CA Test Data Manager - 4.1

Data Generation Functions and Parameters

Date: 11-Aug-2017

technologies

Table of Contents

Data Generation Functions and Parameters

This page contains a comprehensive list of the Datamaker functions and their parameters.

Use double quotes to pass literal commas and spaces in arguments. Arguments to functions can be double quoted, but the behavior between Datamaker and CA TDM Portal varies. CA TDM Portal always strips double quotes, but Datamaker may or may not. Differences are noted in the function list. In Datamaker, @seedlist, @date, @ibann, @jdate fail if supplied with a double quoted string. Single quotes are always returned.

Functions, variables, and column references embedded in a quoted string are still evaluated. Although unintuitive, this behavior is particular to Datamaker and is replicated in CA TDM Portal.

Click to expand table of contents...

Boolean Expressions

A Boolean expression can be a constant representing `true` or `false`.

A `true` value is one of the following:

- the words `true` or `yes`
- a number which is not 0 or -1
- something else which is none of the false values below.

A `false` value is one of the following:

- the numbers 0 or -1
- the words `false`, `no`, or `null`
- an empty string
- a string whose first character is N, n, F, f, l, I, ! or ?.

A Boolean function is one of @and, @not and @or. For more information, see the functions on this page.

A Boolean expression consists of the following:

```
<left hand operand> <logical operator> [<right hand operand>]
```

An operand can be any string or numeric expression made of constants, functions, variables or column references. All these operators work with both numbers and strings. If the operands are strings, then a lexicographic comparison is made, for example, A is less than Z. If the operands are floating point numbers, then tests for equality are done within a fixed precision of 1.0e-10, because floating point numbers do not have a guaranteed accuracy.

A logical operator can be one of:

- = equality test
- <> inequality test
- IS NULL tests if the left hand operand is NULL or empty. This operator does not require a right hand operand.
- > greater than
- < less than
- >= greater or equal to

- <= less than or equal to

In Datamaker, do not enclose just one operand in quotes; for example, the expression `"a"=a` returns false.

A Boolean expression is only evaluated when used in a function that requires a Boolean expression. They are not evaluated when used as an argument to a macro. For example:

- In `@if(a=b,yes,nno)@`, the expression “a=b” will be evaluated to true or false.
- In `@randchars(4,4,a=b)@`, the expression “a=b” is read literally as the characters “a”, “=” and “b”.

11PROOF(SEQUENCE, SIGN)

Finds the next Elf-Proef number from the sequence, according to the Dutch bank account number validation method.

Parameters:

- SEQUENCE — name of the sequence to use. The sequence is created if it does not already exist.
- SIGN — the arithmetic operator + or -

Return value: the Elf-Proef number

Example: @11proof(mysequence,+)@

Example result: 100000010

ABS(NUMBER)

Returns the absolute value of the argument.

Parameters:

- NUMBER — an integer or floating point number.

Return value: An absolute value

Example: @abs(-23)@

Example result: 23

ADD(NUMBER1,NUMBER2)

Adds two values together.

Parameters:

- Two expressions evaluating to integer or floating point numbers.

Return value: The sum of the two numbers.

Example: @add(1,1)@

Example result: 2

Example: @add(~v1~,~v2~)@

Example result: The value of variable v1 plus v2.

ADDCHECKSUM(NUMBER, METHOD)

Adds special LUHN, VERHOEFF, or CB checksum.

Parameters:

- NUMBER — a number
- METHOD — name of the checksum method to be used; one of:
  - LUHN
  - VERHOEFF
  - CB

Return value: The input number with added checksum.

Example: @addchecksum(343, luhn)@

Example result: 3434

ADDDAYS(DATE, DAYS)

Adds a number of days to a date.

Parameters:

- DATE — a date in any of the usual formats, eg dd/mm/yyyy, dd-mm-yyyy and so on. The function will try the project date format first and if that fails, a number of other possible formats.
- DAYS — the number of days to add (or subtract if negative).

Return value: The new date in project date format.

Example: @adddays(12/05/2017,35)@

Example result: 16/06/2017

ADDLUHN(NUMBER)

Adds a Luhn checkdigit. The Luhn algorithm or Luhn formula, also known as the "modulus 10" or "mod 10" algorithm, is a simple checksum formula used to validate a variety of identification numbers, such as credit card numbers, IMEI numbers, National Provider Identifier numbers in US, and Canadian Social Insurance Numbers.

Parameters:

- NUMBER — a number of any length

Return value: number + Luhn checkdigit

Example: `@addluhn(123)@`

Example result: 1236

Examples:

```
@addluhn(@randlov(0, @list(34,37)@)@@nextval(AMEX_SEQ, 100000000000)@)@           ;
AMEX Card
@addluhn(@randlov(0, @list(51,52,53,54,55)@)@@nextval(MC_SEQ, 1000000000000)@)@   ;
MASTER Card
@addluhn(4@nextval(VS13_SEQ,10000000000)@)@                                       ;
VISA 13 digit
@addluhn(4@nextval(VS16_SEQ,10000000000000)@)@                                    ;
VISA 16 digit
```

ADDMICROSECS(TIMESTAMP, MICROSECONDS)

Adds a number of microseconds to a timestamp. The timestamp resolution is 1 millisecond, so adding 1 microsecond will not change the timestamp, but adding 1000 microseconds will.

Parameters:

- TIMESTAMP — a timestamp,
- MICROSECONDS — the number of microseconds to add.

Return value: new timestamp in project timestamp format

Example: @addmicrosecs(~STIMESTAMP~,1000)@

ADDMILLISECS(TIMESTAMP, MILLISECONDS)

Adds a number of milliseconds to a timestamp. The function will use the project format to interpret the timestamp first and then a number of other formats until it finds one that matches. If the timestamp format cannot be recognized, the function fails. For example "08/08/2016 03:36:20.000".

Parameters:

- TIMESTAMP — a timestamp,
- MILLISECONDS — a number of milliseconds to add.

Return value: The new timestamp in project timestamp format

Example: @addmillisecs(~STIMESTAMP~,136)@

Example result: 08/08/2014 03:36:20.136

ADDMOD97(NUMBER)

Adds a modulo97 checksum code to a number.

Parameters:

- NUMBER — A long integer number

Return value: The input value followed by the two checksum digits.

Example: @addmod97(100)@

Example result: 10003

ADDMONTHS(DATE, MONTHS)

Adds a number of months to a date. The function interprets dates in the project format first, followed by a number of other formats, until a match is found.

Parameters:

- DATE — a date
- MONTHS — the number of months to add. Can be negative

Return value: a new date in project format.

Example: @addmonths(05/05/2020,3)@

Example result: 05/08/2020

ADDRAND(NUMBER,MIN,MAX)

Addsa random number between a specified minimum and maximum to a number. If MAX is less than MIN, the value returned is NUMBER plus MIN minus 1.

Parameters:

- NUMBER — a long integer number
- MIN — minimum value for a random number
- MAX — maximum value for a random number

Return value: a new number

Example: @addrand(1,1,5)@

Example result: 2,3,4,5,6

ADDRANDDAYS(DATE,MIN,MAX)

Adds a random number of days between the min and max to the date. If MAX is less than MIN, the value returned is the DATE plus the MIN minus 1.

Parameters:

- DATE — a date in project date format
- MIN — minimum number of days to add
- MAX — maximum number of days to add

Return value: a new date in project format.

Example: @addranddays(13/04/1977,3,10)@

Example result: 22/04/1977

ADDSECONDS(DATETIME, SECONDS)

Adds a number of seconds to a datetime value.

Parameters:

- DATETIME — a date
- SECONDS — number of seconds to add

Return value: a new datetime

Example: @addseconds(~CDATE~,1)@

Example result: 2008/07/04 00:00:01

ADDSECONDS(TIME, SECONDS)

Adds a number of seconds to a time.

Parameters:

- TIME — a time in project format, usually hh:mm:ss
- SECONDS — the number of seconds to add.

Return value: the new time in project format

Example: @addseconds(01:02:00,1)@

Example result: 01:02:01

ADDVERHOEFF(NUMBER)

Adds a checksum using the Verhoeff algorithm.

Parameters:

- NUMBER — a number

Return value: a new number followed by a checksum digit.

Example: @addverhoeff(1234)@

Example result: 12340

ADDYEARS(DATE, YEARS)

Adds a number of years to a date. IF the DATE does not match the project format, the functions attempts to interpret the DATE in other formats.

Parameters:

- DATE — a date in project format
- YEARS — the number of years to add

Return value: a new date in project format.

Example: @addyears(2008/07/01, 1)@

Example result: 2009/07/01

ALPHANUM(STRING)

Remove all non-alphanumeric characters from a string.

Parameters:

- STRING — a character string to be cleaned up

Return value: a new string containing only letters and digits

Example: @alphanum(12345%%abc)@

Example result: 12345abc

ASC(STRING)

Returns the ASCII character code of the first character of the string. This is actually a Unicode code point which will be mostly the same as ASCII for European language strings.
If the string is double quoted, CA Datamaker returns the code of the double quote. CA TDM Portal returns the code for the first character after the double quote.

Parameters:

- STRING — a character string

Return value: a new string. If the input was empty, it returns an empty result.

Example: @asc(Apple)@, **Result**: 65

ASLIST(LIST[,COLUMN1][,COLUMN2][, QUOTED])

Generates a list of comma separated strings from the specified list function.

Usage in TDOD: Specify the two optional columns to define a drop-down variable for use in TDOD screens. Such variables have two attributes: a key value (the value of the variable) and a display value (the value shown in the drop down list in the TDOD UI). COLUMN1 defines the display value and COLUMN2 defines the key value. The list items are returned in the format:
`<COLUMN1> [ <COLUMN1> ]`
TDOD recognizes the item in square brackets as the display value.

Parameters:

- LIST — a list function such as @SEEDLIST or @SQLLIST that generates a list of items.
- COLUMN1 — The name or number of a column 1 in the list. If omitted, column 1 of the LIST is chosen. Used to define a drop-down variable for TDOD screens.
- COLUMN2 — The name of another column in the list. Used to define a drop-down variable for TDOD screens.
- QUOTED — Set this to sq to return the list items single quoted. Set this to dq to return the list items double quoted. This optional parameter is useful when generating lists for use in SQL expressions.

Example: @aslist(@seedlist(“DayOfWeek”)@,dq)@

Example result: “Monday”,”Tuesday”,”Wednesday”…

AND (BOOLEAN, BOOLEAN[,BOOLEAN…])

Performs a logical AND function on a variable list of arguments.

Parameters:

- a list of Boolean valued expressions. For more information, see the Boolean Expressions (see page 23) appendix.

Return value: a Boolean value, either true or false

Example: @AND (A, B)@ where A=true and B=false@

Example result: false

ATSIGN()

Returns the value '@'. This function is useful where you need a literal at-sign, but in a context where you want to avoid it being interpreted as a function invocation.

Example: A@atsign()@B

Example result: A@B

CARET()

Returns the value '^ '. This function is useful where you need a literal caret character, but in a context where you want to avoid it being interpreted as a column reference.

Example: A@caret()@B

Example result: A^B

CASE(TEST1,VALUE1,[TESTn,VALUEn...], ELSEVALUE)

Returns the value associated with the first test that is true, otherwise return the ELSEVALUE.

Parameters:

- TEST1 — a boolean expression. For more information, see the Boolean Expressions (see page 23) appendix.
- VALUE1 — the expression whose value you want returned if TEST1 is true
- TESTn, VALUEn — (Optional) Additional tests and values
- ELSEVALUE — an expression whose value is returned if all tests fail.

Return value: the value of the expression whose test was true.

Example: @case(1=2,A,2=2,B,C)@

Example result: B

CHAR(CODENUMBER)

Returns the character for a specified code number.

Parameters:

- CODENUMBER — A Unicode code point. For English, this is the same as ASCII.

Return value: a single character

Example: @char(65)@

Example result: A

COLLAPSE(STRING)

Removes all space, tab, carriage return, and new line characters from a string.

Parameters:

- STRING — a character string to be cleaned up

Return value: a new string

Example: @collapse(hello there)@

Example result: h (mailto:huwprice@abc.com)ellothere

CONVBASE(NUMBER, BASE)

Converts the number to a different base.

Parameters:

- NUMBER — a decimal number
- BASE — the new base to convert to.

Return value: a new number in the requested base.

Example: @convbase(999,16)@

Example result: 3E7

CONVBASE(NUMBER,BASE,DIGITS)

Converts the string to a different base, represented by specified digits.
The number of digits is equal to the base. For base 8, provide 8 digits. If the number of digits is greater than the base, then only the first digits are taken. If the number of digits is less than the base, then the digits are cycled until the required number is reached. For example, if you specify "AB" for base 5, then the digits are expanded to "ABABA".

Parameters:

- NUMBER — a decimal number to be converted
- BASE — the new base as a decimal number
- DIGITS — a list of digits to represent the converted number in. If you do not define any DIGITS, then the function attempts to guess the digits from the base specified. For example, if the base is 8, then it chooses 1,2,3,4,5,6,7,0. The least significant digit must be last.

Return value: the converted number as string

Example: @convbase(6,3,*&^)@

Example result: &*

COUNT(VALUE[,VALUE...])

Counts the number of items in the list or related rows in another table.

Parameters:

- VALUE — a list of items

Return value: **the** number of items in the list

Example: @count(1,A,Z,G,X)@ @

Example result: 5

Example: @count(^ITEM.ITEMTOT^)@

Example result: 4 (the number of rows in the column "ITEMTOT".

COUNTLIST(@SQLLIST(CONNECTION, SQL)@)

Returns the number of rows that are returned by specified SQL query.

Parameters: A SQL query of the following type @SQLLIST(CONNECTION, SQL)@

- CONNECTION — Defines the the dbms connection type. Choose one of the following:
  - R — Repository
  - S — Source
  - T — Target
  - P*profilename* — connection profile, for example `Pmyconprof`.
- SQL — A SQL query. Only select statements are supported.

Return value: number of rows returned by the query

Example: @COUNTLIST(@SQLLIST(Ptravel, SELECT * FROM TRAVEL.COUNTRIES)@)@

Example result: 293

COUNTLIST(@SEEDLIST(SEEDNAME)@)

Returns the number of rows in a seed list. See also Seed Lists (https://docops.ca.com/display/TDM41/Seed+Lists).

Parameters:

- SEEDNAME — name of a seed list. In Datamaker, do not enclose this string in quotes.

Return value: number of rows in the list

COUNTLIST(@PRIORPUBLISHKEYLIST(LD_ID, TABLENAME)@)

Returns the number of rows contained in the key list of a saved publish job. A publish job can use the saved columns feature to save the data in various columns for use in another publish job to be carried out at a later time. The priorpublishkeylist function retrieves that data. Use the countlist function to determine how many rows were saved.
The data is saved in the repository. If you run the job several times, the function looks for the latest data.

Parameters:

- LD_ID — The id of the generator that performed the publish. Datapainter inserts this id automatically when composing the expression containing countlist.
- TABLENAME — the name of the table whose columns were saved.

Return value: number of rows saved from the named table

Example: @COUNTLIST(@PRIORPUBLISHKEYLIST(1007, PEOPLE)@)@

COUNTLIST(@ALLPAIRS(LIST, LIST,[LIST], ALL_COMBINATIONS)@)

(Datamaker only) Returns the number of rows that are returned by all combinations of all pairs in specified LISTs. This is not supported in TDM Portal 4.0.

Parameters: an SQL query of the following type @ALLPAIRS(LIST, LIST[,LIST], ALL_COMBINATIONS)@) where

- LIST — list
- ALL_COMBINATIONS — constant

Return value: number of rows

COUNTWADL(URL, XPATH)

(Datamaker only) Returns the number of rows that are returned by the REST call that match an XPATH. This function is not supported in TDM Portal.

Parameters:

- URL — specifies the REST call to be made. Only GET is supported.
- XPATH — xpath of the XML element to find

Return value: number of rows

Example: @countwadl(http://server:5091/Service?LIST, Customer/CustomerId)@

CUSTOMLUHN(NUMBER,INDEX,LENGTH, [SEEDVALUE])

Returns a Luhn number based on a custom set of digits. For more information, see also the ADDLUHN function.

Parameters:

- NUMBER — A custom set of digits that stays the same in each generated number,
- INDEX — the position in the number where the additional digits are inserted to generate a Luhn number,
- LENGTH — the target length of the Luhn number,
- SEEDVALUE — an optional seed value to control uniqueness.

Return value: a LUHN number

Example 1: You want to create a Luhn number that starts with 12345 and ends with 56789. You also want the Luhn number to be 16 digits long. The length of prefix and postfix is 5+5=10 digits. You want to generate the remaining 6 digits. You use the function with the following parameters: `@customluhn(1234556789, 5, 6)@`. The function returns "`1234513626656789`".

Example 2: You want to generate a unique list of Luhn values. You define the seedvalue parameter to use the function nextval starting from 1000: `@CUSTOMLUHN(1234556789,5,6,@(nextval (listA, 1000))@)@`. Each time the customluhn function is called, the seedvalue is incremented by 1. The function returns the following:

```
1234501000156789 << seedvalue =1000
1234501001656789 << seedvalue =1001
1234501002056789 << seedvalue =1002
1234501003656789 << seedvalue =1003
1234501004456789 << seedvalue =1004
```

DATE(STRING)

Interprets the input string as a date and converts it to the format defined for the project.

Parameters:

- STRING — A date string. In Datamaker, do not enclose this string in quotes.

Return value: string

Example: @date(26/february/2018)@

Example result: 2018-02-26

DATE(DATESTRING, FORMAT)

Returns a string as a date using format specified.

Parameters:

- DATESTRING — A date string in any format,
- FORMAT — Your required date fromat, for example, dd/mm/yyyy, mm/dd/yyyy, DD-MM-YY. For more information see, date format (see page 88).

Return value: The input date in the specified format.

Example: @date(26/02/2018, DD/MM/YYYY)@

Example result: 2018-02-16 (because project format in this example is YYYY-MM-DD)

DATETIME(DATESTRING)

Converts a date to the datetime format defined for the project, eg yyyy-mm-dd HH:MM:SS.

Parameters:

- DATESTRING — A date string in any format. In Datamaker, do not enclose this string in quotes.

Return value: the input date reformatted as a datetime.

Example: @datetime(26/february/2008)@

Example result: 2008-02-26 00:00:00

DATETIME(STRING, FORMAT)

Returns a specific string as a date and a time, using the format that is specified.

Parameters:

- STRING — a date and time in any format,
- FORMAT — the new date time format.

Return value: date and time string with specified format.

Example: @datetime(12/07/1993, MMDDYYYY)@

Example result: 07121993

DAYSAFTER(STARTDATE, ENDDATE)

Returns the difference in days between the end date and the start date.

Parameters:

- STARTDATE — starting date,
- ENDDATE — ending date

Return value: The number of days between start and end.

Example: @daysafter(26/10/2008,29/10/2008)@

Example result: 3

DBLQUOTE()

Returns a double quote character. Use this function in expressions where a quote could be ambiguous.

Return value: a double quote character

Example: @dblquote()@

Example result: ''

DIVIDE(NUMBER1,NUMBER2)

Divides the first number by the second. The function expects integer or floating point numbers.

Parameters:

- NUMBER1 — dividend,
- NUMBER2 — divisor.

Return value: the quotient

Example: @divide(6,3)@

Example result: 2

DOB(MINAGE,MAXAGE,DATE)

Determines a date of birth, given a random age between a minimum and a maximum, on a specified date.

Parameters:

- MINAGE — minimum age,
- MAXAGE — maximum age,
- DATE — the date at which someone has a randomly selected age in the specified range. In Datamaker, do not enclose this string in quotes.

Return value: a date of birth in project date format

Example: What is your birthday, if you were a random age between 4 and 99 years old on the 8th of December 1901? @dob(4,99,1901-12-08)@

Example result: 1861-09-18

DOW(DATE)

Returns the day of the week for the date specified.

Parameters:

- DATE — a date

Return value: a number representing a day of the week. 1 for Sunday, 2 for Monday, 3 for Tuesday, 4 for Wednesday, 5 for Thursday, 6 for Friday, 7 for Saturday

Example: @dow(14/09/1972)@

Example result: 5

EBCDIC(STRING)

(Datamaker only) Returns the EBCDIC representation of the given string. Useful for generating flat file data for mainframe systems. Not supported on TDM Portal.

Parameters:

- STRING — a string to be converted to ebcdic

Return value: string representation

Example: @ebcdic(Datamaker)@

ELEMENT(STRING, DELIM, ELEMENTNO)

Returns a specified item in a list of items.

Parameters:

- STRING — a string containing a list of items separated by a delimiter character,
- DELIM — the character separating items in the list,
- ELEMENTNO — number of the item to return, starting at 1.

Return value: the requested list item. If ELEMENTNO is out of the range of the list, then the function returns "".

Example: @element("hello,there,folks", "," , 3)@

Example result: folks

ENVVAR(ENV)

(Datamaker only) Reads an environment variable. This function is not supported in TDM Portal 4.0.

Parameters:

- ENV — The name of an environment variable

Return value: The value of this environment variable

Example: @envvar(WINDIR)@

Example result: C:\WINDOWS

EXECSQL(CONNECTION, SQL)

Executes a SQL select query against the repository, a source, a target, or a profile. The SQL can be any SQL compatible with the connection and can include references to variables and functions. The query must return a single row. If more than one column is queried, then the values are returned as a comma separated list.

Parameters:

- CONNECTION — Defines the the dbms connection type. Choose one of the following:
    - R — Repository
    - S — Source
    - T — Target
    - P*profilename* — connection profile, for example `Pmyconprof`.
- SQL — A SQL select query that is compatible with the dbms underlying the specified connection.

Return value: the result of the query

Example: @execsql(PTravel_AW - Personal DB, select distinct expiration_date from credit_cards)@

Result: 1999-12-31

EXECSQLCOUNT(CONNECTION, SQL)

Returns the number of records returned by the specified SQL query, using the specified connection.

Parameters:

- CONNECTION — connection type - R(Repository), S(Source), or T(Target), or P*profilename* a connection profile.
- SQL — a SQL select query that is compatible with the dbms underlying the specified connection.

Return value: the number of rows returned by the query

Example: @execsqlcount(PTravel_AW - Personal DB, select * from countries)@

Example result: 293

EXECSQLPROC(CONNECTION,PROCNAME, PARAMNAME1,PARAMDIRECTION1, PARAMVALUE1, [PARAMNAME2, PARAMDIRECTION2,PARAMVALUE2,...], PARAMOUTNAME)

Execute a stored procedure against a specified connection with a list of parameters, and return the result.

Parameters:

- CONNECTION — connection type - R(Repository), S(Source), or T(Target), or Pprofilename a connection profile. Source and Target are not supported in CA TDM Portal 4.0
- PROCNAME — Name of the stored procedure to be executed.
- PARAMNAME — Name of the first parameter.
- PARAMDIRECTION — Each parameter can be either an input parameter, or an output parameter, or both.
  - (SQL Server) IN, OUT, OUTPUT
  - (Oracle) IN, IN OUT, OUT
  - (DB2 and Teradata) IN, OUT, INOUT
- PARAMVALUE — The value of the parameter named in PARAMNAME.
- (Optional) You can have any number of parameters by repeating PARAMNAME, PARAMDIRECTION and PARAMVALUE.
- PARAMOUTNAME — Name of the output parameter name to be returned as the value of the EXECSQLPROC function.

Return Value: Value that is returned by the stored procedure as defined by PARAMOUTNAME.

Example: @execsqlproc(PTravelX,dbo.extractCounter,paramin1,in,’Test’,paramin2,in,100,paramout1, output,0,paramout2,output,0,paramout2)@
This executes the stored procedure “dbo.extractCounter” against the connection “TravelX”. The procedure is supplied with the input parameters paramin1=’Test’, paramin2=100 and the output parameter paramout2. The value of this parameter is returned as the value of the execsqlproc function call.

Example result: 200

EXP(POWER)

Returns the natural exponent of a specified number.

Parameters:

- POWER — an integer or a floating point number

Return value: natural exponent of the input

Example: @exp(1)@

Example result: 2.718281828459045, because e to the power of 1 is e.

EXP(NUMBER, POWER)

Raises the specified number to a power.

Parameters:

- NUMBER — an integer or floating point number,
- POWER — an integer or floating point number.

Return value: the specified number raised to the power

Example: @exp(10,2)@

Example result: 100

FINNISHPERSONALID(GENDER)

Returns a random Finnish ID based on gender.

Parameters:

- GENDER — Either F for female, M for male, or U for Unisex.

Return value: A random Finnish ID of the format "DDMMYYCZZZQ"

FINNISHPERSONALID(DATE,NUMBER)

Returns a fixed Finnish ID based on DOB and ZZZ number.

Parameters:

- DATE — date of birth in yyyy-mm-dd format
- NUMBER — a personal identification number from 0 to 999.

Return value: A fixed Finnish ID of the format "DDMMYYCZZZQ"

FINNISHPERSONALID(GENDER,STARTDATE, ENDDATE)

Returns a sequential list of Finnish IDs, starting from the start date up to the end date. For any given date, 500 Finnish IDs are generated for either M or F gender, and a 1000 IDs in the case of U.

Parameters:

- GENDER — F female, M male, or U Unisex.
- START DATE — a start date of birth in yyyy-mm-dd format
- END DATE — an end date of birth in yyyy-mm-dd format. Must be equal to or greater than the start date.

Return value: A sequential list of Finnish IDs

GETSQL(PROGNAME)

Returns the specified stored SQL from the repository. This function is useful in conjunction with @sqllist()@, especially for very long queries.

Parameters:

- PROGNAME — name of the sql program to fetch.

Return value: the SQL statements

Example: Fetch the SQL statements stored under the name "getpets". @getsql(getpets)@

Example result: SELECT name, size, type, colour, tail_length FROM dbo.pets

GROUP(ITEMNO, GROUPSIZE)

If you have a number of items grouped into blocks of a certain size, then this function returns the number of the group that a particular item belongs to. This is computed by 1 + (itemno-1)/groupsize.

Parameters:

- ITEMNO — the number of the item
- GROUPSIZE — number of items in each group.

Return value: number of the group the items belongs to.

Example: @group(11,75)@

Example result: 1, because item number 11 belongs to the first group, because that contains the first 75 items.

GROUP(ITEMNO,FROMGROUPSIZE, TOGROUPSIZE,OFFSET)

Transforms an offset from one group to another.

Parameters:

- ITEMNO — the number of the item,
- FROMGROUPSIZE — the size of the group that the items belongs to,
- TOGROUPSIZE — size of the target group that the item is moving to,
- OFFSET — a number added to the new group number.

Return value: the number of the new group

Example: @group(3,5,10,2)@

Example result: 2

GUID(COLLAPSE)

Generates a globally unique identifier (GUID).

Parameters:

- COLLAPSE — optional keyword which, if specified, removes all dashes from the generated guid. If omitted, or if the argument is not the word “collapse”, then the guid is returned with dashes.

Return value: a new GUID.

Example: @guid(collapse)@

Example result: EF33B0BEA5BD44BBB2EDD676CFD64D46

Example: @guid()@

Example result: 3F2504E0-4F89-11D3-9A0C-0305E82C3301

HASH(NUMBER, MAXVAL)

Returns a hash value for a number.

Parameters:

- NUMBER — the number to be hashed,
- MAXVAL — maximum hash value allowed.

Return value: the hash value

Example: @hash(123456,345678)@

Example result: 86656

HASH(NUMBER,MAXVAL,SEED)

Creates a hash value for a number.

Parameters:

- NUMBER — the number to be hashed,
- MAXVAL — maximum allowed hash value,
- SEED — an integer value to be used for randomization of the hash.

Return value: hash value

Example: @hash(42,100)@

Example result: 13

HASHSIGN()

Returns the value '#'. This function is useful where you need a literal hash sign, but in a context where you want to avoid it being interpreted as a variable parameter invocation. For more information, see Create and Manage Variables (https://docops.ca.com/display/TDM41/Create+and+Manage+Variables#CreateandManageVariables-parameterized_variable).

Example: @pos(#string#, @hashsign()@)@

Example result: Finds a hash sign in a string, in a context when the pos() function is used as the value of a parameterized variable.

IBAN()

Creates a random IBAN for a random country.

Return value: a new IBAN

Example: @iban()@

Example result: GL9622756107472084

IBAN(ISOCOUNTRYCODE)

Creates a random IBAN with specified country code.

Parameters:

- ISOCOUNTRYCODE — For more information, see IBAN formats by country (Wikipedia) (http://en.wikipedia.org/wiki/International_Bank_Account_Number#IBAN_formats_by_country). The following country codes are not supported: AL, BE, BA, EE, FO, FI, IT, MK, NO, PT, RS, SI. In Datamaker, do not enclose this string in quotes.

Return value: a new random IBAN

Example: @iban(GB)@

Example result: GB94TOCG73393021580682

IBAN(ISOCOUNTRYCODE, BANKCODE)

Creates a random IBAN with specified country code and bank codes.

Parameters:

- ISOCOUNTRYCODE — A country code. For more information, see IBAN(ISOCOUNTRYCODE),
- BANKCODE — A bank code.

Return value: a new random IBAN

Example: @iban(GB, BARC)@

Example result: GB08BARC57552380947394

IBAN(ISOCOUNTRYCODE,BANKCODE, ACCOUNT)

Creates a random IBAN with a specified country code, bank code, and account number.

Parameters:

- ISOCOUNTRYCODE — A country code. For more information, see IBAN(ISOCOUNTRYCODE),
- BANKCODE — A bank code, ACCOUNT: An account number.

Return value: IBAN

Example: @iban(GB,BARC,41021810000222)@

Example result: GB26BARC41021810000222

IF(BOOLEANEXPR,TRUEEXPR,FALSEEXPR)

If the boolean expression is true, this function returns the value of one expression, otherwise it returns the other.

Parameters:

- BOOLEANEXPR — an expression returning a Boolean value. For more details, see the section on Boolean Expressions.
- TRUEEXPR — An expression whose value is returned if BOOLEANEXPR is true,
- FALSEEXPR — An expression whose value is returned if BOOLEANEXPR is false.

Return value: Either TRUEEXPR or FALSEEXPR.

Example: @if(1<0,Smaller,Bigger)@

Example result: Bigger

IFNULL(EXPR, NULLEXPR)

Evaluatse an expression and if it is NULL or empty, return the value of another expression. If not, return the value of the original expression.

Parameters:

- EXPR — An expression,

- NULLEXPR — An expression.

Return value: An expression

Example: @ifnull(@randlov(0, H)@, hello)@

Example result: hello - because "@randlov(0, H)@" returned an empty string, so the function returns the value of NULLEXPR.

Example: @ifnull(@execsql(Pxyz,select name from pets where id=123)@, hello)@

Example result: tiggy – because the select statement returned a result which is not NULL, so this result is returned.

JDATE(DATE[, PRECISION])

Converts a date to a Julian date with a certain precision.

Parameters:

- DATE — a date in any of the usual formats. In Datamaker, do not enclose this string in quotes.
- PRECISION — (optional argument) number of decimal places in the result.

Return value: the corresponding Julian date rounded to the defined precision.

Example: @jdate(25-02-2016,1)@

Example result: 2457443.5

Example: @jdate(25-02-2016)@

Example result: 2457443.500000

LASTDAY(DATE)

Returns the last day of the month that the specified date is in.

Parameters:

- DATE — A date in any of the usual formats. In Datamaker, do not enclose this string in quotes.

Return value: the date of the last day of the month in which the date resides.

Example: @lastday(12/04/2054)@

Example result: 2054-04-30, because the last day of April is the 30th.

LEFT(STRING, LENGTH)

Returns the leftmost characters from the string.

Parameters:

- STRING — a string to be split,
- LENGTH — number of characters to return.

Return value: a substring starting at position 1 in the string.

Example: @left(ashok, 2)@

Result: as

LEFTPAD(STRING, CHARTOPAD[, LENGTH])

Pads a string up to a specified length by adding characters to the start. If the string is already the length or longer then no characters are added.

Parameters:

- STRING — a string to be padded
- CHARTOPAD — a string to add to the left of the string. Usually a single character is used but any length string can be used.
- LENGTH — (optional argument) the required length of the string. If omitted, the column width is used.

Return value: a string padded with characters up to the specified length

Example: @leftpad(ABCDE, 1, 10)@

Example result: 11111ABCDE

Example: @leftpad(ABCDE, +)@ used where the column width is 10 characters

Result: +++++ABCDE

LEFTTRIM(STRING, CHARTOTRIM)

Remove all occurences of a characters from the left side of a string.

Parameters:

- STRING — a string expression whose value is to be trimmed,
- CHARTOTRIM — the character to remove from left of string. Usually a single character is used, but any length string can be used and that string will be trimmed.

Return value: a possibly shorter string

Example: @lefttrim(++++++hello,+)@

Example result: hello

Example: @lefttrim(ABCDEFGHIJKLMNOPQRSTUVWXYZ, ABC)@

Example result: DEFGHIJKLMNOPQRSTUVWXYZ

LENGTH(STRING)

Returns the number of characters in a string.

Parameters:

- STRING — a string expression. In Datamaker, do not enclose this string in quotes, otherwise they count towards the length.

Return value: number of characters in that string

Example: @length(Contrafibularity)@

Result: 16

LOG(NUMBER)

Returns the natural log of a number.

Parameters:

- NUMBER — a numeric expression

Return value: the natural log of the value of the expression

Example: @log(10)@

Example result: 2.302585092994046

LOGTEN(NUMBER)

Returns the log to the base 10 of a number.

Parameters:

- NUMBER — a numeric expression

Return value: base 10 log of the expression

Example: @logten(1000)@

Example result: 3

LOV(@SQLLIST(CONNECTION, SQL)@, ROW)

Return a specific row from an SQL query.

Parameters:

- SQLLIST — a call to the sqllist function which executes a select statement, returning one or more rows,
- ROW — the number of the row required, starting at 1.

Return value: the required row. if more than one colum is selected, then the results are returned separated by commas.

Example: @lov(@sqllist(Ptravel, select name from cities)@,4)@

Example result: Canterbury

LOWER(STRING)

Return a string in lower case. If the string is double quoted, CA Datamaker returns the quotes as well, but CA TDM Portal does not.

Parameters:

- STRING — a string expression

Return value: a lower case string

Example: @lower(HELLO)@

Example result: hello

LUHN(LENGTH)

Returns a random number of the given length including a check digit using the Luhn algorithm.

Parameters:

- LENGTH — length of the number to be returned

Return value: a number

Example: @luhn(6)@

Example result: 705357

MAX(VALUE1,VALUE2[...,VALUEn])

Returns the maximum value in a list of numbers or strings. If the list contains strings that cannot be converted to numbers, the function makes a string comparison on all items. The maximum value is the one that is lexograpically greater than the others. For example, Z is greater than A. If all list items can be converted to numbers, the function makes a numeric comparison.

Parameters:

- VALUE1...VALUEn — a list of numeric or string expressions. If empty, the function fails.

Return value: The largest item in the list.

Example: @max(^PASSENGER1_MONEY^)@

Example result: the largest value in the money column.

Example: @max(1,2,3,4,5)@

Example result: 5

MEAN(VALUE1,VALUE2, ...VALUEn)

Returns the mean of a list of numeric values.

Parameters:

- VALUE1... VALUEn — a list of numeric expressions

Return value: the mean of the list

Example: @mean(1,2,3)@

Example result: 2

MID(STRING, STARTPOS)

Returns a substring starting from the given position, up to the end of the string.

Parameters:

- STRING — a string expression.
- STARTPOS — starting position (first character is position 1).

Return value: a substring

Example: @mid(ABCDEFGHIJK, 2)@

Example result: BCDEFGHIJK

MID(STRING,STARTPOS,LENGTH)

Returns a substring starting from the given position and of the specified length.

Parameters:

- STRING — a string expression. In Datamaker, do not enclose this string in quotes, otherwise the quotes are included in the start position.
- STARTPOS — starting position (first character is position 1). If the position is past the end of the string, an empty string is returned.
- LENGTH — the required length of the string. If zero, then an empty string is returned.

Return value: a substring

Example: @mid(ABCDEFGHIJKL, 3, 4)@

Example result: CDEF

MIN(VALUE1,VALUE2,...VALUEn)

Return the minimum value in a list of numbers or strings. If the list contains strings that cannot be converted to numbers, then a string comparison is made on all items. The minimum value is the one that is lexograpically less than the others. For example, A is less than Z.
If all list items can be converted to numbers then a numeric comparison is made.

Parameters:

- VALUE1...N — a list of numeric or string expressions. If empty, the function fails.

Return value: the smallest item in the list.

Example: @min(^PASSENGER1_MONEY^)@

Example result: 100

Example: @min(1,2,3,4,5)@

Example result: 1

MOD(NUMBER1, NUMBER2)

Returns the modulo of two numbers. This is the remainder of the first number divided by the second number.

Parameters:

- NUMBER1 and NUMBER2 — numeric expressions

Return value: remainder of NUMBER1 divided by NUMBER2

Example: @mod(5,2)@

Example result: 1

MULTIPLY(NUMBER1, NUMBER2)

Multiplies the two numbers.

Parameters:

- NUMBER1 and NUMBER2 — numeric expressions

Return value: value of NUMBER1 multiplied by NUMBER2

Example: @multiply(2,3)@

Example result: 6

NEXTSTRINGVAL(SEQUENCE, STARTVAL, DIGITS)

Returns the next value in the specified sequence as a string using a list of digits. The sequence is created if it does not exist and assumes a starting value of startval. The list of digits is used to represent a number in a base equal to the length of the string. For example, abcd represents base 4 using those digits, [0-9] would represent decimal and [0-9][A-F] would represent hexadecimal. The sequence is used to remember the previous value as a decimal number. The function increments the value of the sequence and converts the decimal value to a number in the requested base using the digits.

Parameters:

- SEQUENCE — the name of a sequence,
- STARTVAL — starting value of the sequence if it does not already exist,
- DIGITS — a list of characters to be used to represent the next value. You can specify letters, numbers and ranges in the form [a-z], for example abc[0-9].

Return value: a string representing the next value in the SEQUENCE using the supplied digits.

Example: @nextstringval(myseq,7,12345670)@

Example result: 10 (This example returned the next value in myseq, assuming base 8 (octal), and starting the sequence at 7.)

NEXTVAL(SEQUENCE)

Return the next value in the specified sequence. The sequence is created if it does not exist. The sequence starts at 1.

Parameters:

- SEQUENCE — name of a sequence

Return value: the next value in the sequence

Example: @nextval(mysequence)@

Example result: 1

NEXTVAL(SEQUENCE, STARTVAL)

Return the next value in the specified sequence. The sequence is created if it does not exist and assumes a starting value of startval. If no startval is given, sequence always starts from 1.

Parameters:

- SEQUENCE — name of a sequence,
- STARTVAL — a numeric expression giving the starting value

Return value: the next value in the sequence

Example: @nextval(EXAMPLE, 10)@

Example result: 10

NOT(BOOLEAN)

Applies a logical NOT operation to a Boolean expression.

Parameters:

- BOOLEAN — a boolean expression

Return value: true when BOOLEAN=false and false when BOOLEAN=true

Example: @not(1=1)@

Example result: false

OCCURS(STRING, STRINGTOFIND)

Counts the number of occurrences of a string in another string.

Parameters:

- STRING — a string expression,
- STRINGTOFIND — a string expression to search for

Return value: number of occurrences of STRINGTOFIND

Example: @occurs(hello hello there folks,hello)@

Example result: 2

OCCVAL(NUMBER %STRING,NUMBER%STRING [,NUMBER%STRING...])

Return a random value from a list of strings. The list is composed using arguments of the form: number%string. In Datamaker, do not enclose these arguments in quotes.

Parameters:

- NUMBER — the number of times to repeat the following string
- STRING — a string expression to be repeated

Return value: a random value from the composed list.

Example: @occval(100%A,300%B)@

Example result: B, because the function assumes a list composed of 100 A’s and 300 Bs and then selects a random item from that list.

OR(BOOLEAN1, BOOLEAN2[,BOOLEAn...])

Perform a logical OR operation on a list of Boolean expressions.

Parameters:

- BOOLEAN1...BOOLEANn — a Boolean expression

Return value: true if any expression is true, and false if they are all false.

Example: @or(~var1~,1=1,1=2,1=3)@

Example result: true

PERCVAL(N%STRING, N%STRING[,N% STRING...])

Returns a random value from a list of values generated using a percentage. Each argument has the form: <percentage of the total list>%<string expression>. All the percentages must add up to 100 otherwise the function fails. For example, an argument list of “30%fred,70%jim” would assume a list that contains 30% freds and 70% jims.
The difference between @randlov(0, @perclist(50%DEBIT,50%CREDIT)@)@ and @percval(50%DEBIT, 50%CREDIT)@ is stability. Multiple calls to the former, within a row, always returns the same value, whereas multiple calls to the latter, within a row, do not. This is true for all @randlov() functions - they all return the same value in the same row.
In Datamaker, do not enclose these arguments in quotes.

Parameters:

- N — the percentage of the associated STRING present in the list,
- STRING — a string expression.

Return value: A random value from a list

Example: @percval(10%A,90%B)@

Example result: B, because the function assumes a list composed of 10 A’s and 90 Bs and then selects a random item from that list.

POS(STRING, STRINGTOFIND)

Returns the first position of stringtofind in a string.

Parameters:

- STRING — a string expression to be searched,
- STRINGTOFIND — a string expression to find

Return value: First position of STRINGTOFIND

Example: @pos(^LAST_NAME^,a)@

Example result: 3 (where LAST_NAME=Stacey)

POS(STRING, STRINGTOFIND, STARTPOS)

Return the first position of stringtofind in the string starting at startpos.

Parameters:

- STRING — a string expression to be searched,
- STRINGTOFIND — a string expression to find

Return value: First position of STRINGTOFIND after STARTPOS

Example: @pos(banana,a,3)@

Example result: 4

RANDCHARS(MINLEN,MAXLEN,CHARLIST)

Returns a random string of a length between minlen and maxlen, consisting only of the specified characters.

Parameters:

- MINLEN — minimum length of the resulting string,
- MAXLEN — maximum length,
- CHARLIST — a string

Return value: a string of characters to be used in the random string

Example: @randchars(3,10, AEIOURSTLNE)@

Example result: EEOLES

RANDDATE(MIN, MAX)

Returns a random date between the minimum and maximum values.

Parameters:

- MIN — minimum date,
- MAX — maximum date

Return value: A random date in project format

Example: @randdate(2009/01/01,2010/01/01)@

Example result: 2009/12/07

RANDDIGITS(MINLEN, MAXLEN)

Returns a random string of digits of a length between minlen and maxlen.

Parameters:

- MINLEN — minimum length of the resulting string,
- MAXLEN — maximum length

Return value: String of digits

Example: @randdigits(3,10)@

Example result: 2103

RANDEXP(MIN, MAX, MEAN)

Returns a positive number on an exponential distribution with a mean as specified. Only values between min and max are returned.

Parameters:

- MIN — minimum value,
- MAX — maximum value,

- MEAN — mean value

Return value: A positive number

Example: @randexp(1,10,4)@

Example result: 5

RANDHEX(MINBYTES, MAXBYTES)

Returns a random hex string of a random length between the minimum and maximum length.

Parameters:

- MINBYTES — minimum length of resulting string,
- MAXBYTES — maximum length

Return value: A random hexadecimal string

Example: @randhex(1,4)@

Example result: 8A8172

RANDLOV(PERCNULL,@DIRLIST(DIRECTORY)@)

(Datamaker only) Returns a random value (with percentage of nulls that are specified by percnull) from the list of files that are contained in the specified directory. All @randlov() functions return the same value in the same row.

Parameters:

- PERCNULL — percentage of nulls to be expected over a large number of calls. For example, a value of 50% would mean that there is a 50-50 chance of the value being a null.,
- DIRECTORY — the directory to search for files

Return value: A random value from the list of files in the named directory

Example: @randlov(0, @dirlist(C:\TEMP)@)@

Example result: a random filename from the c:\temp directory.

RANDLOV(PERCNULL, @LIST(STRING, STRING[, STRING])@)

Returns a random selection from a list of values. All @randlov() functions return the same value in the same row.

Parameters:

- PERCNULL — percentage of nulls to be expected over a large number of calls. For example, a value of 50% would mean that there is a 50-50 chance of the value being a null.
- LIST — a list of string expressions and values that are separated by commas. In Datamaker, quoted strings are returned with their quotes; CA TDM Portal removes them.

Return value: A randomly chosen value from the list or maybe a null.

Example: @randlov(0, @list(Devon,Cornwall,Surrey)@)@

Example result: Cornwall

RANDLOV(PERCNULL, @OCCLIST(N%STRING, N%STRING[,N%STRING...])@)

Returns a random value (with percentage of nulls that are specified by percnull) from the specified occurrence list. All @randlov() functions return the same value in the same row.

Parameters:

- PERCNULL — percentage of nulls,
- OCCLIST — occurrence list,
- N — a percent value

Return value: A random value from a list

Example: @randlov(0, @occlist(50%VI,50%MC)@)@

Example result: MC

RANDLOV(PERCNULL, @OCCLIST(N%STRING, N%STRING[,N%STRING…])@)

Returns a random value from the specified occurrence list. All @randlov() functions return the same value in the same row.

Parameters:

- PERCNULL — percentage of nulls to be expected over a large number of calls. For example, a value of 50% would mean that there is a 50-50 chance of the value being a null.
- OCCLIST — an occurrence list. This is a list created from a set of items in the form: <number of items>%<a string expression>. For example, 10%A,20%B would create a list containing 10 A's and 20 B's.

Return value: A random value from the list or NULL if percnull > 0.

Example: @randlov(0, @occlist(50%VI,50%MC)@)@

Example result: MC

RANDLOV(PERCNULL, @PERCLIST(N%STRING, N%STRING[,N%STRING])@)

Return a random selection from a list of values generated by a @PERCLIST. The difference between @randlov(0, @perclist(50%DEBIT,50%CREDIT)@)@ and @percval(50%DEBIT,50%CREDIT)@ is stability. Multiple calls to the former, within a row, always returns the same value, whereas multiple calls to the latter, within a row, do not. This is true for all @randlov() functions - they all return the same value in the same row.

Parameters:

- PERCNULL — percentage of nulls to be expected over a large number of calls. For example, a value of 50% would mean that there is a 50-50 chance of the value being a null.
- PERCLIST — a percentage occurrence list. This is a list created from a set of items in the form: <% of items>%<a string expression>. For example, 10%A,90%B would create a list containing 10% A's and 90% B's.

Return value: A random value from the list or NULL if percnull > 0.

Example: @randlov(0, @perclist(10%Devon,20%Cornwall,70%Surrey)@)@

Example result: Cornwall

RANDLOV(PERCNULL,@SEEDLIST(SEEDNAME)@[,SEEDCOLUMN,][INVALIDVAL])

Return a random value from a column in a seed list. All @randlov() functions return the same value in the same row. Some seed lists have more than one column, for example, DayOfWeek contains two columns, one for the day number and the other for the day name. See also Seed Lists (https://docops.ca.com/display/TDM41/Seed+Lists).

Parameters:

- PERCNULL — percentage of nulls to be expected over a large number of calls. For example, a value of 50% would mean that there is a 50-50 chance of the value being a null.
- SEEDNAME — name of a seed list, eg DayOfWeek. If the seed list has more than one column then the first column is chosen. In Datamaker, do not enclose this string in quotes.
- INVALIDVAL — (optional) a value from the seed list that will not be returned. If a randomly chosen item in the seed list has this value then it is skipped and another value is chosen.
- SEEDCOLUMN — (optional) the column in the SEEDLIST to be fetched. This can be the column number or name.

Return value: A random value from the specified column in the seed list, or NULL if percnull > 0.

Example: @randlov(0,@seedlist(streetname)@,1)@

Example result: 9 St Thomas Street

Example: @randlov(0,@seedlist(DayOfWeek)@,Numeric Day)@

Example result: 2

Example: @randlov(0,@seedlist(month)@)@

Example result: April

RANDLOV(PERCNULL, @SQLLIST(CONNECTION, SQL)@[,COLUMN][, INVALIDVAL])

Returns a random value from a sql query. All @randlov() functions return the same value in the same row.

Parameters:

- PERCNULL — percentage of nulls to be expected over a large number of calls. For example, a value of 50% would mean that there is a 50-50 chance of the value being a null.

- CONNECTION — Defines the the dbms connection type. Choose one of the following:
    - R — Repository
    - S — Source
    - T — Target
    - P*profilename* — connection profile, for example `Pmyconprof`.
- SQL — A SQL query. Only select statements are supported.
- COLUMN — (optional argument). The number or name of the column to be returned. If this is omitted and multiple columns are present in the query then those columns are returned separated by commas.
- INVALIDVAL — (optional argument). A invalid value that is never returned. If a randomly chosen row has this value then it is skipped and another row is chosen.

Return value: a random item from the query or maybe a null if percnull > 0

Example: @randlov(0, sqllist(Ptravel, select names from cities))@

Example result: Albury

Example: @randlov(0, sqllist(Ptravel, select distinct expiration_date from credit_cards),1,2000/01/01)@

Example result: 2008/05/01

RANDLOV(PERCNULL, @PRIORPUBLISHKEYLIST (LD_ID, TABLENAME)@)

Returns a random row from the key list of a prior publish (for level ID (LD_ID) and table (TABLENAME)). All @randlov() functions return the same value in the same row.

Parameters:

- PERCNULL — percentage of nulls to be expected over a large number of calls. For example, a value of 50% would mean that there is a 50-50 chance of the value being a null.
- LD_ID — the id of the generator that performed the publish. This is automatically inserted by datapainter when composing the expression containing countlist.
- TABLENAME — the name of the table whose columns were saved.
- COLUMN — (optional argument). The number or name of the column to be returned. If this is omitted and multiple columns are present in the query then those columns are returned separated by commas.

- INVALIDVAL — (optional argument). A invalid value that is never returned. If a randomly chosen row has this value then it is skipped and another row is chosen.

Return value: A random row from a previously published table

Example: @randlov(0, @priorpublishkeylist(1007, PEOPLE)@)@

RANDLOV(PERCNULL, @PRIORTESTMATCHLIST (LD_ID, TESTNAME)@)

(Datamaker only) Returns a random row from the key list of a prior test match for level ID (LD_ID) and test name (TESTNAME). All @randlov() functions return the same value in the same row. This function is only supported in Datamaker and not in TDM Portal 4.0.

Parameters:

- PERCNULL — percentage of nulls to be expected over a large number of calls. For example, a value of 50% would mean that there is a 50-50 chance of the value being a null.
- LD_ID — the id of the generator that performed the publish. This is automatically inserted by datapainter when composing the expression containing countlist.
- TESTNAME — test name

Return value: A random row from the test match results

Example: @randlov(0, @priortestmatchlist(4000,TEST1)@)@

RANDLOV(PERCNULL, @WADLLIST(URL, COLUMNNAME)@)

(Datamaker only) Returns a random row from a REST call. This function is only supported in Datamaker and not in TDM Portal 4.0.

Parameters:

- PERCNULL — percentage of nulls to be expected over a large number of calls. For example, a value of 50% would mean that there is a 50-50 chance of the value being a null.
- URL — a REST UIRL to be called. Only GET is supported.
- COLUMNNAME — name of the element or an XPATH of the element to be returned

Return value: A random row

Example: @randlov(0, wadllist(http://server:5091/Service?LIST, CustomerName)@)@

RANDLOV(PERCNULL, SOURCES[,INVALIDVAL])

(Datamaker only) This function allows you to specify a list of sources (A to J) from which the list is drawn. This function is only supported in Datamaker and not in TDM Portal 4.0.

Parameters:

- PERCNULL — percentage of nulls to be expected over a large number of calls. For example, a value of 50% would mean that there is a 50-50 chance of the value being a null.
- SOURCES — sources(A to J)
- INVALIDVAL — (optional argument) An invalid value that is never returned. If one is chosen, then it is skipped and another is chosen.

Return value: A random item from the list

Example: @randlov(0, G)@,

Example result: Base US Visa

Example: @randlov(0,G,UK-US Visa)@

Example result: Standard US Visa

RANDNORM(MIN,MAX,MEAN,STDDEV)

Returns a normally distributed number between the min and max values with the specified mean and standard deviation.

Parameters:

- MIN — minimum value
- MAX — maximum value
- MEAN — mean value
- STDDEV — standard deviation

Return value: a floating point value

Example: @randnorm(1,100,50,12)@

Example result: 43.1

RANDNULL(PERCNULL, EXPR)

Returns the value of the expression or randomly a null.

Parameters:

- PERCNULL — percentage of nulls to be expected over a large number of calls. For example, a value of 50% would mean that there is a 50-50 chance of the value being a null.
- EXPR — a numeric or string expression

Return value: the value of the expression or maybe a null if percnull > 0

Example: @randnull(50, @randtext(1,20)@)@

Exam result: yuniuidfniub

RANDRANGE(MIN, MAX,[WIDTH])

Returns a random integer between min and max.

Parameters:

- MIN — minimum allowed
- MAX — maximum allowed
- WIDTH — (optional argument) if specified then the result is zero padded up to this width

Return value: A random integer

Example: @randrange(1,2)@,

Example result: 1,2

Example: @randrange(100,1000,8)@,

Example result: 00000234

RANDTEXT(MINLEN,MAXLEN[,CASE])

Create random text of a length between the min and max.

Parameters:

- MIN — minimum length of result

- MAX — maximum length
- CASE — (optional argument) If omitted, capitalized text is assumed.
  - U for upper case text,
  - M for capitalized text (Default)
  - L for lower case text.

Return value: a random string

Example: @randtext(3,50, M)@,

Example result: Ghjhj GGY hhjh

RANDTIME(MIN, MAX)

Returns a random time between a min and max.

Parameters:

- MIN — minimum time allowed in the format hh:mm
- MAX — maximum time allowed

Return value: A random time

Example: @randtime(00:00:00,23:59:59)@

Example result: 12:34:01

RANDVAL(SEQUENCE, MAXVAL)

Generates a more or less unique but not sequential number. The function can take any sequence name. The sequence is created if it does not exist. Remember that the sequence is in the repository, not the target.

Parameters:

- SEQUENCE — name of a sequence. If the sequence does not exist, it is created.
- MAXVAL — maximum allowed value. The parameter defines the size of the range of values the function returns. The smaller maxval, the greater the rate at which duplicates are generated.

Return value: A unique number

Example: @randval(myseq, 999999)@

Example Result: 1245 depending on how many times it is called.

REPLACE(STRING,STRINGTOREPLACE, REPLACEMENTSTRING)

Returns the string with substrings replaced as specified.

Parameters:

- STRING — a string expression to be edited
- STRINGTOREPLACE — the substring in the string to be replaced. If blank then the original string is returned.
- REPLACEMENTSTRING — the replacement string

Return value: a string

Example: @replace(King Kong,ng,la)@

Example result: Kila Kola

REPEAT(EXPR,OCCURS,SEPARATOR)

Generates a string by evaluating an expression a number of times and concatenating the values separated by a character. The expression is evaluated multiple times as it may return a different result each time (for example, a random value from a seed list). It is not assumed that it has a constant value.

Parameters:

- EXPR — a numeric or string expression
- OCCURS — the number of times to repeat
- SEPARATOR — the separator character or string

Return value: A string containing a set of values of the expression.

Example: @repeat(hello,5,”,”)@

Example result: hello,hello,hello,hello,hello

REVERSE(STRING)

Returns the reverse of a string.

Parameters:

- STRING — a string expression. In Datamaker, quoted strings are returned with their quotes. Portal removes them.

Return value: A string

Example: @reverse(ABC)@

Example result: CBA

RIGHT(STRING, LENGTH)

Return the specified number of characters from the right of a string.

Parameters:

- STRING — a string expression
- LENGTH — number of characters to return.

Return value: A substring ending at the right of the source string

Example: @right(hello, 2)@

Example result: lo

RIGHTPAD(STRING,CHARTOPAD[,LENGTH])

Pads the string to the required length by adding chartopad to the right.

Parameters:

- STRING — a string expression
- CHARTOPAD — padding character or string
- LENGTH — (optional argument) the required string length. If omitted, the column width is used.

Return value: A string padded to the required length

Example: @rightpad(ABCDE,#,10)@

Result: ABCDE#####

RIGHTTRIM(STRING, CHARTOTRIM)

Trims a substring from the right-hand end of a string.

Parameters:

- STRING — a string expression.
- CHARTOTRIM — character to trim

Return value: A trimmed string

Example: @righttrim(hello*****,*)@

Example result: hello

ROUND(NUMBER, DECPLACES)

Round a number up to a number of decimal places.

Parameters:

- NUMBER — a numeric expression
- DECPLACES — the number of decimal places

Return value: a number

Example: @round(1.262776,2)@

Example result: 1.27

SECONDSAFTER(STARTDATETIME, ENDDATETIME)

Return the number of seconds between a start and end datetime or time.

Parameters:

- STARTDATETIME — start time or datetime in project or other format, eg 20-01-2017 14:42
- ENDDATETIME — end time datetime

Return value: The number of seconds between those datetimes.

Example: @secondsafter(26/10/2008:01:30:56,26/11/2008:01:30:59)@

Example result: 2678403

Example: @secondsafter(01:30:56,01:30:59)@

Example result: 3

SEQLOV(PERCNULL, @ALLPAIRS(LIST1,LIST2, [LIST], ALL_COMBINATIONS)@, COLUMN)

(Datamaker only) Returns the next item from a list of all combinations of pairs for LIST1, LIST2... for a particular column COLUMN. Each time the function is called, the next item is returned. When the end of the list is reached, the list wraps back to the beginning. This is only supported in Datamaker and not TDM Portal 4.0.

Parameters:

- PERCNULL — percentage of nulls to be expected over a large number of calls. For example, a value of 50% would mean that there is a 50-50 chance of the value being a null.
- @ALLPAIRS — a list generated from all combinations of two lists
- COLUMN — name or number of the column to be returned if the lists have multiple columns.

Return value: the next item from the all pairs list.

Example: @seqlov(0, @allpairs(@list(abc,bcd,def)@, @list(kkk,llll,mmm)@, N)@,1)@,

Example Result: [(abc, kkk), (abc,llll), (abc,mmm), (bcd,kkk), (bcd, llll), (bcd,mmm), (def, kkk), (def, llll), (def, mmm)]

SEQLOV(PERCNULL,@DIRLIST(DIRECTORY)@)

Returns a sequential value (with percentage of nulls that are specified by percnull) from the list of files that are contained in the specified directory.

Parameters:

- PERCNULL — percentage of nulls to be expected over a large number of calls. For example, a value of 50% would mean that there is a 50-50 chance of the value being a null.
- DIRECTORY — the directory to search for files

Return value: the next value from the list of files in the named directory.

Example: @seqlov(0, @dirlist(C:\TEMP)@)@

Example result: c:\temp\fred.txt

SEQLOV(PERCNULL, @LIST(STRING, STRING[, STRING])@)

This function allows for sequential selection from a list of values. It includes any values of interest inside the @LIST(...)@ construct, separated by commas.

Parameters:

- PERCNULL — percentage of nulls to be expected over a large number of calls. For example, a value of 50% would mean that there is a 50-50 chance of the value being a null.
- @LIST: a list of string expressions and values separated by commas

Return value: The next value from the list or maybe a null.

Example: @seqlov(0, @list(Devon,Cornwall,Surrey)@)@

Example result: Devon – the next call would return Cornwall.

SEQLOV(PERCNULL, @OCCLIST(N%STRING, N%STRING[N%STRING...])@)

Returns a the next sequential value from an occurrence list.

Parameters:

- PERCNULL — percentage of nulls to be expected over a large number of calls. For example, a value of 50% would mean that there is a 50-50 chance of the value being a null.
- OCCLIST — an occurrence list. This is a list created from a set of items in the form: <number of items>%<a string expression>. For example, 10%A,20%B would create a list containing 10 A's and 20 B's.

Return value: The next value from the list or maybe a null if percnull > 0.

Example: @seqlov(0, @occlist(50%VI,50%MC)@)@

Example result: VI

SEQLOV(PERCNULL, @PERCLIST(N%STRING, N%STRING[,N%STRING])@)

Return the next sequential item from a list of values generated by a @PERCLIST.

The difference between @randlov(0, @perclist(50%DEBIT,50%CREDIT)@)@ and @percval(50%DEBIT, 50%CREDIT)@ is stability. Multiple calls to the former, within a row, always returns the same value, whereas multiple calls to the latter, within a row, do not. This is true for all @randlov() functions - they all return the same value in the same row.

Parameters:

- PERCNULL — percentage of nulls to be expected over a large number of calls. For example, a value of 50% would mean that there is a 50-50 chance of the value being a null.
- PERCLIST — a percentage occurrence list. This is a list created from a set of items in the form: <% of items>%<a string expression>. For example, 10%A,90%B would create a list containing 10% A's and 90% B's.

Return value: The next value from the list or NULL if percnull > 0.

Example: @seqlov(0, @perclist(10%Devon,20%Cornwall,70%Surrey)@)@

Example result: Devon

SEQLOV(PERCNULL, @PRIORPUBLISHKEYLIST (LD_ID, TABLENAME)@[, COLUMN][, INVALIDVAL])

(Datamaker only) Returns the next sequential item from the key list of a prior publish (for level ID (LD_ID) and table (TABLENAME)). All @seqlov() functions return the same value in the same row. This function is only supported in Datamaker and not in TDM Portal 4.0.

Parameters:

- PERCNULL — percentage of nulls to be expected over a large number of calls. For example, a value of 50% would mean that there is a 50-50 chance of the value being a null.
- LD_ID — the id of the generator that performed the publish. This is automatically inserted by datapainter when composing the expression containing countlist.
- TABLENAME — the name of the table whose columns were saved.
- COLUMN — (optional argument). The number or name of the column to be returned. If omitted, the first column is assumed.

- INVALIDVAL — (optional argument). A invalid value that is never returned. If a randomly chosen row has this value then it is skipped and another row is chosen.

Return value: A random row from a previously published table

Example: @seqlov(0, @priorpublishkeylist(1007, PEOPLE)@,1)@

SEQLOV(PERCNULL, @PRIORTESTMATCHLIST (LD_ID, TESTNAME)@[, COLUMN])

(Datamaker only) Returns a sequential row from the key list of a prior test match for level ID (LD_ID) and test name (TESTNAME). (COLUMN) is the column number from the list to extract values from. This function is only supported in Datamaker and not in TDM Portal 4.0.

Parameters:

- PERCNULL — percentage of nulls to be expected over a large number of calls. For example, a value of 50% would mean that there is a 50-50 chance of the value being a null.
- LD_ID — the id of the generator that performed the publish. This is automatically inserted by datapainter when composing the expression containing countlist.
- TESTNAME — the name of the testmatch previously run.
- COLUMN — (optional argument). The number or name of the column to be returned. If omitted, the first column is assumed.
- INVALIDVAL — (optional argument). A invalid value that is never returned. If a randomly chosen row has this value then it is skipped and another row is chosen

Return value: The next sequential value

Example: @seqlov(0, @priortestmatchlist(4000,TEST1)@,1)@

SEQLOV(PERCNULL,@SEEDLIST(SEEDNAME[,S])@,SEEDCOLUMN, INVALIDVAL)

Returns the next sequential value from a column in a seed list. All @seqlov() functions return the same value in the same row. Some seed lists have more than one column, for example, DayOfWeek contains two columns, one for the day number and the other for the day name. See also Seed Lists (https://docops.ca.com/display/TDM41/Seed+Lists).

Parameters:

- PERCNULL — percentage of nulls to be expected over a large number of calls. For example, a value of 50% would mean that there is a 50-50 chance of the value being a null.

- SEEDNAME — name of a seed list, eg DayOfWeek. If the seed list has more than one column then the first column is chosen. In Datamaker, do not enclose this string in quotes.
- S — (optional) If the literal argument S is specified then the seed list is randomly shuffled before returning the first item. Subsequent calls do not re-shuffle.
- INVALIDVAL — (optional) a value from the seed list that will not be returned. If a randomly chosen item in the seed list has this value then it is skipped and another value is chosen.
- Return value — A random value from the seed list or NULL if percnull > 0.
- SEEDCOLUMN — the column in the SEEDLIST to be fetched. This can be the column number or name.

Return value: The next sequential value from the specified column in the SEEDLIST, or possibly a null value if percnull > 0.

Example: @seqlov(0,seedlist(streetname),1,Oxford Circus)@

Example result: 10 Wall Street

SEQLOV(PERCNULL, @SQLLIST(CONNECTION, SQL[,S])@,COLUMN, INVALIDVAL)

Return the next sequential value from a sql query. All @seqdlov() functions return the same value in the same row.

Parameters:

- PERCNULL — percentage of nulls to be expected over a large number of calls. For example, a value of 50% would mean that there is a 50-50 chance of the value being a null.
- CONNECTION — Defines the the dbms connection type. Choose one of the following:
  - R — Repository
  - S — Source
  - T — Target
  - P*profilename* — connection profile, for example `Pmyconprof`.
- SQL — A SQL query. Only select statements are supported.
- S — (optional) If the literal argument S is specified then the rows returned by the query are randomly shuffled before returning the first item. Subsequent calls do not re-shuffle.

- COLUMN — (optional) The number or name of the column to be returned. If this is omitted and multiple columns are present in the query then those columns are returned separated by commas.

- INVALIDVAL — (optional) An invalid value that is never returned. If an item has this value then it is skipped and another item is chosen.

Return value: The next sequential item from the query or maybe a null if percnull > 0

Example: @randlov(0, sqllist(Ptravel, select names from cities))@

Example: @seqlov(0, sqllist(Ptravel, select distinct expiration_date from credit_cards),1, 2000/01/01)@

Example result: 2008/05/01 (2000/01/01 will never be returned)

Example: @seqlov(0, sqllist(Ptravel, select distinct expiration_date from credit_cards,S),1)@

Example result: 2010/11/01 (the list is shuffled first)

Example: @seqlov(0, sqllist(Ptravel, select names from cities, S))@

Example result: Aalborg

SEQLOV(PERCNULL,@UITESTLIST(LENGTH)@, REPEATTYPE)

Returns a sequential value from the UI Test List specified by UITESTLIST. Values have the specified maximum length (exception is hex strings, which represent bytes). Find the list of Valid Values for RepeatType .

Parameters:

- PERCNULL — percentage of nulls

- UITESTLIST — UI Test List

- LENGTH — length

- REPEATTYPE — RepeatType Values

Return value: The next sequential value rom the uitest list

Example: @seqlov(0, @uitestlist(5)@, NUMERIC)@

SEQLOV(PERCNULL, @WADLLIST(URL, COLUMNNAME)@)

(Datamaker only) Returns the next sequential row from a REST call. This function is only supported in Datamaker and not in TDM Portal 4.0.

Parameters:

- PERCNULL — percentage of nulls to be expected over a large number of calls. For example, a value of 50% would mean that there is a 50-50 chance of the value being a null.
- URL — a REST URL to be called. Only GET is supported.
- COLUMNNAME — name of the element or an XPATH of the element to be returned

Return value: The next sequential row from the results of the REST call

Example: @seqlov(0, wadllist(http://server:5091/Service?LIST, CustomerName)@)@

SEQLOV(PERCNULL, SOURCES[,INVALIDVAL])

Selects a sequential list of values from the source, a percentage of the values are designated as null. Source can be A to J.

Parameters:

- PERCNULL — percentage of nulls
- SOURCES — source (A to J)
- INVALIDVAL — (optional) An invalid value that is never returned. If an item has this value then it is skipped and another item is chosen.

Return value: The next sequential list value from the list

Example: @seqlov(0, G)@, m

Example result: Base US Visa

SIGN(NUMBER)

Returns the sign of a number.

Example: @sign(-110)@

Example result: - (that is, the character 'minus')

SINE(COUNTER, PERIOD)

Return the sine of the counter with respect to the period, that is, it returns sin((2+counter+π)/period)

Parameters:

- COUNTER — a numeric expression
- PERIOD — a numeric expression

Return value: the value

Example: @sine(4,32)@

Example result: 0.7071067811865475

SINE(MAGNITUDE,COUNTER,PERIOD)

Returns the value of magnitude*sin(2+counter + π)/period).

Parameters:

- MAGNITUDE — a numeric expression
- COUNTER — a numeric expression
- PERIOD — a numeric expression

Return value: the sine value

Example: @sine(4,23,12)@

Result: -2.000000000000006

SQLLIST(CONNECTION, SQL[,S])

Execute a SQL query on a specified connection and return a list. This function can only be used inside functions such as SEQLOV or RANDLOV which pick an item from the list. If you try to use it outside, it will only return a list identification string (for example Q1) which only the LOV functions understand.

Parameters:

- CONNECTION — Defines the the dbms connection type. Choose one of the following:
  - R — Repository

    - S — Source
    - T — Target
    - P*profilename* — connection profile, for example `Pmyconprof`.
- SQL — A SQL query. Only select statements are supported.
- S — (optional) If the literal argument S is specified then the rows returned by the query are randomly shuffled.

STDDEV(NUMBER[,NUMBER...])

Returns the standard deviation for the specified list of numbers.

Parameters:

- NUMBER... — a list of numeric expressions

Return value: the standard deviation of the list

Example: @stddev(23,42)@

Example result: 9.5

STRING(DATE, DATEFORMAT)

Returns a date in the given date format. Parts of the date can be returned, not just whole dates.

Parameters:

- DATE — a date - usually a system or user variable or date valued function or a column reference.
- DATEFORMAT — the required date format, e.g. MM for just a month number, DD-MM-YYYY for a date

Return value: The date in the new format

Example: @string(~CDATE~, MM)@

Example result: 07

STRING(NUMBER, NUMBERFORMAT)

Convert the number into string that is based on the number format. This is not supported in TDM Portal 4.0: it simply returns the number unchanged.

Parameters:

- NUMBER — A numeric expression
- NUMBERFORMAT — format of a number. This is a string of digits chosen from this list:
  - 9 — a corresponding digit from the number
  - . — a decimal point
  - 0 — a leading zero or a digit

Return value: A formatted number

Example: @string(45,0000)@

Example result: 0045

STRING(TIME, TIMEFORMAT)

Returns the specified time as a string in the specified format.

Parameters:

- TIME — a time, usually a system or user variable or time valued function or a column reference.
- TIMEFORMAT — the required time format.

Return value: Time as a string

Example: @string(~STIME~, HH:MM:SS)@

Example result: 16:54:23

SUBTRACT(NUMBER1,NUMBER2)

Subtracts one number from another.

Parameters:

- NUMBER1,2 — numeric expressions

Return value: NUMBER1-NUMBER2

Example: @subtract(3,1)@

Example result: 2

SUM(NUMBER[,NUMBER...])

Returns the sum of a list of numbers.

Parameters:

- NUMBER... — a list of numeric expressions

Return value: The sum of the list of number values.

Example: @sum(1,2,3)@

Example result: 6

SWEDEN_SSN(SEPARATOR, GENDER)

Returns a Swedish Social Security Number with a specified separator and gender.

Parameters:

- SEPARATOR — a separator character; must be either + or – (minus).
- GENDER — the string Male or Female or M or F

Return value: A Swedish Social Security Number

Example: @sweden_ssn(|, Female)@

Example result: 281002|403

SWEDEN_SSN(DOB, GENDER)

Returns a Swedish Social Security Number with the specified date of birth and gender.

Parameters:

- DOB — date of birth
- GENDER — the string Male or Female or M or F

Return value: A string representing the Swedish Social Security Number as a string

Example: @sweden_ssn(1901-03-06, Male)@

Example result: 010306+191

TCKID([NUMBER[, WIDTH]])

(Datamaker only) Returns an 11-digit Turkish National ID, according to the Turkish MERNIS project. If you provide the optional NUMBER parameter, the first 4 digits are randomly generated, and the next 5 digits are fixed values. The optional WIDTH parameter defines the number of characters that will be fixed values. This function is supported in Datamaker and CA TDM Portal 4.1.

Parameters:

- NUMBER — a number
- WIDTH — a number.

Default: If you provide no parameters, all first 9 digits are randomly generated.

Example: @tckid(@nextval(abc)@)@ generates 99,999 unique numbers.

Example: @tckid(@nextval(abc)@, 8)@ generates 99,999,999 values.

Return value: a Turkish ID

TILDE()

Returns the tilde character ~. This function is useful in expressions where the tilde might be confused with a variable reference.

TIME(DATETIME)

Returns the time portion of a datetime in the project format.

Parameters:

- DATETIME — a datetime in one of a number of formats. In Datamaker, quoted strings will fail to be interpreted as a date time.

Return value: the time part of the DATETIME

Example: @time(26/02/2008:12:23:22)@

Example result: 12:23:22

TRIM(STRING, CHARTOTRIM)

Trims a substring from both ends of a string.

Parameters:

- STRING — a string expression
- CHARTOTRIM — characters to trim from each end

Return value: a possibly shorter string

Example: @trim(***hello*****,*)@

Example result: hello

UK_NINO()

Returns a random UK National Insurance number.

Example: @uk_nino()@

Result: WW212145A

UPPER(STRING)

Upper-cases a string. If the string is double quoted, Datamaker returns the quotes as well, but TDM Portal does not.

Parameters:

- STRING — a string expression

Return value: the upper cased STRING

Example: @upper(hello)@

Example result: HELLO

US_SSN(SEPARATOR)

Return a random US social security number with the specified separator.

Parameters:

- SEPARATOR — a character

Return value: US social security number with specified SEPARATOR

Example: @us_ssn(-)@

Example result: 729-60-7933

VERHOEFF(LENGTH)

Return a random number of a specified length with a Verhoeff checksum.

Parameters:

- LENGTH — number of digits required

Return value: a number with Verhoeff checksum test

Example: @verhoeff(123)@

Example result:
4689006505332434039238153758272904570067046028547392630365646452819191890746820629

WORDCAP(STRING)

Capitalizes a string. If the string is double quoted, Datamaker returns the quotes as well, but TDM Portal does not.

Parameters:

- STRING — a string to capitalize

Return value: an upper case STRING

Example: @wordcap(hello folks)@

Example result: Hello Folks

XLOOKUP(LIST, OLDVAL)

Retrieves the new value that is associated with the given old value from the given cross reference list. Use xref() to set up a cross reference list. Useful for maintaining referential integrity across multiple databases when performing updates.

Parameters:

- LIST — a cross reference list,
- OLDVAL — value

Return value: new value

Example: @xlookup(PEOPLE, Joe Bloggs)@

XREF(LIST,OLDVAL,NEWVAL)

Adds the pair (old value, new value) to the given cross reference list. Returns the new value. Use xlookup() to retrieve these values later.
You typically use @xref and @lookup in a datapool that you have constructed to clone data between a source and target. Typically an identifying ID on a column is read from the source, and a new ID is generated to be used on the target. The xref function maps the old (source value) and new ID (generated value) to one another. In a later expression, you use xlookup to retrieve the mapping of the two values.

Parameters:

- LIST — a cross reference list
- OLDVAL — value
- NEWVAL — value

Return value: new value

Example: @xref(PEOPLE, Joe Bloggs, Vincent Van Gogh)@

XREFPERSIST(TABLENAME, COLUMNNAME , OLDVAL, NEWVAL)

Performs the same basic function as xref, but additionally stores the specified old and new values in the repository. The mappings of OLDVAL to NEWVAL are stored in a list with the name TABLENAME. COLUMNNAME. Xrefpersist stores the old and new values in the gtrep_pj_key_values table when the publish completes. CA TDM portal stores only the table-column combinations that use xrefpersist in the repository. In CA Datamaker, using @xrefpersist once causes all values used in both @xref and @xrefpersist to be stored to the repository. Use the @xlookup method to query the mapping of values later.

Parameters:

- TABLENAME — table name where this value has been generated,
- COLUMNNAME — column name where this value has been generated,
- OLDVAL — value,
- NEWVAL — value.

Return value: new value

Example: @xrefpersist(PEOPLE, NAME, Joe Bloggs, Vincent Van Gogh)@

Function Date Formats

These are the valid date formats to be used as function parameters in Datamaker:

```
DD-MMM-YYYY, DD:MMM:YYYY, DD MMM YYYY, DD/MMM/YYYY, DD.MMM.YYYY, DDMMMYYYY,
DD-MM-YYYY,  DD:MM:YYYY,  DD MM YYYY,  DD/MM/YYYY,  DD.MM.YYYY,  DDMMYYYY,
DD-MMM-YY,   DD:MMM:YY,   DD MMM YY,   DD/MMM/YY,   DD.MMM.YY,   DDMMMYY,
DD-MM-YY,    DD:MM:YY,    DD MM YY,    DD/MM/YY,    DD.MM.YY,    DDMMYY,
MMM-DD-YYYY, MMM:DD:YYYY, MMM DD YYYY, MMM/DD/YYYY, MMM.DD.YYYY, MMMDDYYYY,
MM-DD-YYYY,  MM:DD:YYYY,  MM DD YYYY,  MM/DD/YYYY,  MM.DD.YYYY,  MMDDYYYY,
MMM-DD-YY,   MMM:DD:YY,   MMM DD YY,   MMM/DD/YY,   MMM.DD.YY,   MMMDDYY,
MM-DD-YY,    MM:DD:YY,    MM DD YY,    MM/DD/YY,    MM.DD.YY,    MMDDYY,
YYYY-DD-MMM, YYYY:DD:MMM, YYYY DD MMM, YYYY/DD/MMM, YYYY.DD.MMM, YYYYDDMMM,
YYYY-DD-MM,  YYYY:DD:MM,  YYYY DD MM,  YYYY/DD/MM,  YYYY.DD.MM,  YYYYDDMM,
YY-DD-MMM,   YY:DD:MMM,   YY DD MMM,   YY/DD/MMM,   YY.DD.MMM,   YYDDMMM,
YY-DD-MM,    YY:DD:MM,    YY DD MM,    YY/DD/MM,    YY.DD.MM,    YYDDMM,
YYYY-MMM-DD, YYYY:MMM:DD, YYYY MMM DD, YYYY/MMM/DD, YYYY.MMM.DD, YYYYMMMDD,
YYYY-MM-DD,  YYYY:MM:DD,  YYYY MM DD,  YYYY/MM/DD,  YYYY.MM.DD,  YYYYMMDD,
YY-MMM-DD,   YY:MMM:DD,   YY MMM DD,   YY/MMM/DD,   YY.MMM.DD,   YYMMMDD,
YY-MM-DD,    YY:MM:DD,    YY MM DD,    YY/MM/DD,    YY.MM.DD,    YYMMDD
```

Function Sources

This is the list of values of Sources used in Datamaker Functions:

A Values from DDL

B Values from Production

C Values from Development

D Invalid Values

E Used Values from Test Data Repository

F Used Values from Data Target

G Used Values from Data Source

H Related Keys from Test Data Repository

I Related Keys from Data Target

J Related Keys from Data Source

Function Time Formats

These are the time formats accepted as function parameters in Datamaker:

```
HH-MM, THH:MMZ, HH:MM, HH MM, HH.MM, HHMM
HH-MM-SS, THH:MM:SSZ, HH:MM:SS, HH MM SS, HH.MM.SS, HHMMSS
HH-MM-SS.FFF, THH:MM:SS.FFFZ, HH:MM:SS.FFF, HH:MM:SS:FFF, HH MM SS.FFF, HH.MM.SS.FFF,
HHMMSSFFF, HHMMSS.FFF,
HH-MM-SS.FFFFFF, THH:MM:SS.FFFFFFZ, HH:MM:SS.FFFFFF, HH:MM:SS:FFFFFF, HH MM SS.
FFFFFF, HH.MM.SS.FFFFFF, HHMMSSFFFFFF, HHMMSS.FFFFFF,
HH-MM-SS.FFFFFFFFF, HH:MM:SS.FFFFFFFFF, HH:MM:SS:FFFFFFFFF, HH MM SS.FFFFFFFFF, HH.MM.
SS.FFFFFFFFF, HHMMSSFFFFFFFFF, HHMMSS.FFFFFFFFF
```

Values for REPEATYPE Functions

Below is a list of Valid Values for REPEATTYPE functions and their meanings:

TEXT: Valid Text

TEXT-SPACE: Text and Spaces

TEXT-SPECIALCHAR: Special Characters and Text

SPECIALCHAR: Special Characters

SINGLESPACE (sic): Single Space

NUMERIC: Numbers Stored As Text

EMPTY: Empty

ALPHA-NUMERIC: Alphanumeric Values

ALLSPACE: All spaces

STD-ASC: Standard ASCII Values (between 32 and 127 inclusive)

NONSTD-ASC: Non-Standard ASCII Values (ASCII < 32 or > 127)

HEXCHAR: Hex Characters