Pattern Definition

This document describes data generation engine compatible patterns.
The engine uses one pattern to generate set (or family) of values with same properties.

Any pattern is a string, i.e. a sequence of items and constant elements (letters, digits, signs, etc) without delimiters. The data generation engine copies constant elements to output without processing.

Formal Definition (BNF)

```
<pattern> ::= [<items>]
<items> ::= <item>[<items>]
<item> ::= <constant>|<symbol>|<block>|<block-reference>|<optional>|<column-reference>|<list>|<function-call>|<expression>
<constant> ::= <immediate-constant>|<escaped-constant>
<escaped-constant> ::= \<immediate-constant>
<immediate-constant> ::= <any-printable-character>
<symbol> ::= <single-symbol>[<repeater>]
<single-symbol> ::= a|A|N|X
<repeater> ::= {<positive-number>}|{=<positive-number>}|{<number>:<positive-number>}
<block> ::= (<pattern>)
<block-reference> ::= #
<optional> ::= '['<pattern>']'
<column-reference> ::= @<number>|@@<number>|@'<name>'|@@'<name>'
<list> ::= '<'<list-items>'>'
<list-items> = <pattern>[|<list-items>]
<function-call> ::= $<name>(<parameters>)
<parameters> ::= <empty-string>|<pattern>[,<parameters>]
<expression> ::= $$(<term>)
<term> ::= (<term>)|-<term>|<number>|<date>|<time>|<string>|<function-call>|<term><add-op><term>|<term><mul-op><term>|<term><cmp-op><term>|<term><log-op><term>
<add-op> ::= +|-
<mul-op> ::= *|/
<cmp-op> ::= '>'|'<'|=|!=|>=|<=
<log-op> ::= &|AND|OR|'|'
```

'\c' means the engine should not use 'c' as pattern item, i.e. pass it to the output stream without processing.
Sample symbols those should be escaped: A, a, N, X, @, $, <, >, (,), #, [,].

Examples

'\Number' pattern produces "Number" string when 'Number' generates "1umber" or something like.

See also: $Const function.

Pattern Items

There are item types:

- Symbol produces one or more random symbols.
- Block, reference to some pattern's part.
- Optional part, optional part of the pattern.
- Column reference refers to already generated value for another column.
- List, predefined value list.
- Functions call, built-in functions.
- Expression, numeric or date/time expressions.

Symbols

Symbol produces single digit or letter. The pattern can contain following symbols:

- A - produces random upper letter between 'A' and 'Z'
- a - produces random lower letter between 'a' and 'z'
- N - produces random number between '0' and '9'
- X - produces random hexadecimal digit: 0..9,A..F

See also

- Symbol escaping
- The repeater can be applied to any symbol.

Repeater

The **repeater** is a one of the following sequences: {n}, {=n}, {n:m}

{n} means to repeat a symbol from 1 to n times. {} and {0} means do not repeat, i.e. {1}
Example: A{3}

{=n} means to repeat a symbol exact n times.
Example: N{=6}

{n:m} means to repeat a symbol from n to m times. 'n' can be '0' in this case. 'm' must be greater than 'n'.
Example X{2:5}

Important: the repeater can be applied to mentioned symbols only. Please consider block with the optional feature for functions or expressions.

Examples

($Lib(Cities))[#][#][#] repeats the same city between 1 and 4 times when ($Lib(Cities)[$Lib(Cities)][$Lib(Cities)][$Lib(Cities)] generates different cities between 1 and 4 times.

(pattern) is pattern's part value that can be referred by # item.

Examples

- (NNN)-# produces strings like 345-345, 043-043, etc.
- (NNN)-#-(A)-# produces strings like 385-385-B-B, 943-943-S-S, etc.
- (NN(N))-# generates 385-5, 943-3, 801-1, etc. See limitation #2 for explanation.

Limitations

- '#' can't be used in the expressions.
- New block definition overrides the previous block. # can be used between '()' and next '()' including nested or end of the pattern only.

Optional Part

[pattern] means optional part of the pattern with 50/50 probability.

Examples

[-]NNN will generate -NNN or NNN.

Hint: you should use $IfR function to specify probability.
Example for 10% negative values: $IfR(10,-,)NNN

List of items

<subpattern1|subpattern2|...|subpatternN> - the engine will use one of the list item randomly.

Examples

<+|->NNN

Limitations

- The nested list is not supported
- The list format does not allow the user to specify value probability. Please use $List function call for this purpose.

Expressions

$$(expression) specifies numeric, logical (boolean) or date/time expression.

An expression can contain:

- Numeric constants and quoted strings (example: 'null')
- Function calls
- Date or time constants created by $Date or $Time function
- Operations ('+',' -','*','/') and comparison operators ('>','>=','<','<=','=', '!=')
- '&' and '|' for AND and OR logicals operators

Operations

Operations '*','/' and unary ' -' are acceptable for numeric operands only.

Only comparison operations can be applied to strings.

Logical operations '>','>=','<','<=','=','!=' are acceptable for numeric, date and time operands.

'&' and '|' are acceptable for logical (boolean) operands only.

The '+' operation

	Right operand			
Left operand	**Integer**	**Float**	**Date**	**Time**
Integer	+	+ (float)	Add N days	Add N seconds
Float	+ (float)	+	N/A	N/A
Date	add N days	N/A	N/A	N/A
Time	add N seconds	N/A	N/A	N/A

The '-' operation

	Right operand			
Left operand	**Integer**	**Float**	**Date**	**Time**
Integer	-	- (float)	N/A	N/A
Float	- (float)	+	N/A	N/A
Date	subtract N days	N/A	N/A	N/A
Time	subtract N seconds	N/A	- (int)*	N/A

Notes and limitations

- '(' and ')' can be used with any acceptable data type.
- Logical values have only integer presentation. 1 for 'true' and '0' for 'false'.
- Complex nested expressions like $$($$(8-2)-2) are not acceptable, use '()' instead of: $$((8-2)-2)
- You should use @'name' column reference format in expressions. @number format is not supported for some kind of expressions.
- * - approximate number of days between two dates.

@n - from current row by column position. The first column is 1
@'name' - from current row by name
@@n - from the previous row by column position. The first column is 1
@@'name' - from the previous row by name

Limitation: forward references are not allowed for '@' operation.

Groups

The group is a set of columns. The set consists of the main column and dependent column or columns.

There are functions that can be used to create a group:

- File Group
- Library Group
- Table Group
- Microsoft Excel Group
- Database Query based Group
- User defined script (Perl, Python, etc) based Group

The Group function should be used to access depended group item.

Date and Time Format Strings

Date Format

The format string for date can contain following items:
DD - day, integer between 01 and 31
MM - month, integer between 01 and 12
MON - upper case month name, the string between 'JAN' and 'DEC'
mon - lower case month name, the string between 'jan' and 'dec'
YY - two-digits year, integer between 00 and 99.
YYYY - four-digits year, integer.
C - century, integer 0 to 8 value: 0 means 19xx, 1 means 20xx, etc. It is not compatible with YYYY.

Default date format depends on local user's settings.

Time Format

The format string for time can contain following items:
HH - hour, an integer between 00 and 23 (or 01 to 12).
mm - minute, an integer between 00 and 59. **MM** is also acceptable but not recommended.
SS - second, an integer between 00 and 59.
sss - milliseconds, integer between 000 and 999.
AM - AM or PM sign.
zz - time zone between -11:00 and +11:00.

Default time format depends on local user's settings.

Limitation: comma can't be used as a separator.

Notes

- All mentioned format items are case sensitive. For example, you can't use 'yyyy' instead of 'YYYY'.
- All digits mentioned in the format are mandatory. I.e. '1' can't be used for 'DD' format instead of '01'.

Format Strings

A format string consists of its format specification and the rest of characters that are put into the result unchanged.

There are two format string styles: C/C++/Java and C#. The acceptable style depends on the software product you have.

A C++ format specification always starts from the % sign. If it is necessary to use this character as a character instead of a formatting element, it should be doubled.

C# format string has following format: {n} or {n:format} where 'n' is position, 0 for most cases.

The most useful format types are:

C++/Java format	.net/C# format	Description
%d	D	Signed decimal integer. You can use width for this field, for example, %5d. If you specify the width as %05d, the engine will add non-significant zeros to reach the specified width.
%I64d		Huge integer number. Currently, it can be used with $Inc and $RInt functions only.
%x	X	hexadecimal integer. Uses lowercase letters for hexadecimal integers. %X uses uppercase letters.
%f	F	value in the form dddd.dddd, where dddd is one or more decimal digits. It is possible to specify sizes in the form %5.2f (2 corresponds to the number of decimal digits here, while 5 is the total).
%s		String. '-' (minus) means alignment to the right side if it is specified together with the width.

Source	Code	Text	Note
Expression	80	Factor error: expect ')'	
Expression	81	Factor error: expect value instead of '...'	
Expression	82	Factor error: Unexpected empty lexema	
Expression	83	Factor error: Unexpected empty factor	
Expression	84	Expression: unexpected operation ot token '...'s	
Function	90	Parameter(s) missed.	There is no parameter list, even empty.
Function	91	Brackets coordination error.	
Function	92	Could not resolve nested call or reference.	
Function	100	Could not recognize '...' as correct function name.	The specified function does not exist.
Group	101	Specified group N not found.	
Group	102	Parameter list must contain 2 items.	
Repeater	103	Incomplete repeater value.	Value or range expression is not completed well.
List	104	<...> list is empty.	
Repeater	105	Incorrect repeater value.	Incorrect range separator or another problem.
Truncate	106	Parameter list must contain 2 items.	
Truncate	107	Truncate: string size must be positive integer value.	
Variables	108	Parameter list must contain 1 item	
Variable access	109	Incorrect modifier for '...'	
Variable access	110	'...' variable not found	
Repeater	111	Incorrect high border of the repeater.	Hight border is empty or incorrect
List	112	<...> list closed incorrectly.	
Optional	113	Optional block closed incorrectly.	
RDate	200	'From' date '...' does not correspond to format '...'	
RDate	201	'To' date '...' does not correspond to format '...'	
RDate	204	Could not recognize date format '...' for 'from' date [...]	
RDate	205	Could not recognize date format '...' for 'to' date [...]	

RDate	206	Invalid long year in the date format '...'	For example, two digits year for YYYY format.
RDate	207	Invalid short year in the date format '...'	For example, four digits year for YY format.
RDate	208	Invalid date format '...'	No format items in the format string found.
RTime	300	'From' time '...' does not correspond to format '...'	Constant and format specification are different
RTime	301	'To' time '...' does not correspond to format '...'	Constant and format specification are different
RTime	302	Incorrect or incomplete time format '...'	
Time	303	Time constant '...' does not correspond to format '...'	Wrong time constant or format
Date	304	Date constant '...' does not correspond to format '...'	Wrong date constant or format
Geometry	400	Geometry expects not more than one parameter.	0 or 1 parameter is required.
Geometry	401	Parameter '...' is incorrect, expects positive integer.	
Geography	500	Geography has no parameters.	Unexpected parameters were passed.
Call	600	Function $Call expects one parameter.	
Call	601	Named generators do not passed to pattern engine.	No named generators defined.
Call	602	Named generator '...' not found.	
Column reference	700	Definitions are not present.	Columns are not defined.
Column reference	701	Column reference without column name	@ sign without number or name
Column reference	702	Invalid column index or forward reference	@n with out of range or incorrect 'n' value
Column reference	703	Column name without quotation	@name instead of @'name'
Column reference	704	Column name quotation error.	
Column reference	705	Specified column '...' not found or forward reference.	
MSExcel	1000	Function expects at least 3 parameters.	
MSExcel	1001	Could not open '...' file	
ExcelGroup	1050	Function expects at least 5 parameters.	
ExcelGroup	1051	Could not open '...' file	
ExcelGroup	1052	<Excel driver-specific error message>	
Query	1100	No query present	
Query	1101	Could not connect to custom data source.	
Query	1102	No default or custom database connection specified.	

Query	1103	<Database-specific error message>	Could not execute database query
Query	1104	Invalid query or expression	Could not resolve or prepare query text
Query	1105	Empty query provided	
Query	1106	The Query execution error	
File	1200	Function expects at least one parameter.	
File	1201	Could not access or open source file '...'.	Check path and access rights for the file
File	1202	Could not calculate item '...' of the list	File contains incorrect engine call
FileGroup	1300	Function expects at least 3 parameters.	
FileGroup	1301	Could not open source file '...'.	Check path and access rights for the file
FileGroup	1302	Could not calculate item.	The file contains incorrect engine call
MSAccess	1400	Function expects at least 3 parameters.	
MSAccess	1401	Could not open '...' file.	Check path, format and access rights for the file
Table	1500	Function expects at least 2 parameters.	
Table	1501	No default or custom database connection specified.	
Table	1502	Could not connect to custom data source.	
TableGroup	1600	Function expects at least 3 parameters	
TableGroup	1601	<database-specific error message>	
TableGroup	1602	This call version expects at least 6 parameters	
TableGroup	1604	Could not connect to custom data source.	
TableGroup	1608	No default or custom database connection specified.	
TableGroup	1610	Group item is out of range	Incorrect item of the group number
GroupByQuery	1700	Function expects at least 2 parameters: group and query.	
GroupByQuery	1701	<database-specific error message>	Could not execute the query or retrieve requested rows.
GroupByQuery	1702	No default or custom database connection specified.	
GroupByQuery	1704	Could not connect to custom data source.	
GroupByQuery	1705	Incorrect engine call as query text	
			Wrong number of the group

GroupByQuery	1706	Group item is out of range	item
Library	1800	Function expects at least one parameter.	
Library	1801	Could not open library at '...'.	Check for library location and accessibility
Library engine	1850	<database-specific error message>	GetColumn returns nothing
Library engine	1860	ListValue could not find value with enough length.	All values are too long
LibGroup	1900	Function expects at least 3 parameters.	
LibGroup	1901	Could not open library at '...'.	Check for library location and accessibility
LibGroup	1902	Group item is out of range	Incorrect reference to the item of the group
If	2000	Function expects 3 parameters	
If	2001	The first parameter must be between 0 and 100.	
If	2002	Could not compile the first pattern '...'	Check the first pattern
If	2003	Could not compile the second pattern '...'	Check the second pattern
If	2004	Could not calculate the item.	
Unique	2100	Function expects single parameter.	
Unique	2101	Could not compile nested pattern.	Check the nested pattern
Unique	2102	Could not find unique value provided by nested pattern.	Not enough unique data items provided
Unique	2103	Could not calculate nested pattern.	Check the nested pattern
List	2200	Function expects at least 2 parameters	
List	2201	Incomplete pair of parameters	Check number of parameters
List	2202	Wrong probability value	Expects positive integer between 1 and 100.
List	2204	Could not calculate item '...' of the list	Incorrect engine call for list item.
Sequence	2300	Function expects exact 2 parameters.	
Sequence	2301	Incorrect pattern provided	Please check nested call or expression
Sequence	2304	Could not calculate counter value	Please check call or expression for "counter" parameter
XML	2500	The function has two parameters.	
XML	2501	'...' file inaccessible or does not exist.	
XML	2502	Could not load XML file.	Check that file has correct

____			XML format
XML	2503	Could not load XML from web resource.	
DLL	2600	Expects at least one parameter	
DLL	2601	DLL not found or inaccessible	
DLL	2602	'Install' entry point not found.	The DLL does not export 'Install' entry point.
DLL	2603	'GetValue' entry point not found.	The DLL does not export 'GetValue' entry point.
BLOB loader	2700	Expects one parameter	
BLOB loader	2701	No source directory found or inaccessible	
BLOB loader	2702	No files found in the directory	
ListPattern	2800	Function expects 2 parameters	
ListPattern	2801	Function expects positive integer as 1st parameter	
ListPattern	2802	Function could not generate unique value	List is too small to find another unique value
ListPattern	2803	Incorrect pattern or expression	
ByExample	2900	Function expects at least one parameter	
ByExample	2901	Function could not recognize any pattern	Try to provide more examples or another example list
ByExample	2902	Could not calculate the item	
RString	3000	Function expects at least 2 parameters	
IncDate	3100	Wrong step size: 'D','M','Y','H','m' or 'S' expected	
IncDate	3101	Incorrect call for Step (...)	Check built-in function call for 'Step' parameter
IncDate	3102	Incorrect call for From (...)	Check built-in function call for 'From' parameter
IncDate	3103	Could not calculate last value	Check table name and column reference for 'From'
Inc	3150	Incorrect call for Step (...)	Check built-in function call for 'Step' parameter
Inc	3151	Incorrect call for First (...)	Check built-in function call for 'First' parameter
Inc	3152	Could not calculate last value	
Inc	3153	Incorrect call for sequence border (...)	Check built-in function call for 'sequence border' parameter
Inc	3154	Incorrect call for Reuse counter (...)	Check built-in function call for 'Reuse counter' parameter
IncTime	3200	Wrong step size'H','M','m' or 'S' expected	
IncTime	3201	Incorrect call for From (...)	Check built-in function call

			for 'From' parameter
IncTime	3202	Incorrect call for Step (...)	Check built-in function call for 'Step' parameter
IncTime	3203	Could not calculate last value	Check table and column name for 'From'
IncFloat	3250	Incorrect call for Step (...)	Check built-in function call for 'Step' parameter
IncFloat	3251	Incorrect call for sequence border (...)	Check built-in function call for 'sequence border' parameter
IncFloat	3252	Incorrect call for Reuse (...)	Check built-in function call for 'Reuse' parameter
IncFloat	3253	Could not calculate last value	
IncFloat	3254	Incorrect call for First (...)	Check built-in function call for 'First' parameter
Format	3300	The function expects exact 2 parameters	
Lower	3400	The function expects single mandatory parameter	
Upper	3500	The function expects single mandatory parameter	
Quote	3600	The function expects at least one parameter	
Pattern	3700	The function expects single mandatory parameter	
WebFile	3800	The function expects at least one parameter	
WebFile	3801	Could not load the resource	
WebFile	3802	Could not calculate item	Value is incorrect engine call
WebFileGroup	3900	The function expects at least 3 parameters	
WebFileGroup	3901	Could not load the resource	
WebFileGroup	3902	Could not calculate value	File contains incorrect engine call
WebFileGroup	3904	Empty value	The group was not defined properly
WebFileGroup	3903	Group item is out of range	Wrong item of the group number
Case	4000	The function expects at least 3 parameters	
Case	4001	Could not calculate the selector.	Invalid engine call for selector expression
Case	4002	Could not calculate the value.	Invalid engine call for value
Script	4100	The function expects at least 2 parameters	
Script	4101	Unknown script type '...'	Type have to be 'python' for this version of the engine
Script	4102	Script file '...' not found or inaccessible.	

Script	4103	Could not load data provided by script	
ScriptGroup	4200	The function expects at least four parameters	
ScriptGroup	4202	Script file '...' not found or inaccessible.	
ScriptGroup	4204	Group item is out of range	Incorrect number of the group item used
Script	4203	Could not load data provided by script	
JSON File	4300	The function expects at least two parameters	
JSON File	4301	JSON data file inaccessible or does not exist	
CVT function	4400	The function expects at least three parameters	
CVT function	4401	Pattern provides no data. Incompatible or empty data source provided	
CVT function	4402	Invalid target format. XML or JSON expected	
RIntN function	4500	The function expects at least 5 parameters.	
RIntN function	4501	Could not compile call.	Check parameters 1,2 and 3
RIntN function	4502	Could not calculate the value.	
RStringN function	4600	The function expects at least 4 parameters	
RStringN function	4601	Could not compile call.	Check parameters 1 and 2
ListN function	4700	The function expects at least 5 parameters	
ListN function	4701	Could not compile call	
Repeat function	4800	The function expects 5 parameters	
Repeat function	4801	Could not compile call	Check the nested pattern
MakeJSONArray function	4900	The function expects 3 parameters	
MakeJSONArray function	4901	Incorrect item ID	Positive integer expected
MakeJSONArray function	4902	Could not compile call	Check the nested pattern
MakeJSONObject function	5000	The function expects at least 4 parameters	
MakeJSONObject function	5001	Incorrect item ID	Positive integer expected
MakeJSONObject function	5002	Could not compile call	Check the nested pattern
MakeJSONItem function	5100	The function expects at least 4 parameters	
MakeJSONItem function	5101	Could not compile call	Check the nested pattern

function

MakeJSON function	5102	Could not compile subitem	Check the nested pattern
First function	5200	Function expects two mandatory parameters	
Last function	5300	Function expects two mandatory parameters	
Const function	5400	Function expects one mandatory parameter	

Engine Editions

There are three editions of the pattern engine: native Windows (32 and 64 bit), .net and multiplatform (Java).

The following table describes differences between editions:

Engine Feature	Windows Native	.net	Multiplatform (Java)
$BLOB function support	Yes	Limited	No
$DLL function support	Yes	Yes	No
$MSAccess function support	Yes	Yes	No
$MSExcel function support	Yes	Yes	No
$Script function support	Yes	No	No
Array functions support	Yes	No	No
Make JSON functions support	Yes	No	No
Format Strings	C++ style	C# style	C++ style

Function call

$FunctionName([agruments_list])

There are built-in functions [function groups]

- BLOB - BLOB loader
- ByExample - by user-provided examples
- Call - call named generator
- Case - select value from list by selector
- Const - constant. It copies argument to output
- CVT - convert data set to document
- Date - date constant
- DLL - call custom generator from external DLL
- File - data from text file
- FileGroup- group based on text file
- First - extracts first symbols from the string
- Format - apply format string to any pattern execution result
- Geography - generate Geography data in WKT format
- Geometry - generate Geometry data in WKT format
- Group - dependent group item
- Guid - unique identifier
- If - select own of two patterns by expression value
- IfR - select own of two patterns by specified probability
- Inc - auto incremental or decremental integer value
- IncChar - auto incremental char/symbol
- IncDate - auto incremental date
- IncFloat - auto incremental numeric value
- IncTime - auto incremental time
- JSON - use JSON file as external data source
- Last - extracts last symbols of the string
- LibGroup - group based on library table
- Library - value from library
- List - complex cases for value list
- ListN - array on predefined values
- ListPattern - unique value list generated by pattern
- Lower - convert to lower case
- MSAccess - data from Microsoft Access database
- MSExcel - data from Microsoft Excel
- Now - current time
- Old - existing value for update and data scrambling
- Pattern - calculate parameter as a pattern
- Query - list of values based on database query
- QueryGroup - group based on database query
- Quote - quote pattern execution results if necessary
- RDate - random date
- Regexp - value defined by regular expression
- Repeat - array on custom values
- RFloat - random float value
- RInt - random integer value
- RIntN - array on integers
- RString - random string
- RStringN - array on strings
- RTime - random time
- Script - run user-defined script as test data provider
- ScriptGroup - group of value based on user-defined script
- Sequence - use same generated value a few times
- TableGroup - group based on database table
- Table - data from database table
- Text - random text
- Time - time constant

-
- Today - current date
- Truncate - truncates string to required length if necessary
- Unique - unique value based on specified pattern
- Upper - convert to upper case
- Variables - resolves references and variables in the pattern
- Web File - loads data from the net via HTTP or FTP
- Web File Group - makes group based on data loaded from the net via HTTP or FTP
- XML - uses data from XML documents

Important

- () pair required even argument list is empty. Example: $Rint()
- Quotation is required for arguments with ','. Example: $FileGroup (1,1,d:\countries.txt,0,0,(,))

Function list grouped by Category

Random Data

- RDate - random date
- RInt - random integer value
- RFloat - random float value
- RString - random string
- RTime - random time
- Text - random text

Database Access

- MSAccess - Microsoft Access data
- MSExcel - Microsoft Excel data
- Query - database query based list
- QueryGroup - database query based group
- Table - database table data
- TableGroup - table based group

Auto Incremental

- Inc - auto incremental or decremental integer
- IncChar - auto incremental char/symbol
- IncDate - auto incremental date
- IncFloat - auto incremental float value
- IncTime - auto incremental time

Conversions

- Case - select value from list by selector
- CVT - convert data set to document
- First - extracts first symbols from the string
- Format - apply format string to any pattern
- Last - extracts last symbols of the string
- Lower - convert to lower case
- Pattern - calculate parameter as a pattern
- Quote - quote pattern execution results if necessary
- Upper - convert to upper case
- Truncate - truncate string if necessary

Date and Time functions

- Date - date constant
- Now - current time
- Today - current date
- Time - time constant

Make JSON functions

- MakeJSONArray - creates JSON-style array
- MakeJSONObject - creates JSON-style

Flow Control

- Call - named generator call
- Case - select value from list by selector
- Group - dependent group item
- If - select own of two patterns by expression value
- IfR - select own of two patterns by specified probability
- List - complex cases for value list
- ListPattern - unique value list generated by pattern
- Old - existing value for update and data scrambling
- Sequence - use same generated value a few times
- Unique - unique value based on specified pattern
- Variables - resolves references and variable calls

External Files

- File - data from text file
- FileGroup- group based on text file
- BLOB - BLOB loader
- JSON - use JSON file as external data source
- XML - uses data from XML documents

Arrays

- RIntN - array on integers
- RStringN - array on strings
- ListN - array on predefined values
- Repeat - array on custom values

Other Data Sources

- ByExample - by user-provided examples
- Guid - unique identifier
- DLL - call custom generator from external DLL
- Geography - generate Geography data in WKT format
- Geometry - generate Geometry data in WKT format
- LibGroup - group based on library table
- Library - value from library
- Regexp - value defined by regular expression
- Script - run user-defined script as test data provider
- ScriptGroup - group of value based on user-defined script
- Web File - HTTP or FTP request based value list

- object
- MakeJSONItem - creates object or array
- MakeJSON - generates JSON document

- Web File Group - HTTP or FTP request based group

BLOB Function

$BLOB function scans a directory for files content. It uses the content of one random file as a source. By default, it uses hexadecimal presentation but text format is also available. The function has one mandatory and one optional parameter:

1. Full path to source folder.
2. (optional) integer value. 1 means 'use text presentation of the content', 0 (default) - hexadecimal.

Example

$BLOB(d:\images)

Important: to create correct BLOB presentation the user has to add depended on the database prefix or quoting. There are samples:
\x'$BLOB(d:\images)' - for MySQL and SQLite database.
0\x$BLOB(d:\images) - for Microsoft SQL Server.
\X'$BLOB(d:\images)' or HEX('...') - for IBM DB2 database.
use TO_LOB conversion function in Oracle, etc.

Notes

1. .NET edition of the engine supports text presentation only
2. Java edition of the engine does not support this function yet

Data by Example Function

The **$ByExample** function generates test data based on sample values provided by the user.
The function accepts a list of samples divided by ','.

Notes

- Minimum recommended number of examples is 3.
- Use extra () quoting for values with comma inside.

Examples

1. $ByExample(Abc,Cdb,Edc) will generate strings with 3 letters that starts with capitalized letter.
2. $ByExample((A,c),(R,b),(E,c)) will generate strings with upper letter, comma and lower letter.

By Query Function

$Query function allows you to generate data based on the database query. There are two versions of the function. The first uses the default connection. The second allows you to specify custom connection (predefined ODBC data source name). Both have two parameters. They are:

1. SQL statement or function (like $Pattern) call
2. (optional) Query Mode, the default is 0.
   '1' means use data from result set sequentially i.e. the first row will be used after the last if necessary.
   '2' means rerun the query for each output row [1]. It is suitable for Oracle sequence requests, UID requests, etc.

Also, the second version has three additional parameters:

1. Predefined **Data Source Name**[2]. String. Use ODBC Administrator to prepare or manage it.
2. (optional) **Login**[3]. String, default is empty.
3. (optional) **Password**. String, default is empty.

[1] - this mode uses the first value of the resultset only.
[2] - use JDBC driver name for Java edition of the engine.
[3] - use connect string for Java edition of the engine.

Function returns NULL value for the empty recordset.

Examples

1. $Query(select [Name] from Customers,0)
2. $Query(select ProfileName from Profile,1,supv3) uses 'supv3' data source name with empty user name and password.
3. $Query($Vars(select [Name] from Customers where ID=@1)) - refers to column #1 in the WHERE clause.

Notes

- Additional quotation by '(' and ')' is required for query that contains ','.
- Identifiers must be quoted if necessary.

Case Function

$Case (or $Switch) calculates expression and uses it as the selector to find depended value by the label. The function has following parameters:

1. Selector expression - the program will use it to find required pair
2. Pair1 definition
3. PairN definition
4. (optional) default value. It will be used if no label found

The pair definition is: <label>:<pattern>, for example: 5:five or 3:$Rint(0,100)

Examples

1. $Case(5,1:one,2:two,3:three,4:four,5:$Rint(0,5)) - random number between 0 and 5 will be generated
2. $Case(0,1:one,2:two,3:three,4:four,5:five,other) - 'other' will be generated
3. $Case(@'F1',1:one,2:two,3:three,4:four,5:five,other) - uses F1 column as value of the selector expression

Constant Value

$Const function copies argument to output without any changes. It allows the user to avoid a lot of escaping or control symbols of the string.

Conversion function

$CVT function converts complete result data set to XML or JSON document. The function has three mandatory parameters:

1. Pattern engine function call*. String.
2. Item name. String.
3. Format. String, XML or JSON.

* - $CVT function applicable to following calls only: $File, $Script, $XML, $WebFile, $JSON, $List, $ListPattern, $Access, $Excel, $Table, $Query

Limitation: only C++ edition of the engine supports this function at the moment.

Example

$CVT($File(d:\Users.txt),UserID,JSON)

Today Function

$Today function retrieves current date in specified format. The only optional parameter is the format. The default format is DD.MM.YYYY

Examples

1. $Today()
2. $Today(MM/DD/YYYY)

Now Function

$Now function retrieves current time. The only optional parameter is the time format. The default format is HH:MM:SS

Examples

1. $Now()
2. $Now(HHMM)

Date Value

$Date function can be used within expressions to define a date value.
It has following parameters:

1. Date constant or expression*. Default is 01.01.1970
2. (optional) Date format, default is DD.MM.YYYY

* - functions call, expression or column-to-column references are acceptable as a value.

Examples

- $Date(04.12.1967,DD.MM.YYYY)
- $Date(@'DateColumnAsString',DD.MM.YYYY)

Form Table Function

$Table function allows you to use a list of value from a database table. There are two usages of the $Table function. The first uses the default connection. The second allows you to use a custom ODBC data source name. Both have the following parameters:

1. **Table Name**. String, identifier quotation not required
2. **Column Name**. String, identifier quotation not required
3. (optional) **Use sequentially**. Integer, default is 0 (false), 1 means true
4. (optional) **Where***. String, default is empty (use all data rows)

Optionally, three additional parameters are available:

1. Predefined **Data Source Name**[1]. String. Use ODBC Administrator to prepare or manage it.
2. (optional) **Login**[2]. String, default is empty.
3. (optional) **Password**. String, default is empty.

[1] - use JDBC driver name for Java edition of the engine.
[2] - use connect string for Java edition of the engine.

Notes

* - the multiplatform edition of the pattern engine does not support 'where' clause. Please use $Query instead.
** - you should use 32 or 64 bit version of the administrator depend on application version. Function returns NULL value for empty table

Examples

1. $Table(Orders,OrderID) uses values from 'OrderID' column in the 'Orders' table.
2. $Table(Orders,OrderID,1,OrderID>100) uses sequentially values greater 100 from 'OrderID' column in the 'Orders' table.
3. $Table(Orders,OrderID,,,MyDsn,Admin,Password) uses 'MyDsn' data source with user name 'Admin' and password 'Password'.

Call Custom generator from DLL

The **$DLL** function provides access to the custom generator. There are two parameters:

1. DLL name*
2. (optional) generator parameters, String.

* - absolute or relative current executable file.

Examples

$DLL(gendll/ssn.dll)

Important: this function is supported by Windows editions (native and .net) of the engine only.

File Function

$File function uses value list from the text file. The engine expects one value per line file format.
The function has the following parameters:

1. **File Name>***. String, quotation not required.
2. (optional) Use data from the file sequentially. 0 or 1, default is 0 (false). The first row will be used after the last of necessary.
3. (optional) Values are patterns. 0 or 1, default is 0 (false).

* - the following macros can be used as a part of the file name: %APPDATA%, %DOCUMENTS%, %USERPROFILE%, %ALLUSERSPROFILE%

Examples

- $File(c:\Program Files\DataBase\sales.txt) - use local file
- $File(\\192.168.137.2\C\dates.txt) - use file from the network share

First Function

The **$First** function extracts one or more first symbols from the source string. Parameters:

1. Source string
2. Number of characters to be extracted

Examples

$First($Lib(Cities),2)

Format Function

$Format function applies format string to the first parameter. There are two mandatory parameters:

1. Any pattern
2. Format string. The engine supports 'd','u','l','x','I','f', and 's' format strings only.

Example

$Format($Table(ti1,A12),%04d)

From Access Database Function

The **$MSAccess** function allows you to use data from Microsoft Access database as a data source. This function has following parameters:

1. **File Name***. String, the quotation is not required.
2. **Table Name**. String, the quotation is not required.
3. **Column Name**. String, the quotation is not required.

* - the following macros can be used as a part of the file name: %APPDATA%, %DOCUMENTS%, %USERPROFILE%, %ALLUSERSPROFILE%

Examples

$MSAccess(d:\databases\sales.mdb,Customers,Contact Name)

Limitation: the function is available for Windows systems only (C++ and .net editions of the pattern engine).

$MSExcel allows the user to add data from the Excel spreadsheet to output data set. The parameters are:

1. **File Name***. String, the quotation is not required.
2. **Spreadsheet Name**. String, the quotation is not required.
3. **Column Name**. String, the quotation is not required.
4. **Use sequentially** (optional). Integer, default if 0 that means 'random'. '1' or 'yes' means sequentially.

* - the following macros can be used as a part of the file name: %APPDATA%, %DOCUMENTS%, %USERPROFILE%, %ALLUSERSPROFILE%

Example

$MSExcel(d:\sales.xls,Contracts,AgentName)

Limitation: the function is available on Windows systems only, not in Java edition.

Geography Data Function

The **$Geography** function generates geography data in WKT format. The function has no parameters.

Example

$Geography()

Geometry Data Function

The **$Geometry** function generates geometry data in WKT format.
The only parameter defines how many items should be generated.

The default value is 1. The function generates POINT, LINESTRING or POLYGON in this case.
Otherwise, it will produce MULTIPOINT, MULTILINESTRING or NULTIPOLIGON.

Examples

1. $Geometry()
2. $Geometry(5)

GUID Function

The **$GUID** function generates the unique identifier. It has two parameters:

- (optional) Use upper case. If this parameter is 1 or 'true' the engine creates output in upper case. The default is 0 (means 'false').
- (optional) Remove group separators. If this parameter is 1 or 'true' the engine removes '-' from output string. The default is 0 (means 'false').

Examples

1. $GUID() generates GUID using default presentation.
2. $GUID(true,true)) generates GUID using upper case without '-' group separators.

Random "if" function

The **$IfR** function returns one of two patterns based on random value. It has the following parameters:

1. Probability, an integer between 0 and 100 percents. The engine will use the first pattern with specified probability and the second otherwise.
2. Pattern 1
3. Pattern 2

Examples

$IfR(10,null,255) means 10 percent for 'null' and 90 percent for 255 value.

See Also

$if function.

Conditional function

$If function returns one of two patterns based on expression value. It has the following parameters:

1. Expression.
2. Pattern 1 (expression is "true")
3. Pattern 2 (expression is "false")

Important

- The engine uses 0 and negative integer values as 'false' and other integers as 'true'. It converts float values to integer by truncating: '0.9' and ' -9' are 'false'.
- If the expression contains '(' and ')' please add extra brackets for whole expression.
- We do not recommend to use string, date or time values as condition directly. Please compare it with another value instead.

Examples

1. $If(@'CustomerID'>255,null,255) means 'null' if 'CustomerID' column greater then 255 or 'null' otherwise.
2. $if(((@'ID'>=2) & (@'ID'<8)),10,0) generates 10 for 'ID' between 1 and 8 (i.e. 2,3,4,5,6,7) and 0 otherwise.
3. $If(@'CustomerName'='null', $CALL(COMPANY),0) means 0 if 'CustomerID' column contains 'null' string or use named generator COMPANY otherwise.

See Also

$ifr function.

Incremental Symbol Function

$IncChar generates a sequence of single char strings and has following parameters:

1. (optional) first symbol. Optional, default is 'a'
2. (optional) last symbol. Optional, default is 'z'
3. (optional) Step. Integer, default is 1. Negative value means decrement.
4. (optional) Reuse counter, integer, default is 1. The generator will return same value a few times if this parameter is positive.

Examples

1. $IncChar() - sequence from 'a' to 'z' and repeat. Output: a,b,c,d,e,f,...
2. $IncChar(A,F) - sequence from 'A' to 'F' and repeat. Output: A,B,C,D,E,F,A,B,C,D,E,F,A,B,C,D,E,F,...
3. $IncChar(c,g,2) - sequence from 'c' to 'g' with step 2 and repeat. Output: c,e,g,c,e,g,c,e,g,c,e,g,...
4. $IncChar(z,q,-1) - sequence from 'z' to 'q' with step -1. Output: z,y,x,w,v,u,t,s,r,q,z,y,x,w,v,u...

Incremental Date Function

$IncDate has two forms. In the first the function has following parameters:

1. (optional) Date and optional time format. See date and time formats for details. The default is DD.MM.YYYY
2. (optional) Initial value*, default is 01.01.1970 (and 00:00:00 if time present).
3. (optional) Step*. Integer, default is 1. Negative values are acceptable.
4. (optional) Step size. 'Y', 'M', 'D', 'H', 'm' and 'S' for year, month, day, hour, minute and second. The default is 'D'.
5. (optional) Sequence border**, the function backs to the first value if the current value greater than this parameter. Date. Empty ("No cycle") is default.
6. (optional) Reuse counter, integer, default is 1. The generator will return same value a few times if this parameter is positive.

In the second form "Initial value" parameter should be replaced by table name and column name. In this case, the engine will start incrementing after the latest available value of the column.

Important: use the $IncTime function instead for time increment only. It works faster.

* - the function call (like $Pattern) is acceptable for 'Initial value' and 'Step' instead of constant. Performance warning: it can work a few times slowly for some cases.

** - applicable to positive steps only.

Examples

1. $IncDate() - same as $IncDate(DD.MM.YYYY,01.01.1970,1,D). Output: 01.01.1970,02.01.1970,03.01.1970,04.01.1970,...
2. $IncDate(DDMMYY,150412,10,M) - start from 15/04/12, step is 10 months. Output: 150412, 150213,151213,151014,150815,...
3. $IncDate(DD/MM/YYYY,10/01/2000,-1,M) - decrement from 10/01/2000 by one month. Output: 10/01/2000,10/12/1999,10/11/1999,10/10/1999,...
4. $IncDate(DDMMYYYY,01012000,1,D,,3) - reuse each date 3 times. Output: 01012000,01012000,01012000, 02012000,02012000,02012000, 03012000,03012000...
5. $IncDate(DDMMYYYY,01012000,1,D,06012000) - sequence border is 06012000. Output: 01012000,02012000, 03012000,04012000, 05012000,06012000, 01012000,02012000,...
6. $IncDate(YYYY-MM-DD HH:mm:SS,,,M) - date with time format, increment by one month starting 1970-01-01. Output: 1970-01-01 00:00:00,1970-02-01 00:00:00,1970-03-01 00:00:00,...
7. $IncDate(,,$Vars(#DateStep)) - start from 01.01.1970 and use #DateStep variable as step value.
8. $IncDate(DD-MON-YYYY, $Pattern(@1),1,M) - use column #1 as initial value, step is 1 month.
9. $IncDate(DD-MON-YYYY,Orders,OrderDate,2,M) - the engine will find maximum value of 'OrderDate' column in 'Orders' table, adds 2 months and use results as initial value

Incremental Integer Number Function

The incremental function **$Inc** has **two forms**.
In the first form it has the following parameters:

1. (optional) Initial value*. Integer, default is 1.
2. (optional) Step*. Integer, default is 1.
3. (optional) Format. String, default is '{0}' for .net edition of the library, '%I64i' for Win32 and '%d' for Java editions.
4. (optional) Sequence border*, the function backs to the first value if the current value greater than this parameter. Integer. 0 is the default value that means "No cycle".
5. (optional) Reuse counter*. Integer, default is 1. How many times the engine should return each generated value.

* - the function call (like $Pattern) is acceptable instead of constant value for: 'Initial value', 'Step', 'Cycle length' and 'Reuse counter'. Performance warning: it can work a few times slowly for some cases.

Examples

1. $Inc() - same as $Inc(1,1). Output: 1,2,3,4,5,6,7,8,9,...
2. $Inc(10,5). Output: 10,15,20,25,30,35,...
3. $Inc(0,-5,%d). Output: 0,-5,-10,-15,-20,-25,...
4. $Inc(0,10,%04d,40,2). Output: 0000,0000,0010,0010,0020,0020,0030,0030,0040,0040,0000,0000,0010,0010,...
5. $Inc($Pattern(@1), $Vars(#step)) - use column #1 as a initial value and local variable '#step' as Step value.

In the **second form** the first parameter should be replaced to table name and column name. The function will use the next value after last found.
I.e. it will execute "select max(column) from table", add "step value" to retrieved number and use the result as the initial value.

Examples

1. $Inc(Orders,OrderID)
2. $Inc(Orders,OrderID,2,%04d)

Limitation: all integer parameters must be between -9223372036854775808 and 9223372036854775807

See also: incremental float number,incremental date, incremental time, incremental symbol.

Incremental Float Value Function

The incremental function **$IncFloat** has **two forms**.
In the first form it has the following parameters:

1. (optional) Initial value*. Numeric, default is 1.0
2. (optional) Step*. Numeric, default is 1.0
3. (optional) Format. String, default is '{0}' for .net edition of the library, '%I64i' for Win32 and '%d' for Java editions.
4. (optional) Sequence border*, the function backs to the first value if the current value greater than this parameter. Float. "No cycle" or 0 by default.
5. (optional) Reuse counter*. Integer, default is 1. How many times the engine should return each generated value.

* - the function call (like $Pattern) is acceptable instead of constant value for: 'Initial value', 'Step', 'Cycle length' and 'Reuse counter'. Performance warning: it can work a few times slowly for some cases.

Examples

1. $IncFloat() - same as $IncFloat(1.0,1.0). Output: 1.0,2,0,3.0,4.0,...
2. $IncFloat(10,0.5) - 10 is initial step is 0.5. Output: 10.0,10.5,11.0,11.5,12.0,12.5,13.0,13.5,...
3. $IncFloat(0,-0.25) - negative step is -0.25. Output: 0.0, -0.25,-0.50,-0.75,-1.0,...
4. $IncFloat(0,0.5,,,2) - use each value twice. Output: 0.0,0.0, 0.5,0.5, 1.0,1.0...
5. $IncFloat(0,0.5,,2.0) - 2.0 is high border. Output: 0.0,0.5,1.0,1.5,2.0,0.0,0.5,1.0,1.5,2.0...
6. $IncFloat($Pattern(@1), $Vars(#step)) - use column #1 as a initial value and local variable '#step' as Step value.

In the **second form** the first parameter should be replaced to table name and column name. The function it will execute "select max(column) from table", add "step" to retrieved value and use result as initial value.

Examples

$IncFloat(Orders,LocationX,2.5)

See also: incremental integer, incremental date, incremental time, incremental symbol.

Incremental Time Function

$IncTime has two modes. The first one has following parameters:

1. (optional) Time Format. See date and time formats for details. Optional, default is HH:MM:SS
2. (optional) Initial value*. Optional, default is 00:00:00.
3. (optional) Step*. Integer. Optional, default is 1. Negative values are applicable.
4. (optional) Step size. 'H', 'M'/'m' or 'S' for hour, minute or second. Optional, default is 'S'.
5. (optional) Sequence border**, the function backs to the first value if the current value greater than this parameter. Time. Empty ("No cycle") is default.
6. (optional) Reuse counter, integer, default is 1. The generator will return same value a few times if this parameter is positive.

In the second mode, the "Initial value" references to the maximum value of some column and will start after it. In this case, table name and column name should be provided instead of immediate value.

* - the function call (like $Pattern) is acceptable for 'Initial value' and 'Step' instead of constant. Performance warning: it can work a few times slowly for some cases.

** - applicable to positive step values only.

Examples

1. $IncTime() - same as $IncTime(HH:MM:SS,00:00:00,1,S). Output: 00:00:00,00:00:01,00:00:02,00:00:03,00:00:04,00:00:05,...
2. $IncTime(HHMM,1000,10,M) - start from 10:00, step is 10 minutes. Output: 1000,1010,1020,1030,1040,1050,1100,1110,...
3. $IncTime(HH:MM,03:00,-1,H) - decrement by 1 hour starting 03:00. Output: 03:00,02:00,01:00,00:00,23:00,22:00,21:00,...
4. $IncTime(HH:MM,03:00,1,H,09:00) - sequence limited by 09:00 value. Output: 03:00,04:00,05:00,06:00,07:00,08:00,09:00, 03:00,04:00,06:00,...
5. $IncTime(HH:MM,03:00,1,H,06:00,2) - limited by 09:00 value with each value twice usage. Output: 03:00,03:00,04:00,04:00,05:00,05:00,06:00,06:00, 03:00,03:00,...
6. $IncTime(HH:MM,$Pattern(@1),1,H) - uses column #1 as initial value, step is 1 hour.
7. $IncTime(HH:MM:SS,Activations,ActivationTime,1,S) - the program will use 'ActivationTime' of 'Activations' table + one second as initial value.

JSON function

$JSON function enables users to get data values or arrays from the file with JSON data. The function has two mandatory parameters:

1. JSON file[1] or resource[2] name with source data. String.
2. Item name[3]. String.

[1] - the following macros can be used as a part of the file name: %APPDATA%, %DOCUMENTS%, %USERPROFILE%, %ALLUSERSPROFILE%
[2] - http:// or ftp:// prefix required to access web-based JSON document.
[3] - the function will use any scalar value or array with this name.

Limitation: only C++ edition of the engine supports this function at the moment.

Example

$JSON(D:\Projects And Files\widget.json,name)

Last Function

The **$Last** function extracts one or more last symbols of the source string. Parameters:

1. Source string
2. Number of characters to be extracted

Examples

$Last($Lib(Countries),2)

The library is a set of predefined tables. Each table contains well-known data like cities, names or countries.
$Lib or **$Library** function can be used to add these data sets to generated data.

The function has the following parameters:

1. **Data set name**. String.
2. (optional) **Column Name**. String, default is 'Name'.
3. (optional) **Maximum acceptable length**. Integer. Default value is 0 that means 'do not check'.
4. (optional) **Use data sequentially**. Integer. Default value is 0 that means 'random'.
5. (optional) **Where clause**. String, default value is empty that means 'all rows'.

Examples

1. $Lib(Cities) - use all cities.
2. $Lib(Cities,Name,10,1) - use cities with name shorter 11 symbols sequentially (A to Z).
3. $Lib(Cities,Name,0,0,State='Ohio') - use cities where "State" is 'Ohio'.

Note: you can use @'column name' in you 'where' parameter if referred column is located before this.
For example, if you have 'State' and 'City' columns the following call is acceptable:
$Lib(Cities,Name,0,0,State='@'State' ')

See also: library.

List Function

The **$List** function allows the user to specify a more complex type of lists than <...>.

There are two types of the function call. In the first case, the first parameter should be 'S' or 'SN' *. The function uses the following list as a set of value that will be used **sequentially**.

Examples

$List(S,1,2,3,4,5) will generate: 1,2,3,4,5,1,2,3,4,5,1,2,...

The second type of call allows the user to specify each value probability. The first parameter must be 'P' or 'PN' * in this case. Other parameters group to pair [pattern, probability]
The probability is an integer value between 0 and 100 where 0 means 'never' and 100 is 'always'.

* - N suffix means the value is not a pattern and should be used without processing.

Examples

1. $List(PN,Y,90,N,10) - returns 90% of 'Y' value and 10% for 'N'.
2. $List(P, 64,30, $Table(Customers,CompanyName),70) - uses sub-pattern "$Table (Customers,CompanyName)" with 70 percent probability.

List of items by pattern function

The **$ListPattern** function executes provided pattern a few times and uses the created list for data generation.
It has two mandatory parameters:

1. Number of unique values, positive integer.
2. Pattern

Examples

1. $ListPattern(30,$Rint()) - creates list of 30 unique random integers and returns random of them.
2. $ListPattern(10,A) - creates list of 10 unique upper letters and returns random item.

Lower Function

The **$Lower** function converts the string to lower case. The only mandatory parameter is source string.

Examples

1. $Lower(($Lib(Cities)) returns Cities name list in lower case.
2. $Lower(X{=8}) generates 8 hexadecimal digits in lower case.

Call Named Generator Function

The **$Call** function returns value of the named generator. The only parameter is generator name. String, mandatory.

Examples

$Call(IP_ADDRESS)

Old Function

The **$Old** function is available in update and data scrambling mode of the data generation. It returns the existing value from database before the modification. Otherwise it returns empty string.

Sample

$Old()

Pattern Function

The **$Pattern** function uses the mandatory parameter as a pattern and calculates it.

Examples

$Pattern($Table(ti1,A3))
The function calls $Table(ti1,A3) and uses retrieved value list as patterns instead of immediate values.

Quote Function

The **$Quote** function allows to add the quotation to begin and end of the value depend on data type or unconditionally.
There are three parameters:

1. Pattern.
2. (optional) data type, see below. Apply quote always for default (empty parameter).
3. (optional) quotation symbol, ' is default.
4. (optional) 1 for double ' symbol for SQL compatibility. 0 is default.

Data Types

The function accepts numeric codes or string data type names. There is name list to be not quoted:
"integer", "decimal", "numeric", "smallint", "float", "real", "bigint", "tinying", "bit", "double", "date", "time".
The numeric data type codes can be found in the ODBC SDK or in files SQL.H + SQLEXT.H by Microsoft (R).

Examples

1. $Quote(A{=3}) - calculate 'A{=3}' pattern, quote always.
2. $Quote(A{=3},,|) - calculate 'A{=3}' pattern, quote always by '|' symbol.
3. $Quote(A{=3},char) - calculate 'A{=3}' pattern, quote by data type 'char'.

Random Date

The **$RDate** function generates random dates. It accepts the following parameters:

1. (optional) **Format**. String, the default value is 'DD.MM.YYYY'. See date and time formats for details.
2. (optional) **Low border**. Date, the default value is '01.01.1970'.
3. (optional) **High border**. Date, the default value is '31.12.2050'.

Notes:

- Values for 2nd and 3d parameters must use the format specified by the first parameter.
- Function call, local or global variable are acceptable as 2nd and 3d parameters.
- The function fixes a few wrong values automatically: month to [1:12] range, day to [1:31] range, etc.

Implementation note: the engine ignores 'High border' value in case it is less than 'Low border'.

Examples

1. $RDate() means random date between '01.01.1970' and '31.12.2050'.
2. $RDate(YYYYMMDD,20000101,20101231) means random date between '01.01.2000' and '31.01.2010' with 'YYYYMMDD' format.

Random Integer

The **$RInt** function generates random integers. This function has a few parameters. All of them are optional. They are:

1. (optional) Low border, Integer, 0 by default.
2. (optional) High border. Integer, 32000 by default.
3. (optional) Format. String, default is '{0}' for .net edition of the library and '%d' for Win32 and Java editions.

Note: function call, local or global variable are acceptable as the first and second parameter.

Examples

1. $Rint()
2. $Rint(-5,5)
3. $Rint(0,999,%05d)
4. $Rint(@GV,#LV) - value between global variable GV and local variable LV

Limitation: the first and second parameters must be between -9223372036854775808 and 9223372036854775807

Random Float

The **$RFloat** function generates random float value. The function has the following parameters:

1. (optional) Low border. Float, default is 0.
2. (optional) High border. Float, default is 1 000 000.
3. (optional) Decimal digits. Integer, default is 2. The maximum is 10. A negative value means generate rounded values with defined number of 0.
4. (optional) Format. String, default is '{0}' for .net edition of the library and '%f' for Win32 and Java editions.
5. (optional) Distribution*. String, default or the empty string means "even distribution".
6. (optional) Dispersion value (for "Normal" distribution only). The default value is 1.

* - "Even", "Normal" or "Linear" distributions are only acceptable.

Note: function call, local or global variable are acceptable as the first and second parameter.

Examples

1. $RFloat()
2. $RFloat(-100,100)
3. $RFloat(0,2000,3)
4. $RFloat(100,2000,-2) - generates values like 400,1200,800,1900
5. $RFloat(0,20,,%7.3f,Normal,5)
6. $RFloat(@GV,#LV) - value between global variable GV and local variable LV

Random String

The **$RString** function generates random strings. There are parameters:

1. Minimum string length. Integer, mandatory.
2. Maximum string length. Positive integer, mandatory.
3. (optional) Upper letter probability. Integer, default is 5%. Use 0 for all lower case letter.
4. (optional) Space char probability. Integer, default is 5%. Use 0 to disable space chars.
5. (optional) Separators (.,?!) probability. Integer, default is 0%. Use 0 to disable space chars.
6. (optional) List of Unicode blocks*.

* - one or more constants: 'Latin','Greek',
'Cyrillic','Hebrew','Arabic','Syriac','Bengali','Thai','Lao','Mongolian','CJK','Hangul'.
Only Unicode version of the library supports this parameter. ANSI version ignores it.

Examples

1. $RString(1,10) - generates strings between 1 and 10 letters, 5% upper letters and 5% spaces, without separators.
2. $RString(1,10,0,0,0)- generates strings between 1 and 10 letters, without upper letters, spaces and separators.
3. $RString(1,10,,,,Greek,Cyrillic) - generates strings between 1 and 10 letters with Greek and Cyrillic Unicode symbols.

Random Time

The **$RTime** function generates random time and has the following parameters:

1. (optional) Format. See date and time formats for details. The default is HH:MM:SS
2. (optional) Low border. Default is 00:00:00
3. (optional) High border. Default is 23:59:59

Notes

- function call, local or global variable are acceptable for range definition.
- The function fixes wrong parameters automatically: hour to [0:23] range, minutes, and seconds to [0:59] range.

Examples

1. $RTime() - means $RTime(HH:MM:SS,00:00:00,23:59:59)
2. $RTime(HHMM)
3. $RTime(HHMMSS,000000,100000)

Regular Expression Function

The **$Regexp** function uses specified regular expression as a pattern for data generation. The only mandatory parameter is a regular expression, an empty string is acceptable.

Supported Options

- The function supports standard UNIX-style regular expressions except items mentioned in 'Limitations' section.
- \d means digit
- \w means letter digit or '_'
- \s means space symbols
- Repeaters: {n} means exact n times, {n,m} means between n and m times

Limitations

1. The function ignores '$' and '^' signs for begin and end.
2. The function does not support negative ranges like [^0-9].
3. Only \1 to \9 blocks can be used.

Script Function

The **$Script** function allows users to run an external script as a data source. The function has two mandatory parameters:

1. The full path and file name of the script interpreter.
2. File path and name with the script without quoting*.

* - the following macros can be used as a part of the file name: %APPDATA%, %DOCUMENTS%, %USERPROFILE%, %ALLUSERSPROFILE%

Any number of optional parameters will be passed to the script. The column to column reference or expression call is acceptable.

The script should write generated data to standard output stream. One value per line.

Examples

1. $Script(c:\python33\python.exe,c:\my scripts\test.py) - Python script without extra parameters
2. $Script(c:\python33\python.exe,c:\my scripts\test.py,Data1,10) - Python script with two parameters: 'Data1' and '10'

Sample Python script

The script generates and returns squares between 1 and the first argument:

```
import  sys

iter  =  1+int(sys.argv[1])
for  a  in  range(1,iter):
     print(a*a)
```

Sequence Function

$Sequence or **$Seq** allows you to use same generated data a few times. There are two parameters:

1. Pattern.
2. Counter*. Integer, 1 or greater.

* - the function call (like $Pattern) is acceptable for 'Initial value' and 'Step' instead of constant. Performance warning: it can work a few times slowly for complex cases.

Examples

1. $Seq(A{=3},5) - generates value for 'A{=3}' pattern and returns it 5 times, generates next value...
2. $Seq(A{=3},$Vars(#Counter)) - the function uses #Counter local variable for counter.

Text Function

$Text function generates random texts. There are two versions of the function. The first version operates with Value Library. The second one uses user defined phrase list. The phrase list is a text file with one phrase per line.

The first version of the function has two parameters:

1. (optional) **maximum text length**. The default value is 128.
2. (optional) **language***. The default is EN.

* - current version of the value library supports only following languages: EN, DE, FR, IT and RU.

Examples

1. $Text(255)
2. $Text()
3. $Text(500,IT)

The second version of the function also has two parameters. Both are mandatory:

1. **Maximum text length**.
2. **File name***.

* - the following macros can be used as a part of the file name: %APPDATA%, %DOCUMENTS%, %USERPROFILE%, %ALLUSERSPROFILE%

Examples

$Text(200,d:\phrases\estonian.txt)

Time Value

$Time function can be used within expressions to define a time value. The function call has the following parameters:

1. Time constant or expression*. Default value is 00:00:00
2. (optional) Time format, default is HH:MM:SS

* - functions call, expression or column-to-column references are acceptable as a value.

Example

- $Time(0400,HHMM)
- $Time(@'TimeColumn') - default format HH:MM:SS

"Truncate" Function

$Truncate function returns substring of the first parameter with the length defined by the second parameter.
The function has the following parameters:

1. **Pattern**.
2. **Length**, positive integer. If the first parameter's value is longer than "Length" the function returns substring. The whole value will be returned otherwise.

Important: extra () pair is required for patterns that contain ','.

Examples

1. $Truncate($Lib(FirstNames) A. $Lib(LastNames),20)
2. $Truncate(($Rint(1,250), $Lib(Streets)),20)

Unique Expression Value

$Unique (or $Uniq) function must be used together with another pattern. It instructs data generation engine that value, specified by mentioned pattern must be unique. This pattern is the only parameter of the function.

Examples

$Unique(NNN)

See Also

Unique by pattern function.

Upper Function

The **$Upper** function converts the string to upper case. The only mandatory parameter is source string.

Examples

$Upper($Lib(Cities))

Variables Function

The **$Variables** (or **$Vars**) function resolvers the column references and variables (if applicable. Please refer to the data generation software's manual for details) in the provided pattern string.
The only mandatory parameter is source pattern.

Notation

- @n means reference to column #n. Example: @2 means the second column value.
- @'name' means reference to column 'name'. Example: @'ID' means ID column value.
- @name means reference to global variable 'name'. Example: @year refers to variable 'year'
- #name means reference to local variable 'name'.

Notes

- There is no additional quotation is required if the pattern contains ',' signs.
- The engine calculates this function only once before execution even referred by '@' or '@@' column has been changed.

Examples

$Query($Vars(select OrderID from Orders where CustomerID=@1 and OrderNo=#number))

Web File Function

$WebFile function loads text file from specified URL (by HTTP or FTP) or calls web service for a list of data values. The function expects one value per line file format. It has the following parameters:

1. **URL** starts from 'http' or 'ftp'. String, quotation not required.
2. (optional) Use data from the file sequentially. 0 or 1, default is 0 (false). The first row will be used after the last of necessary.
3. (optional) Values are patterns. 0 or 1, default is 0 (false).

Examples

1. $WebFile(http://www.sqledit.com/data/sales.txt)
2. $WebFile(ftp://sqledit.com/sales.txt,1)

XML function

$XML allows the user to extract data lists from the XML file. The function has two mandatory parameters:

1. XML file name[1] or resource[2] with source data. String.
2. The path to value list. String.

Limitation: only C++ and .net editions require full path when Java edition uses node name only.

[1] - the following macros can be used as a part of the file name: %APPDATA%, %DOCUMENTS%, %USERPROFILE%, %ALLUSERSPROFILE%
[2] - http:// or ftp:// prefix is required for web based document access.

Examples

1. $XML (d:\dg.xml,//XML_DIZ_INFO/MASTER_PAD_VERSION_INFO/MASTER_PAD_VERSION) - C++/C# style
2. $XML(\\192.168.137.2 \C\dg.xml,//XML_DIZ_INFO/MASTER_PAD_VERSION_INFO/MASTER_PAD_VERSION) - uses file from the network share
3. $XML(d:\dg.xml,MASTER_PAD_VERSION) - Java style.

Group By Excel Spreadsheet

$MSExcelGroup function can be used to create a value group based on data from Microsoft Excel file.
The function has following parameters:

1. **Group** number. Positive integer.
2. **File Name***. String, a path to the file.
3. **Sheet Name**. String, identifier quotation not required.
4. **Column List**. List of strings, comma delimited. Quotation not required.
5. (optional) Use data from the table **sequentially**. 0 or 1. Optional, default is 0 (false). The first row will be used after the last of necessary.

* - the following macros can be used as a part of the file name: %APPDATA%, %DOCUMENTS%, %USERPROFILE%, %ALLUSERSPROFILE%

Example: $MSExcelGroup(1,c:\data files\countries.xls,Sheet1,Name,Country) - specifies sheet 'Sheet1' in the 'c:\data files\countries.xls' file as data source and uses the first column 'Name'.
$Group(1,2) - refers to the second column 'Country'.

Limitation: Java edition of the pattern engine does not support this function.

Group by Table Function

There are two versions of the "group by table" function: $TableGroup and $TableGroupC. **$TableGroup** function can be used to create a value group based on value list from a database table and uses default connection.

The function has the following parameters:

1. **Group** number. Positive integer.
2. **Table Name**. String, identifier quotation not required.
3. **Column List**. List of strings, comma delimited. Quotation not required.
4. (optional) Use data from the table **sequentially**. 0 or 1, default is 0 (false). The first row will be used after the last if necessary.

Example: $TableGroup(1,City,Name,Country) - specifies table 'City' as data source and uses the first column 'Name'.
$Group(1,2) - refers to the second column 'Country'.

The second version (**$TableGroupC**) requires specified ODBC data source name and, optional, login (user name) and password.

The function has the following parameters:

1. **Group** number. Positive integer.
2. Predefined **Data Source Name**[1]. String. Use ODBC Administrator[*] to prepare or manage it.
3. **Login**[2]. String, default is empty.
4. **Password**. String, default is empty.
5. **Table Name**. String, identifier quotation not required.
6. **Column List**. List of strings, comma delimited. Quotation not required.
7. (optional) Use data from the table **sequentially**. 0 or 1, default is 0 (false). The first row will be used after the last if necessary.

[1] - use JDBC driver name for Java edition of the engine.
[2] - use connect string for Java edition of the engine.

[*] you should use 32 or 64 bit version of the administrator depend on application version.

Note: function returns NULL value for empty table

Examples

$TableGroupC(1,MyDb,UserName,Password,City,Name,Country)

Group by File

The **$FileGroup** function creates a group based on the text file. It expects a set of delimited values in the source file.

The function has following parameters:

1. **Group number**. Positive integer.
2. **Column number**. Positive integer.
3. **Source file name***. String, quotation not required.
4. (optional) Use data from file **sequentially**. 0 or 1. Optional, default is 0 (false). The first row will be used after the last of necessary.
5. (optional) File values are **patterns**. 0 or 1. Optional, default is 0 (false).
6. (optional) Value separator. Default is TAB. Use quoted (,) for comma.

* - the following macros can be used as a part of the file name: %APPDATA%, %DOCUMENTS%, %USERPROFILE%, %ALLUSERSPROFILE%

Examples

1. $FileGroup(1,1,d:\data files\discounts.txt) - specifies 'discounts.txt' as data source and uses the first column.
   $Group(1,2) - refers to the second column.
2. $FileGroup(1,1,d:\countries.txt,0,0,|) - the engine should use pipe as value separator for 'd:\countries.txt' file.
3. $FileGroup(1,1,\\192.168.137.2\C\dates.txt) - use file from the network share

Group By Query

The **$QueryGroup** function can be used to create a value group based on the database query. There are two versions of the function. The first uses the default connection. The second allows the user to specify custom connection (predefined ODBC data source name). Both have the following parameters:

1. **Group No**. Positive Integer.
2. **Query text**. Additional quotation by '(' and ')' is required for the query that contains ','. The function (like $ Pattern) call is acceptable as well.
3. (optional) Query Mode, the default is 0.
   '1' means use data from result set sequentially i.e. the first row will be used after the last if necessary.
   '2' means rerun the query for each output row [1]. It is suitable for Oracle sequence requests, UID requests, etc.

The second version has three additional parameters:

1. Predefined **Data Source Name**. String. Use ODBC Administrator* to prepare or manage it.
2. (optional) **Login**. String, default is empty.
3. (optional) **Password**. String, default is empty.

* you should use 32 or 64 bit version of the administrator depend on application version.

Note: function returns NULL value for an empty recordset

Examples

1. $QueryGroup(1,(select AgentName,Code from Contracts)) - specifies query as data source and uses 'AgentName' column.
   $Group(1,2) - refers to the second column 'Code'.
2. $QueryGroup(1,(select AgentName,Code from Contracts),0,MyDb,UserName,Password) - uses "MyDb" ODBC data source with specified user name and password.

Group Item

$Group function specifies depended item of the group. It has two mandatory parameters: group number and column in the group. Both parameters must be greater than 0.

Notes

- It is strongly recommended to define the group before use. For example, if you want to unite columns #5, #6 and #8 to group, you should define a group for column #5 and use $Group(...) calls for #6 and #8.

Examples

$Group(2,3) call refers to column #3 of the group #2.

Group by Library Function

$LibGroup function allows the user to create a value group based on Value Library's table. The function has the following parameters:

1. **Group number**. Positive Integer.
2. **Data Set Name**. String.
3. **Column List**. Comma-separated string list or * for all columns.
4. (optional, for '*' as column list only) **Where clause**. String, default value is empty that means 'all rows'.

Examples

1. $LibGroup(2,Cities,Name,Country)
   where: 2 is a group name, 'Cities' is list name and 'Name,Country' - columns.
2. $LibGroup(1,Cities,*) - create group #1 based on 'Cities' set with all columns included.
3. $LibGroup(1,Cities,*,Country='USA') - create group #1 based on US Cities list.

Script Group Function

$ScriptGroup function allows users to run an external script as a data source for the group of data. The function has four mandatory parameters:

1. **Group number**. Positive Integer.
2. **Column number**. Positive integer.
3. The full path and file name of the **script interpreter**. String.
4. File path and name with the **script** without quoting*. String.

* - the following macros can be used as a part of the file name: %APPDATA%, %DOCUMENTS%, %USERPROFILE%, %ALLUSERSPROFILE%

Any number of optional parameters will be passed to the script. The column to column reference or expression call is acceptable.

The script should write generated data to standard output stream separated by tab ("\t") symbol.

Examples

1. $ScriptGroup(1,1,c:\python33\python.exe,c:\my scripts\test.py) - Python script without extra parameters crates group #1 and returns 1st column of data
2. $ScriptGroup(2,4,c:\python33\python.exe,c:\my scripts\test.py,Data1,10) - Python script with two parameters: 'Data1' and '10' creates group #2 and returns 4s column

Group by Web File

The **$WebFileGroup** function creates a group based on HTTP or FTP requests. It expects a set of delimited values in the answer, one set per line.

The function has following parameters:

1. **Group number**. Positive integer.
2. **Column number**. Positive integer.
3. **URL** starts with 'http' or 'ftp'. String, quotation not required.
4. (optional) Use data **sequentially**. 0 or 1. Optional, default is 0 (false). The first row will be used after the last of necessary.
5. (optional) File values are **patterns**. 0 or 1. Optional, default is 0 (false).
6. (optional) Value separator. The default separator is TAB. Use quoted (,) for the comma.

Examples

1. $WebFileGroup(1,1,http://www.sqledit.com/data/discounts.txt) - specifies 'discounts.txt' from 'http://www.sqledit.com/data' as data source and uses the first column.
   $Group(1,2) - refers to the second column.
2. $WebFileGroup(1,1,ftp://sqledit.com/countries.txt,0,0,|) - the engine should use pipe as value separator for 'countries.txt' file requested from 'ftp://sqledit.com'.

Array Functions

The array generation functions produce a set of values delimited by comma or user defined separator. The result is suitable for post-relational fields populations like ARRAY in PostgreSQL.

There are array functions:

- $RintN - array of integer values
- $RStringN - strings array
- $ListN - array of predefined values
- $Repeat - general type of array

Array of integer values

The **$RintN** generates a sequence of random integer values. There are parameters:

1. Minimal value, 0 for empty parameter
2. Maximal value, 32000 by default
3. Format string, %d is default
4. Minimal sequence length, default is 1
5. Maximal sequence length, default is 1
6. Array item separator, empty string means comma

Examples

1. $RintN(-10,10,,2,4) - sequence of integers between -10 and 10 with length 2 to 4 and separated by comma.
2. $RintN(0,100,%03d,3,3,|) = array of integer with zero-padding and pipe as separator. length is three: 012|077|002

$RStringN generates a sequence of random strings. The function has following parameters:

1. Minimum string length. Integer, mandatory.
2. Maximum string length. Positive integer, mandatory.
3. Minimal sequence length, default is 1
4. Maximal sequence length, default is 1
5. Quote char, empty string means no quote required
6. Array item separator, empty string means comma

Note: use $Repeat function call instead of this function if you want to tune all $RString options.

Examples

1. $RStringN(5,10,2,4,') - sequence of strings with length between 5 and 10. Sequence length is random between 2 and 4. The delimiter is comma, quote char is '.

Array of Predefined Values

The **$ListN** creates a sequence of predefined values delimited by comma or user defined separator. The parameters are:

1. Minimum string length. Integer, mandatory.
2. Maximum string length. Positive integer, mandatory.
3. Quote char, empty string means no quote required
4. Sequence item separator, empty string means comma
5. Values of list...

Note: use $Repeat call with nested $List call to tune the list generation in more details.

Examples

1. $ListN(4,4,',,Y,\N) generates a sequence of Y and N letters with length 4 and quoted by '. For example: 'Y','Y','N','N'
2. $ListN(0,3,,|,true,false) generates a sequence of 'true' and 'false' strings. The length between 0 and 4 with pipe as delimiter, without quotation. For example: true|true|false

General Type of Array Generation

$Repeat allows the user to create an array based on nested pattern call. It has four parameters:

1. Minimal sequence length, default is 1
2. Maximal sequence length, default is 1
3. Quote char, empty string means no quote required
4. Delimiter, empty string means comma
5. Nested pattern engine expression or function call.

Note: use extra () if the nested expression has comma(s).

Examples

1. $Repeat(1,5,,,$Rint()) - 1 to 5 random integers delimited by comma, no quotation
2. $Repeat(3,3,,*,XXXX) - array of 3 hexadecimal blocks (four digits per block), no quotation. The values are separated by '*' like 'AB67*905D*452C'

JSON Functions

The JSON generation functions create components of document or complete JSON document.

There are functions:

- MakeJSONArray - creates JSON-style array [item1,item2,...,itemN]
- MakeJSONObject - creates JSON-style object {"name":"value"}
- MakeJSONItem - creates object or array
- MakeJSON - generates complex JSON document

Make JSON Array Function

$MakeJSONArray generates a JSON-style array: [item,item,...,item].

The function has following parameters:

- **ID** item ID, unique positive integer value for references.
- **Object Name** optional string, name of object as a constant or as a pattern.
- **Is pattern** 1 or true if the engine have to use object name as a pattern. Default (empty string) or any another value means name is constant.
- **Value definition** is an integer reference item(s) ID (as array of items between '[' and ']' like [3] or [2,3]) or pattern between '(' and ')'.
- **Repeater**, N for 0 to N items, =N for exact N values or N:M for N to M values.

Examples

1. $MakeJSONArray(1,,,12,=5) generates array of 5 items, Each item has reference number 12.
2. $MakeJSONArray(2,HEX,,(XXXX),2:3) generates an object HEX with array of 2 to 3 items. Each item is a sequence of 4 hexadecimal digita like {"HEX":[128F,4FB0,AB82]}

Make JSON Object Function

$MakeJSONObject function generates one or more JSON-style objects as a pair: name:value. The function quotes a name and value if necessary and separates pairs by comma.

The function has following parameters:

- **ID** item ID, unique positive integer value for references.
- **Object Name** string, name or object as a constant or as a pattern
- **Is pattern** 1 or true if the engine have to use object name as a pattern. Default (empty string) or any another value means name is constant.
- **Value definition** is an integer reference item(s) ID (as array in '[]' like [2] or [2,3]). or pattern between '(' and ')'.
- **Repeater**, N for 0 to N items, =N for exact N values or N:M for N to M values.
- **Quote** optional value. Default is 1 means add '{' before and '}' after generated object definition.

Examples

1. $MakeJSONObject(1,Name,,($Lib(FirstNames))) generates a pair of name: {"Name":"Mike"}.
2. $MakeJSONObject(2,Code,,($Lib(Countries,Code2))) generates a pair of country code: {"Code":"UK"}.

Make JSON Item Function

$MakeJSONItem is a wrapper for $MakeJSONArray and $MakeJSONObject call. It analyzes the passed parameters and calls one of mentioned functions.

$MakeJSONItem has following parameters:

- **ID** item ID, unique positive integer value for references.
- **Object Name** string, name or object as a constant or as a pattern. The engine will call $MakeJSONArray if this parameter is empty.
- **Is pattern** 1 or true if the engine have to use object name as a pattern. Default (empty string) or any another value means name is constant.
- **Value definition** is an integer reference item(s) ID (as array in '[]' like [2] or [2,3]) or pattern between '(' and ')'.
- **Repeater**, optional, default is 1. N for 0 to N items, =N for exact N values or N:M for N to M values.

Examples

1. $MakeJSONItem(1,,,($Rint()),=5) will call $MakeJSONArray(1,,,($Rint()),=5)
2. $MakeJSONItem(2,CityN,1,($Lib(Cities)),4:5) will call $MakeJSONObject(2,CityN,1,($Lib(Cities)),4:5)

Make JSON Document Function

$MakeJSON generates complete JSON document based on passed structure. The function call should contain a list of item definitions separated by comma. Each definition must be closed by pair of '(' and ')' for objects or '[' and ']' for arrays. Each item definition is exact same as $MakeJSONItem call.

There are two types of items: root items and references. The function generates values for the first and uses as a reference only for the second. The reference item must be encoded as @(...) and the root one as (...).

Examples

1. $MakeJSON((1,A,,[3],,0),(2,B,,[4]),@[3,,,($Rint()),=5],@(4,CityN,1,($Lib(Cities)),=2)) will generate a document like {"A":[100,43012,7,6578,332],"B": {"City4":"Madrid","City6":"Paris"}}
   (1,A,,3) creates an object with name "A" with reference to array defined by ID #3.
   (2,B,,4) produces an object with name "B" that references to object #4.
2. $MakeJSON((1,A,,[2,3]),@(2,B,,(XXXX),,0),@(3,C,,($Rint()),,0)) generates A with two depended (child) items like {"A":{"B":"874D","C":30976}}

Value Library

The "Value Library" is a set of well-known data. It can be used by pattern engine (see $Lib and $LibGroup functions) to make generated data more realistic.

Currently, the library[1] contains the following data sets:

Data set	Table Name	Data Columns	Values[2]
World cities	Cities	Name, Country, State (for US cities only), State Code (for US cities only)	1263
Closing sizes	Closingsizes	Name	11
Color set	Colors	Name	143
Companies list	Companies	Name, ShortName, Domain	673
Countries list	Countries	Name, Code2, Code3, Capital, Currency, CurrencyCode	246
Week days	Days	Name,Code2,Code3,Idx	7
Sample company departments	Departments	Name	15
File types	FileTypes	Name, MimeType, Description	60
First names	FirstNames	Name, Sex	5146
First names national	FirstName_National	Name, LatinName, Sex, Country	631
Full names	FullNames	Name	373
Gender	Gender	Name, Abbr, Code	5
Sample industries	Industries	Name	41
World languages	Languages	Name, Code2, Code3	143
Last Name Samples	LastNames	Name	370
Legal forms	LegalForms	Name	15
Months	Months	Name	12
MeasureUnits	Name,Code	Name	340
Name prefixes	NamePreffix	Name	7
Name suffixes	NameSuffix	Name	8
Nationality	Nationality	Name	85
Sample occupations or positions	Occupations	Name	74
Region	Region	Name	4
Streets list	Streets	Name, Country	9185
US States	US_States	Name, Code, Capital	52
Zip codes	ZipCodes	Name, Country	670

1. the library has SQLite 3 physical format and can be modified by DTM data Editor or any compatible tool
2. the library database version 10-jan-2016.

Note: for most library functions call you can skip default column name ("Name").

Library Index Table

You should modify library index if you want to add a new table to the library. The index table called "_index". Each row corresponds to the node or leaf.

Each row has unique identifier (ID column) and all child rows must refer to this ID in the "Parent" column. "Hidden" boolean value describes that column should not be used in visual index presentation. "Comment" column is optional.

This section of the document presents a few solutions for most popular data generation tasks.

Task Definition	Pattern	Sample Output
How to get current date and time without delimiters	**$Today(DDMMYYYY)$Now(HHMMSS)**	040420100855
How to generate money with currency code	**$Rfloat(0,100,2) ($Lib (Countries,CurrencyCode)** value between 0 and 100 with 2-digits cent value	70.91 CAD
How to generate debit and credit values	**$ifr(30,\($Rfloat(,,2)\),$Rfloat(,,2))** it produces negative value with 30% probability with 2-digits cent value	100.25 (23401.56)

Examples for Numeric Values

This section presents a few examples of numeric data generation.

Definition	Pattern	Sample output
IP address	$Rint(0,255).$Rint(0,255).$Rint(0,255).$Rint(0,255)	41.107.214.235 187.239.95.95 190.212.237.81
The difference between the first and second columns with a random number added	$$(@1-@2+$RInt())	100.25 23401

Examples for Date and Time Values

This section presents a few examples of date and time values generation.

Definition	Pattern	Sample output
One year after entered date	$$($Date(12.10.2008,DD.MM.YYYY)+365)	12.10.2009
Random date between 1999-01-01 and today	$RDate(YYYY-MM-DD,1999-01-01,$Today(YYYY-MM-DD))	2009-10-04
Random date between "BirthDay" column + 18 years and today	$RDate(YYYY-MM-DD,$$(@'BirthDate' +365*18),$Today(YYYY-MM-DD))	2000-12-01
Random date between 1999-01-01 and today - 18 years	$RDate(YYYY-MM-DD,1930-01-01,$$($Today(YYYY-MM-DD)-18*365))	1999-11-21

Other example

This section presents a few examples for some specific case.

Definition	Pattern	Sample output
Boolean value	<True|F\alse>	True
Random hexadecimal color	\#X{=6}	#3E41E8
Random monochrome color	\#(XX)#\#	#9C9C9C
Sample MAC address	XX:XX:XX:XX:XX:XX	D7:7F:0D:A9:AB:E5
Random IPv6 address	XXXX:XXXX:XXXX: XXXX:XXXX:XXXX:XXXX:XXXX	07FD:F848:74E2: FBB5:8A52:2C90: 41C6:BF9B
Longitude	$RInt(0,179)°$Rint(0,59)'$Rint(0, 59)"<\N|E>	14°14'23"N
Latitude	$RInt(0,89)°$Rint(0,59)'$Rint(0, 59)"<W|E>	59°30'44"E