selligent MARKETING CLOUD

Functions

PRODUCT MANUAL | Jan 17, 2018

1 Foreword

Copyright

The contents of this manual cover material copyrighted by Selligent. Selligent reserves all intellectual property rights on the manual, which should be treated as confidential information as defined under the agreed upon software licence/lease terms and conditions.

The use and distribution of this manual is strictly limited to authorised users of the Selligent Interactive Marketing Software (hereafter the “Software”) and can only be used for the purpose of using the Software under the agreed upon software licence/lease terms and conditions. Upon termination of the right to use the Software, this manual and any copies made must either be returned to Selligent or be destroyed, at the latest two weeks after the right to use the Software has ended.

With the exception of the first sentence of the previous paragraph, no part of this manual may be reprinted or reproduced or distributed or utilised in any form or by any electronic, mechanical or other means, not known or hereafter invented, included photocopying and recording, or in any information storage or retrieval or distribution system, without the prior permission in writing from Selligent.

Selligent will not be responsible or liable for any accidental or inevitable damage that may result from unauthorised access or modifications.

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.

2 Table of Contents

1 Available SELLIGENT functions

1.1 String operations

1.1.1 Overview

1.1.2 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(@NAME) / LEN('Selligent')
Output:	9

1.1.3 CHARINDEX / FIND

Format: CHARINDEX(*string* needle, *string* haystack)
FIND(*string* needle, *string* haystack)

Use: Searches haystack for the first occurrence of needle.

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

Example:

Function:	CHARINDEX('e', @NAME)
Output:	5
Function:	CHARINDEX('e', 'Optizen')
Output:	5

1.1.4 FINDONEOF

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

Use: Searches haystack for the first occurrence of one of the characters 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 needles found: offset of character in haystack

If none found: -1

Example:

Function:	FINDONEOF('a,e', 'I love cats')
Output:	5

1.1.5 LASTCHARINDEX / REVERSEFIND

Format: LASTCHARINDEX(*string* needles, *string* haystack)
REVERSEFIND(*string* needles, *string* haystack)

Use: Searches haystack for the last occurrence of needle

Parameters:

Needles (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:	LASTCHARINDEX('a', 'I love cats') REVERSEFIND('a', 'I love cats')
Output:	8

1.1.6 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

1.1.7 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

1.1.8 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

1.1.9 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

1.1.10 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

1.1.11 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

1.1.12 UCASE

Format: UCASE(*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:	UCASE('I love cats')
Output:	I LOVE CATS

1.1.13 LCASE

Format: LCASE(*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:	LCASE('I love cats')
Output:	i love cats

1.1.14 PCASE

Format: PCASE(*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:	PCASE('I love cats')
Output:	I Love Cats

1.1.15 TRANSLATE

Format: TRANSLATE(*string* fieldname, *string* multivalue, *string* separator1, *string* separator2)

Use: Translates the multivalue, will separate all values with separator1, except for the last value; that will be separated by separator2.

Parameters:

Fieldname (string): Name of the field to which the option list is linked
Multivalue (string): string to be translated
Seperator1 (string): Seperator1
Seperator2 (string): Seperator2

Return value:

string: returns the converted translated value

Example:

This example is for a user with a value “1,2,3,6” in the INTEREST-field on his profile

On this field the following field options are defined:

1 Sports

2 Politics

3 Movies

4 Cars

5 Animals

6 Nature

While ~$INTERESTS~ would return “Sports, Politics, Movies, Nature”, this function allows us to set the separators

Function: TRANSLATE('INTERESTS',INTERESTS,',',' and') where INTERESTS is the name of the field in the audience list

Output: Sports, Politics, Movies and Nature

Function: '<UL><LI>' & TRANSLATE('INTERESTS', INTERESTS, '</LI><LI>', '</LI><LI>') & '</LI></UL>'

Output: <UL><LI>Sports</LI><LI>Politics</LI><LI>Movies</LI><LI>Nature</LI></UL>

1.1.16 &

Format: *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:	'I' & ' Love ' & 'Cats'
Output:	I Love Cats

1.1.17 CONTAINSTAG

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

Use: search for a string within a string

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

Return value:
string: returns 1 if needle was found

Example:

Function:	CONTAINSTAG('CAT,DOG,Frog,doggybag,hotdog','dog')
Output:	1
Function:	CONTAINSTAG('CAT,DOG,Frog,doggybag,hotdog','doggy')
Output:	0

1.1.1 JSREPLACE

Format: JSREPLACE(*string* variable)

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

Parameters:
variable (string): string to be searched

Return value:
string: returns the escapes/modified string

Example:

Function:	JSREPLACE(@VARIABLE)//where @VARIABLE is ab(cd'ef)gh)
Output:	abcd\'efgh

1.1.2 GETPROP (New 6.3.4)

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 seperators

Return value:
string

Example:

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

1.2 Math functions

1.2.1 Overview

1.2.2 ABS

Format: ABS(*integer* number)

Use: number, without sign. (basically converts negative numbers to positive ones).

Parameters:
Number (integer): Number to convert

Return value:
integer: returns the converted number

Example:

Function:	ABS(5)
Output:	5
Function:	ABS(-5)
Output:	5

1.2.3 AVG

Format: AVG(*integer* number_1, *integer* number_2, …. *Integer* number_n)

Use: Returns the average of the provided numbers

Parameters:
Number_n (integer):

Return value:
integer: returns the calculated number

Example:

Function:	AVG(1,2,3,4,5)
Output:	3

1.2.4 FMUL

Format: FMUL(*float* number_1, *float* number_2)

Use: Returns the floating point result of multiplying number_1 with number_2.

Parameters:
Number_1,number_2 (float):numbers to multiply

Return value:
float

Example:

Function:	FMUL(1.2,3.4)
Output:	4.08

1.2.5 FDIV

Format: FDIV(*float* nominator, *float* denominator)

Use: Returns the floating point result of the division of nominator with denominator.

Parameters:
nominator (float)
denominator (float)

Return value:
float

Example:

Function:	FDIV(3.4, 1.2)
Output:	2.83

1.2.6 FADD

Format: FADD(*float* number_1, *float* number_2)

Use: Returns the floating point result of adding number_1 with number_2.

Parameters:
Number_1 (float)
Number_2 (float)

Return value:
float

Example:

Function:	FADD(3.4, 1.2)
Output:	4.60

1.2.7 FSUB

Format: FSUB(*float* number_1,*float* number_2)

Use: Returns the floating point result of subtracting number_2 from number_1.

Parameters:
Number_1 (float)
Number_2 (float)

Return value:
float

Example:

Function:	FSUB(3.4, 1.2)
Output:	2.20

1.2.8 FABS

Format: FABS(*float* number)

Use: converts negative float numbers to positive ones.

Parameters:
Number (float): number to convert

Return value:
float: the converted number

Example:

Function:	FABS(5.23)
Output:	5.23
Function:	FABS(-5.23)
Output:	5.23

1.2.9 FROUND

Format: FROUND(*float* number, *int* digits)

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

Parameters:
Number (float): number to convert

Digits (int): number of digits

Return value:
string: returns the rounded number

Example:

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

1.2.10 FCOMPARE

Format: FCOMPARE(*float* number1, *float* number2)

Use: compares two float numbers and returns a number indicating if they are equal or not.

Parameters:

Number1 (float): number to evaluate

Number2 (float): number to compare with

Return value:

Int: returns 0 is the numbers are equal, 1 if Number 1 is bigger than Number2 and -1 if Number1 is smaller than Number 2

Example:

Function:	FCOMPARE(1.23,3.456)
Output:	-1
Function:	FCOMPARE(1.23, 0-3.456)
Output:	1
Function:	FCOMPARE(4.563,1.23)
Output:	1
Function:	FCOMPARE(1.23,1.23)
Output:	0

1.2.11 % (MODULO)

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

Example:

Function:	3%2
Output:	1
Function:	12%8
Output:	4

1.2.12 FORMATNUMBER

Format: FORMATNUMBER(*float* value, *int* decimals, *string* thousandSeparator, *string* decimalSeparator, *string* prefix, *string* postfix)

Use: Formats a given number with custom separators, prefix and postfix.

Parameters:
Value (float): The number to format

Decimals (int): The number of decimals to show

ThousandSeparator (string: The character to show in between thousands-groups

DecimalSeparator (string: The character to show between the whole- and the decimal-part

Prefix (string): The text to show in front of the number

Postfix (string): The text to show at the end of the number

Return value:
string: The formatted number

Example:

Function:	FORMATNUMBER(1234567.1234, 2, '.', ',', '€ ', ' ,-')
Output:	€ 1.234.567,12 ,-
Function:	FORMATNUMBER(1234567.1234, 3, ',', '.', '$ ', '##')
Output:	$ 1,234,567,123##

Function:	FORMATNUMBER(123.456, 2, ',', '.', '¥', '')
Output:	¥ 123.46

1.3 Date functions

1.3.1 Overview

1.3.2 DATEADD

Format: DATEADD(*constant* datetype, *integer* amount, *datetime* date)

Use: Returns datetime, added with an amount of years, months, days,... depending of datetype. If the function is used with negative numbers, use singles quotes around the number.

Parameters:

datetype (constant)

yyyy,yy amount in years
mm.m amount in months
dd.d amount in days
hh.h amount in hours
mi.n amount in minutes
ss.s amount in seconds
amount (integer): Number of units of type datetype to add

date (datetime): date to add to

Return value:

Datetime: Returns the resulting datetime value

Example:

Function:	DATEADD('dd',4, '2009-09-24')
Output:	2009-09-28 00:00:00
Function with negative number: DATEADD('dd','-1',GetDate())	

1.3.3 DATEPART

Format: DATEPART(*constant* datetype, *datetime* date)

Use: Returns the part of the date, indicated by datetype

Parameters:

datetype (constant):

yyyy,yy amount in years
mm.m amount in months
wk.wks amount of weeks (Sunday based weeks)
wkm amount of weeks (Monday based weeks)
dd.d amount in days
hh.h amount in hours
mi.n amount in minutes
ss.s amount in seconds

date (datetime): date to add to

Return value:

Integer: Returns the requested part of date, indicated by datetype

Example:

Function:	DATEPART('yyyy', '2009-09-24')
Output:	2009
Function:	DATEPART('mm', '2009-09-24')
Output:	9

1.3.4 DATEDIFF

Format: DATEDIFF(*constant* datetype, *datetime* date_1,*datetime* date_2)

Use: Returns the difference between the part of date_1 and date_2, indicated by datetype.

Parameters:

datetype (constant):

yyyy,yy amount in years
mm.m amount in months
dd.d amount in days
hh.h amount in hours
mi.n amount in minutes
ss.s amount in seconds

Date_1 (datetime)
Date_2 (datetime)

Return value:
Integer: Returns the difference between the requested part of date_1 – the part of date_2

Example:

Function:	DATEDIFF('mm', '2009-01-12','2009-09-24')
Output:	8

1.3.5 GETDATE

Format: GETDATE()

Use: Returns the current system datetime.

Parameters:
none

Return value:
datetime: Returns the current system datetime

Example: On September 24th 2009 at 1PM:

Function:	GETDATE()
Output:	2009-09-24 13:00:00

1.3.6 ISDATE

Format: ISDATE(*string* date)

Use: Returns the current system datetime.

Parameters:
Date (datetime): date to be checked

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

Example:

Function:	ISDATE('2009-01-12')
Output:	1

Function:	ISDATE('2009-02-31')
Output:	0

1.3.7 FORMATDATETIME

Format: FORMATDATETIME(*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):

%a: Abbreviated weekday name
%A: Full weekday name
%b: Abbreviated month name
%B: Full month name
%c: Date and time representation appropriate for locale
%d: Day of month as decimal number (01 – 31)
%H: Hour in 24-hour format (00 – 23)
%I: Hour in 12-hour format (01 – 12)
%j: Day of year as decimal number (001 – 366)
%m: Month as decimal number (01 – 12)
%M: Minute as decimal number (00 – 59)
% p: Current locale's A.M./P.M. indicator for 12-hour clock
%S: Second as decimal number (00 – 59)
%U: Week of year as decimal number, with Sunday as first day of week (00 – 53)
%w: Weekday as decimal number (0 – 6; Sunday is 0)
%W: Week of year as decimal number, with Monday as first day of week (00 – 53)
%x: Date representation for current locale
%X: Time representation for current locale
%y: Year without century, as decimal number (00 – 99)
%Y: Year with century, as decimal number
%z,%Z: Time-zone name or abbreviation; no characters if time zone is unknown
%%: Percent sign

Return value:

string: Returns the formatted string

Example: On September 24th 2009 at 1PM:

Function:	FORMATDATETIME(GETDATE(), '%A, %B %d, %Y')
Output:	Thursday, September 24, 2009

Function: FORMATDATETIME(GETDATE(),'%c')

Output: 09/24/09 13:53:25

Function: FORMATDATETIME(GETDATE(),'%j')

Output: 267

1.3.8 YEAR

Format: YEAR(*string* date)

Use: Returns the year of the specified datetime.

Parameters:
Date (datetime):

Return value:
integer:

Example:

Function: YEAR('2009-01-12 12:34:56')

Output: 2009

1.3.9 MONTH

Format: MONTH(*string* date)

Use: Returns the month of the specified datetime.

Parameters:
Date (datetime):

Return value:
integer:

Example:

Function: MONTH('2009-01-12 12:34:56')

Output: 1

1.3.10 DAY

Format: DAY(*string* date)

Use: Returns the day of the specified datetime.

Parameters:
Date (datetime):

Return value:
integer:

Example:

Function:	DAY('2009-01-12 12:34:56')
Output:	12

1.3.11 HOUR

Format: HOUR(*string* date)

Use: Returns the hour of the specified datetime.

Parameters:
Date (datetime):

Return value:
integer:

Example:

Function:	HOUR('2009-01-12 12:34:56')
Output:	12

1.3.12 MINUTE

Format: MINUTE(*string* date)

Use: Returns the hour of the specified datetime.

Parameters:
Date (datetime):

Return value:
integer:

Example:

Function:	MINUTE('2009-01-12 12:34:56')
Output:	34

1.3.13 SECOND

Format: SECOND(*string* date)

Use: Returns the second of the specified datetime.

Parameters:
Date (datetime):

Return value:
integer:

Example:

Function:	SECOND('2009-01-12 12:34:56')
Output:	56

1.3.14 DAYOFWEEK

Format: DAYOFWEEK(*string* date)

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

Parameters:
Date (datetime):

Return value:
integer:

Example:

Function:	DAYOFWEEK('2009-01-12 12:34:56')
Output:	2

1.3.15 GETCURRENTUNIXTIMESTAMP()

Format: GETCURRENTUNIXTIMESTAMP()

Use: retrieves the unix timestamp in milliseconds

Parameters: none

Return value: datetime

1.3.16 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

1.4 Other functions

1.4.1 Overview

CHKPROP	PROPCOUNT	MATCHCOUNT	MAKELEN
IF	URLENCODE	URLDECODE	H2U
H2A	U2H	A2H	CHKMAIL
ISNUMERIC	CONVERTPHONENUMBER	UTF8	UTF8DECODE
CONVERTCHARSET	MDENCODE	GETCAMPAIGNID	GETACTIONID
GETLISTID	GETUSERID	GETPROBEID	GETTRIGGERID
GETCATALOGLISTID	GETCATALOGITEMID		CHKBANKACCOUNT
GENERATEPWD	CHKREGEX	GENERATEGUID	

1.4.2 CHKPROP

Format: CHKPROP(*string* multivalue, *string* value)

Use: Searches a SELLIGENT multivalue (pipe or comma separated string) for the occurrence of value.

Parameters:
multivalue (string): string to search
value (string); string to find

Return value:
Boolean: If value found; returns 1, otherwise 0

Example:

Function:	CHKPROP('a,b,c,d,e','b')
Output:	1
Function:	CHKPROP('a,b,c,d,e','z')
Output:	0

1.4.3 PROPCOUNT

Format: PROPCOUNT(*string* multivalue)

Use: the amount of values, contained in the SELLIGENT multivalue.

Parameters:
multivalue (string): string to search

Return value:
integer: The amount of contained values

Example:

Function:	PROPCOUNT('a,b,c,d,e')
Output:	5

1.4.4 MATCHCOUNT

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

Use: Counts the amount of values that match between the 2 SELLIGENT multivalues.

Parameters:
Multivalue_1 (string)
Multivalue_2 (string)

Return value:
integer: The amount of matched values found

Example:

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

1.4.5 MAKELEN

Format: MAKELEN(*int* number, *int* desiredlen)

Use: If the amount of decimals in number is lower than desiredlen, the number is prefixed with '0's, until the length matches with desiredlen.

Parameters:
number (integer): number
desiredlen (integer): string to find

Return value:
integer: number decreased or completed to the desired length

Example: For user 1 in the list:

Function:	MAKELEN(ID,5)
Output:	00001

1.4.6 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

Example:

Function:	IF('A'='A','Identical', 'Different')
Output:	Identical

Function:	IF('A'='B','Identical', 'Different')
Output:	Different

1.4.7 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

1.4.8 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

1.4.9 H2U

Format: H2U(*string* str)

Use: Converts the html encoded values contained in str to a Unicode string (this function only performs a conversion on a Unicode installation; for ansi installations, the returnvalue is identical to the supplied parameter, without doing a conversion)..

Parameters:
str (string): String containing html encoded values (&# + ASCII-value) to convert to unicode

Return value:
String: converted str

Example:

Function:	H2U('@')
Output:	@

1.4.10 H2A

Format: H2A(*string* str)

Use: Converts the html encoded values contained in str to an Ansi string (this function works on both Ansi and Unicode installations, but will only perform a conversion of Htmlencoded values within the ansi range (0-0xff)).

Parameters:
str (string): String containing html encoded values to convert to ansi

Return value:
String: converted str

Example:

Function:	H2A('@')
Output:	@

1.4.11 U2H (only available in Unicode version of SELLIGENT)

Format: U2H(*string* str)

Use: Converts the Unicode characters in str that are out of the standard range (>0x7F) to their html encoded values. (this function only performs a conversion on a Unicode installation; for ansi installations, the return value is identical to the supplied parameter, without doing a conversion).

Parameters:
str (string): String containing unicode characters that are out of the standard range

Return value:
String: converted str

Example:

Function:	U2H('@')
Output:	@ (character is in standard range, hence no transformation)
Function:	U2H('ö')
Output:	ö

1.4.12 A2H (only available in Unicode version of SELLIGENT)

Format: A2H(*string* str)

Use: Converts the ansi characters in str that are out of the standard range (>0x7F) to their html encoded values. (this function works on both Ansi and Unicode installations, but will only perform a conversion of values within the ansi range (0-0xff)).

Parameters:
str (string): String containing ansi characters that are out of the standard range

Return value:
String: converted str

Example:

Function:	A2H('@')
Output:	@ (character is in standard range, hence no transformation
Function:	A2H('ö')
Output:	ö

1.4.13 CHKMAIL

Format: CHKMAIL(*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 emailaddess on the appropriate domain

Parameters:
Emailaddress (string): Email address to check
timeout (integer): Optional; unit is in milliseconds

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

Example:

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

Output: 1

Function: CHKMAIL('infoselligent.com')

Output: 0

Function: CHKMAIL('info@selligent.com',3000)

Output: 1

1.4.14 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

Example:

Function: ISNUMERIC('abc')

Output: 0

Function: ISNUMERIC('123')

Output: 1

Function: ISNUMERIC('123abc')

Output: 0

1.4.15 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. Shortcodes, shorter than the specified shortcodelength will not be..

Parameters:
normalized

Return value:

string: Converted phone number

Example:

Function: CONVERTPHONENUMBER('+32 477 112345',32,11,4)

Output: +32477112345

Function: CONVERTPHONENUMBER('0032 477 112345',32,11,4)

Output: +32477112345

Function: CONVERTPHONENUMBER('0477 112345',32,11,4)

Output: +32477112345

1.4.16 UTF8

Format: UTF8(*string* value)

Use: Converts the supplied value to UTF8.

Parameters:
Value (string): string to convert to UTF8

Return value:
string: Converted value

Example:

Function:	UTF8('Bàrt')
Output:	bÃ rt

1.4.17 UTF8DECODE

Format: UTF8DECODE(*string* value, string charset)

Use: Converts the supplied UTF8-value to the provided charset.

Parameters:
Value (string): UTF8-String to convert to UTF8
charset (string): the charset to convert the value to

Return value:
string: Converted value

Example:

Function:	UTF8DECODE(UTF8('bàéèrt'),'iso-8859-1')
Output:	bàéèrt

1.4.18 CONVERTCHARSET

Format: CONVERTCHARSET(*string* value, *string* charset1, *string* charset2)

Use: Converts the supplied value from the provided charset to the second provided charset.

Parameters:
Value (string): UTF8-String to convert to UTF8
charset1 (string): the charset to convert the value to

Return value:
string: Converted value

Example:

Function:	CONVERTCHARSET('bàrt','iso-8859-1','utf-8')
Output:	bÃ rt

1.4.19 MDENCODE

Format: MDENCODE(*string* value)

Use: Returns the MD5-encoded value of the provided value.

Parameters:
Value (string): String to be encoded

Return value:
string: Converted value

Example:

Function:	MDENCODE('abcdefgh')
Output:	E8DC4081B13434B45189A720B77B6818

1.4.20 MDENCODE_W

Format: MDENCODE_W(*string* value)

Use: Returns the MD5-encoded value after a conversion to UTF-8 of the provided value

Parameters:

Value (string): string to be encoded

Return value:

String: encoded value

Example:

Function: MDENCODE_W(MAIL) where MAIL=Ben.demoree@selligent.com	
Output:	**3000DA4541BEAD6FFB5E548A425DE7A2**

1.4.21 GETCAMPAIGNID

Format: GETCAMPAIGNID(*string* hashcode)

Use: Returns the ID of the campaign the component is in.

Parameters:
hashcode (string): hashcode to decode the ID from

Return value:
string: Current CampaignID

Example:

Function:	GETCAMPAIGNID(@ID)
Output:	1

1.4.22 GETACTIONID

Format: GETACTIONID(*string* hashcode)

Use: Returns the ID of the active component in the current campaign.

Parameters:
hashcode (string): hashcode to decode the ID from

Return value:
string: Current ActionID

Example:

Function:	GETACTIONID(@ID)
Output:	4

1.4.23 GETLISTID

Format: GETLISTID(*string* hashcode)

Use: Returns the ID of the list the current campaign is working on.

Parameters:
hashcode (string): hashcode to decode the ID from

Return value:
string: Current ListID

Example:

Function:	GETLISTID(@ID)
Output:	12

1.4.24 GETUSERID

Format: GETUSERID(*string* hashcode)

Use: Returns the ID of the current user.

Parameters:
hashcode (string): hashcode to decode the ID from

Return value:
string: Current UserID

Example:

Function:	GETUSERID(@ID)
Output:	27

1.4.25 GETPROBEID

Format: GETPROBEID(*string* hashcode)

Use: Returns the ID of the clicked sensor.

Parameters:
hashcode (string): hashcode to decode the ID from

Return value:
string: Current ProbeID

Example:

Function:	GETPROBEID(@ID)
Output:	100

1.4.26 GETTRIGGERID

Format: GETTRIGGERID(*string* hashcode)

Use: Returns the ID of the trigger that initiated the execution of the campaign-flow.

Parameters:
hashcode (string): hashcode to decode the ID from

Return value:
string: Current TriggerID

Example:

Function:	GETTRIGGERID(@ID)
Output:	0

1.4.27 GETCATALOGLISTID

Format: GETCATALOGLISTID(*string* hashcode)

Use: Returns the ID of the article list that contained the link that initiated the current step.

Parameters:
hashcode (string): hashcode to decode the ID from

Return value:
string: Current CatalogListID

Example:

Function:	GETCATALOGLISTID(@ID)
Output:	344

1.4.28 GETCATALOGITEMID

Format: GETCATALOGITEMID(*string* hashcode)

Use: Returns the ID of the article that contained the link that initiated the current step.

Parameters:
hashcode (string): hashcode to decode the ID from

Return value:
string: Current CatalogItemID

Example:

Function:	GETCATALOGITEMID(@ID)
Output:	17

1.4.29 CHKBANKACCOUNT

Format: CHKBANKACCOUNT(*string* accountnumber, *string* countrycode)

Use: Checks the format of an account number.

BE Checks if the length of the accountnumber is 12, performs a modulo-check (%97) and checks the checksum-digits.

NL Checks if the lengt is less than 11.
In case the length is less than 8, the accountnumber is accepted as 'GIRO'number.

If the length equals 9 or 10, a modulo-check (%11) is performed and checked with the checksum-digits.

Parameters:
Accountnumber (string): the account number to be checked
Countrycode (string): the ISO- country code. Possible values are BE and NL

Return value:
integer:

Example:

Function:	CHKBANKACCOUNT('1234', 'NL')
Output:	1 (valid - Giro)
Function:	CHKBANKACCOUNT('12345678912', 'NL')
Output:	0 (invalid)
Function:	CHKBANKACCOUNT('000-0000011-11', 'BE')
Output:	1 (valid)

1.4.30 CHKIBAN (NEW for 6.3.4)

Format: CHKBIBAN(*string* accountnumber)

Use: Checks if an account number is a valid IBAN number

Parameters:
Accountnumber (string): the account number to be checked

Return value:
1: valid or 0: not valid

Example:

Function:	CHKBIBAN('BE6853900754034')
Output:	0

1.4.31 GENERATEPWD

Format: GENERATEPWD(*int* 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 (int): the length of the password to generate

Return value:
integer: indicating true or false

Example:

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

1.4.32 CHKREGEX

Format: CHKREGEX (*string* RegularExpression , *string* fieldname)

Use: Validates a value with Regular Expression.

Parameters:

RegularExpression (string): the regular expression to compare with

Fieldname (string): the variable to be validated

Return value:

integer: true or false

Example:

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

Output: 1 (valid)

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

Output: 1 (valid)

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

Output: 0 (invalid)

1.4.33 GENERATEGUID

Format: GENERATEGUID( )

Use: Validates a unique string.

Parameters:

None

Return value:

string: Unique ID

Example:

Function: GENERATEGUID()

Output: 956e45d6443c4e3617ee64

1.4.34 WORDWRAP

Format: WORDWRAP(*int* LineLength , *string* Text , *int* TabLength)

Use: returns wrapped text.

Parameters:

LineLength: the maximum length of a rule of text

Text: the text that needs to be wrapped

TabLenght: number of spaces a tab takes

Return value:

string: wrapped text

Note: this function only works correctly in the text version of the message.

1.4.35 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, P, tags by a linefeed or LI tags are replaced by lists.

Parameters:

Html: the html to parse

IncludeTags: the list of tags to strip/remove. If multiple tags they are pipe separated

ExcludeTags: the list of tags that cannot be stripped/removed. If multiple tags they are pipe separated.

Return value:

string: stripped HTML

Example:

Var Strip1=' hello World This is an example '

STRIPHTML(Strip1)

Hello world

This is an example

Var Strip1=' hello World This is an example ', '', 'strong'

STRIPHTML(Strip1)

Hello world

This is an example

Var Strip1='
item 1
item 2
item 3
'

STRIPHTML(Strip1)

- item 1

- item 2

- item 3

1.4.36 SHAENCODE

Format: SHAENCODE(SHAVersion, toEncode)

Use: encodes a string in SHA-version number

Parameters:

SHAVersion: the encoding to be applied. Possible options are 1, 224, 256, 384 and 512. Any other version will result in an empty string

toEncode: the string to encode

Return value: string: encoded string

1.4.37 BASE64ENCODE

Format: BASE64ENCODE(toEncode)

Use: encodes a string in BASE64

Parameters:

toEncode: the string to encode

Return value: string: encoded string

1.4.38 BASE64DECODE

Format: BASE64DECODE(toDecode)

Use: decodes a BASE64 to a string

Parameters:

toDecode: the encoded string to decode

Return value: string: decoded string

1.4.39 TOBOOL

Format: TOBOOL(*var*).

Use: converts the value to a boolean (1 or 0).

Parameters:
var to convert

Return value:

A boolean

1.4.40 TOINT

Format: TOINT(*var*).

Use: converts the value to an integer

Parameters:
var to convert

Return value:

An integer

1.4.41 TOFLOAT

Format: TOFLOAT(*var*).

Use: converts the value to a float

Parameters:
var to convert

Return value:

A float

1.4.42 TODATE

Format: TODATE(*var*).

Use: converts the value to a date.

Parameters:
var to convert

Return value:

A date

1.4.43 TODATETIME

Format: TODATETIME(*ar*).

Use: converts the value to a datetime.

Parameters:
var to convert

Return value:

A datetime

1.4.44 TOSTRING

Format: TOSTRING(*var*).

Use: converts the value to a string

Parameters:
var to convert

Return value:

A string

1.4.45 LOCALIZE

Format: LOCALIZE(var,locale)

Use: use this function to localize a field (eg. Date)

Parameters:

Var representing the field to localize

Local: the local to which the variable should be localized

Return value: depends on the settings of the sytem.

1.5 Template functions

1.5.1 Overview

COUNT/ ARTICLE_COUNT ARTICLE_PROPERTY CONVERT

1.5.2 COUNT/ARTICLE_COUNT

Format: COUNT(*string* ArticleContainerName [\ *string* FilterName])
ARTICLE_COUNT(*string* ArticleContainerName [\ *string* FilterName])

Use: Returns the number of articles in the article-container or article-container-list.

Parameters:
ArticleContainerName (string): name of the article container
FilterName (string): Name of the filter on the article container

Return value:
string: the number of articles in the container

Example:

Function:	COUNT('ARTICLES') COUNT('ARTICLES\TOC')
Output:	5

1.5.3 ARTICLE_PROPERTY

Format: ARTICLE_PROPERTY(string ArticleContainerName [\ string *FilterName*],
int ArticleIndex, string FieldName)

Use: Returns the content of the *FieldName* from a specific article (ArticleIndex) in a specific article container (ArticleContainerName).

Parameters:

ArticleContainerName (string): name of the article container
FilterName (string): Name of the filter on the article container
ArticleIndex (int): ID of the article container
Fieldname (string): Name of the field to get the content from

Return value:

string: the content retrieved from the article field

Example:

Function:	ARTICLE_PROPERTY('ARTICLES\TOC',1, 'Title')
Output:	I love cats

1.5.4 CONVERT

Format: CONVERT ('datetime', *string* value, *int* format)

Use: Converts provided value to type datetime

Parameters:

value (string): string to be converted to date
format (integer): format to convert the date to. This value will always be 120

Return value:

string: returns the converted datetime

Example:

Function:	CONVERT('datetime', '2009-06-13', 120)
Output:	2009-06-13 00:00:00