
ddd	The abbreviated name of the day of the week.
dddd	The full name of the day of the week.
G	The period or era.
h	The hour, using a 12-hour clock from 1 to 12.
hh	The hour, using a 12-hour clock from 01 to 12.
HH	The hour, using a 24-hour clock from 00 to 23.
mm	The minute, from 00 through 59.
M	The month, from 1 through 12.
MM	The month, from 01 through 12.
MMM	The abbreviated name of the month.
MMMM	The full name of the month.
ss	The second, from 00 through 59.
tt	The AM/PM designator.
T	The time (hours, minutes, seconds) in AM/PM format.
yy	The year, from 00 to 99.
yyyy	The year as a four-digit number.

Return value:

string: Returns the formatted string

Example:

On January 11th 2023 at 10:29 AM :

Function: format(sysdate(), 'dddd MMMM d yyyy T')

Output: Wednesday January 11 2023 10:29:05 AM

2.4 Logical functions

2.4.1 eq

Format: eq(*variant* var1, *variant* var2)

Use: compare 2 variables of the same type.
Use the appropriate conversion-function on each variable that is not of type string.

Parameters:
var1 (variant): value to be compared with var2
var2 (variant): value to be compared with var1

Return value:
boolean: true if both variables are equal.

Examples:

Function:	eq('A', 'A')
Output:	true
Function:	eq([MASTER.LONGFIELD], toint('0'))
Output:	true
Function:	eq(tobool([VARIABLE.isVisible]), tobool('TRUE'))
Output:	true

2.4.2 ne

Format: ne(*variant* var1, *variant* var2)

Use: compare 2 variables of the same type.
Use the appropriate conversion-function on each variable that is not of type string.

Parameters:
var1 (variant): value to be compared with var2
var2 (variant): value to be compared with var1

Return value:
boolean: true if both variables are not equal.

Examples:

Function:	ne('A', 'A')
Output:	false
Function:	ne([MASTER.LONGFIELD], toint('0'))
Output:	true
Function:	ne(tobool([VARIABLE.isVisible]), tobool('FALSE'))
Output:	true

2.4.3 lt

Format: lt(*variant* var1, *variant* var2)

Use: checks if var1 is less than var2.

Parameters:
var1 (variant): value to be compared with var2
var2 (variant): value to be compared with var1

Return value:
boolean: true if var1 is less than var2.

Example:

Function:	lt(1, 5)
Output:	true

2.4.4 gt

Format: lt(*variant* var1, *variant* var2)

Use: checks if var1 is greater than var2.

Parameters:
var1 (variant): value to be compared with var2
var2 (variant): value to be compared with var1

Return value:
boolean: true if var1 is greater than var2.

Example:

Function:	gt(1, 5)
Output:	false

2.4.5 le

Format: le(*variant* var1, *variant* var2)

Use: checks if var1 is less than or equal to var2.

Parameters:
var1 (variant): value to be compared with var2
var2 (variant): value to be compared with var1

Return value:
boolean: true if var1 is less than or equal to var2.

Example:

Function:	le(1, 5)
Output:	true

2.4.6 ge

Format: ge(*variant* var1, *variant* var2)

Use: checks if var1 is greater than or equal to var2.

Parameters:
var1 (variant): value to be compared with var2
var2 (variant): value to be compared with var1

Return value:
boolean: true if var1 is greater than var2.

Example:

Function:	ge(1, 5)
Output:	false

2.4.7 between

Format: between(*variant* value, *variant* minValue, *variant* maxValue)

Use: checks if value is between minValue and maxValue (numeric, date or datetime)

Parameters:

value (variant): value to be compared with minValue and maxValue
minValue (variant): the minimum value
maxValue (variant): the maximum value

Return value:

boolean: true if value is between minValue and maxValue.

Examples:

Function: between(1, 5, 8)

Output: false

Function: between(5, 5, 8)

Output: false

Function: between(6, 5, 8)

Output: true

Function: between([MASTER.NUMBER], toint('10'), toint('16'))

Output: false

Function: between([MASTER.BIRTHDAY], todatetime('2018-03-01 00:00:00','yyyy-MM-dd HH:mm:ss'), todatetime('2018-06-01 00:00:00','yyyy-MM-dd HH:mm:ss'))

Output: true

2.4.8 and / all

Format: and(*boolean* bool1, *boolean* bool2)
all(*boolean* bool1, *boolean* bool2)

Use: checks if both bool1 and bool2 are applicable.

Parameters:
bool1 (boolean): first boolean to be evaluated
bool2 (boolean): second boolean to be evaluated

Return value:
boolean: true if both bool1 and bool2 evaluate “true”.

Examples:

For a contact from the Audience List with gender equal to male (‘m’) :

Function: and(eq(1,5), eq([MASTER.GENDER], 'm'))

Output: false

Function: and(eq(5,5), eq([MASTER.GENDER], 'm'))

Output: true

Function: all(eq(1,5), eq([MASTER.GENDER], 'm'))

Output: false

Function: all(eq(5,5), eq([MASTER.GENDER], 'm'))

Output: true

2.4.9 or / any

Format: or(*boolean* bool1, *boolean* bool2)
any(*boolean* bool1, *boolean* bool2)

Use: checks if at least one of bool1 or bool2 is applicable.

Parameters:
bool1 (boolean): first boolean to be evaluated
bool2 (boolean): second boolean to be evaluated

Return value:
boolean: true if bool1 and/and bool2 evaluates “true”.

Examples:

For a contact from the Audience List with gender equal to male (‘m’) :

Function: or(eq(1,5), eq([MASTER.GENDER], 'f'))

Output: false

Function: or(eq(1,5), eq([MASTER.GENDER], 'm'))

Output: true

Function: any(eq(5,5), eq([MASTER.GENDER], 'f'))

Output: true

Function: any(eq(5,5), eq([MASTER.GENDER], 'm'))

Output: true

2.4.10 not

Format: not(*boolean* bool1)

Use: inverts the result from the evaluation of bool1.

Parameters:
bool1 (boolean): boolean to be evaluated

Return value:
boolean: true if bool1 evaluates “false”, false if bool1 evaluates “true”.

Example:

Function: not(eq(5,5))

Output: false

2.4.11 isempty

Format: isempty(*variant* value)

Use: checks if value is empty

Parameters:
value (variant): value to be checked

Return value:
boolean: true if value is empty

Examples:

Function:	isempty([MASTER. BOUNCEDT])
Output:	true
Function:	isempty([MASTER.NAME])
Output:	false

2.4.12 isnotempty

Format: isnotempty(*variant* value)

Use: checks if value is not empty

Parameters:
value (variant): value to be checked

Return value:
boolean: true if value is not empty

Examples:

Function:	isnotempty([MASTER. BOUNCEDT])
Output:	false
Function:	isnotempty([MASTER.NAME])
Output:	true

2.5 Array functions

2.5.1 union

Format: union(*array* arr1, *array* arr2)

Use: combine 2 arrays into 1 array

Parameters:

arr1: array of objects
arr2: array of objects

Return value:

Array: combined array containing all elements from arr1 and all elements from arr2.

Example:

```
Function: union(array('1', '2'), array('3', '4'))

Output: ['1', '2', '3', '4']
```

2.5.2 intersect

Format: intersect(*array* arr1, *array* arr2)

Use: Get the intersection of 2 arrays

Parameters:

arr1: array of objects
arr2: array of objects

Return value:

Array: array containing all common elements from arr1 and arr2.

Example:

```
Function: intersect(array('1', '2', '4'), array('1', '3', '4'))

Output: ['1', '4']
```

2.5.3 distinct

Format: intersect(*array* arr1, *array* arr2)

Use: Get the distinct values of 2 arrays

Parameters:

arr1: array of objects
arr2: array of objects

Return value:

Array: array containing all unique elements from arr1 and arr2.

Example:

Function: distinct(array('1', '2', '4'), array('1', '3', '4'))

Output: ['1', '2', '3', '4']

2.5.4 join

Format: join(array arr1, string separator)

Use: join all elements from arr1 into a string, with the separator in between

Parameters:

arr1 (array): the array of elements to be joined into a string
separator (string): the string to glue the elements together

Return value:

string

Example:

Function: join(array('1', '2', '3', '4'), '|'))

Output: 1|2|3|4

2.6 Other functions

2.6.1 propcount

Format: propcount(*string* multivalue)

Use: The amount of values, contained in a multi-value field.

Parameters:
multivalue (*string*) : string to search
separators can be "," or "|"

Return value:
integer: The amount of contained values

Examples:

Function:	propcount ('a,b,c,d,e')
Output:	5
Function:	propcount ('a|b|c|d')
Output:	4

2.6.2 matchcount

Format: matchcount(*string* multivalue_1, *string* multivalue_2)

Use: Counts the amount of values that match between 2 multi-value fields.

Parameters:
Multivalue_1 (string)
Multivalue_2 (string)

Return value:
integer: The amount of matched values found

Examples:

Function:	matchcount ('a,b,c,d,e','b,d,e,f,g')
Output:	3

2.6.3 if

Format: if(*boolean* constraint, *variant* ifTrue, *variant* ifFalse)

Use: ifTrue if constraint evaluates to true, otherwise returns ifFalse.

Parameters:

constraint (boolean): Constraint to evaluate. Can be a combination or nesting of every supported SELLIGENT function
ifTrue (variant): Return value if constraint evaluates to true
ifFalse (variant): Return value if constraint evaluates to false

Return value:

variant: ifTrue or ifFalse, depending of the evaluation of constraint

Examples:

Function:	if(eq('A','A'),'Identical', 'Different')
Output:	Identical
Function:	if(eq('A','B'),'Identical', 'Different')
Output:	Different
Function:	if(eventvalue('Boolean'), 'correct', 'incorrect')
Output:	Correct in case of Boolean = true, Incorrect in case of Boolean = false

2.6.4 itemValue

Format: itemValue(*string* dataSelection, *int* itemIndex, *string* fieldName)

Use: value from field fieldName of the *itemIndex* item in the data selection dataselection.

Parameters:

dataSelection (string): Name of the data selection defined in the message/template
itemIndex (int): Index of the item in the data selection (zero-based)
fieldName (string): Name of the field to return the value from

Return value:

variant: fieldvalue of given field in the given item in the given data selection

Example:

Function:	itemValue('Productlist', 0, 'NAME')

Output:	iPhone 7

2.6.5 cartValue

Format:
- inside a Repeater : cartValue(*string* fieldName)
- outside a Repeater : cartValue(*string* dataSelection, *int* itemIndex, *string* fieldName)

Use: returns the value from the field fieldName of an abandoned cart journey.
This uses data coming from the Site cart. When no Site cart data can be found for the selected field, then by default the value is taken from the data selection list in Selligent Marketing Cloud.
The cartValue expression is available for every field in the data selection. However, currently only *Price* and *Quantity* are supported.

Parameters:
- dataSelection (string): Name of the data selection defined in the message/template
- itemIndex (int): Index of the item in the data selection (zero-based)
- fieldName (string): Name of the field to return the value from

Return value:
variant: fieldvalue of the given field in the given data selection (coming from Site abandoned cart data when available).

Examples:

Function:	cartValue('PRICE')
Output:	299
Function:	cartValue('QUANTITY')
Output:	8
Function:	cartValue('Products',0,'PRICE')
Output:	299

2.6.6 journeyLookupValue

Format:
- without scope : journeyLookupValue(*string* list_api_name, *string* fieldName)
- with scope : journeyLookupValue(*string* list_api_name, *string* fieldName, *string* scopeName)

Use: In a Lookup component in a Custom Journey, the property 'Load data' can be selected. When selecting this option, the loaded data will be available in the journey components that follow the Lookup component and for personalization in resulting pages.

The journeyLookupValue function can also be used with a third parameter, namely the 'scope'. The scope should be used when multiple Lookup components, used in the same Custom Journey, perform a lookup on the same list. The scope can then retrieve the right data from the right lookup component.

Parameters:

listName (string): Name of the list in which the lookup is done

fieldName (string): Name of the list field to return the value from

scopeName (string): Name of the scope (optional), in case multiple identical lookups are performed, to retrieve the data of that one specific scope

Return value:

variant: fieldvalue of the given field in the given list (with the given scope)

Example:

Function:	journeyLookupValue('Orders', 'ProductName')
Output:	iPhone 11
Function:	journeyLookupValue('Orders', 'ProductName', 'FirstOrder')
Output:	Samsung Galaxy S20

2.6.7 mobileValue

Format:
mobileValue(*string* fieldName)

Use: For personalization in Selligent Marketing Cloud mobile messages, mobileValue('FIELD_NAME') can be used to retrieve the data from the specified field from the Devices list (which is linked to the Audience list). As the properties of a mobile message define a mobile app or the default one, the right scope is used automatically.

Parameters:

fieldName (string): Name of the list field from the Devices list to return the value from

Return value:

variant: field value of the given field in the Devices list

Example:

Function:	mobileValue('SYSTEM_VERSION')
Output:	Android 9 Pie

2.6.8 translateValue

Format: translateValue(*string* fieldname)

Use: Translates a value of a field that's linked to an option list.

Parameters:

Fieldname (string): Name of the field to which the option list is linked

Return value:
string: returns the converted translated value

Example:

Function: translateValue('MASTER.GENDER')

where the GENDER field in the user list contains the value '**m**' for male users with English translation '**male**' and French translation '**masculin**' (translations in the Option List)

Output: **male**
when the selected/user language is EN

masculin
when the selected/user language is FR

2.6.9 translateMobileValue

Format:
translateMobileValue(*string* fieldName)

Use: For personalization in Selligent Marketing Cloud mobile messages, mobileValue('FIELD_NAME') can be used to retrieve the data from the specified field from the Devices list (which is linked to the Audience list). As the properties of a mobile message define a mobile app or the default one, the right scope is used automatically.
When the field contains Option list field values though, translateMobileValue('FIELD_NAME') needs be to used to retrieve the actual translated field values. Otherwise, the Option list codes are returned instead.

Parameters:
fieldName (string): Name of the list field from the Devices list to return the translated value from

Return value:
variant: translated field value of the given field in the Devices list

Example:

The field 'DEVICE_COUNTRY' in the Devices list contains values linked to an Option list.
For a record, containing the code 'FR' for that field, the English translation in the Option list is 'France'.

When using mobileValue :
Function: mobileValue('DEVICE_COUNTRY')
Output: FR

When using translateMobileValue :

Function: translateMobileValue('DEVICE_COUNTRY')
Output: France

2.6.10 requestValue

Format:
requestValue(fieldName)

Use: lookup and return a posted value from a form field

Parameters:
fieldName: Name of the form field for which the value needs to be returned

Return value:
variant: field value of the form field

Example:

A form is submitted containing an email address in the 'MAIL' form field.

Function: requestValue(MAIL)
Output: john.doe@selligent.com

2.6.11 listValue

Format:
listValue([listName.fieldName])

Use: lookup and return a field value in the lookup list (current scope) or in any (1:1 or 1:N) linked list

Parameters:
[listName.fieldName] : Name of the list and field for which the value needs to be returned

Return value:
variant: field value of the list

Example:

The lookup list (current scope) is the main Audience List, with a linked list 'Preferences' that contains a field 'Interests' from which we'd like to get the value from.

Function: listValue([Preferences.Interests])
Output: fashion

2.6.12 profileValue

Format:
profileValue([listName.fieldName])

Use: lookup and return a field value in the Audience List (or in a 1:1 profile extension of that Audience List)

Parameters:
[listName.fieldName] : Name of the list and field for which the value needs to be returned

Return value:
variant: field value of the list

Example:

The current scope is the 'Products' List (which is a Data List that's not linked to the main Audience List). We'd like to get the contact's email address from the Audience List.

Function: profileValue([MASTER.MAIL])
Output: john.doe@selligent.com

2.6.13 componentValue

Format:
componentValue('scope.variable')

Use: When using a 'Custom Component' or 'Custom Channel Component' with output parameters in a Custom Journey, the output can be accessed through the componentValue function.

Parameters:
- The 'scope' is defined in the properties of the Custom Component/Custom Channel Component.
- The 'variable' is the name of the output parameter (for example to show the value in an email).

Return value:
variant: output parameter value

Example:

There are two Custom Components added to a Custom Journey. Each one generates an output parameter called Product. The first Custom Component is given the scope 'Delivery' and the second is given the scope 'Invoice'. To be able to use the information returned by the Custom Components, the appropriate scope is used.

Function: componentValue('Delivery.Product')
Output: iPhoneX64slv

Function: componentValue('Invoice.Product')
Output: iPhone X 64GB Silver

2.6.14 eventValue

Format:
eventValue('fieldname')

Use: When using Custom Events, to retrieve the value of a Custom Event field.
Or to add transactional field values to a message.

Parameters:
fieldname : Name of the Custom Events list field or transactional field for which the value needs to be returned

Return value:
variant: field value

Example:

When using Custom Events, to retrieve the value of a Custom Event field called 'Favoriteshops' :
Function: eventValue('Favoriteshops')
Output: Parana Store Brussels

For a transactional field called 'Client' :
Function: eventValue('Client')
Output: Selligent

2.6.15 jsonValue

Format:
jsonValue(field, JSON key)

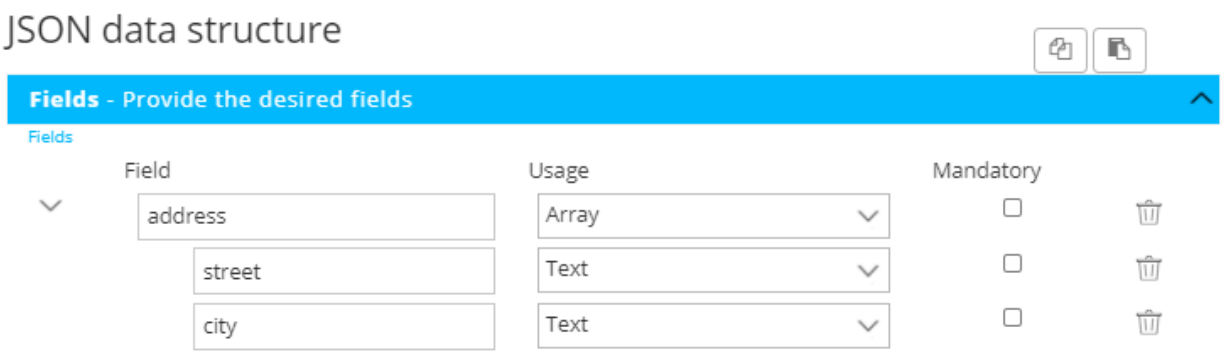
Use: To use the data returned in a Custom Events list field of type JSON (array).

Parameters:
- field = an expression returning the JSON field, e.g.: eventValue('FAVORITESHOPS')
- JSON key = the path to navigate to the specific key in the JSON (array). Use "." as a separator and "[x]" as (zero-based) index operator.

Return value:
variant: JSON field value of the Custom Events list

Example:

A Custom Events field called FAVORITESHOPS contains this JSON structure :



Function: jsonValue(eventValue('FAVORITESHOPS'), 'ADDRESS[0].CITY')
Output: Brussels (= the 'city' from the first element in the ADDRESS array)

Function: jsonValue(eventValue('FAVORITESHOPS'), 'ADDRESS[1].CITY')
Output: Paris (= the 'city' from the second element in the ADDRESS array)

2.6.16 loadValue

Format: loadValue(Value1, Value2, DefaultValue)

Use: To load page form values (for example a Combobox) or parameter values in an Input Component in a Custom Journey. The loadValue function is commonly used in combination with the requestValue function.

Parameters:
- Value1 : The value that will be loaded, when available.
For example, requestValue('NAME') to load a name entered by a contact on a page form.

- Value2 : The value that will be used when Value1 can't be loaded/is empty.
For example, the value from the Audience List field 'NAME', when on an initial page view no name value has been entered in the page form yet.

- DefaultValue : In case both Value1 and Value2 are empty, it's possible to display a default value.
For example, during a registration process of a new user, where no name value has been entered in the page form yet and the Name field in the Audience List is still empty, as it's a new user.

Return value:
string: loaded value

Examples:

A registration page form is shown to a new contact that contains, among other things, a name field.
Function: loadValue(requestValue('NAME'),'NAME','Please enter your name here')
Output: Please enter your name here
It's a new contact, for whom we have no info yet.
Value1 (requestValue('NAME')) doesn't exist.
Value2 ('NAME') is empty in the Audience List, as this is a new contact.
The third parameter is used in this case, which is a default value that we've set.

The name ‘Jane’ is entered in the registration page form, which we want to load in a next step in a journey.
Function: loadValue(requestValue('NAME'),'NAME','')
Output: Jane
The first parameter requestValue('NAME') is used in this case, which is the value ‘Jane’, as that’s what she entered in the form.

After Jane registered, she logs in. She then receives a new page form to enter her newsletter preferences.
The form contains a name field with loaded data.
Jane doesn’t enter her name (yet) in the name field.
Function: loadValue(requestValue('NAME'),'NAME','')
Output: Jane
Jane didn’t enter any value.
Value1 (requestValue('NAME')) couldn’t be loaded.
The second parameter 'NAME' is used in this case, which gets the Name value from the Audience List field ‘NAME’.

2.6.17 urlencode

Format: urlencode(*string* value)

Use: Returns the urlencoded version of value.

Parameters:
value (string): value to convert

Return value:
string: Urlencoded version of value

Example:

Function: urlencode('info@selligent.com')

Output: info%40selligent%2Ecom

2.6.18 urldecode

Format: urldecode(*string* encodedvalue)

Use: Returns the urldecoded version of value.

Parameters:
encodedvalue (string): value to convert

Return value:
string: Urldecoded version of encodedvalue

Example:

Function:	urldecode('info%40selligent%2Ecom')
Output:	info@selligent.com

2.6.19 chkmail / ismail

Format: chkmail (*string* emailaddress [,*integer* timeout])
-or- ismail (*string* emailaddress [,*integer* timeout])

Use: Performs a basic syntax check of the supplied email address

If the optional parameter timeout has been specified, the function will perform a physical check of the email address on the appropriate domain.

Parameters:

emailaddress (string): email address to check

timeout (integer): unit is in milliseconds (optional)

Return value:

boolean: If emailaddress is valid, the return value is 1, otherwise 0

Examples:

Function: chkmail('info@selligent.com') *-or-* ismail('info@selligent.com')
Output: 1

Function: chkmail('infoselligent.com') *-or-* ismail('infoselligent.com')
Output: 0

Function: chkmail('info@selligent.com',3000) *-or-* ismail('info@selligent.com',3000)
Output: 1

2.6.20 isnumeric

Format: isnumeric(*variant* param)

Use: Checks if param is (or can be converted to) a valid numeric value.

Parameters:

param (variant)

Return value:

boolean: If param appears to be numeric, the return value is 1, otherwise 0

Examples:

Function:	isnumeric('abc')
Output:	0
Function:	isnumeric('123')
Output:	1
Function:	isnumeric('123abc')
Output:	0

2.6.21 convertphonenumber

Format: convertphonenumber(*string* phone_nr, *integer* country_dial_prefix, *integer* maxshortcodelength)

Use: Normalizes a phone number to the international format if possible.
If the phone number doesn't contain the country dial prefix, the provided country_dial_prefix parameter will be used.
When the maxshortcodelength parameter contains a greater value than the length of the phone number, the number is untouched.
When only 2 parameters are used (no maxshortcodelength parameter) or when more than 3 parameters are used (which doesn't have any added value), the maxshortcodelength defaults to value 4.

Parameters:
normalized

Return value:
string: Converted phone number

Examples:

Function:	convertphonenumber('0477 112345',32,11)
Output:	0477112345
→ *Untouched, as the maxshortcodelength is greater than the length of the phone number*	
Function:	convertphonenumber('+32 477 112345',32,4)

Output: +32477112345

Function: convertphonenumber('0032 477 112345',32,4)

Output: +32477112345

Function: convertphonenumber('0477 112345',32,4)

Output: +32477112345

Function: convertphonenumber('0477 112345',32,11,0,0,0)

Output: +32477112345

→ *Formatted, as the maxshortcodelength defaults to value 4*

2.6.22 hash

Format: hash(*string* value, *string* encoding)

Use: Returns the hashed value of the provided value.

Parameters:

value (string): String to be encoded
encoding (string): name of the encoding to be used.
Possible values are “md5”, “sha256”, “sha512”

Return value:

string: Converted value

Examples:

Function: hash('abcdefgh', 'md5')

Output: E8DC4081B13434B45189A720B77B6818

Function: hash('abcdefgh', 'sha256')

Output: 9C56CC51B374C3BA189210D5B6D4BF57790D351C96C47C02190ECF1E430635AB

2.6.23 generatepwd

Format: generatepwd(*integer* length)

Use: Returns a strong password of the desired length. Minimum length is 7; if a length smaller than 7 is specified, the function will default to 7.

Parameters:
length (*integer*): the length of the password to generate

Return value:
integer: indicating true or false

Examples:

Function:	generatepwd (1)
Output:	qB1Ps!K
Function:	generatepwd (10)
Output:	4hNNMD?t9g

2.6.24 regex

Format: regex (*string* value, *string* regularExpression)

Use: Validates a value with Regular Expression.

Parameters:
value (string): the variable to be validated
regularExpression (string): the regular expression to compare with

Return value:
integer: true or false

Examples:

Function:	regex(MAIL, '^([0-9a-zA-Z]([-.\w]*[0-9a-zA-Z])*@(([0-9a-zA-Z])+([-\w]*[0-9a-zA-Z])*\.)+[a-zA-Z]{2,9})$')
Output:	1 (valid)
Function:	regex(MAIL, '^([0-9a-zA-Z]([-.\w]*[0-9a-zA-Z])*@(([0-9a-zA-Z])+([-\w]*[0-9a-zA-Z])*\.)+[a-zA-Z]{2,9})$')

Output:	1 (valid)
Function:	regex(NAME, '^([0-9a-zA-Z]([-.\w]*[0-9a-zA-Z])*@(([0-9a-zA-Z])+([-\w]*[0-9a-zA-Z])*\.)+[a-zA-Z]{2,9})$')
Output:	0 (invalid)

2.6.25 guid

Format: guid( )

Use: Validates a unique string.

Parameters:
None

Return value:
string: Unique ID

Example:

Function:	guid()
Output:	7f57c9ad-13f5-4a97-9da7-1b913f2e4888

2.6.26 striphtml

Format: striphtml(*string* HTML , *string* IncludeTags , *string* ExcludeTags)

Use: Returns an HTML in which some tags have been removed. Typically, this function replaces BR-tags and P-tags by a linefeed or LI-tags are replaced by lists.

Parameters:
HTML: The html to parse.
IncludeTags (optional): The list of tags to strip/remove. In case of multiple tags, they're pipe separated.
ExcludeTags (optional): The list of tags that can't be stripped/removed. In case of multiple tags, they're pipe separated.

Return value:
string: stripped HTML

Examples:

```
Var Strip1='<p>hello <strong>World</strong><p><p>This is an example</p>'
```

Function: striphtml (Strip1)

Output:

Hello world

This is an example

Var Strip2=' hello World This is an example ', '', 'strong'

Function: striphtml (Strip2)

Output:

Hello world

This is an example

Var Strip3='
item 1
item 2
item 3
'

Function: striphtml (Strip3)

Output:

- item 1

- item 2

- item 3

2.6.27 encode (base64)

Format: encode(string, 'base64')

Use: encodes a specific string in base64.

Parameters:

string : The string value to be encoded (between single quotes).
'base64': Type of coding (between single quotes). This is a fixed value.

Return value:

string: encoded value

Example:

Function: encode('Password', 'base64')
Output: UGFzc3dvcmQ=

2.6.28 decode (base64)

Format: decode(string, 'base64')

Use: decodes a base64 encoded string value.

Parameters:

string : The encoded string to decode (between single quotes).
'base64': Type of coding (between single quotes). This is a fixed value.

Return value:

string: decoded value

Example:

Function:	decode('SGVsbG8gd29ybGQ=', 'base64')
Output:	Hello world

2.6.29 link

Format: link(id)

Use: Returns the optiextension url with the linkid as query string.

Parameters:

id: Selligent MC link ID

Return value:

string: optiextension url with linkid

Example:

Function:	link(100)
Output:	http://acme-campaign.selligent.local/optiext/optiextension.dll?ID=_ybRiQFKotGvrCkU2%2BeisQrZoCaRlvUSpYkt0avMFlrsOI8QWS6eKHiRfuScf2cvwfF2pNTFYCqPZ5N9YfVWJFq__%2B

2.6.30 linkid

Format: linkid(id)

Use: Returns the linkid.

Parameters:

id: Selligent MC link ID

Return value:
string: linkid

Example:

Function:	linkid(100)
Output:	_ybRiQFKotGvrCkU2%2BeisQrZoCaRlvUSpYkt0avMFlrsOI8QWS6eKHiRfuScf2cvwfF2pNTFYCqPZ5N9YfVWJFq__%2B

2.6.31 journey

Format: journey('name') *or* journey ('run_dt')

Use: Returns the journey name, or the most recent run date/time of the journey.

Parameters:
one of these strings can be used, including the single quotes : 'name' *or* 'run_dt'

Return value:
string: name of the journey (when 'name' is used) *or*
string: most recent run date/time of the journey (when 'run_dt' is used)

Example:

Function:	journey('name')
Output:	Newsletter Journey
Function:	journey('run_dt')
Output:	2021-01-26 11:51:04

2.6.32 position

Format: position()

Use: Returns the index position of a Data Selection item inside a Repeater.

Parameters:
None

Return value:
numeric value of the index position of the Data Selection item inside the Repeater

Example:

As example, let's say the DTS items in the Repeater in ascending order are : Smartphone A, Laptop B, Tablet C.

Function: position()
Output: 0 -> *for the first item Smartphone A*
1 -> *for the second item Laptop B*
2 -> *for the third item Tablet C*

2.6.33 count

Format: count(string DataSelectionName)

Use: Returns the number of items within a Data Selection.

Parameters:
DataSelectionName (string) : name of the Data Selection setup in the properties of the message (that's linked to a Data Selection List containing the data)

Return value:
numeric value of the number of items within the given Data Selection

Example:

In the message properties, the data selection 'DTS_prods' has been setup (linked to the Data Selection List 'Products'), that contains 145 product items.

Function: count('DTS_prods')
Output: 145

Addendum

Boolean expressions

Boolean expressions should not be used to display the outcome directly. They only return a **'true'** or **'false'** value.

Boolean expressions are useful in the following context :

1. in an **IF** expression
   Example : [% if(isnumeric([MASTER.cardid]), 'Your card id is correct', 'Your card id is incorrect. It should be a numeric value.')) %]
   => If the card id value is numeric, the first message is displayed. If not, the second message is shown.
2. in a **visibility constraint**
   Example : <sg:content expression="all(eq(tobool([VARIABLE.Loyal]),tobool('true')))">
   => The text content is only visible when the boolean variable Loyal is set to true.
3. in an expression in a **Split** component.

List of all boolean expressions :

4. contains
5. startswith
6. endswith
7. notcontains
8. isdate
9. eq
10. ne
11. lt
12. gt
13. le
14. ge
15. between
16. and / all
17. or / any
18. not
19. chkmail
20. isnumeric
21. regex

22. isempty
23. isnotempty

Grid support

All the Selligent Marketing Cloud functions are supported on Grid, **except** the following ones :

- regex
- union
- distinct
- explode
- intersect
- join