

Expression Editor and Listen Node



Original Instruction

The information contained herein is the property of Techman Robot Inc. (hereinafter referred to as the Corporation). No part of this publication may be reproduced or copied in any way, shape or form without prior authorization from the Corporation. No information contained herein shall be considered an offer or commitment. It may be subject to change without notice. This Manual will be reviewed periodically. The Corporation will not be liable for any error or omission.



REVISION HISTORY TABLE.....	11
1. EXPRESSION	12
1.1 Types.....	12
1.2 Variables and Constants	12
1.3 Array	16
1.4 Operator Symbols.....	18
1.5 Data type conversion.....	20
1.6 Warning.....	21
2. FUNCTIONS	24
2.1 Byte_ToInt16().....	24
2.2 Byte_ToInt32().....	25
2.3 Byte_ToFloat()	26
2.4 Byte_ToDouble().....	27
2.5 Byte_ToInt16Array()	28
2.6 Byte_ToInt32Array()	29
2.7 Byte_ToFloatArray().....	30
2.8 Byte_ToDoubleArray()	31
2.9 Byte_ToString().....	32
2.10 Byte_Concat()	33
2.11 String_ToInteger()	36
2.12 String_ToFloat()	38
2.13 String_ToDouble().....	40
2.14 String_ToByte().....	42
2.15 String_IndexOf()	43
2.16 String_LastIndexOf()	44
2.17 String_Substring().....	45
2.18 String_Split().....	48

2.19	String_Replace()	49
2.20	String_Trim()	49
2.21	String_ToLower()	51
2.22	String_ToUpper()	52
2.23	Array_Append()	52
2.24	Array_Insert()	53
2.25	Array_Remove()	54
2.26	Array_Equals()	55
2.27	Array_IndexOf()	57
2.28	Array_LastIndexOf()	58
2.29	Array_Reverse()	59
2.30	Array_Sort()	61
2.31	Array_SubElements()	62
2.32	ValueReverse()	64
2.33	GetBytes()	68
2.34	GetString()	72
2.35	GetToken()	80
2.36	GetAllTokens()	89
2.37	GetNow()	91
2.38	GetNowStamp()	93
2.39	GetVarValue()	96
2.40	Length()	97
2.41	Ctrl()	98
2.42	XOR8()	100
2.43	SUM8()	101
2.44	SUM16()	102
2.45	SUM32()	105
2.46	CRC16()	106

2.47	CRC32()	108
2.48	ListenPacket()	110
2.49	ListenSend().....	111
2.50	VarSync()	113
3.	MATH FUNCTIONS	115
3.1	abs()	115
3.2	pow().....	116
3.3	sqrt()	118
3.4	ceil()	118
3.5	floor()	119
3.6	round()	120
3.7	random()	122
3.8	d2r()	123
3.9	r2d()	124
3.10	sin()	124
3.11	cos()	125
3.12	tan()	126
3.13	asin()	127
3.14	acos().....	127
3.15	atan().....	128
3.16	atan2().....	129
3.17	log().....	130
3.18	log10().....	131
3.19	norm2()	132
3.20	dist().....	133
3.21	trans().....	133
3.22	inversetrans()	134
3.23	applytrans()	136

3.24	interpoint()	138
3.25	changeref()	138
3.26	points2coord()	140
3.27	intercoord()	142
4.	FILE FUNCTIONS	145
4.1	File_ReadBytes()	146
4.2	File_ReadText()	146
4.3	File_ReadLines()	147
4.4	File_NextLine()	149
4.5	File_NextEOF()	153
4.6	File_WriteBytes()	154
4.7	File_WriteText()	159
4.8	File_WriteLine()	162
4.9	File_WriteLines()	166
4.10	File_Exists()	169
4.11	File_Length()	169
4.12	File_Delete()	170
4.13	File_Copy()	171
4.14	File_CopyImage()	172
4.15	File_Replace()	175
4.16	File_GetToken()	176
4.17	File_GetAllTokens()	180
5.	SERIAL PORT FUNCTIONS	183
5.1	com_read()	183
5.2	com_read_string()	189
5.3	com_write()	195
5.4	com_writeline()	197
6.	SOCKET FUNCTIONS	200

6.1	socket_read()	200
6.2	socket_read_string().....	206
6.3	socket_send()	212
6.4	socket_sendline()	214
7.	PARAMETERIZED OBJECTS	217
7.1	Point.....	218
7.2	Base	219
7.3	TCP	220
7.4	VPoint	221
7.5	IO	222
7.6	Robot	225
7.7	FT	226
8.	EXTERNAL SCRIPT.....	229
8.1	Listen Node	229
8.2	ScriptExit()	230
8.3	Communication Protocol	230
8.4	TMSCT	232
8.5	TMSTA.....	234
8.6	CPERR.....	238
8.7	Priority Commands	240
9.	ROBOT MOTION FUNCTIONS.....	244
9.1	QueueTag().....	244
9.2	WaitQueueTag()	245
9.3	StopAndClearBuffer().....	247
9.4	Pause()	247
9.5	Resume()	249
9.6	PTP().....	249
9.7	Line()	253

9.8	Circle().....	255
9.9	PLine().....	258
9.10	Move_PTP().....	260
9.11	Move_Line()	262
9.12	Move_PLine()	264
9.13	ChangeBase().....	266
9.14	ChangeTCP().....	267
9.15	ChangeLoad()	270
9.16	PVTEnter()	271
9.17	PVTExit().....	272
9.18	PVTPoint()	272
9.19	PVTPause()	273
9.20	PVTResume()	274
	Pose Configuration Parameters: [Config1, Config2, Config3]	275
9.21	Vision_DoJob().....	275
9.22	Vision_DoJob_PTP().....	276
9.23	Vision_DoJob_Line()	277
9.24	Vision_IsJobAvailable()	278
10.	MODBUS FUNCTIONS	279
10.1	modbus_read().....	279
10.2	modbus_read_int16().....	282
10.3	modbus_read_int32().....	284
10.4	modbus_read_float().....	286
10.5	modbus_read_double()	288
10.6	modbus_read_string()	290
10.7	modbus_write().....	292
11.	TM ETHERNET SLAVE	297
11.1	GUI Setting	297

11.2	svr_read()	298
11.3	svr_write()	298
11.4	Data Table	299
11.5	Communication Protocol	301
11.6	TMSVR.....	302
0.	Mode = 0 (the status the server responds to the client command processing)	304
1.	Mode = 1 BINARY.....	306
2.	Mode = 2 STRING.....	308
3.	Mode = 3 JSON	309
11.	Mode = 11 BINARY (Request read).....	310
12.	Mode = 12 STRING (Request read)	312
13.	Mode = 13 JSON (Request read).....	313
12. PROFINET FUNCTIONS		315
12.1	profinet_read_input().....	316
12.2	profinet_read_input_int().....	320
12.3	profinet_read_input_float().....	323
12.4	profinet_read_input_string()	325
12.5	profinet_read_input_bit().....	326
12.6	profinet_read_output()	330
12.7	profinet_read_output_int()	333
12.8	profinet_read_output_float()	336
12.9	profinet_read_output_string().....	338
12.10	profinet_read_output_bit()	339
12.11	profinet_write_output()	341
12.12	profinet_write_output_bit()	350
13. ETHERNET/IP FUNCTIONS.....		356
13.1	eip_read_input().....	357
13.2	eip_read_input_int().....	360

13.3	eip_read_input_float().....	363
13.4	eip_read_input_string().....	365
13.5	eip_read_input_bit().....	366
13.6	eip_read_output()	369
13.7	eip_read_output_int()	373
13.8	eip_read_output_float()	375
13.9	eip_read_output_string().....	377
13.10	eip_read_output_bit()	378
13.11	eip_write_output()	381
13.12	eip_write_output_bit()	389

Revision History Table

Revision	Date	Revised Content
01	July 2019	Original Release
02	February 2020	Minor context revision.
03	August 2020	Added the followings: ● File functions ● Safety SI / SO of IO Parameterization ● Mode 11/12/13 Request read of TM Ethernet Slave ● More examples for TMSVR response messages ● GetToken () support to get last token by count is -1 ● RefCoordF3D and RefCoordT3D in FT ● The unit of X Y Z RX RY RZ in the math function ● Profinet functions and modified: ● The unit of RX RY RZ in the motion functions such as PTP / Line / Move to ° ● The example description of modbus_read_string ● The description of TMSVR ● The example description of TMSVR binary transmission ● The modbus chapter and adjust it to the back of the motion command
04	October 2021	In addition to the previous software version, this version: ● added more descriptions of Retrieve a substring from input string with a prefix or a suffix. ● changed f to f0 in the note of changeref(). ● added the description of using numbers as constants. ● updated figures in Chapter 11.

1. Expression

1.1 Types

Different data types of variable can be declared in Variables Manager.

byte	8bit integer	unsigned	0 to 255	significant digit 3
int	32bit integer	signed	-2147483648 to 2147483647	significant digit 10
float	32bit floating-point	signed	-3.40282e+038f to 3.40282e+038f	significant digit 7
double	64bit floating-point	signed	-1.79769e+308 to 1.79769e+308	significant digit 15
bool	boolean		true or false	
string	string			

For int type variable, both int16 and int32 are supported. The default type is int 32.

int16	16bit integer	signed	-32768 to 32767	significant digit 5
int32	32bit integer	signed	-2147483648 to 2147483647	significant digit 10

1.2 Variables and Constants

1. Variables

In the naming rule of variables, only the numbers, under line and the upper case and lower case English characters are supported.

Numbers 0123456789

Characters a-z, A-Z, _

Example

```
Int var_i = 0
```

```
string var_s = "ABC"
```

```
string var_s1 = "DEF"
```

```
string var_s2 = "123"
```

Without double quotation marks, strings will be taken as variables.

```
var_s = var_s1 + " and " + var_s2      // var_s = "DEF and 123"
                                         // var_s, var_s1, var_s2 are variable, and " and " is a string.
```

In addition to variables, the naming rule also applies to constants, numbers, strings, and Booleans except that string constants need to be enclosed in double quotes.

When a variable is generated in TMflow, a prefix is added based on the source. To use the variable for writing or reading, users must enter the full name including the prefix word. For the rules of adding prefixes, refer to the respective description in variable setting pages.

2. Numbers

- Decimal integer, decimal floating-point, binary, hexadecimal integer and scientific notation are supported.

Decimal integer	123
	-123
	+456
Decimal float	34.567
	-8.9
Binary	0b0000111
	0B1110000
Hexadecimal integer	0x123abc
	0X00456DEF
Scientific notation	3.4e5
	2.3E-4

- For binary and hexadecimal notation, there is no floating-point.
- The notation of number is not case sensitive.

For example:

0b0011 equals to 0B0011
0xabcD equals to 0XABCD, 0xABCd, 0Xabcd etc.
3.4e5 equals to 3.4E5

- The system determines the data types of numbers automatically when using numbers as constants.

The rule is to conclude bit types from the smaller to the greater such as

100 // data type: byte // 100 is in the value range of data type: byte.
1000 // data type: int
1.11 // data type: float //1.11 is in the value range of data type: float.

To assign the data type, use variable declaration or conversion to do so such as

```
byte b = 100    // variable b = 100 as data type: byte
```

```
int i = 100     // variable i = 100 as data type: int
```

```
(int)100        // constant 100 as data type: int
```

```
(float)100       // constant 100 as data type: float
```

For function calls, the system determines the data types of arguments and selects the respective syntax. If there is no respective syntax, the system determines the compatible syntax with the rule concluding bit types from the smaller to the greater such as

```
GetBytes(100, 0, 0) // {0x64,0x00,0x00,0x00} // 100 and the 0s come with the data type: byte, but the
                    // syntax goes with GetBytes(int, int, int). The system, therefore, converts 100 and the
                    // 0s into the data type: int to proceed with the call syntax.
```

```
GetBytes(100)       // {0x64} // 100 comes with the data type: byte. in the syntax GetBytes(?) to go with
                    // any data types. Therefore, the system takes 100 as a constant in data type: byte to
                    // proceed the call syntax.
```

- The transforming between floating-point and byte array may cause discrepancy in value

For example:

```
float    5.317302E+030    ➔ float to byte[] {0x72,0x86,0x3A,0x42}
```

```
byte[]   {0x72,0x86,0x3A,0x43} ➔ byte[] to float 5.317302E+030
```

- Byte can only present unsigned numbers from 0 to 255. As a result, if negative number is assigned to byte type variable directly or through calculation, only 8 bit unsigned value will be kept.

For example:

```
byte var_b = -100    // var_b = 156 // -100 is present as 0xFFFFF9C by 16bit notation.
```

```
// Because byte can only keep 8 bit data, that is 0x9C (156), b will equals to 156
```

3. String

When inputting string constant, double quotation marks shall be placed in pairs around the string to avoid the recognition error of variable and string. °

For example

```
"Hello World! "
```

```
"Hello TM""5" (If " is one of the character in the string, use two ("") instead of one (").
```

- Control character in double quotation mark are not supported.

For example:

"Hello World!\r\n" (the output would be **Hello World!\r\n** string)

- Without double quotation marks, the compiling will follow the rules below
 1. Numbers will be viewed as numbers
 2. The combination of numbers and characters will be viewed as variable as long as the variable does exist.
 3. If the variable does not exist, it will be compiled as string with warning message.
- The combination of string and variable

1. Inside double quotation marks, variables will not be combined as variables

For example:

```
var_s = "TM5"           // var_s = "TM5"
var_s1 = "Hi, s Robot"  // var_s1 = "Hi, s Robot"
```

2. To input the combination of variables and strings, double quotation marks need to be placed around the string, and plus sign (+) shall be used to link variables and numbers

Example:

```
var_s1 = "Hi, " + var_s + " Robot"    // var_s1 = "Hi, TM5 Robot"
```

3. To be compatible with the old version software, the single quotation marks can be placed around the variables, but a warning message will be sent out

For example:

```
single quotation marks  "Hi, 's' Robot"    // var_s1 = "Hi, TM5 Robot"
"Hi, 'x' Robot"         // var_s1 = "Hi, 'x' Robot"    // Because variable x does not exist, 'x' is
                        viewed as string
```

4. Single quotation marks do not support element value retrieval with array indexes. The standard format with double quotation marks should be used.

For example

```
string[] var_ss = {"Techman", "Robot"}
"Hi, 'var_s' 'ss[0]' Robot"           // var_s1 = "Hi, TM5 'ss[0]' Robot" // 'ss[0]' is invalid
"Hi, " + var_s + " " + var_ss[0] + " Robot"    // var_s1 = "Hi, TM5 Techman Robot"
```

5. Single quotation marks cannot be presented by `'`. If users would like to input `'(variable name)'`, The standard format with double quotation marks should be used.

For example

```
"Hi, 's' Robot"           // var_s1 = "Hi, TM5 Robot"
```

// If var_s1 = "Hi, 's' Robot" is what you want, please use the following syntax.

```
"Hi, " + "s" + " Robot"    // var_s1 = "Hi, 's' Robot"
```

- For control character, e.g. new line, please use Ctrl() command.

For example

```
var_s1 = "Hi, " + Ctrl("\r\n") + s + " Robot"    or    "Hi, " + NewLine + var_s + " Robot"
```

Hi,

TM5 Robot

- Reserved characters is similar to variables, no double quotation marks is needed. (But single quotation mark is not supported)
 1. empty empty string, equals to ""
 2. newline or NewLine new line, equals to Ctrl("\r\n") or Ctrl(0x0D0A)

4. Boolean

True or false value of logic.

	true		false
Denote true value	True	Denote false value	False

The Boolean value is case sensitive. Misuses of capital letters such as TRue will be taken as a variable or a string.

1.3 Array

- Array is a set of data with the same data type. The initial value is assigned with {}, and every element remains the characteristic of its data type.

For example

```
int[] var_i = {0,1,2,3}           // elements in number data type
string[] var_s = {"ABC", "DEF", "GHI"} // elements in string data type
bool[] var_bb = {true, false, true} // elements in boolean data type
```

- By utilizing index, the value of specified element can be get, the index is start from 0

For example

index	0	1	2	3	4	5	6	7	
array									eight elements in total
	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	

Valid index values [0] .. [7], an error will occur with invalid index number.

- Only one degree array is supported. The maximum index number is 2048.
- The array size may alter according to the return value of functions or assigned values. The maximum element number is 2048. This feature makes array meet the needs of different functions and applications in Network Node.

For Example:

```
string[] var_ss = {empty, empty, empty}           // The initial size of string array is 3 elements
var_ss = String_Split("A_B_C_D_F_G_H", "_")      // After splitting string, the string array has 7 elements
var_len = Length(var_ss)                          // var_len = 7
var_ss = String_Split("A,B", ",")                // After splitting string, the string array has 2 elements
var_len = Length(var_ss)                          // var_len = 2
```

1.4 Operator Symbols

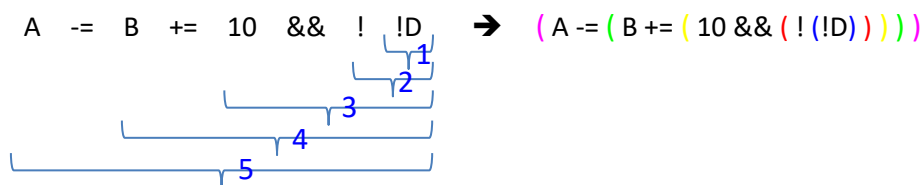
- The operator table is listed below.
- The calculation follows the precedence of operator first then the associativity.

For example

left-to-right associativity

$$A = A * B / C \% D \quad \rightarrow \quad (A = (((A * B) / C) \% D))$$


right-to-left associativity

$$A -= B += 10 \&\& ! !D \quad \rightarrow \quad (A -= (B += (10 \&\& (! (!D)))))$$


- The calculation will proceed by the type of the operand.
 1. When both values come as the integer type, the calculation will proceed by the integer type such as


```
int var_a = 10
int var_b = 3
float var_c = var_a / var_b
```

By the operator priority, the calculation goes / first and then =.

$var_a / var_b = 10 / 3 = 3$ (both var_a and var_b are integers)

$var_c = 3$ (The integer 3 assigns to the floating point number var_c)
 2. When one of the two values comes as the floating-point type, the calculation will proceed by the

floating-point type such as

```
int var_a = 10
float var_b = 3
float var_c = var_a / var_b
```

$var_a / var_b = 10 / 3 = 3.333333$ (for var_b is a floating-point number)

$var_c = 3.333333$ (the floating-point number 3.333333 assigns to var_c)

```
var_c = var_a / 3
```

$var_a / 3 = 10 / 3 = 3$ (both var_a and 3 are integers)

$var_c = 3$

```
var_c = var_a / 3.0
```

$var_a / 3.0 = 10 / 3.0 = 3.333333$ (for 3.0 is a floating-point number)

$var_c = 3.333333$

Precedence High to low	Operator	Name	Example	Requirement	associativity
17	++	Postfix increment	i++	Integer variable	left-to-right
	--	Postfix decrement	i--		
	()	Function call	int x = f()		
	[]	Allocate storage	array[4] = 2	Array variable	
16	++	Prefix increment	++i	Integer variable	right-to-left
	--	Prefix decrement	--i		
	+	Unary plus	int i = +1	Numeric variable, Constant	
	-	Unary minus	int i = -1		
	!	Logical negation (NOT)	if (!done) ...	Boolean	
	~	Bitwise NOT	flag1 = ~flag2	Integer	
14	*	Multiplication	int i = 2 * 4	Numeric variable, Constant	left-to-right
	/	Division	float f = 10.0 / 3.0		
	%	Modulo (integer)	int rem = 4 % 3		
13	+	Addition	int i = 2 + 3	Numeric variable, Constant	
	-	Subtraction	int i = 5 - 1		
12	<<	Bitwise left shift	int flags = 33 << 1	Integer variable, Constant	
	>>	Bitwise right shift	int flags = 33 >> 1		
11	<	Less than	if (i < 42) ...	Numeric variable, Constant	
	<=	Less than or equal to	if (i <= 42) ...		
	>	Greater than	if (i > 42) ...		
	>=	Greater than or equal to	if (i >= 42) ...		
10	==	Equal to	if (i == 42) ...		
	!=	Not equal to	if (i != 42) ...		
9	&	Bitwise AND	flag1 = flag2 & 42	Integer variable, Constant	
8	^	Bitwise XOR	flag1 = flag2 ^ 42		
7		Bitwise OR	flag1 = flag2 42		
6	&&	Logical AND	if (conditionA && conditionB)		
5		Logical OR	if (conditionA conditionB)		
4	c ? t : f	Ternary conditional	int i = a > b ? a : b		right-to-left
3	=	Basic assignment	int a = b	Left side: Numeric	
	+=	Addition assignment	a += 3		
	-=	Subtraction assignment	b -= 4		

	*=	Multiplication assignment	a *= 5	variable	
	/=	Division assignment	a /= 2	Right side:	
	%=	Modulo assignment	a %= 3	Numeric variable, Constant	
	<<=	Bitwise left shift assignment	flags <<= 2	Left side: Integer variable	
	>>=	Bitwise right shift assignment	flags >>= 2	Right side: Integer variable	
	&=	Bitwise AND assignment	flags &= new_flags	variable, Constant	
	^=	Bitwise XOR assignment	flags ^= new_flags		
	=	Bitwise OR assignment	flags = new_flags		

1.5 Data type conversion

- Data types can be converted to each other and used in variables/constants or arrays.
- Conversions must be in the same format of the containers such as variable/constant conversions or array conversions. It is not permitted to convert a variable to an array or an array to a variable.

Native type	Conversion type	Example	Result
byte	int	int i = (int)100	i = 100
	float	float f = (float)100	f = 100
	double	double d = (double)100	d = 100
	bool	bool flag = (bool)0	flag = true (not null)
	string	string s = (string)100	s = "100"
int	byte	byte b = (byte)1000	b = 232
	float	float f = (float)1000	f = 1000
	double	double d = (double)1000	d = 1000
	bool	bool flag = (bool)1000	flag = true (not null)
	string	string s = (string)1000	s = "1000"
float	byte	byte b = (byte)1.23	b = 1
	int	int i = (int)1.23	i = 1
	double	double d = (double)1.23	d = 1.23
	bool	bool flag = (bool)1.23	flag = true
	string	string s = (string)1.23	s = "1.23"
double	byte	byte b = (byte)1.23	b = 1
	int	int i = (int)1.23	i = 1
	float	float f = (float)1.23	f = 1.23

	bool	bool flag = (bool)1.23	flag = true
	string	string s = (string)1.23	s = "1.23"
bool	byte	byte b = (byte)True	Error
	int	int i = (int)False	Error
	float	float f = (float>true	Error
	double	double d = (double>false	Error
	string	string s = (string)True	s = "True"
string	byte	byte b1 = (byte)"1.23" byte b2 = (byte)"XYZ"	1 Error
	int	int i = (int)"1.23"	1
	float	float f1 = (float)"1.23" float f2 = (float)"XYZ"	1.23 Error
	double	double d = (double)"1.23"	1.23
	bool	bool flag1 = (bool)"1.23" bool flag2 = (bool)""	flag1 = true (not null) flag2 = false (null)

- The conversion method of arrays is in accordance with the table above. The conversion is performed for each element in the array.

```
string[] var_ss = {"1.23", "4.56", "0.789"}
```

```
int[] var_i_array = (int[])var_ss // var_i_array = {1, 4, 0}
```

```
float[] var_f_array = (float[])var_ss // var_f_array = {1.23, 4.56, 0.789}
```

- Error messages will be returned should the conversions below occur.
 - Fail to convert to numeric correctly such as Booleans (true/false) or non-numeric strings ("XYZ").

```
int var_value = (int>true // Error
```

```
int var_value = (int)"XYZ" // Error
```

- Invalid floating-point numbers to convert to floats or doubles such as NaN or Infinity.

```
string var_dvalue = "1.79769e+308"
```

```
float var_f = (float) var_dvalue // Error 1.79769e+308 is a valid double type and unable to convert to the float type.
```

1.6 Warning

A warning message will prompt, under the condition listed below.

- Double quotation marks does not placed around the string constant.

```
string var_s = Hello // warningHello
```

- There is single quotation mark inside the string constant.

```
string var_s0 = "World"
string var_s1 = "Hello 's0'" // warning's0'
```

- When assigning float value to integer constant, some digits may get lost such as

```
int var_i = 1.234 // warning var_i = 1
float var_f = 1.234
var_i = var_f // warning var_i = 1
```

- When assigning value to variables with fewer digits, some digits may get lost such as

```
byte var_b = 100
int var_i = 1000
float var_f = 1.234
double var_d = 2.345
var_b = var_i // warning var_b = 232// byte can contain values from 0 to 255
var_f = var_d // warning var_f = 2.345
```

- When assigning string value to numeric variable, a conversion from string to number will be applied. If the conversion is executable, a warning message will prompt, or the project will be stopped by error such as

```
int var_i = "1234" // warning var_i = 1234
int var_j = "0x89AB" // warning var_j = 35243
int var_k = "0b1010" // warning var_k = 10
string var_s1 = 123 // warning var_s1 = 123 // Number to string
string var_s2 = "123"
int var_x = var_s2 // warning // string to number
// The following code can be compiled with warning, but will be stopped by error when executing.
var_s2 = "XYZ"
var_x = var_s2 // warning // Stop executing by error
// var_s = "XYZ" cannot be converted to number

var_s2 = ""
var_x = var_s2 // warning // Stop executing by error
// var_s = "" cannot be converted to number
```

- String parameters are used as numeric parameters in functions such as

```
Ctrl(0x0A0B0C0D0E) // warning // 0x0A0B0C0D0E is not int type (over 32bit)
// Because there is another syntax, Ctrl(string), the parameter
would be applied to Ctrl(string)
```

Although the project can still be executed with a warning message, correcting all the errors in a warning message is highly recommended to eliminate unpredictable problems and prevent the project being stopped by

errors.

- How to fix the error messages

1. Use double quotations with the string constants

```
string var_s = "Hello"
```

2. Use + to link the string constant and the string variable

```
string var_s0 = "World"
```

```
string var_s1 = "Hello " + var_s0
```

3. Specify the type clearly for numerical conversions

```
float var_f = 1.234
```

```
int var_i = (int) var_f    // Use (int) for type conversion, var_i = 1 while processing // It turns the  
                           number in floating-point to an integer.
```

2. Functions

2.1 Byte_ToInt16()

Transform the first two bytes of the assigned byte array to integer, and returns in int type.

Syntax 1

```
int Byte_ToInt16(  
    byte[],  
    int,  
    int  
)
```

Parameters

<code>byte[]</code>	Byte array
<code>int</code>	Byte array follows the Little Endian or Big Endian
0	Little Endian (Default)
1	Big Endian
<code>int</code>	Transfer to signed int16 (Signed Number) or unsigned int16 (Unsigned Number)
0	signed int16 (Default)
1	unsigned int16

Return

`int` A signed or unsigned int16 formed by 2 bytes beginning at index [0].
Because only 2 bytes is needed, the index of byte array will be [0][1]. If the data is not long enough, it would be filled to 2 bytes before transforming.

Note

```
byte[] var_bb1 = {0x90, 0x01, 0x05}
```

```
byte[] var_bb2 = {0x01} // Cause var_bb2[] does not fill 2 bytes. It would be filled to 2 bytes before transforming.
```

```
var_value = Byte_ToInt16(var_bb1, 0, 0) // 0x0190 var_value = 400  
var_value = Byte_ToInt16(var_bb1, 0, 1) // 0x0190 var_value = 400  
var_value = Byte_ToInt16(var_bb1, 1, 0) // 0x9001 var_value = -28671  
var_value = Byte_ToInt16(var_bb1, 1, 1) // 0x9001 var_value = 36865  
var_value = Byte_ToInt16(var_bb2, 0, 0) // 0x0001 var_value = 1  
var_value = Byte_ToInt16(var_bb2, 0, 1) // 0x0001 var_value = 1  
var_value = Byte_ToInt16(var_bb2, 1, 0) // 0x0100 var_value = 256  
var_value = Byte_ToInt16(var_bb2, 1, 1) // 0x0100 var_value = 256
```

Syntax 2

```
int Byte_ToInt16 (
    byte[],
    int
)
```

Note

Similar to Syntax 1 with return value as signed int16

Byte_ToInt16(var_bb1, 0) => **Byte_ToInt16**(var_bb1, 0, 0)

Syntax 3

```
int Byte_ToInt16 (
    byte[]
)
```

Note

Similar to Syntax 1 with little endian input and return value as signed int16

Byte_ToInt16(var_bb1) => **Byte_ToInt16**(var_bb1, 0)

2.2 Byte_ToInt32()

Transform the first four bytes of byte array to integer, and return in int type.

Syntax 1

```
int Byte_ToInt32 (
    byte[],
    int
)
```

Parameters

byte[] The input byte array

int The input byte array follows Little Endian or Big Endian

- 0** Little Endian (Default)
- 1** Big Endian

Return

int An unsigned int32 formed by 4 bytes beginning at index [0].

Because only 4 bytes is needed, the index of byte array will be [0][1][2][3]. If the data is not long enough, it would be filled to 4 bytes before transforming.

Note

```
byte[] var_bb1 = {0x01, 0x02, 0x03, 0x4F, 1}
byte[] var_bb2 = {0x01, 0x02, 0x03}
// Cause var_bb2[] does not fill 4 bytes. It would be filled to 4 bytes before
transforming.
var_value = Byte_ToInt32(var_bb1, 0) // 0x4F030201 var_value = 1325597185
var_value = Byte_ToInt32(var_bb1, 1) // 0x0102034F var_value = 16909135
var_value = Byte_ToInt32(var_bb2, 0) // 0x00030201 var_value = 197121
var_value = Byte_ToInt32(var_bb2, 1) // 0x01020300 var_value = 16909056
```

Syntax 2

```
int Byte_ToInt32 (
    byte[]
)
```

Note

Similar to Syntax 1 with little endian input

Byte_ToInt32(var_bb1) => **Byte_ToInt32**(var_bb1, 0)

2.3 Byte_ToFloat()

Transform the first four bytes of byte array to floating-point number, and return in floating-point type.

Syntax 1

```
float Byte_ToFloat (
    byte[],
    int
)
```

Parameters

byte[] The input byte array

int The input byte array follows Little Endian or Big Endian

- 0 Little Endian (Default)
- 1 Big Endian

Return

float A floating-point number formed by 4 bytes beginning at index [0].
Because only 4 bytes is needed, the index of byte array will be [0][1][2][3]. If the data is not long enough, it would be filled to 4 bytes before transforming.

Note

```
byte[] var_bb1 = {0x01, 0x02, 0x03, 0x4F, 1}
byte[] var_bb2 = {0x01, 0x02, 0x03} // Cause bb2[] does not fill 4 bytes. It would be filled to 4 bytes before
                                     transforming.

var_value = Byte_ToFloat(var_bb1, 0) // 0x4F030201 var_value = 2.197947E+09
var_value = Byte_ToFloat(var_bb1, 1) // 0x0102034F var_value = 2.38796E-38
var_value = Byte_ToFloat(var_bb2, 0) // 0x00030201 var_value = 2.762254E-40
var_value = Byte_ToFloat(var_bb2, 1) // 0x01020300 var_value = 2.387938E-38
```

Syntax 2

```
float Byte_ToFloat (
    byte[]
)
```

Note

Similar to Syntax 1 with little endian input

Byte_ToFloat(var_bb1) => Byte_ToFloat(var_bb1, 0)

2.4 Byte_ToDouble()

Transform the first eight bytes of byte array to floating-point number, and return in double type.

Syntax 1

```
double Byte_ToDouble (
    byte[],
    int
)
```

Parameters

byte[] The input byte array

int The input byte array follows Little Endian or Big Endian

- 0** Little Endian (Default)
- 1** Big Endian

Return

double A floating-point number formed by 8 bytes beginning at index [0].

Because only 8 bytes is needed, the index of byte array will be [0][1][2][3][4][5][6][7]. If the data is not long enough, it would be filled to 8 bytes before transforming.

Note

```

byte[] var_bb1 = {0x01, 0x02, 0x03, 0x4F, 1}    // Cause bb1[] does not fill 8 bytes. It would be filled to 8
                                                bytes before transforming.
byte[] var_bb2 = {0x01, 0x02, 0x03}    // Cause bb1[] does not fill 8 bytes. It would be filled to 8 bytes before
                                                transforming.
var_value = Byte_ToDouble(var_bb1, 0) // 0x000000014F030201 var_value = 2.77692782029764E-314
var_value = Byte_ToDouble(var_bb1, 1) // 0x0102034F01000000 var_value = 8.20840179153173E-304
var_value = Byte_ToDouble(var_bb2, 0) // 0x00000000000030201 var_value = 9.73907141738724E-319
var_value = Byte_ToDouble(var_bb2, 1) // 0x0102030000000000 var_value = 8.20785244926136E-304

```

Syntax 2

```

double Byte_ToDouble (
    byte[]
)

```

Note

Similar to Syntax 1 with little endian input

Byte_ToDouble(var_bb1) => Byte_ToDouble(var_bb1, 0)

2.5 Byte_ToInt16Array()

Transform byte array to integer every 2 bytes, and return in int[] type.

Syntax 1

```

int[] Byte_ToInt16Array (
    byte[],
    int,
    int
)

```

Parameters

byte[]	The input byte array
int	The input byte array follows Little Endian or Big Endian
0	Little Endian (Default)
1	Big Endian
int	Transfer to signed int16 (Signed Number) or unsigned int16 (Unsigned Number)
0	signed int16 (Default)
1	unsigned int16

Return

int[] A integer array formed by every 2 bytes of byte array beginning at index [0]

Note

byte[] var_bb1 = {0x90, 0x01, 0x02, 0x03, 0x04} // When the remaining part does not fill 2 byte, it would be filled to 2 bytes before transforming.

byte[] var_bb2 = {1, 2, 3, 4}

var_value = **Byte_ToInt16Array**(var_bb1, 0, 0) // {0x0190, 0x0302, 0x0004} var_value = {400, 770, 4}

var_value = **Byte_ToInt16Array**(var_bb1, 0, 1) // {0x0190, 0x0302, 0x0004} var_value = {400, 770, 4}

var_value = **Byte_ToInt16Array**(var_bb1, 1, 0) // {0x9001, 0x0203, 0x0400} var_value = {-28671, 515, 1024}

var_value = **Byte_ToInt16Array**(var_bb1, 1, 1) // {0x9001, 0x0203, 0x0400} var_value = {36865, 515, 1024}

var_value = **Byte_ToInt16Array**(var_bb2, 0, 0) // {0x0201, 0x0403} var_value = {513, 1027}

var_value = **Byte_ToInt16Array**(var_bb2, 0, 1) // {0x0201, 0x0403} var_value = {513, 1027}

var_value = **Byte_ToInt16Array**(var_bb2, 1, 0) // {0x0102, 0x0304} var_value = {258, 772}

var_value = **Byte_ToInt16Array**(var_bb2, 1, 1) // {0x0102, 0x0304} var_value = {258, 772}

Syntax 2

```
int[] Byte_ToInt16Array(  
    byte[],  
    int  
)
```

Note

Similar to Syntax 1 with return value as signed int16

Byte_ToInt16Array(var_bb1, 0) => **Byte_ToInt16Array**(var_bb1, 0, 0)

Syntax 3

```
int[] Byte_ToInt16Array(  
    byte[]  
)
```

Note

Similar to Syntax 1 with little endian input and return value as signed int16

Byte_ToInt16Array(var_bb1) => **Byte_ToInt16Array**(var_bb1, 0)

2.6 Byte_ToInt32Array()

Transform byte array to integer every 4 bytes, and return in int[] type

Syntax 1

```
int[] Byte_ToInt32Array(  
    byte[],  
    int  
)
```

Parameters

`byte[]` The input byte array

`int` The input byte array follows Little Endian or Big Endian

- 0 Little Endian (Default)
- 1 Big Endian

Return

`int[]` A integer array formed by every 4 bytes of byte array beginning at index [0]

Note

`byte[] var_bb1 = {0x01, 0x02, 0x03, 0x04, 0x05}` // When the remaining part does not fill 4 byte, it would be filled to 4 bytes before transforming.

`byte[] var_bb2 = {1, 2, 3, 4}`

`var_value = Byte_ToInt32Array(var_bb1, 0)` // {0x04030201, 0x00000005} `var_value = {67305985, 5}`

`var_value = Byte_ToInt32Array(var_bb1, 1)` // {0x01020304, 0x05000000} `var_value = {16909060, 83886080}`

`var_value = Byte_ToInt32Array(var_bb2, 0)` // {0x04030201} `var_value = {67305985}`

`var_value = Byte_ToInt32Array(var_bb2, 1)` // {0x01020304} `var_value = {16909060}`

Syntax 2

```
int[] Byte_ToInt32Array(  
    byte[]  
)
```

Note

Similar to Syntax 1 with little endian input.

`Byte_ToInt32Array(var_bb1) => Byte_ToInt32Array(var_bb1, 0)`

2.7 Byte_ToFloatArray()

Transform byte array to integer every 4 bytes, and return in `float[]` type.

Syntax 1

```
float[] Byte_ToFloatArray(  

```

```

    byte[],
    int
)

```

Parameters

`byte[]` The input byte array

`int` The input byte array follows Little Endian or Big Endian

- 0 Little Endian (Default)
- 1 Big Endian

Return

`float[]` A floating-point number array formed by every 4 bytes of byte array beginning at index [0]

Note

```

byte[] var_bb1 = {0x01, 0x02, 0x03, 0x04, 0x05}

// When the remaining part does not fill 4 byte, it would be filled to 4 bytes before transforming.

byte[] var_bb2 = {1, 2, 3, 4}

var_value = Byte_ToFloatArray(var_bb1, 0)
// {0x04030201, 0x00000005} var_value = {1.53999E-36, 7.006492E-45}
var_value = Byte_ToFloatArray(var_bb1, 1)
// {0x01020304, 0x05000000} var_value = {2.387939E-38, 6.018531E-36}
var_value = Byte_ToFloatArray(var_bb2, 0) // {0x04030201} var_value = {1.53999E-36}
var_value = Byte_ToFloatArray(var_bb2, 1) // {0x01020304} var_value = {2.387939E-38}

```

Syntax 2

```

float[] Byte_ToFloatArray (
    byte[]
)

```

Note

Similar to Syntax 1 with little endian input

```
Byte_ToFloatArray(var_bb1) => Byte_ToFloatArray(var_bb1, 0)
```

2.8 Byte_ToDoubleArray()

Transform byte array to double every 8 bytes, and return in double[] type.

Syntax 1

```
double[] Byte_ToDoubleArray (
```

```

    byte[],
    int
)

```

Parameters

`byte[]` The input byte array

`int` The input byte array follows Little Endian or Big Endian

- 0 Little Endian (Default)
- 1 Big Endian

Return

`double[]` A floating-point number array formed by every 8 bytes of byte array beginning at index [0]

Note

`byte[] var_bb1 = {0x01, 0x02, 0x03, 0x04, 0x05}` // When the remaining part does not fill 8 byte, it would be filled to 8 bytes before transforming.

`byte[] var_bb2 = {1, 2, 3, 4}` // When the remaining part does not fill 8 byte, it would be filled to 8 bytes before transforming.

```

var_value = Byte_ToDoubleArray(var_bb1, 0) // {0x0000000504030201} var_value = {1.06432325297744E-313}
var_value = Byte_ToDoubleArray(var_bb1, 1) // {0x0102030405000000} var_value = {8.20788039849233E-304}
var_value = Byte_ToDoubleArray(var_bb2, 0) // {0x0000000004030201} var_value = {3.32535749480063E-316}
var_value = Byte_ToDoubleArray(var_bb2, 1) // {0x0102030400000000} var_value = {8.2078802626846E-304}

```

Syntax 2

```

double[] Byte_ToDoubleArray (
    byte[]
)

```

Note

Similar to Syntax 1 with little endian input

`Byte_ToDoubleArray(var_bb1) => Byte_ToDoubleArray(var_bb1, 0)`

2.9 Byte_ToString()

Transform byte array to string

Syntax 1

```

string Byte_ToString (
    byte[],

```

```
int
```

```
)
```

Parameters

`byte[]` The input byte array

`int` The character encoding rules applied to input byte array

0 UTF8 (Default) (0x00 END)

1 HEX BINARY

2 ASCII (0x00 END)

Return

`string` String formed by byte array. The transformation begins from index [0].

Note

```
byte[] var_bb1 = {0x31, 0x32, 0x33, 0x00, 0x4F, 1}
```

```
byte[] var_bb2 = {0x01, 0x54, 0x4D, 0x35, 0xE6, 0xA9, 0x9F, 0xE5, 0x99, 0xA8, 0xE4, 0xBA, 0xBA}
```

```
var_value = Byte_ToString(var_bb1, 0) // var_value = "123" (UTF8 stop at 0x00)
```

```
var_value = Byte_ToString(var_bb1, 1) // var_value = "313233004F01"
```

```
var_value = Byte_ToString(var_bb1, 2) // var_value = "123" (ASCII stop at 0x00)
```

```
var_value = Byte_ToString(var_bb2, 0) // var_value = "\u01TM5機器人" (UTF8)
```

```
var_value = Byte_ToString(var_bb2, 1) // var_value = "01544D35E6A99FE599A8E4BABA"
```

```
var_value = Byte_ToString(var_bb2, 2) // var_value = "\u01TM5?????????" (ASCII)
```

* `\u01` represents the SOH control character, not the string value.

Syntax 2

```
string Byte_ToString(  
    byte[]  
)
```

Note

Similar to Syntax 1 with UTF8 character encoding rules

```
Byte_ToString(var_bb1) => Byte_ToString(var_bb1, 0)
```

2.10 Byte_Concat()

Concatenate two byte arrays, or concatenate one array with a byte value.

Syntax 1

```
byte[] Byte_Concat(  
    byte[],
```

```
byte
```

```
)
```

Parameters

`byte[]` The input byte array

`byte` The byte value concatenated after the byte array

Return

`byte[]` The byte array formed by the input byte array and byte value

Note

```
byte[] var_bb1 = {0x31, 0x32, 0x33, 0x00, 0x4F, 1}
```

```
var_value = Byte_Concat(var_bb1, 12) // var_value = {0x31, 0x32, 0x33, 0x00, 0x4F, 0x01, 0x0C}
```

Syntax 2

```
byte[] Byte_Concat (  
    byte[] ,  
    byte[]  
)
```

Parameters

`byte[]` The input byte array1

`byte[]` The input byte array2, would be concatenated to the end of array1

Return

`byte[]` Byte array formed from concatenating input arrays.

Note

```
byte[] var_bb1 = {0x31, 0x32, 0x33, 0x00, 0x4F, 1}
```

```
byte[] var_bb2 = {0x01, 0x02, 0x03}
```

```
var_value = Byte_Concat(var_bb1, var_bb2)  
// var_value = {0x31, 0x32, 0x33, 0x00, 0x4F, 0x01, 0x01, 0x02, 0x03}
```

Syntax 3

```
byte[] Byte_Concat (  
    byte[] ,  
    byte[] ,  
    int  
)
```

Parameters

<code>byte[]</code>	The input byte array1
<code>byte[]</code>	The input byte array2, would be concatenated after the end of array1
<code>int</code>	The number of element in array2 to be concatenated
<code>0..the length of array2</code>	Valid number
<code><0</code>	Invalid. Length of array2 will be applied instead.
<code>> the length of array2</code>	Invalid. Length of array2 will be applied instead.

Return

`byte[]` Byte array formed from concatenating input arrays.

Note

```
byte[] var_bb1 = {0x31, 0x32, 0x33, 0x00, 0x4F, 1}
```

```
byte[] var_bb2 = {0x01, 0x02, 0x03}
```

```
var_value = Byte_Concat(var_bb1, var_bb2, 2) // var_value = {0x31, 0x32, 0x33, 0x00, 0x4F, 0x01, 0x01, 0x02} //
```

Concatenate only 2 elements from array2

[illegible][illegible]

```
// Length() can be utilized to acquire the array size
```

```
var_value = Byte_Concat(var_bb1, var_bb2, Length(var_bb2))  
// var_value = {0x31, 0x32, 0x33, 0x00, 0x4F, 0x01, 0x01, 0x02, 0x03}
```

Syntax 4

```
byte[] Byte_Concat(  
    byte[],  
    int,  
    int,  
    byte[],  
    int,  
    int  
)
```

Parameters

`byte[]` The input byte array1

```
int
```

 The starting index of array1

0..(length of array1)-1 Valid

<0 The starting index would be 0

`>=(length of array1)` The starting index would be the length of array2 (For index over the length of array2, an empty value would be captured)

`int` The number of element in array1 to be concatenated
`0.. (length of array1)` Valid
`<0` Invalid · length of array1 will be applied instead
`>(length of array1)` Invalid · length of array1 will be applied instead

If the total number of starting index and assigning elements exceeds the length of array1, the surplus index will be suspended.

`byte[]` The input byte array2 · would be concatenated after the end of array1

`int` The starting index of array2
`0.. (length of array2)-1` Valid
`<0` The starting index would be 0
`>=(length of array2)` The starting index would be the length of array2 (For index over the length of array2, an empty value would be captured)

`int` The number of element in array2 to be concatenated
`0.. (length of array2)` Valid
`<0` Invalid. Length of array2 will be applied instead.
`>(length of array2)` Invalid. Length of array2 will be applied instead.

If the total number of starting index and assigning elements exceeds the length of array2, the surplus index will be suspended.

Return

`byte[]` Byte array formed from concatenating input arrays.

Note

`byte[] var_bb1 = {0x31, 0x32, 0x33, 0x00, 0x4F, 1}`

`byte[] var_bb2 = {0x01, 0x02, 0x03}`

`var_value = Byte_Concat(var_bb1, 1, 3, var_bb2, 1, 2)` `// var_value = {0x32, 0x33, 0x00, 0x02, 0x03}`

`var_value = Byte_Concat(var_bb1, -1, 3, var_bb2, 3, -1)` `// var_value = {0x31, 0x32, 0x33}`

2.11 String_ToInteger()

Transform string to integer (int type)

Syntax 1

```
int String_ToInteger (
    string,
    int
```

)

Parameters

`string` The input string.

`int` The input string's notation is decimal, hexadecimal or binary

- `10` decimal or auto format detecting (Default)
- `16` hexadecimal
- `2` binary

String's notation

- `123` decimal
- `0x7F` hexadecimal
- `0b101` binary

Return

`int` The integer value formed from input string. If notation is invalid, returns 0.

Note

```
var_value = String_ToInteger("1234", 10)    // var_value = 1234
var_value = String_ToInteger("1234", 16)    // var_value = 4660
var_value = String_ToInteger("1234", 2)     // var_value = 0      // Invalid binary format
var_value = String_ToInteger("1100", 2)     // var_value = 12
var_value = String_ToInteger("0x1234", 10)  // var_value = 4660    // Hexadecimal format by auto
                                           // detecting
var_value = String_ToInteger("0x1234", 16)  // var_value = 4660
var_value = String_ToInteger("0x1234", 2)   // var_value = 0      // Invalid binary format
var_value = String_ToInteger("0b1100", 10)  // var_value = 12     // Binary format by auto detecting
var_value = String_ToInteger("0b1100", 16)  // var_value = 725248 // Valid Hexadecimal number
var_value = String_ToInteger("0b1100", 2)   // var_value = 12
var_value = String_ToInteger("+1234", 10)   // var_value = 1234
var_value = String_ToInteger("-1234", 10)   // var_value = -1234
var_value = String_ToInteger("-0x1234", 16) // var_value = 0      // Invalid hex format
var_value = String_ToInteger("-0b1100", 2)  // var_value = 0      // Invalid binary format
```

Syntax 2

```
int String_ToInteger (
    string
)
```

Note

Similar to syntax1 with decimal format or auto format detection

String_ToInteger(str) => String_ToInteger(str, 10)

Syntax 3

```
int[] String_ToInteger(  
    string[],  
    int  
)
```

Parameters

<code>string[]</code>	Input string array
<code>int</code>	The notation of element in input string array is decimal, hexadecimal or binary
10	decimal or auto format detecting (Default)
16	hexadecimal
2	binary
String's notation	
123	decimal
0x7F	hexadecimal
0b101	binary
* The notations of all the elements in a single array have to be identical	

Return

`int[]` The integer array formed from input string array. If notation is invalid, returns 0.

Note

```
var_ss = {"12", "ab", "cc", "dd", "10"}  
var_value = String_ToInteger(var_ss) // var_value = {12, 0, 0, 0, 10} // "ab","cc","dd" are invalid decimal  
                                         numbers  
var_value = String_ToInteger(var_ss, 2) // var_value = {0, 0, 0, 0, 2} // "12","ab","cc","dd" are invalid binary  
                                         numbers  
var_value = String_ToInteger(var_ss, 16) // var_value = {18, 171, 204, 221, 16}  
var_value = String_ToInteger(var_ss, 10) // var_value = {12, 0, 0, 0, 10} // "ab","cc","dd" are invalid decimal  
                                         numbers
```

2.12 String_ToFloat()

Transform string to floating-point (floating-point type)

Syntax 1

```
float String_ToFloat(  
    string,  
    int  
)
```

Parameters

`string` Input string

`int` Input string's notation is decimal, hexadecimal or binary format

- `10` decimal or auto format detecting (Default)
- `16` hexadecimal
- `2` binary

String's notation

- `123` decimal
- `0x7F` hexadecimal
- `0b101` binary

Return

`float` The floating-point number formed from input string. If notation is invalid, returns 0.

Note

```
var_value = String_ToFloat("12.34", 10) // var_value = 12.34
var_value = String_ToFloat("12.34", 16) // var_value = 0 // Invalid hexadecimal format
var_value = String_ToFloat("12.34", 2) // var_value = 0 // Invalid binary format
var_value = String_ToFloat("11.00", 2) // var_value = 0 // Invalid binary format
var_value = String_ToFloat("0x1234", 10) // var_value = 6.530051E-42 // Hexadecimal format by auto
                                         detecting
var_value = String_ToFloat("0x1234", 16) // var_value = 6.530051E-42
var_value = String_ToFloat("0x1234", 2) // var_value = 0 // Invalid binary format
var_value = String_ToFloat("0b1100", 10) // var_value = 1.681558E-44 // Binary format by auto
                                         detecting
var_value = String_ToFloat("0b1100", 16) // var_value = 1.016289E-39 // Valid hexadecimal format
var_value = String_ToFloat("0b1100", 2) // var_value = 1.681558E-44
var_value = String_ToFloat("+12.34", 10) // var_value = 12.34
var_value = String_ToFloat("-12.34", 10) // var_value = -12.34
var_value = String_ToFloat("-0x1234", 16) // var_value = 0 // Invalid hex format
var_value = String_ToFloat("-0b1100", 2) // var_value = 0 // Invalid format
```

Syntax 2

```
float String_ToFloat (
    string
)
```

Note

Similar to syntax1 with decimal format or auto format detection

String_ToFloat(str) => String_ToFloat(str, 10)

Syntax 3

```
float[] String_ToFloat(
    string[],
    int
)
```

Parameters

<code>string[]</code>	Input string array
<code>int</code>	The notation of elements in input string array is decimal, hexadecimal or binary
<code>10</code>	decimal or auto format detecting (Default)
<code>16</code>	hexadecimal
<code>2</code>	binary
	String's notation
<code>123</code>	decimal
<code>0x7F</code>	hexadecimal
<code>0b101</code>	binary
	* The notation of all the elements in a single array have to be identical

Return

`float[]` The floating-point number array formed from input string array. If notation is invalid, returns 0.

Note

```
var_ss = {"12.345", "ab", "cc", "dd", "10.111"}
var_value = String_ToFloat(var_ss)           // var_value = {12.345, 0, 0, 0, 10.111}
var_value = String_ToFloat(var_ss, 2)        // var_value = {0, 0, 0, 0, 0}
var_value = String_ToFloat(var_ss, 16)       // var_value = {0, 2.39622E-43, 2.858649E-43, 3.09687E-43, 0}
var_value = String_ToFloat(var_ss, 10)       // var_value = {12.345, 0, 0, 0, 10.111}
```

2.13 String_ToDouble()

Transform string to floating-point number (double type)

Syntax 1

```
double String_ToDouble(
    string,
    int
)
```

Parameters

<code>string</code>	Input string
<code>int</code>	Input string's notation is decimal, hexadecimal or binary format
<code>10</code>	decimal or auto format detecting (Default)

16 hexadecimal
 2 binary
 String's notation
 123 decimal
 0x7F hexadecimal
 0b101 binary

Return

`double` The floating-point number formed from input string. If notation is invalid, returns 0.

Note

```
var_value = String_ToDouble("12.34", 10)    // var_value = 12.34
var_value = String_ToDouble("12.34", 16)    // var_value = 0      // Invalid hexadecimal format
var_value = String_ToDouble("12.34", 2)     // var_value = 0      // Invalid binary format
var_value = String_ToDouble("11.00", 2)     // var_value = 0      // Invalid binary format
var_value = String_ToDouble("0x1234", 10)   // var_value = 2.30234590962021E-320 // Hexadecimal
                                           // format by auto detecting
var_value = String_ToDouble("0x1234", 16)   // var_value = 2.30234590962021E-320
var_value = String_ToDouble("0x1234", 2)    // var_value = 0      // Invalid binary format
var_value = String_ToDouble("0b1100", 10)   // var_value = 5.92878775009496E-323 // Binary format by
                                           // auto detecting
var_value = String_ToDouble("0b1100", 16)   // var_value = 3.58320121515072E-318 // Valid hexadecimal
                                           // format
var_value = String_ToDouble("0b1100", 2)    // var_value = 5.92878775009496E-323
var_value = String_ToDouble("+12.34", 10)    // var_value = 12.34
var_value = String_ToDouble("-12.34", 10)    // var_value = -12.34
var_value = String_ToDouble("-0x1234", 16)   // var_value = 0      // Invalid hex format
var_value = String_ToDouble("-0b1100", 2)    // var_value = 0      // Invalid binary format
```

Syntax 2

```
double String_ToDouble (
    string
)
```

Note

Similar to syntax1 with decimal format or auto format detection

`String_ToDouble(str) => String_ToDouble(str, 10)`

Syntax 3

```
double[] String_ToDouble (
    string[],
```

```
int
```

```
)
```

Parameters

`string[]` Input string array

`int` The notation of elements in input string array is decimal, hexadecimal or binary

`10` decimal or auto format detecting (Default)

`16` hexadecimal

`2` binary

String's notation

`123` decimal

`0x7F` hexadecimal

`0b101` binary

* The notation of all the elements in a single array has to be identical

Return

`double[]` The floating-point number array formed from input string array. If notation is invalid, returns 0.

Note

```
var_ss = {"12.345", "ab", "cc", "dd", "10.111"}
```

```
var_value = String_ToDouble(var_ss) // var_value = {12.345, 0, 0, 0, 10.111}
```

```
var_value = String_ToDouble(var_ss, 2) // var_value = {0, 0, 0, 0, 0}
```

```
var_value = String_ToDouble(var_ss, 16) // var_value = {0, 8.44852254388532E-322, 1.00789391751614E-321, 1.09188507730915E-321, 0}
```

```
var_value = String_ToDouble(var_ss, 10) // var_value = {12.345, 0, 0, 0, 10.111}
```

2.14 String_ToByte()

Transform string to byte array

Syntax 1

```
byte[] String_ToByte (  
    string,  
    int  
)
```

Parameters

`string` Input string

`int` The character encoding rules applied to input string

`0` UTF8 (Default)

`1` HEX BINARY // Stop at invalid Hex value

Return

`byte[]` The byte array formed from input string

Note

```
var_value = String_ToByte("12345", 0)           // var_value = {0x31, 0x32, 0x33, 0x34, 0x35}
var_value = String_ToByte("12345", 1)           // var_value = {0x12, 0x34, 0x50} // the insufficient part will be
                                                // filled with 0
var_value = String_ToByte("12345", 2)           // var_value = {0x31, 0x32, 0x33, 0x34, 0x35}
var_value = String_ToByte("0x12345", 0)          // var_value = {0x30, 0x78, 0x31, 0x32, 0x33, 0x34, 0x35}
var_value = String_ToByte("0x12345", 1)          // var_value = {0x00}           // Only 0 be transformed, cause x is an
                                                // invalid Hex value
var_value = String_ToByte("0x12345", 2)          // var_value = {0x30, 0x78, 0x31, 0x32, 0x33, 0x34, 0x35}

var_value = String_ToByte("TM5\\u4eba", 0)        // var_value = {0x54, 0x4D, 0x35, 0xE6, 0xA9, 0x9F, 0xE5, 0x99, 0xA8, 0xE4,
                                                // 0xBA, 0xBA}
var_value = String_ToByte("TM5\\u4eba", 1)        // var_value = {0x00}           // T is an invalid Hex value
var_value = String_ToByte("TM5\\u4eba", 2)        // var_value = {0x54, 0x4D, 0x35, 0x3F, 0x3F, 0x3F}
var_value = String_ToByte("0123456", 1)          // var_value = {0x01, 0x23, 0x45, 0x60}
var_value = String_ToByte("01234G5", 1)          // var_value = {0x01, 0x23, 0x40} // G is an invalid Hex value
```

Syntax 2

```
byte[] String_ToByte (
    string
)
```

Note

Similar to syntax1 with UTF8 format

`String_ToByte(str) => String_ToByte(str, 0)`

2.15 String_IndexOf()

Report the zero-based index of the first occurrence of a specified string

Syntax 1

```
int String_IndexOf (
    string,
    string
```

)

Parameters

`string` Input string

`string` The specified string to be searched. The zero-based index of the first occurrence is to be found.

Return

`int` `0..(Length of string)-1` If the specified string is found, returns the index number

`-1` Not found

`0` The specified string is "" or empty

Note

```
var_value = String_IndexOf("012314", "1") // var_value = 1
```

```
var_value = String_IndexOf("012314", "") // var_value = 0
```

```
var_value = String_IndexOf("012314", empty) // var_value = 0
```

```
var_value = String_IndexOf("012314", "d") // var_value = -1
```

```
var_value = String_IndexOf("", "d") // var_value = -1
```

2.16 String_LastIndexOf()

Report the zero-based index position of the last occurrence of a specified string

Syntax 1

```
int String_LastIndexOf (  
    string,  
    string  
)
```

Parameters

`string` Input string

`string` The specified string to be searched. The zero-based index of the last occurrence is to be found.

Return

`int` `0..(Length of string)-1` If the specified string is found, returns the index number

`-1` Not found

`0` The specified string is "" or empty

Note

```
var_value = String_LastIndexOf("012314", "1") // var_value = 4
```

```
var_value = String_LastIndexOf("012314", "") // var_value = 5
```

```
var_value = String_LastIndexOf("012314", empty) // var_value = 5
```

```
var_value = String_LastIndexOf("012314", "d") // var_value = -1
```

```
var_value = String_LastIndexOf("", "d")
```

```
// var_value = -1
```

2.17 String_Substring()

Retrieve a substring from input string

Syntax 1

```
string String_Substring(  
    string,  
    int,  
    int  
)
```

Parameters

string Input string

int The starting character position of sub string (0 .. (length of input string)-1)

int The length of substring

Return

string Substring

If **starting character position < 0**, returns empty string

If **starting character position >= length of input string**, returns empty string

If **length of substring < 0**, the substring ends at the last character of the input string

If the sum of starting character position and length of substring exceeds the length of input string, the substring ends at the last character of the input string

Note

```
var_value = String_Substring("0x12345", 2, 4) // var_value = "1234"
```

```
var_value = String_Substring("0x12345", -1, 4) // var_value = ""
```

```
var_value = String_Substring("0x12345", 7, 4) // var_value = ""
```

```
var_value = String_Substring("0x12345", 2, -1) // var_value = "12345"
```

```
var_value = String_Substring("0x12345", 2, 100) // var_value = "12345"
```

Syntax 2

```
string String_Substring(  
    string,  
    int  
)
```

Note

Similar to syntax1 with the substring ends at the last character of the input string

String_Substring(str, 2) => String_Substring(str, 2, maxlen)

Syntax 3

```
string String_Substring(  
    string,  
    string,  
    int  
)
```

Parameters

string Input string

string The target string to be searched, the substring will start at its position, if it is found

int The length of substring

Return

string Substring

If **the target string is empty**, the substring start at index zero

If **the target string is not found**, returns empty string

If **length of substring < 0**, the substring ends at the last character of the input string

If the sum of starting character position and length of substring exceeds the length of input string, the substring ends at the last character of the input string

Note

This syntax is the same as **String_Substring(str, String_IndexOf(str1), int)**

```
var_value = String_Substring("0x12345", "1", 4)      // var_value = "1234"  
var_value = String_Substring("0x12345", "", 4)       // var_value = "0x12"  
var_value = String_Substring("0x12345", "7", 4)      // var_value = ""  
var_value = String_Substring("0x12345", "1", -1)     // var_value = "12345"  
var_value = String_Substring("0x12345", "1", 100)    // var_value = "12345"
```

Syntax 4

```
string String_Substring(  
    string,  
    string  
)
```

Note

Similar to Syntax 3 with the substring ends at the last character of the input string

String_Substring(str, "1") => String_Substring(str, "1", maxlen)

Syntax 5

```
string String_Substring(
    string,
    string,
    string,
    int
)
```

Parameters

string Input string

string Prefix. The leading element of the substring

string Suffix. The trailing element of the substring

int The number of occurrence

Return

string Substring

If **prefix and suffix are empty string**, returns input string

If **the number of occurrence<=0**, returns empty string

Note

<code>var_value = String_Substring("0x12345", "", "", 0)</code>	<code>// var_value = "0x12345"</code>
<code>var_value = String_Substring("0x12345", "1", "4", 1)</code>	<code>// var_value = "1234"</code>
<code>var_value = String_Substring("0x12345", "1", "4", 2)</code>	<code>// var_value = ""</code>
<code>var_value = String_Substring("0x12345", "1", "4", 0)</code>	<code>// var_value = ""</code>
<code>var_value = String_Substring("0x123450x12-345", "1", "4", 1)</code>	<code>// var_value = "1234"</code>
<code>var_value = String_Substring("0x123450x12-345", "1", "4", 2)</code>	<code>// var_value = "12-34"</code>
<code>var_value = String_Substring("0x123450x12-345", "1", "4", 3)</code>	<code>// var_value = ""</code>
<code>var_value = String_Substring("0x12345122", "1", "", 1)</code>	<code>// var_value = "12345122" // All the trailing characters after the prefix</code>
<code>var_value = String_Substring("0x12345122", "1", "", 2)</code>	<code>// var_value = "122" // All the trailing characters after the 2nd occurred prefix</code>
<code>var_value = String_Substring("0x12345122", "1", "", 4)</code>	<code>// var_value = ""</code>
<code>var_value = String_Substring("0x12345433", "", "4", 1)</code>	<code>// var_value = "0x123454" // All the leading characters before the suffix</code>
<code>var_value = String_Substring("0x12345433", "", "4", 2)</code>	<code>// var_value = "0x1234" // All the leading characters before the 2nd occurred suffix</code>

Syntax 6

```
string String_Substring(
    string,
    string,
    string
)
```

Note

Similar to Syntax 5 with the substring start at the first occurrence

String_Substring(str, prefix, suffix) => **String_Substring**(str, prefix, suffix, 1)

2.18 String_Split()

Split the string using specified separator.

Syntax 1

```
string[] String_Split(  
    string,  
    string,  
    int  
)
```

Parameters

string Input string

string Separator (String)

int Format

- 0 Split and keep the empty strings
- 1 Split and eliminate the empty strings
- 2 Split with the elements inside double quotation mark skipped, and keep the empty strings
- 3 Split with the elements inside double quotation mark skipped, and eliminate the empty strings

Return

string[] Split substring

If input string is empty, return substring have only one element. [0] = empty

If separator is empty, return substring have only one element. [0] = Input string

Note

```
var_value = String_Split("0x112345", "1", 0)    // var_value = {"0x", "", "2345"}  
var_value = String_Split("0x112345", "", 0)     // var_value = {"0x112345"}  
var_value = String_Split("0x112345", "1", 1)     // var_value = {"0x", "2345"}  
var_s1 = "123, ""456,67"" ,89"  
var_value = String_Split(var_s1, ",", 0)         // var_value = {"123", ""456", "67""", "89"} // length = 4  
var_value = String_Split(var_s1, ",", 2)         // var_value = {"123", ""456,67""", "89"} // length = 3
```

Syntax 2

```
string[] String_Split(  

```

```
    string,  
    string  
)
```

Note

Similar to `Syntax1` with splitting and keeping the empty strings

`String_Split(str, separator) => String_Split(str, separator, 0)`

2.19 String_Replace()

Return a new string in which all occurrences of a specified string in the input string are replaced with another specified string

Syntax 1

```
string String_Replace(  
    string,  
    string,  
    string  
)
```

Parameters

`string` Input string

`string` Old value, the string to be replaced

`string` New value, the string to replace all occurrences of old value

Return

`string` The string formed by replacing the old value with new value in input value. If the old value is empty, returns the input string

Note

```
var_value = String_Replace("0x112345", "1", "2")    // var_value = "0x222345"  
var_value = String_Replace("0x112345", "", "2")    // var_value = "0x112345"  
var_value = String_Replace("0x112345", "1", "")    // var_value = "0x2345"
```

2.20 String_Trim()

Return a new string in which all leading and trailing occurrences of specified characters or white-space characters from the input string are removed

Syntax 1

```
string String_Trim(  
    string
```

```
string
```

```
)
```

Parameters

`string` Input string

Return

`string` String formed by removing all leading and trailing occurrences of white-space characters

Note

```
var_value = String_Trim("0x112345 ") // var_value = "0x112345"
```

```
var_value = String_Trim(" 0x112345") // var_value = "0x112345"
```

```
var_value = String_Trim(" 0x112345 ") // var_value = "0x112345"
```

White-space characters

\u0020	\u1680	\u2000	\u2001	\u2002	\u2003	\u2004
\u2005	\u2006	\u2007	\u2008	\u2009	\u200A	\u202F
\u205F	\u3000					
\u2028						
\u2029						
\u0009	\u000A	\u000B	\u000C	\u000D	\u0085	\u00A0
\u200B	\uFEFF					

Syntax 2

```
string String_Trim(  
    string,  
    string  
)
```

Parameters

`string` Input string

`string` Specified characters to be removed from leading occurrences

Return

`string` String formed by removing all leading occurrences of specified characters

Syntax 3

```
string String_Trim(  
    string,  
    string,  
    string
```

)

Parameters

`string` Input string

`string` Specified characters to be removed from leading occurrences

`string` Specified characters to be removed from trailing occurrences

Return

`string` String formed by removing all leading and trailing occurrences of the specified characters

Note

```
var_string s1 = "Hello  Hello  World  Hello  World"
var_string s2 = "HelloHelloWorldHelloWorld"
var_value = String_Trim(var_s1, "Hello")           // var_value = "  Hello  World  Hello  World"
var_value = String_Trim(var_s1, "World")           // var_value = "Hello  Hello  World  Hello  World"
var_value = String_Trim(var_s1, "", "Hello")       // var_value = "Hello  Hello  World  Hello  World"
var_value = String_Trim(var_s1, "", "World")       // var_value = "Hello  Hello  World  Hello  "
var_value = String_Trim(var_s1, "Hello", "World")  // var_value = "  Hello  World  Hello  "
var_value = String_Trim(var_s2, "Hello")           // var_value = "WorldHelloWorld"
var_value = String_Trim(var_s2, "World")           // var_value = "HelloHelloWorldHelloWorld"
var_value = String_Trim(var_s2, "", "Hello")       // var_value = "HelloHelloWorldHelloWorld"
var_value = String_Trim(var_s2, "", "World")       // var_value = "HelloHelloWorldHello"
var_value = String_Trim(var_s2, "Hello", "World")  // var_value = "WorldHello"
```

2.21 String_ToLower()

Change all the characters in a string to lower case

Syntax 1

```
string String_ToLower(
    string
)
```

Parameters

`string` Input string

Return

`string` The string formed by converting all the English character into lower case. Non-English character will be remained the same.

Note

```
var_value = String_ToLower("0x11Acz34") // var_value = "0x11acz34"
```

2.22 String_ToUpper()

Change all the characters in a string to upper case

Syntax 1

```
string String_ToUpper(  
    string  
)
```

Parameters

`string` Input string

Return

`string` The string formed by converting all the English character into upper case. Non-English character will remain the same.

Note

```
var_value = String_ToUpper("0x11Acz34") // var_value = "0X11ACZ34"
```

2.23 Array_Append()

Add new data as the elements in the end of the array.

Syntax 1

```
?[] Array_Append(  
    ?[],  
    ? or ?[]  
)
```

Parameters

`[]` Parameter 1, the array to be appended. Available types: byte, int, float, double, bool, and string.

`? or ?[]` Parameter 2, the data or the array to add. The type must be the same with the type of the array to be appended.

*Both parameters must go with the same type.

Return

`[]` The new array with the parameter 2 elements appended to the parameter 1.

Note

```
? byte[] var_n1 = {100, 200, 30}
```

```
byte[] var_n2 = {40, 50, 60}
var_n3 = Array_Append(var_n1, var_n2) // var_n3 = {100, 200, 30, 40, 50, 60}
var_n1 = Array_Append(var_n1, 100) // var_n1 = {100, 200, 30, 100}
var_n1 = Array_Append(var_n1, var_n3) // var_n1 = {100, 200, 30, 100, 100, 200, 30, 40, 50, 60}
```

? float[] var_n1 = {1.1, 2.2, 3.3}

```
float[] var_n2 = {0.4, 0.5}
```

```
var_n3 = Array_Append(var_n1, var_n2) // var_n3 = {1.1, 2.2, 3.3, 0.4, 0.5}
var_n4 = Array_Append(var_n3, 5.678) // var_n4 = {1.1, 2.2, 3.3, 0.4, 0.5, 5.678}
```

? string[] var_n1 = {"123", "ABC", "456", "DEF"}

```
string[] var_n2 = {"ABC", "123", "XYZ"}
```

```
var_n3 = Array_Append(var_n1, var_n2) // var_n3 = {"123", "ABC", "456", "DEF", "ABC", "123", "XYZ"}
var_n4 = Array_Append(var_n2, "Hello World") // var_n4 = {"ABC", "123", "XYZ", "Hello World"}
```

2.24 Array_Insert()

Insert data as the elements in the array.

Syntax1

```
?[] Array_Insert (
    [],
    int,
    ? or ?[]
)
```

Parameters

?[]	Parameter 1, the array to be inserted. Available types: byte, int, float, double, bool, and string.
int	The index starting address of the parameter 1.
0	The length of the array 1 - 1 Legal value
>=	The length of the array 1 Legal value, and will insert the value in the end of the parameter 1.
< 0	Illegal value, the project will stop by error.
? or ?[]	Parameter 2, the data or the array to insert. The type must be the same with the type of the array to be appended.
* Both parameters must go with the same type.	

Return

? [] The new array with the parameter 2 elements inserted to the index starting address of the parameter 1.

Note

```
? int[] var_n1 = {100, 200, 30}
int[] var_n2 = {40, 50, 60}
var_n3 = Array_Insert(var_n1, 0, var_n2)    // var_n3 = {40, 50, 60, 100, 200, 30}
                                           // Insert to the index 0
var_n4 = Array_Insert(var_n1, 2, var_n2)    // var_n4 = {100, 200, 40, 50, 60, 30}
                                           // Insert to the index 2
var_n5 = Array_Insert(var_n1, -1, var_n2)   // var_n5 = {}
                                           // The project will stop by error. Illegal index to start with

? double[] var_n1 = {1.4, 2.6, 3.9}
double[] var_n2 = {0.5, 0.7}
var_n3 = Array_Insert(var_n1, 1, var_n2)   // var_n3 = {1.4, 0.5, 0.7, 2.6, 3.9}
var_n4 = Array_Insert(var_n3, 4, 1.2345)   // var_n4 = {1.4, 0.5, 0.7, 2.6, 1.2345, 3.9}
var_n5 = Array_Insert(var_n3, 100, 9)      // var_n5 = {1.4, 0.5, 0.7, 2.6, 3.9, 9}
                                           // Out of the index. The value will insert in the end of the array.
```

2.25 Array_Remove()

Delete data as the elements in the array.

Syntax1

```
? [] Array_Remove (
    ? [],
    int,
    int
)
```

Parameters

? []	Parameter 1, the array to be inserted. Available types: byte, int, float, double, bool, and string.
int	The index starting address of the parameter 1 to remove.
0	The length of the parameter 1 - 1 Legal value
>=	The length of the parameter 1 Illegal value, the project will stop by error.
< 0	Illegal value, the project will stop by error.

<code>int</code>	The number of the elements to remove
<code>> 0</code>	The number of the elements to remove from the index starting address or until the end of the array.
<code>< 0</code>	The number will be 0 and no element will be removed.

Return

<code>? []</code>	The new array with elements removed after the index starting address.
-------------------	---

Syntax2

```
? [] Array_Remove (
    ? [],
    int
)
```

Note

Same as syntax 1. The default number of the elements to remove is 1.

```
? int[] var_n1 = {100, 200, 30, 40, 50, 60}
var_n3 = Array_Remove(var_n1, -1)    // var_n3 = {}
                                     // The project will stop by error. Illegal value to start with.
var_n4 = Array_Remove(var_n1, 100)   // var_n4 = {}
                                     // The project will stop by error. Illegal value to start with.
var_n5 = Array_Remove(var_n1, 0)     // var_n5 = {200, 30, 40, 50, 60}    // Remove index 0
var_n6 = Array_Remove(var_n1, 1, 2)  // var_n6 = {100, 40, 50, 60}
                                     // Remove 2 elements from index 1
var_n7 = Array_Remove(var_n1, 1, 100) // var_n7 = {100}
                                     // Remove 100 elements from index 1 (remove to the end of the
                                     array)
var_n8 = Array_Remove(var_n1, Length(var_n1)-1) // var_n8 = {100, 200, 30, 40, 50}
                                               // Remove from the last of index
var_n9 = Array_Remove(var_n1, Length(var_n1))  // var_n9 = {}
                                               // The project will stop by error. Illegal value to start with.
```

2.26 Array_Equals()

Determine whether the specified two arrays are identical.

Syntax 1

```
bool Array_Equals (
    ? [],
```

```
?[]
```

```
)
```

Parameters

`?[]` Input array1 (Data type can be byte, int, float, double, bool, string)

`?[]` Input array2 (Data type can be byte, int, float, double, bool, string)

* The data type of array1 and array2 must be identical.

Return

`bool` Two arrays are identical or not?

`true` two arrays are identical

`false` two arrays are not identical

Syntax 2

```
bool Array_Equals (
```

```
    ?[], vv
```

```
    int,
```

```
    ?[],
```

```
    int,
```

```
    int
```

```
)
```

Parameters

`?[]` Input array1 (Data type can be byte, int, float, double, bool, string)

`int` The starting index of array1 (0 .. (length of array1)-1)

`?[]` Input array2 (Data type can be byte, int, float, double, bool, string)

`int` The starting index of array2 (0 .. (length of array2)-1)

`int` The number of elements to be compared (0: return true)

* The data type of array1 and array2 must be identical.

Return

`bool` The assigned elements in two arrays are identical or not?

`true` identical

`false` not identical (or parameters are not valid)

Note

```
? byte[] var_n1 = {100, 200, 30}
```

```
byte[] var_n2 = {100, 200, 30}
```

```
Array_Equals(var_n1, var_n2) // true
```

```
Array_Equals(var_n1, 0, var_n2, 0, 3) // true
```

```
Array_Equals(var_n1, 0, var_n2, 0, Length(var_n2)) // true
```

```

?   int[] var_n1 = {1000, 2000, 3000}

    int[] var_n2 = {1000, 2000, 3000, 4000}

    Array_Equals(var_n1, var_n2)                // false
    Array_Equals(var_n1, 0, var_n2, 0, Length(var_n2)) // false // compare 4 elements
    Array_Equals(var_n1, 0, var_n2, 0, 3)         // true

?   float[] var_n1 = {1.1, 2.2, 3.3}

    float[] var_n2 = {1.1, 2.2}

    Array_Equals(var_n1, var_n2)                // false
    Array_Equals(var_n1, 0, var_n2, 0, Length(var_n2)) // true // compare 2 elements
    Array_Equals(var_n1, 0, var_n2, 0, Length(var_n1)) // false

?   double[] var_n1 = {100, 200, 300, 3.3, 2.2, 1.1}

    double[] var_n2 = {100, 200, 400, 3.3, 2.2, 4.4}

    Array_Equals(var_n1, var_n2)                // false
    Array_Equals(var_n1, 0, var_n2, 0, Length(var_n2)) // false
    Array_Equals(var_n1, 0, var_n2, 0, 2)         // true
    Array_Equals(var_n1, 3, var_n2, 3, 2)         // true

?   bool[] var_n1 = {true, false, true, true, true}

    bool[] var_n2 = {true, false, true, false, true}

    Array_Equals(var_n1, var_n2)                // false
    Array_Equals(var_n1, 0, var_n2, 0, -1)       // false
    Array_Equals(var_n1, 0, var_n2, 0, 0)         // true // compare 0 element

?   string[] var_n1 = {"123", "ABC", "456", "DEF"}

    string[] var_n2 = {"123", "ABC", "456", "DEF"}

    Array_Equals(var_n1, var_n2)                // true
    Array_Equals(var_n1, -1, var_n2, 0, 4)       // false // Invalid starting index

```

2.27 Array_IndexOf()

Search for the specified element and returns the index of its first occurrence in the input array

Syntax 1

```

int Array_IndexOf (
    ?[],
    ?
)

```

Parameters

? [] input array (Data type can be byte, int, float, double, bool, string)

? The target element to search (The data type needs to be the same as the input array ? [], but not an array)

Return

int 0 .. (length of input array)-1 If the element is found , returns the index value

-1 No element found

Note

```
? byte[] var_n = {100, 200, 30}
var_value = Array.IndexOf(var_n, 200)           // 1
var_value = Array.IndexOf(var_n, 2000)          // error // 2000 is not byte data

? int[] var_n = {1000, 2000, 3000}
var_value = Array.IndexOf(var_n, 200)           // -1

? float[] var_n = {1.1, 2.2, 3.3}
var_value = Array.IndexOf(var_n, 1.1)           // 0

? double[] var_n = {100, 200, 300, 3.3, 2.2, 1.1}
var_value = Array.IndexOf(var_n, 1.1)           // 5

? bool[] var_n = {true, false, true, true, true}
var_value = Array.IndexOf(var_n, true)           // 0

? string[] var_n = {"123", "ABC", "456", "DEF"}
var_value = Array.IndexOf(var_n, "456")         // 2
```

2.28 Array_LastIndexOf()

Search for the specified element and returns the index of the last occurrence within the entire Array.

Syntax 1

```
int Array_LastIndexOf (
    ? [],
    ?
)
```

Parameters

? [] input array (Data type can be byte, int, float, double, bool, string)

? The target element to search (The data type needs to be the same as the input array ? [], but not an array)

Return

<code>int</code>	<code>0..(length of input array)-1</code>	If the element is found , returns the index value
	<code>-1</code>	No element found

Note

```
? byte[] var_n = {100, 200, 30}
var_value = Array_LastIndexOf(var_n, 200)           // 1
var_value = Array_LastIndexOf(var_n, 2000)          // error // 2000 is not byte data

? int[] var_n = {1000, 2000, 3000}
var_value = Array_LastIndexOf(var_n, 200)           // -1

? float[] var_n = {1.1, 2.2, 3.3}
var_value = Array_LastIndexOf(var_n, 1.1)           // 0

? double[] var_n = {100, 200, 300, 3.3, 2.2, 1.1}
var_value = Array_LastIndexOf(var_n, 1.1)           // 5

? bool[] var_n = {true, false, true, true, true}
var_value = Array_LastIndexOf(var_n, true)          // 4

? string[] var_n = {"123", "ABC", "456", "DEF"}
var_value = Array_LastIndexOf(var_n, "456")         // 2
```

2.29 Array_Reverse()

Reverse the sequence of the elements in the array

Syntax 1

```
? [] Array_Reverse (
    ? []
)
```

Parameters

`?` [] input array (Data type can be byte, int, float, double, bool, string)

Return

`?` [] The reversed array

Note

```
? byte[] var_n = {100, 200, 30}
var_n = Array_Reverse(var_n)           // var_n = {30, 200, 100}

? int[] n = {1000, 2000, 3000}
var_n = Array_Reverse(var_n)           // var_n = {3000, 2000, 1000}

? float[] var_n = {1.1, 2.2, 3.3}
var_n = Array_Reverse(var_n)           // var_n = {3.3, 2.2, 1.1}
```

```
? double[] var_n = {100, 200, 300, 3.3, 2.2, 1.1}
var_n = Array_Reverse(var_n) // var_n = {1.1, 2.2, 3.3, 300, 200, 100}

? bool[] var_n = {true, false, true, true, true}
var_n = Array_Reverse(var_n) // var_n = {true, true, true, false, true}

? string[] var_n = {"123", "ABC", "456", "DEF"}
var_n = Array_Reverse(var_n) // var_n = {"DEF", "456", "ABC", "123"}
```

Syntax 2

```
? [] Array_Reverse (
    ? [],
    int
)
```

Parameters

? [] input array (Data type can be byte, int, float, double, bool, string)

int the number of elements to be viewed as a section to be reversed

2 2 elements as a section

4 4 elements as a section

8 8 elements as a section

* The sequence of the elements in the same section will be reversed, but the sequence of the sections will remain the same

Return

? [] The reversed array

Note

```
? byte[] var_n = {100, 200, 30}
var_n = Array_Reverse(var_n, 2) // var_n = {200, 100, 30}
// 2 elements as a section, that is {100,200}{30}

var_n = Array_Reverse(var_n, 4) // var_n = {30, 200, 100}
// 4 elements as a section, that is {100,200,30}

var_n = Array_Reverse(var_n, 8) // var_n = {30, 200, 100}

? int[] var_n = {100, 200, 300, 400}
var_n = Array_Reverse(var_n, 2) // var_n = {200, 100, 400, 300}
// 2 elements as a section, that is {100,200}{300,400}

var_n = Array_Reverse(var_n, 4) // var_n = {400, 300, 200, 100}
// 4 elements as a section, that is {100,200,300,400}

var_n = Array_Reverse(var_n, 8) // var_n = {400, 300, 200, 100}

? float[] var_n = {1.1, 2.2, 3.3, 4.4, 5.5}
```

```

var_n = Array_Reverse(var_n, 2) // var_n = {2.2, 1.1, 4.4, 3.3, 5.5}
                                   // 2 elements as a section, that is {1.1,2.2}{3.3,4.4}{5.5}
var_n = Array_Reverse(var_n, 4) // var_n = {4.4, 3.3, 2.2, 1.1, 5.5}
                                   // 4 elements as a section, that is {1.1,2.2,3.3,4.4}{5.5}
var_n = Array_Reverse(var_n, 8) // var_n = {5.5, 4.4, 3.3, 2.2, 1.1}
? double[] var_n = {100, 200, 300, 400, 4.4, 3.3, 2.2, 1.1, 50, 60, 70, 80}
var_n = Array_Reverse(var_n, 2) // var_n = {200, 100, 400, 300, 3.3, 4.4, 1.1, 2.2, 60, 50, 80, 70}
var_n = Array_Reverse(var_n, 4) // var_n = {400, 300, 200, 100, 1.1, 2.2, 3.3, 4.4, 80, 70, 60, 50}
var_n = Array_Reverse(var_n, 8) // var_n = {1.1, 2.2, 3.3, 4.4, 400, 300, 200, 100, 80, 70, 60, 50}
? bool[] var_n = {true, false, true, true, true, false, true, false}
var_n = Array_Reverse(var_n, 2) // var_n = {false, true, true, true, false, true, false, true}
var_n = Array_Reverse(var_n, 4) // var_n = {true, true, false, true, false, true, false, true}
var_n = Array_Reverse(var_n, 8) // var_n = {false, true, false, true, true, true, false, true}
? string[] var_n = {"123", "ABC", "456", "DEF", "000", "111"}
var_n = Array_Reverse(var_n, 2) // var_n = {"ABC", "123", "DEF", "456", "111", "000"}
var_n = Array_Reverse(var_n, 4) // var_n = {"DEF", "456", "ABC", "123", "111", "000"}
var_n = Array_Reverse(var_n, 8) // var_n = {"111", "000", "DEF", "456", "ABC", "123"}

```

2.30 Array_Sort()

Sort the elements in a array

Syntax 1

```

?[] Array_Sort (
    ?[],
    int
)

```

Parameters

?[]	input array (Data type can be byte, int, float, double, bool, string)
int	Sorting direction
0	Ascending Order (Default)
1	Descending Order

Return

?[]	The array after sorting
-----	-------------------------

Syntax 2

```

?[] Array_Sort (

```

```
? []
```

```
)
```

Note

Similar to Syntax1 with sorting direction as ascending order

Array_Sort(array[]) => **Array_Sort**(array[], 0)

```
? int[] var_n = {1000, 2000, 3000}
```

```
var_n = Array_Sort(var_n) // var_n = {1000, 2000, 3000}
```

```
? double[] var_n = {100, 200, 300, 3.3, 2.2, 1.1}
```

```
var_n = Array_Sort(var_n, 1) // var_n = {300, 200, 100, 3.3, 2.2, 1.1}
```

```
? bool[] var_n = {true, false, true, true, true}
```

```
var_n = Array_Sort(var_n, 1) // var_n = {true, true, true, true, false}
```

```
? string[] var_n = {"123", "ABC", "456", "DEF"}
```

```
var_n = Array_Sort(var_n) // var_n = {"123", "456", "ABC", "DEF"}
```

2.31 Array_SubElements()

Retrieve the sub-elements from input array

Syntax 1

```
? [] Array_SubElements (  
    ? [],  
    int,  
    int  
)
```

Parameters

? [] Input array (Data type can be byte, int, float, double, bool, string)

int The starting index of sub-elements. (0 .. (length of array)-1)

int The number of element in sub-elements

Return

? [] The sub-elements from input arrays

If **starting index < 0**, sub-elements equals to empty array

If **starting index >= length of input array**, sub-elements equals to empty array

If **sub-element number < 0**, sub-elements starts at starting index to the last element of input array

If the sum of starting index and the number of element exceeds the length of the input array, sub-elements starts at starting index to the last element of input array

Syntax 2

```
?[] Array_SubElements(  
    [],  
    int  
)
```

Note

Similar to Syntax 1, but the sub-elements starts at starting index to the last element of input array

Array_SubElements(array[], 2) => Array_SubElements(array[], 2, maxlen)

```
? byte[] var_n = {100, 200, 30}  
  
var_n1 = Array_SubElements(var_n, 0)    // var_n1 = {100, 200, 30}  
var_n1 = Array_SubElements(var_n, -1)   // var_n1 = {}  
var_n1 = Array_SubElements(var_n, 0, 3) // var_n1 = {100, 200, 30}  
var_n1 = Array_SubElements(var_n, 1, 3) // var_n1 = {200, 30}  
var_n1 = Array_SubElements(var_n, 2)    // var_n1 = {30}  
var_n1 = Array_SubElements(var_n, 3, 3) // var_n1 = {}  
  
? int[] var_n = {1000, 2000, 3000}  
  
var_n1 = Array_SubElements(var_n, 0)    // var_n1 = {1000, 2000, 3000}  
var_n1 = Array_SubElements(var_n, -1)   // var_n1 = {}  
var_n1 = Array_SubElements(var_n, 1, 3) // var_n1 = {2000, 3000}  
var_n1 = Array_SubElements(var_n, 2)    // var_n1 = {3000}  
  
? float[] var_n = {1.1, 2.2, 3.3}  
  
var_n1 = Array_SubElements(var_n, 0)    // var_n1 = {1.1, 2.2, 3.3}  
var_n1 = Array_SubElements(var_n, -1)   // var_n1 = {}  
var_n1 = Array_SubElements(var_n, 1, 3) // var_n1 = {2.2, 3.3}  
var_n1 = Array_SubElements(var_n, 2)    // var_n1 = {3.3}  
  
? double[] var_n = {100, 200, 3.3, 2.2, 1.1}  
  
var_n1 = Array_SubElements(var_n, 0)    // var_n1 = {100, 200, 3.3, 2.2, 1.1}  
var_n1 = Array_SubElements(var_n, -1)   // var_n1 = {}  
var_n1 = Array_SubElements(var_n, 1, 3) // var_n1 = {200, 3.3, 2.2}  
var_n1 = Array_SubElements(var_n, 2)    // var_n1 = {3.3, 2.2, 1.1}  
  
? bool[] var_n = {true, false, true, true, true}  
  
var_n1 = Array_SubElements(var_n, 0)    // var_n1 = {true, false, true, true, true}  
var_n1 = Array_SubElements(var_n, -1)   // var_n1 = {}  
var_n1 = Array_SubElements(var_n, 1, 3) // var_n1 = {false, true, true}  
var_n1 = Array_SubElements(var_n, 2)    // var_n1 = {true, true, true}  
  
? string[] var_n = {"123", "ABC", "456", "DEF"}  
  
var_n1 = Array_SubElements(var_n, 0)    // var_n1 = {"123", "ABC", "456", "DEF"}  
var_n1 = Array_SubElements(var_n, -1)   // var_n1 = {}
```

```
var_n1 = Array_SubElements(var_n1, 1, 3)    // var_n1 = {"ABC", "456", "DEF"}
var_n1 = Array_SubElements(var_n1, 2)      // var_n1 = {"456", "DEF"}
```

2.32 ValueReverse()

Reverse the sequence of byte units inside input data (int 2 bytes or 4 bytes, float 4 bytes, double 8 bytes); or reverse the sequence of character of string.

Syntax 1

```
int ValueReverse (
    int,
    int
)
```

Parameters

int Input value

int The input value follows int32 or int16 format

- 0 int32 (Default)
- 1 int16. If the data does not meets int16 format, int32 will be applied instead.
- 2 int16. Forced to apply int16 format. For int32 data input, there could be some bytes missing

Return

int The value formed from reversing the sequence of byte units inside the input value. For Int32 data, reverse with 4 bytes. For int16 data, reverse with 2 bytes.

Note

```
int var_i = 10000
var_value = ValueReverse(var_i, 0)  // 10000=0x00002710 → 0x10270000 // var_value = 270991360
var_value = ValueReverse(var_i, 1)  // 10000=0x2710 → 0x1027 // var_value = 4135
var_i = 100000                      // int32 value
var_value = ValueReverse(var_i, 0)  // 100000=0x000186A0 → 0xA0860100 // var_value = -1601830656
var_value = ValueReverse(var_i, 1)  // 100000=0x000186A0 → 0xA0860100 // var_value = -1601830656
var_value = ValueReverse(var_i, 2)  // 100000=0x000086A0 → 0xA0860000 // var_value = -24442
```

Syntax 2

```
int ValueReverse (
    int
)
```

Parameters

`int` Input value

Note

Similar to Syntax1 with int32 input format

ValueReverse(int) => **ValueReverse**(int, 0)

Syntax 3

```
float ValueReverse (  
    float  
)
```

Parameters

`float` Input value

Return

`float` The value formed from reversing the sequence of byte units inside the input value. For float data, reverse 4 bytes.

Note

`float var_i = 40000`

`var_value = ValueReverse(var_i)                      // 40000=0x471C4000 → 0x00401C47 // var_value = 5.887616E-39`

Syntax 4

```
double ValueReverse (  
    double  
)
```

Parameters

`double` Input value

Return

`double` The value formed from reversing the sequence of byte units inside the input value. For double data, reverse 8 bytes.

Note

`double var_i = 80000`

`var_value = ValueReverse(var_i)                      // 80000=0x40F3880000000000 → 0x000000000088F340 // var_value = 4.43432217445369E-317`

Syntax 5

```
string ValueReverse (  
    string  
)
```

Parameters

`string` Input string

Return

`string` The value formed from reversing the sequence of characters of input string.

Note

```
string var_i = "ABCDEF"
var_value = ValueReverse(var_i)    // var_value = "FEDCBA"
```

Syntax 6

```
int[] ValueReverse (
    int[],
    int
)
```

Parameters

`int[]` Input array value

`int` The input value follows int32 or int16 format

0 int32 (Default)

1 int16. If the data does not meets int16 format, int32 will be applied instead.

2 int16. Forced to apply int16 format. For int32 data input, there could be some bytes missing

Return

`int[]` The array formed from reversing the sequence of byte units inside every element of the input array.

Note

```
int[] var_i = {10000, 20000, 60000, 80000}
var_value = ValueReverse(var_i, 0)    // var_value = {270991360, 541982720, 1625948160, -2143813376}
var_value = ValueReverse(var_i, 1)    // var_value = {4135, 8270, 1625948160, -2143813376}
var_value = ValueReverse(var_i, 2)    // var_value = {4135, 8270, 24810, -32712}
```

Syntax 7

```
int[] ValueReverse (
    int[]
)
```

Parameters

`int[]` Input array value

Note

Similar to Syntax6 with input integer as int32

ValueReverse(int[]) => **ValueReverse**(int[], 0)

Syntax 8

```
float[] ValueReverse (  
    float[]  
)
```

Parameters

float[] Input array value

Return

float[] The array formed from reversing the sequence of byte units inside every element of the input array.

Note

```
float[] var_i = {10000, 20000}  
var_value = ValueReverse(var_i)           // var_value = {5.887614E-39, 5.933532E-39}
```

Syntax 9

```
double[] ValueReverse (  
    double[]  
)
```

Parameters

double[] Input array value

Return

double[] The array formed from reversing the sequence of byte units inside every element of the input array.

Note

```
double[] var_i = {10000, 20000}  
var_value = ValueReverse(var_i)           // var_value = {4.42825109579759E-317, 4.43027478868296E-317}
```

Syntax 10

```
string[] ValueReverse (  
    string[]  
)
```

Parameters

string[] Input string array

Return

`string[]` The string array formed from reversing the string inside every element of the input string array.

Note

```
string[] var_i = {"ABCDEFGH", "12345678"}  
var_value = ValueReverse(var_i)           // var_value = {"GFEDCBA", "87654321"}
```

2.33 GetBytes()

Convert arbitrary data type to byte array.

Syntax 1

```
byte[] GetBytes (  
    ?,  
    int  
)
```

Parameters

`?` The input data. Data type can be int, float, double, bool, string or array.

`int` The input data as integers and floating points follows Little Endian or Big Endian

- 0 Little Endian (Default)
- 1 Big Endian

The input data as string arrays separates with 0x00 0x00 for each element

- 0 Not separate with 0x00 0x00 (Default)
- 1 Separate with 0x00 0x00

Return

`byte[]` The byte array formed by input data

Syntax 2

```
byte[] GetBytes (  
    ?  
)
```

Note

Same as syntax 1 with Little Endian or Big Endian defaults to 0 such as returns based on Little Endian

getBytes(?) => GetBytes(?, 0)

```
? byte var_n = 100  
var_value = GetBytes(var_n)           // var_value = {0x64}
```

```
var_value = GetBytes(var_n, 0)    // var_value = {0x64}
var_value = GetBytes(var_n, 1)    // var_value = {0x64}
```

? **byte[]** var_n = {100, 200} // Convert every element of the array to byte, 1 byte as a single unit.

```
var_value = GetBytes(var_n)        // var_value = {0x64, 0xC8}
var_value = GetBytes(var_n, 0)     // var_value = {0x64, 0xC8}
var_value = GetBytes(var_n, 1)     // var_value = {0x64, 0xC8}
```

? **int**

```
var_value = GetBytes(123456)       // var_value = {0x40, 0xE2, 0x01, 0x00}
var_value = GetBytes(123456, 0)    // var_value = {0x40, 0xE2, 0x01, 0x00}
var_value = GetBytes(0x123456, 0)  // var_value = {0x56, 0x34, 0x12, 0x00}
var_value = GetBytes(0x1234561, 1) // var_value = {0x01, 0x23, 0x45, 0x61}
```

? **int[]** var_n = {10000, 20000, 80000}

// Convert every single element of the array to byte. For int32 data, works on 4 bytes sequentially.

```
var_value = GetBytes(var_n)
// var_value = {0x10, 0x27, 0x00, 0x00, 0x20, 0x4E, 0x00, 0x00, 0x80, 0x38, 0x01, 0x00}
var_value = GetBytes(var_n, 0)
// var_value = {0x10, 0x27, 0x00, 0x00, 0x20, 0x4E, 0x00, 0x00, 0x80, 0x38, 0x01, 0x00}
var_value = GetBytes(var_n, 1)
// var_value = {0x00, 0x00, 0x27, 0x10, 0x00, 0x00, 0x4E, 0x20, 0x00, 0x01, 0x38, 0x80}
```

? **float**

```
var_value = GetBytes(123.456, 0)   // var_value = {0x79, 0xE9, 0xF6, 0x42}
float var_n = -1.2345
var_value = GetBytes(var_n, 0)     // var_value = {0x19, 0x04, 0x9E, 0xBF}
var_value = GetBytes(var_n, 1)     // var_value = {0xBF, 0x9E, 0x04, 0x19}
```

? **float[]** var_n = {1.23, 4.56, -7.89}

// Convert every single element of the array to byte. For float data, works on 4 bytes sequentially.

```
var_value = GetBytes(var_n)
// var_value = {0xA4, 0x70, 0x9D, 0x3F, 0x85, 0xEB, 0x91, 0x40, 0xE1, 0x7A, 0xFC, 0xC0}
var_value = GetBytes(var_n, 0)
// var_value = {0xA4, 0x70, 0x9D, 0x3F, 0x85, 0xEB, 0x91, 0x40, 0xE1, 0x7A, 0xFC, 0xC0}
var_value = GetBytes(var_n, 1)
// var_value = {0x3F, 0x9D, 0x70, 0xA4, 0x40, 0x91, 0xEB, 0x85, 0xC0, 0xFC, 0x7A, 0xE1}
```

? **double** var_n = -1.2345

var_value = **GetBytes**(var_n, 0) // var_value = {0x8D, 0x97, 0x6E, 0x12, 0x83, 0xC0, 0xF3, 0xBF}

var_value = **GetBytes**(var_n, 1) // var_value = {0xBF, 0xF3, 0xC0, 0x83, 0x12, 0x6E, 0x97, 0x8D}

? **double[]** var_n = {1.23, -7.89}

// Convert every single element of the array to byte. For double data, works on 8 bytes sequentially.

var_value = **GetBytes**(var_n)

// var_value = {0xAE, 0x47, 0xE1, 0x7A, 0x14, 0xAE, 0xF3, 0x3F, 0x8F, 0xC2, 0xF5, 0x28, 0x5C, 0x8F, 0x1F, 0xC0}

var_value = **GetBytes**(var_n, 0)

// var_value = {0xAE, 0x47, 0xE1, 0x7A, 0x14, 0xAE, 0xF3, 0x3F, 0x8F, 0xC2, 0xF5, 0x28, 0x5C, 0x8F, 0x1F, 0xC0}

var_value = **GetBytes**(var_n, 1)

// var_value = {0x3F, 0xF3, 0xAE, 0x14, 0x7A, 0xE1, 0x47, 0xAE, 0xC0, 0x1F, 0x8F, 0x5C, 0x28, 0xF5, 0xC2, 0x8F}

? **bool** var_flag = true // true is converted to 1 ; false is converted to 0

var_value = **GetBytes**(flag) // var_value = {1}

var_value = **GetBytes**(flag, 0)

// var_value = {1} // Because bool is 1 byte, Endian Parameters are not sufficient.

var_value = **GetBytes**(flag, 1) // var_value = {1}

? **bool[]** var_flag = {true, false, true, false, false, true, true}

var_value = **GetBytes**(flag) // var_value = {1, 0, 1, 0, 0, 1, 1}

var_value = **GetBytes**(flag, 0) // var_value = {1, 0, 1, 0, 0, 1, 1}

var_value = **GetBytes**(flag, 1) // var_value = {1, 0, 1, 0, 0, 1, 1}

? **string** var_n = "ABCDEFGH" // string to encode in UTF8

var_value = **GetBytes**(var_n) // var_value = {0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47}

var_value = **GetBytes**(var_n, 0)

// var_value = {0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47} // Endian Parameters not sufficient

var_value = **GetBytes**(var_n, 1) // var_value = {0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47}

? **string[]** var_n = {"ABC", "DEF", "達明機器人"}

var_value = **GetBytes**(var_n)

// var_value = {0x41, 0x42, 0x43, 0x44, 0x45, 0x46,

0xE9, 0x81, 0x94, 0xE6, 0x98, 0x8E, 0xE6, 0xA9, 0x9F, 0xE5, 0x99, 0xA8, 0xE4, 0xBA, 0xBA}

var_value = **GetBytes**(var_n, 1)

// var_value = {0x41, 0x42, 0x43, 0x00, 0x00, 0x44, 0x45, 0x46, 0x00, 0x00,

0xE9, 0x81, 0x94, 0xE6, 0x98, 0x8E, 0xE6, 0xA9, 0x9F, 0xE5, 0x99, 0xA8, 0xE4, 0xBA, 0xBA}

*Conversion of string[] to byte[] without separation bytes will save the content completely, but it is unable to turn

byte[] back to string[] effectively.

*It is effective to turn byte[] back to string[] by inserting separation bytes (2 consecutive 0x00s) between the elements in the array, but it is possible to find conversion errors if the value of the string come with 0x00 0x00.

Syntax 3

Convert integer (int type) to byte array.

```
byte[] GetBytes (  
    int,  
    int,  
    int  
)
```

Parameters

int	The input integer (int type)
int	The input value follows Little Endian or Big Endian
0	Little Endian (Default)
1	Big Endian
int	The input integer value's data type is int32 or int16
0	int32 (Default)
1	int16. If the data does not meets int16 format, int32 will be applied instead.
2	int16. Forced to apply int16 format. For int32 data input, there could be some bytes missing.

Return

byte[] The byte array formed by input integer. For int32 data, convert with 4 bytes. For int16 data, convert with 2 bytes.

Note

var_value = GetBytes(12345, 0, 0)	// var_value = {0x39, 0x30, 0x00, 0x00}
var_value = GetBytes(12345, 0, 1)	// var_value = {0x39, 0x30}
var_value = GetBytes(12345, 0, 2)	// var_value = {0x39, 0x30}
var_value = GetBytes(0x123456, 0, 0)	// var_value = {0x56, 0x34, 0x12, 0x00}
var_value = GetBytes(0x123456, 0, 1)	// var_value = {0x56, 0x34, 0x12, 0x00}
var_value = GetBytes(0x123456, 0, 2)	// var_value = {0x56, 0x34} // bytes missing
var_value = GetBytes(0x1234561, 1, 0)	// var_value = {0x01, 0x23, 0x45, 0x61}
var_value = GetBytes(0x1234561, 1, 1)	// var_value = {0x01, 0x23, 0x45, 0x61}
var_value = GetBytes(0x1234561, 1, 2)	// var_value = {0x45, 0x61} // bytes missing

Syntax 4

Convert the integer array (int[] type) to byte array

```
byte[] GetBytes (  
    int[],  
    int,  
    int  
)
```

Parameters

<code>int[]</code>	The input integer array (int[] type)
<code>int</code>	The input integer array follows Little Endian or Big Endian
0	Little Endian (Default)
1	Big Endian
<code>int</code>	The input integer array's data type is int32 or int16
0	int32 (Default)
1	int16. If the data does not meets int16 format, int32 will be applied instead
2	int16. Forced to apply int16 format. For int32 data input, there could be some bytes missing.

Return

`byte[]` The byte array formed by input integer array. Every element is converted independently and forms an array. For int32 data, convert with 4 bytes. For int16 data, convert with 2 bytes.

Note

```
var_i={10000, 20000, 80000}  
var_value = GetBytes(var_i, 0, 0)  
    // var_value = {0x10, 0x27, 0x00, 0x00, 0x20, 0x4E, 0x00, 0x00, 0x80, 0x38, 0x01, 0x00}  
var_value = GetBytes(var_i, 0, 1)    // var_value = {0x10, 0x27, 0x20, 0x4E, 0x80, 0x38, 0x01, 0x00}  
var_value = GetBytes(var_i, 0, 2)    // var_value = {0x10, 0x27, 0x20, 0x4E, 0x80, 0x38} // bytes missing  
var_value = GetBytes(var_i, 1, 0)  
    // var_value = {0x00, 0x00, 0x27, 0x10, 0x00, 0x00, 0x4E, 0x20, 0x00, 0x01, 0x38, 0x80}  
var_value = GetBytes(var_i, 1, 1)    // var_value = {0x27, 0x10, 0x4E, 0x20, 0x00, 0x01, 0x38, 0x80}  
var_value = GetBytes(var_i, 1, 2)    // var_value = {0x27, 0x10, 0x4E, 0x20, 0x38, 0x80} // bytes missing
```

2.34 GetString()

Convert arbitrary data type to string

Syntax 1

```
string GetString (  
    ?,  
    int,
```

`int`

)

Parameters

`?` The input data. Data type can be int, float, double, bool, string or array.

`int` The output string's notation is decimal, hexadecimal or binary (Can be only applied to hexadecimal or binary number)

`10` decimal

`16` hexadecimal

`2` binary

String's notation

`123` decimal

`0x7F` hexadecimal

`0b101` binary

When the input value is a string array, the output string value is in standard string format or not.

`0` or `10` Automatic detection. If the values in the string come with double quotations or commas, it converse to standard string format.

`1` Mandatory conversion to standard string format

`Other` No conversion

`int` The output string format (Can be applied to hexadecimal or binary number only)

`0` Fill up digits. Add prefix 0x or 0b, e.g. 0x0C or 0b00001100

`1` Fill up digits. No prefix 0x or 0b, e.g. 0C or 00001100

`2` Don't fill up digits. Add prefix 0x or 0b, e.g. 0xC or 0b1100

`3` Don't fill up digits. No prefix 0x or 0b, e.g. C or 1100

Return

`string` String converted from input data. If the input data cannot be converted, returns empty string. If the input data is array, every element is converted respectively, and returned in "{ , , }" format

Syntax 2

```
string GetString(  
    ?,  
    int  
)
```

Note

Similar to Syntax1 with filling up digits and adding prefix 0x or 0b.

GetString(?, 16) => GetString(?, 16, 0)

Syntax 3

```
string GetString(  
    ?  
)
```

Note

Same as syntax 1. The output string's notation defaults to 10 and the output string format defaults to 0.

GetString(?) => GetString(?, 10, 0)

GetString(?) => GetString(?, 0, 0) // supposed ? is a string array

? **byte** var_n = 123

```
var_value = GetString(var_n)           // var_value = "123"  
var_value = GetString(var_n, 10)       // var_value = "123"  
var_value = GetString(var_n, 16)       // var_value = "0x7B"  
var_value = GetString(var_n, 2)        // var_value = "0b01111011"  
var_value = GetString(var_n, 16, 3)    // var_value = "7B"  
var_value = GetString(var_n, 2, 2)     // var_value = "0b1111011"
```

? **byte[]** var_n = {12, 34, 56}

```
var_value = GetString(var_n)           // var_value = "{12,34,56}"  
var_value = GetString(var_n, 10)       // var_value = "{12,34,56}"  
var_value = GetString(var_n, 16)       // var_value = "{0x0C,0x22,0x38}"  
var_value = GetString(var_n, 2)        // var_value = "{0b00001100,0b00100010,0b00111000}"  
var_value = GetString(var_n, 16, 3)    // var_value = "{C,22,38}"  
var_value = GetString(var_n, 2, 2)     // var_value = "{0b1100,0b100010,0b111000}"
```

? **int** var_n = 1234

```
var_value = GetString(var_n)           // var_value = "1234"  
var_value = GetString(var_n, 10)       // var_value = "1234"  
var_value = GetString(var_n, 16)       // var_value = "0x000004D2"  
var_value = GetString(var_n, 2)        // var_value = "0b0000000000000000000010011010010"  
var_value = GetString(var_n, 16, 3)    // var_value = "4D2"  
var_value = GetString(var_n, 2, 2)     // var_value = "0b10011010010"
```

? **int[]** var_n = {123, 345, -123, -456}

```

var_value = GetString(var_n)           // var_value = "{123,345,-123,-456}"
var_value = GetString(var_n, 10)       // var_value = "{123,345,-123,-456}"
var_value = GetString(var_n, 16)       // var_value = "{0x0000007B,0x00000159,0xFFFFF85,0xFFFFFE38}"
var_value = GetString(var_n, 2)        // var_value = "{0b0000000000000000000000001111011,
                                         0b000000000000000000000000101011001,
                                         0b11111111111111111111111110000101,
                                         0b1111111111111111111111111000111000}"

var_value = GetString(var_n, 16, 3)    // var_value = "{7B,159,FFFFFF85,FFFFFFE38}"
var_value = GetString(var_n, 2, 2)     // var_value = "{0b1111011,
                                         0b101011001,
                                         0b11111111111111111111111110000101,
                                         0b1111111111111111111111111000111000}"

```

? float var_n = 12.34

```

var_value = GetString(var_n)           // var_value = "12.34"
var_value = GetString(var_n, 10)       // var_value = "12.34"
var_value = GetString(var_n, 16)       // var_value = "0x414570A4"
var_value = GetString(var_n, 2)        // var_value = "0b01000001010001010111000010100100"
var_value = GetString(var_n, 16, 3)    // var_value = "414570A4"
var_value = GetString(var_n, 2, 2)     // var_value = "0b1000001010001010111000010100100"

```

? float[] var_n = {123.4, 345.6, -123.4, -456.7}

```

var_value = GetString(var_n)           // var_value = "{123.4,345.6,-123.4,-456.7}"
var_value = GetString(var_n, 10)       // var_value = "{123.4,345.6,-123.4,-456.7}"
var_value = GetString(var_n, 16)       // var_value = "{0x42F6CCCD,0x43ACCCCD,0xC2F6CCCD,0xC3E4599A}"
var_value = GetString(var_n, 16, 3)    // var_value = "{42F6CCCD,43ACCCCD,C2F6CCCD,C3E4599A}"

```

? double var_n = 12.34

```

var_value = GetString(var_n)           // var_value = "12.34"
var_value = GetString(var_n, 10)       // var_value = "12.34"
var_value = GetString(var_n, 16)       // var_value = "0x4028AE147AE147AE"
var_value = GetString(var_n, 16, 3)    // var_value = "4028AE147AE147AE"

```

? double[] var_n = {123.45, 345.67, -123.48, -456.79}

```

var_value = GetString(var_n)           // var_value = "{123.45,345.67,-123.48,-456.79}"
var_value = GetString(var_n, 10)       // var_value = "{123.45,345.67,-123.48,-456.79}"
var_value = GetString(var_n, 16)       // var_value = "{0x405EDCCCCCCCCCD,0x40759AB851EB851F,
                                         0xC05EDEB851EB851F,0xC07C8CA3D70A3D71}"
var_value = GetString(var_n, 16, 3)    // var_value = "{405EDCCCCCCCCCD,40759AB851EB851F,

```

```
? bool var_n = true
var_value = GetString(var_n)           // var_value = "true"
var_value = GetString(var_n, 16)        // var_value = "true"
var_value = GetString(var_n, 2)         // var_value = "true"
var_value = GetString(var_n, 16, 3)     // var_value = "true"

? bool[] var_n = {true, false, true, false, false, true}
var_value = GetString(var_n)           // var_value = "{true,false,true,false,false,true}"
var_value = GetString(var_n, 16)        // var_value = "{true,false,true,false,false,true}"
var_value = GetString(var_n, 2)         // var_value = "{true,false,true,false,false,true}"
var_value = GetString(var_n, 16, 3)     // var_value = "{true,false,true,false,false,true}"

? string var_n = "1234567890"
var_value = GetString(var_n)           // var_value = "1234567890"
var_value = GetString(var_n, 16)        // var_value = "1234567890"
var_value = GetString(var_n, 2)         // var_value = "1234567890"
var_value = GetString(var_n, 16, 3)     // var_value = "1234567890"

? string[] var_n = {"123.45", "345.67", "-12""3.48", "-45A6.79"}
var_value = GetString(var_n)           // var_value = "{123.45,345.67,-12""3.48,-45A6.79}"
var_value = GetString(var_n, 16)        // var_value = "{123.45,345.67,-12""3.48,-45A6.79}"
var_value = GetString(var_n, 2)         // var_value = "{123.45,345.67,-12""3.48,-45A6.79}" // -12""3.48 displayed as
-12"3.48
var_value = GetString(var_n, 16, 3)     // var_value = "{123.45,345.67,-12""3.48,-45A6.79}"
//use automatic detection as the default
```

Syntax 4

```
string GetString (
    ?,
    string,
    int,
    int
)
```

Parameters

- ? The input data. Data type can be int, float, double, bool, string or array.
- string Separator for output string (Only effective to array input)

<code>int</code>	The output string's notation is decimal, hexadecimal or binary (Can be only applied to hexadecimal or binary number)
<code>10</code>	decimal
<code>16</code>	hexadecimal
<code>2</code>	binary
	String's notation
<code>123</code>	decimal
<code>0x7F</code>	hexadecimal
<code>0b101</code>	binary
	When the input value is a string array, the output string value is in standard string format or not.
<code>0</code> or <code>10</code>	Automatic detection. If the values in the string come with double quotations or separation symbols, it converse to standard string format.
<code>1</code>	Mandatory conversion to standard string format
<code>Other</code>	No conversion
<code>int</code>	The output string format (Can be only applied to hexadecimal or binary number)
<code>0</code>	Fill up digits. Add prefix 0x or 0b, e.g. 0x0C or 0b00001100
<code>1</code>	Fill up digits. No prefix 0x or 0b, e.g. 0C or 00001100
<code>2</code>	Don't fill up digits. Add prefix 0x or 0b, e.g. 0xC or 0b1100
<code>3</code>	Don't fill up digits. No prefix 0x or 0b, e.g. C or 1100

Return

`string` String converted from input data. If the input data cannot be converted, returns empty string.
If the input data is array, every element is converted respectively, and returned as a string with the assigned separator

Syntax 5

```
string GetString (
    ?,
    string,
    int
)
```

Note

Same as Syntax 4 with filling up digits and adding prefix 0x or 0b

`GetString(?, str, 16) => GetString(?, str, 16, 0)`

Syntax 6

```
string GetString (
    ?,
```

```
string
```

```
)
```

Note

Same as Syntax 4. The output string's notation defaults to 10 and the output string format defaults to 0.

GetString(?) => **GetString(?, 10, 0)**

GetString(?) => **GetString(?, 0, 0)** // supposed ? is a string array

? **byte** var_n = 123

var_value = **GetString**(var_n) // var_value = "123"

var_value = **GetString**(var_n, ";", 10) // var_value = "123"

var_value = **GetString**(var_n, "-", 16) // var_value = "0x7B"

var_value = **GetString**(var_n, "#", 2) // var_value = "0b01111011"

var_value = **GetString**(var_n, ",", 16, 3) // var_value = "7B"

var_value = **GetString**(var_n, "", 2, 2) // var_value = "0b1111011"

* Separator is effective to array input only.

? **byte[]** var_n = {12, 34, 56}

var_value = **GetString**(var_n, "-") // var_value = "12-34-56"

var_value = **GetString**(var_n, Ctrl("\r\n"), 10) // var_value = "12\u00D0A34\u00D0A56"

var_value = **GetString**(var_n, newline, 16) // var_value = "0x0C\u00D0A0x22\u00D0A0x38"

var_value = **GetString**(var_n, NewLine, 2) // var_value =
"0b00001100\u00D0A0b00100010\u00D0A0b00111000"

var_value = **GetString**(var_n, "-", 16, 3) // var_value = "C-22-38"

var_value = **GetString**(var_n, "", 2, 2) // var_value = "0b1100-0b100010-0b111000"

* **\u00D0A** is Newline control character, not string value.

? **string[]** var_n = {"123.45", "345.67", "-12""3.48", "-45A6.79"}

var_value = **GetString**(var_n, "-") // var_value = "123.45-345.67--12""3.48"--45A6.79""

var_value = **GetString**(var_n, "-", 1) // var_value = ""123.45"--345.67"--12""3.48"--45A6.79""

var_value = **GetString**(var_n, "-", 2) // var_value = "123.45-345.67--12"3.48--45A6.79"

// Troubled for identifying the separation symbols and the negative signs.

Syntax 7

```
string GetString (
```

```
?,
```

```
string,
```

```
string,
```

```

    int,
    int
)

```

Parameters

- ?** The input data. Data type can be int, float, double, bool, string or array.
- string** The index of the output string for array input. (Only effective to ? as array type data)
 - * Support numeric format strings
- string** Separator for output string (Only effective to array input)
- int** The output string's notation is decimal, hexadecimal or binary (Can be only applied to hexadecimal or binary number)
 - 10 decimal
 - 16 hexadecimal
 - 2 binary

String's notation

 - 123 decimal
 - 0x7F hexadecimal
 - 0b101 binary

When the input value is a string array, the output string value is in standard string format or not.

 - 0 or 10 Automatic detection. If the values in the string come with double quotations or separation symbols, it converse to standard string format.
 - 1 Mandatory conversion to standard string format
 - Other No conversion
- int** The output string format (Can be only applied to hexadecimal or binary number)
 - 0 Fill up digits. Add prefix 0x or 0b, e.g. 0x0C or 0b00001100
 - 1 Fill up digits. No prefix 0x or 0b, e.g. 0C or 00001100
 - 2 Don't fill up digits. Add prefix 0x or 0b, e.g. 0xC or 0b1100
 - 3 Don't fill up digits. No prefix 0x or 0b, e.g. C or 1100

Return

- string** Converse the value to the string to return. If unable to converse, it returns an empty string. If the type is array, elements in the array will be conversed to strings with prefixes of the element index value format string separated by separation symbols to return. There will be no braces.

Syntax 8

```

string GetString(
    ?,
    string,

```

```

    string,
    int
)

```

Note

Similar to Syntax7 with filling up digits and adding prefix.

GetString(?, str, str, 16) => **GetString**(?, str, str, 16, 0)

Syntax 9

```

string GetString (
    ?,
    string,
    string
)

```

Note

Similar to Syntax7 with decimal output, with filling up digits and adding prefix.

GetString(?, str, str) => **GetString**(?, str, str, 10, 0)

? **byte** var_n = 123

```

var_value = GetString(var_n)           // var_value = "123"
var_value = GetString(var_n, "[0]=", ";", 10) // var_value = "123"
var_value = GetString(var_n, "[0]=", "-", 16) // var_value = "0x7B"
var_value = GetString(var_n, "[0]=", "#", 2) // var_value = "0b01111011"

```

* Index and separator are only effective to array input.

? **byte[]** var_n = {12, 34, 56}

```

var_value = GetString(var_n, "[0]=", "-")           // var_value = "[0]=12-[1]=34-[2]=56"
var_value = GetString(var_n, "[0]=", Ctrl("\r\n"), 10) // var_value = "[0]=12\u00D0A[1]=34\u00D0A[2]=56"
var_value = GetString(var_n, "[0]=", newline, 16)    // var_value =
                                                    "[0]=0x0C\u00D0A[1]=0x22\u00D0A[2]=0x38"

var_value = GetString(var_n, "[0]=", "-", 16, 3)    // var_value = "[0]=C-[1]=22-[2]=38"
var_value = GetString(var_n, "[0]=", "-", 2, 2)    // var_value =
                                                    "[0]=0b1100-[1]=0b100010-[2]=0b111000"

```

* "[0]=" Support numeric format strings

* **\u00D0A** is Newline control character, not string value.

2.35 GetToken()

Retrieve a substring from input string, or the sub-array from the input byte[] array

Syntax 1

```
string GetToken (  
    string,  
    string,  
    string,  
    int,  
    int  
)
```

Parameters

string	Input string
string	Prefix. The leading element of the substring
string	Suffix. The trailing element of the substring
int	The number of the matched substring to retrieve
>=1	Retrieve the n th matched substring
-1	Retrieve the last matched substring
int	Remove options
0	The 1 st matched not in the start of the input string, and not remove the prefix and the suffix. (default)
1	The 1 st matched not in the start of the input string, and remove the prefix and the suffix.
2	The 1 st matched in the start of the input string, and not remove the prefix and the suffix.
3	The 1 st matched in the start of the input string, and remove the prefix and the suffix.

Return

string String formed by part of the input string

If the **prefix and suffix are empty strings**, returns the input string

If **the number of the matched substrings <=0 or larger than the number of the total matched substrings**, returns empty string

If the remove option is 2 or 3, the first match retrieved must be at the start of the input string; otherwise, it returns an empty string.

Syntax 2

```
string GetToken (  
    string,  
    string,  
    string,  
    int  
)
```

Note

Similar to Syntax1 with reserving prefix and suffix.

GetToken(str,str,str,1) => GetToken(str,str,str,1,0)

Syntax 3

```
string GetToken (  
    string,  
    string,  
    string  
)
```

Note

Similar to Syntax1 with returning the first occurrence, and reserving prefix and suffix.

GetToken(str,str,str) => GetToken(str,str,str,1,0)

```
string var_n = "$abcd$1234$ABCD$"  
var_value = GetToken(var_n, "", "", 0)           // var_value = "$abcd$1234$ABCD$"  
var_value = GetToken(var_n, "$", "$")           // var_value = "$abcd$"  
var_value = GetToken(var_n, "$", "$", 0)         // var_value = ""  
var_value = GetToken(var_n, "$", "$", 1)         // var_value = "$abcd$"  
var_value = GetToken(var_n, "$", "$", 2)         // var_value = "$ABCD$"  
var_value = GetToken(var_n, "$", "$", 3)         // var_value = ""  
var_value = GetToken(var_n, "$", "$", -1, 1)      // var_value = "ABCD"  
var_value = GetToken(var_n, "$", "$", 1, 1)      // var_value = "abcd"  
var_value = GetToken(var_n, "$", "$", 2, 1)      // var_value = "ABCD"  
var_value = GetToken(var_n, "$", "", 1)          // var_value = "$abcd"  
var_value = GetToken(var_n, "$", "", 2)          // var_value = "$1234"  
var_value = GetToken(var_n, "$", "", 3)          // var_value = "$ABCD"  
var_value = GetToken(var_n, "$", "", 4)          // var_value = "$"  
var_value = GetToken(var_n, "", "$", 1)          // var_value = "$"  
var_value = GetToken(var_n, "", "$", 2)          // var_value = "abcd$"  
var_value = GetToken(var_n, "", "$", 3)          // var_value = "1234$"  
var_value = GetToken(var_n, "", "$", 4)          // var_value = "ABCD$"  
string var_n = "$abcd$1234$ABCD$" + Ctrl("\r\n") + "56\r\n78$"  
var_value = GetToken(var_n, "$", Ctrl("\r\n"), 1) // var_value = "$abcd$1234$ABCD$\u000A"  
var_value = GetToken(var_n, "$", newline, 2)     // var_value = ""  
var_value = GetToken(var_n, "$", NewLine, 1, 1)   // var_value = "abcd$1234$ABCD$"  
                                                  // Remove prefix and suffix  
var_value = GetToken(var_n, Ctrl("\r\n"), "$", 1) // var_value = "\u000A56\r\n78$"  
var_value = GetToken(var_n, newline, "$", 2)     // var_value = ""  
var_value = GetToken(var_n, NewLine, "$", 1, 1)   // var_value = "56\r\n78$"  
* \u000A is Newline control character, not string value.
```

```
string n = "#abcd$1234#ABCD$5678#"
```

```

value = GetToken(n, "", "", 0) // value = "#abcd$1234#ABCD$5678#"
value = GetToken(n, "$", "$") // value = "$1234#ABCD$"
value = GetToken(n, "$", "$", 0) // value = ""
value = GetToken(n, "$", "$", 1) // value = "$1234#ABCD$"
value = GetToken(n, "$", "$", 2) // value = ""
value = GetToken(n, "$", "$", 3) // value = ""
value = GetToken(n, "$", "$", -1, 0) // value = "$1234#ABCD$"
value = GetToken(n, "$", "$", -1, 1) // value = "1234#ABCD"
value = GetToken(n, "$", "$", 1, 1) // value = "1234#ABCD"
value = GetToken(n, "$", "$", 2, 1) // value = ""
value = GetToken(n, "$", "", 1) // value = "$1234#ABCD"
value = GetToken(n, "$", "", 2) // value = "$5678#"
value = GetToken(n, "$", "", 3) // value = ""
value = GetToken(n, "$", "", 4) // value = ""
value = GetToken(n, "", "$", 1) // value = "#abcd$"
value = GetToken(n, "", "$", 2) // value = "1234#ABCD$"
value = GetToken(n, "", "$", 3) // value = ""
value = GetToken(n, "", "$", 4) // value = ""
value = GetToken(n, "$", "$", 1, 2) // value = ""
// The string matched $ not in the start of the input string.

value = GetToken(n, "$", "$", -1, 2) // value = ""
// The string matched $ not in the start of the input string.

value = GetToken(n, "#", "", 1, 3) // value = "abcd$1234"
value = GetToken(n, "#", "", 1, 2) // value = "#abcd$1234"
value = GetToken(n, "#", "", 2, 2) // value = "#ABCD$5678"
value = GetToken(n, "#", "", 3, 2) // value = "#"
value = GetToken(n, "#", "", 4, 2) // value = ""
value = GetToken(n, "#", "", -1, 2) // value = "#"
value = GetToken(n, "#", "", -1, 3) // value = ""
value = GetToken(n, "#", "$", 1, 2) // value = "#abcd$"
value = GetToken(n, "#", "$", 1, 3) // value = "abcd"

```

Syntax 4

```

string GetToken (
    string,
    byte[],
    byte[],
    int,
    int
)

```

Parameters

<code>string</code>	Input string
<code>byte[]</code>	Prefix. The leading element of the substring, <code>byte[]</code> type
<code>byte[]</code>	Suffix. The trailing element of the substring, <code>byte[]</code> type
<code>int</code>	The number of the matched substring to retrieve
<code>>=1</code>	Retrieve the n^{th} matched substring
<code>-1</code>	Retrieve the last matched substring
<code>int</code>	Remove options
<code>0</code>	The 1 st matched not in the start of the input string, and not remove the prefix and the suffix. (default)
<code>1</code>	The 1 st matched not in the start of the input string, and remove the prefix and the suffix.
<code>2</code>	The 1 st matched in the start of the input string, and not remove the prefix and the suffix.
<code>3</code>	The 1 st matched in the start of the input string, and remove the prefix and the suffix.

Return

`string` String formed by part of the input string

If the [prefix and suffix are empty strings](#), returns the input string

If [the number of the matched substrings <=0 or larger than the number of the total matched substrings](#), returns empty string

If the remove option is 2 or 3, the first match retrieved must be at the start of the input string; otherwise, it returns an empty string.

Syntax 5

```
string GetToken (  
    string,  
    byte[],  
    byte[],  
    int  
)
```

Note

Similar to Syntax4 with reserving prefix and suffix

`GetToken(str,byte[],byte[],1) => GetToken(str,byte[],byte[],1,0)`

Syntax 6

```
string GetToken (  
    string,  
    byte[],  
    byte[]  
)
```

Note

Similar to Syntax 4 with the first occurrence and reserving prefix and suffix

GetToken(str,byte[],byte[]) => **GetToken**(str,byte[],byte[],1,0)

string var_n = "\$abcd\$1234\$ABCD\$"

byte[] var_bb0 = {}, var_bb1 = {0x24} // 0x24 is \$

var_value = **GetToken**(var_n, var_bb0, var_bb0, 0) // var_value = "\$abcd\$1234\$ABCD\$"

var_value = **GetToken**(var_n, var_bb1, var_bb1) // var_value = "\$abcd\$"

var_value = **GetToken**(var_n, var_bb1, var_bb1, 0) // var_value = ""

var_value = **GetToken**(var_n, var_bb1, var_bb1, 1) // var_value = "\$abcd\$"

var_value = **GetToken**(var_n, var_bb1, var_bb1, 2) // var_value = "\$ABCD\$"

var_value = **GetToken**(var_n, var_bb1, var_bb1, 3) // var_value = ""

var_value = **GetToken**(var_n, var_bb1, var_bb1, 1, 1) // var_value = "abcd"

var_value = **GetToken**(var_n, var_bb1, var_bb1, 2, 1) // var_value = "ABCD"

var_value = **GetToken**(var_n, var_bb1, var_bb0, 1) // var_value = "\$abcd"

var_value = **GetToken**(var_n, var_bb1, var_bb0, 2) // var_value = "\$1234"

var_value = **GetToken**(var_n, var_bb1, var_bb0, 3) // var_value = "\$ABCD"

var_value = **GetToken**(var_n, var_bb1, var_bb0, 4) // var_value = "\$"

var_value = **GetToken**(var_n, var_bb0, var_bb1, 1) // var_value = "\$"

var_value = **GetToken**(var_n, var_bb0, var_bb1, 2) // var_value = "abcd\$"

var_value = **GetToken**(var_n, var_bb0, var_bb1, 3) // var_value = "1234\$"

var_value = **GetToken**(var_n, var_bb0, var_bb1, 4) // var_value = "ABCD\$"

string var_n = "\$abcd\$1234\$ABCD\$" + Ctrl("\r\n") + "56\r\n78\$"

byte[] var_bb0 = {0x0D,0x0A}, var_bb1 = {0x24} // 0x24 is \$ // 0x0D,0x0A is \u000A

var_value = **GetToken**(var_n, var_bb1, var_bb0, 1) // var_value = "\$abcd\$1234\$ABCD\$\u000A"

var_value = **GetToken**(var_n, var_bb1, var_bb0, 2) // var_value = ""

var_value = **GetToken**(var_n, var_bb1, var_bb0, 1, 1) // var_value = "abcd\$1234\$ABCD\$"
// Removing the prefix and the suffix

var_value = **GetToken**(var_n, var_bb0, var_bb1, 1) // var_value = "\u000A56\r\n78\$"

var_value = **GetToken**(var_n, var_bb0, var_bb1, 2) // var_value = ""

var_value = **GetToken**(var_n, var_bb0, var_bb1, 1, 1) // var_value = "56\r\n78"

* \u000A is the Newline control character, not the string content.

Syntax 7

```
byte[] GetToken (  
    byte[],  
    string,  
    string,  
    int,  
    int  
)
```

Parameters

<code>byte[]</code>	The input <code>byte[]</code>
<code>string</code>	Prefix. The leading element of the output <code>byte[]</code> , <code>byte[]</code> type
<code>string</code>	Suffix. The trailing element of the output <code>byte[]</code> , <code>byte[]</code> type
<code>int</code>	The number of the matched substring to retrieve
<code>>=1</code>	Retrieve the n^{th} matched substring
<code>-1</code>	Retrieve the last matched substring
<code>int</code>	Remove options
<code>0</code>	The 1 st matched not in the start of the input string, and not remove the prefix and the suffix. (default)
<code>1</code>	The 1 st matched not in the start of the input string, and remove the prefix and the suffix.
<code>2</code>	The 1 st matched in the start of the input string, and not remove the prefix and the suffix.
<code>3</code>	The 1 st matched in the start of the input string, and remove the prefix and the suffix.

Return

`byte[]` The `byte[]` formed from part of the input `byte[]`

If the **prefix and suffix are empty**, returns the input array

If **the number of the matched substrings ≤ 0 or larger than the number of the total matched substrings**, returns empty array

If the remove option is 2 or 3, the first match retrieved must be at the start of the input string; otherwise, it returns an empty string.

Syntax 8

```
byte[] GetToken (  
    byte[],  
    string,  
    string,  
    int  
)
```

Note

Similar to Syntax7 with reserving prefix and suffix

GetToken(byte[],str,str,1) => GetToken(byte[],str,str,1,0)

Syntax 9

```
byte[] GetToken (  
    byte[],  
    string,  
    string  
)
```

Note

Similar to Syntax7 with returning the first occurrence, and reserving prefix and suffix.

```
GetToken(byte[],str,str) => GetToken(byte[],str,str,1,0)
```

```
string var_s="$abcd$1234$ABCD$"
```

```
byte[] var_n = GetBytes(var_s)
```

```
var_value = GetToken(var_n, "", "", 0)
```

```
// var_value = {0x24,0x61,0x62,0x63,0x64,0x24,0x31,0x32,0x33,0x34,0x24,0x41,0x42,0x43,0x44,0x24}
```

```
var_value = GetToken(var_n, "$", "$") // var_value = {0x24,0x61,0x62,0x63,0x64,0x24}
```

```
var_value = GetToken(var_n, "$", "$", 0) // var_value = {}
```

```
var_value = GetToken(var_n, "$", "$", 1) // var_value = {0x24,0x61,0x62,0x63,0x64,0x24}
```

```
var_value = GetToken(var_n, "$", "$", 2) // var_value = {0x24,0x41,0x42,0x43,0x44,0x24}
```

```
var_value = GetToken(var_n, "$", "$", 1, 1) // var_value = {0x61,0x62,0x63,0x64}
```

```
var_value = GetToken(var_n, "$", "$", 2, 1) // var_value = {0x41,0x42,0x43,0x44}
```

```
var_value = GetToken(var_n, "$", "", 1) // var_value = {0x24,0x61,0x62,0x63,0x64}
```

```
var_value = GetToken(var_n, "$", "", 2) // var_value = {0x24,0x31,0x32,0x33,0x34}
```

```
var_value = GetToken(var_n, "$", "", 3) // var_value = {0x24,0x41,0x42,0x43,0x44}
```

```
var_value = GetToken(var_n, "$", "", 4) // var_value = {0x24}
```

```
var_value = GetToken(var_n, "", "$", 1) // var_value = {0x24}
```

```
var_value = GetToken(var_n, "", "$", 2) // var_value = {0x61,0x62,0x63,0x64,0x24}
```

```
var_value = GetToken(var_n, "", "$", 3) // var_value = {0x31,0x32,0x33,0x34,0x24}
```

```
var_value = GetToken(var_n, "", "$", 4) // var_value = {0x41,0x42,0x43,0x44,0x24}
```

```
string var_var_s="$abcd$1234$ABCD$" + Ctrl("\r\n") + "56\r\n78$"
```

```
byte[] var_n = GetBytes(var_s)
```

```
var_value = GetToken(var_n, "$", Ctrl("\r\n"), 1)
```

```
// var_value = {0x24,0x61,0x62,0x63,0x64,0x24,0x31,0x32,0x33,0x34,0x24,0x41,0x42,0x43,0x44,0x24,0x0D,0x0A}
```

```
var_value = GetToken(var_n, "$", Ctrl("\r\n"), 1, 1)
```

```
// var_value = {0x61,0x62,0x63,0x64,0x24,0x31,0x32,0x33,0x34,0x24,0x41,0x42,0x43,0x44,0x24}
```

```
// Removing prefix and suffix
```

```
var_value = GetToken(var_n, Ctrl("\r\n"), "$", 1)
```

```
// var_value = {0x0D,0x0A,0x35,0x36,0x5C,0x72,0x5C,0x6E,0x37,0x38,0x24}
```

```
var_value = GetToken(var_n, Ctrl("\r\n"), "$", 1, 1)
```

```
// var_value = {0x35,0x36,0x5C,0x72,0x5C,0x6E,0x37,0x38}
```

Syntax 10

```
byte[] GetToken (
```

```
byte[] ,
```

```
byte[] ,
```

```
byte[] ,
```

```
int ,
```

```
int
```

)

Parameters

<code>byte[]</code>	The input <code>byte[]</code> array
<code>byte[]</code>	Prefix. The leading element of the output <code>byte[]</code>
<code>byte[]</code>	Suffix. The trailing element of the output <code>byte[]</code>
<code>int</code>	The number of the matched substring to retrieve
<code>>=1</code>	Retrieve the n^{th} matched substring
<code>-1</code>	Retrieve the last matched substring
<code>int</code>	Remove options
<code>0</code>	The 1 st matched not in the start of the input string, and not remove the prefix and the suffix. (default)
<code>1</code>	The 1 st matched not in the start of the input string, and remove the prefix and the suffix.
<code>2</code>	The 1 st matched in the start of the input string, and not remove the prefix and the suffix.
<code>3</code>	The 1 st matched in the start of the input string, and remove the prefix and the suffix.

Return

`byte[]` The `byte[]` formed from part of the input `byte[]`

If the [prefix and suffix are empty](#), returns the input array

If [the number of the matched substrings <=0 or larger than the number of total matched substrings](#), returns empty array

If the remove option is 2 or 3, the first match retrieved must be at the start of the input string; otherwise, it returns an empty string.

Syntax 11

```
byte[] GetToken (  
    byte[] ,  
    byte[] ,  
    byte[] ,  
    int  
)
```

Note

Similar to Syntax10 with reserving the prefix and suffix

`GetToken(byte[],byte[],byte[],1) => GetToken(byte[],byte[],byte[],1,0)`

Syntax 12

```
byte[] GetToken (  
    byte[] ,  
    byte[] ,  
    byte[]  
)
```

Note

Similar to Syntax10 with returning the first occurrence, and reserving prefix and suffix.

```
GetToken(byte[],byte[],byte[]) => GetToken(byte[],byte[],byte[],1,0)
```

```
string var_s="$abcd$1234$ABCD$"
```

```
byte[] var_n = GetBytes(var_s)
```

```
byte[] var_bb0 = {}, var_bb1 = {0x24} // 0x24 is $
```

```
var_value = GetToken(var_n, var_bb0, var_bb0, 0)
```

```
// var_value = {0x24,0x61,0x62,0x63,0x64,0x24,0x31,0x32,0x33,0x34,0x24,0x41,0x42,0x43,0x44,0x24}
```

```
var_value = GetToken(var_n, var_bb1, var_bb1) // var_value = {0x24,0x61,0x62,0x63,0x64,0x24}
```

```
var_value = GetToken(var_n, var_bb1, var_bb1, 0) // var_value = {}
```

```
var_value = GetToken(var_n, var_bb1, var_bb1, 1) // var_value = {0x24,0x61,0x62,0x63,0x64,0x24}
```

```
var_value = GetToken(var_n, var_bb1, var_bb1, 2) // var_value = {0x24,0x41,0x42,0x43,0x44,0x24}
```

```
var_value = GetToken(var_n, var_bb1, var_bb1, 1, 1) // var_value = {0x61,0x62,0x63,0x64}
```

```
var_value = GetToken(var_n, var_bb1, var_bb1, 2, 1) // var_value = {0x41,0x42,0x43,0x44}
```

```
var_value = GetToken(var_n, var_bb1, var_bb0, 1) // var_value = {0x24,0x61,0x62,0x63,0x64}
```

```
var_value = GetToken(var_n, var_bb1, var_bb0, 2) // var_value = {0x24,0x31,0x32,0x33,0x34}
```

```
var_value = GetToken(var_n, var_bb1, var_bb0, 3) // var_value = {0x24,0x41,0x42,0x43,0x44}
```

```
var_value = GetToken(var_n, var_bb0, var_bb1, 1) // var_value = {0x24}
```

```
var_value = GetToken(var_n, var_bb0, var_bb1, 2) // var_value = {0x61,0x62,0x63,0x64,0x24}
```

```
var_value = GetToken(var_n, var_bb0, var_bb1, 3) // var_value = {0x31,0x32,0x33,0x34,0x24}
```

```
string var_s="$abcd$1234$ABCD$" + Ctrl("\r\n") + "56\r\n78$"
```

```
byte[] var_n = GetBytes(var_s)
```

```
byte[] var_bb0 = {0x0D,0x0A}, var_bb1 = {0x24}
```

```
var_value = GetToken(var_n, var_bb1, var_bb0, 1)
```

```
// var_value = {0x24,0x61,0x62,0x63,0x64,0x24,0x31,0x32,0x33,0x34,0x24,0x41,0x42,0x43,0x44,0x24,0x0D,0x0A}
```

```
var_value = GetToken(var_n, var_bb1, var_bb0, 1, 1)
```

```
// var_value = {0x61,0x62,0x63,0x64,0x24,0x31,0x32,0x33,0x34,0x24,0x41,0x42,0x43,0x44,0x24}
```

```
// Remove prefix and suffix
```

```
var_value = GetToken(var_n, var_bb0, var_bb1, 1)
```

```
// var_value = {0x0D,0x0A,0x35,0x36,0x5C,0x72,0x5C,0x6E,0x37,0x38,0x24}
```

```
var_value = GetToken(var_n, var_bb0, var_bb1, 1, 1)
```

```
// var_value = {0x35,0x36,0x5C,0x72,0x5C,0x6E,0x37,0x38}
```

2.36 GetAllTokens()

Retrieve all the substrings from input string, which meets the given condition

Syntax 1

```
string[] GetAllTokens (  
    string,  
    string,  
    string,  
    int  
)
```

Parameters

string	Input string
string	Prefix. The leading element of the substring
string	Suffix. The trailing element of the substring
int	Remove options
0	The 1 st matched not in the start of the input string, and not remove the prefix and the suffix. (default)
1	The 1 st matched not in the start of the input string, and remove the prefix and the suffix.
2	The 1 st matched in the start of the input string, and not remove the prefix and the suffix.
3	The 1 st matched in the start of the input string, and remove the prefix and the suffix.

Return

string[] String array formed from retrieving all the substrings from input string
If the **prefix and suffix are empty**, returns the input array
If the remove option is 2 or 3, the first match retrieved must be at the start of the input string; otherwise, it returns an empty string.

Syntax 2

```
string[] GetAllTokens (  
    string,  
    string,  
    string  
)
```

Note

Similar to Syntax1 with reserving prefix and suffix

GetAllTokens(str,str,str) => GetAllTokens(str,str,str,0)

string var_n = "\$abcd\$1234\$ABCD\$"

var_value = GetAllTokens(var_n, "", "")	// var_value = {"\$abcd\$1234\$ABCD\$"}
var_value = GetAllTokens(var_n, "\$", "\$")	// var_value = {"\$abcd\$", "\$ABCD\$"}
var_value = GetAllTokens(var_n, "\$", "\$", 1)	// var_value = {"abcd", "ABCD"}
var_value = GetAllTokens(var_n, "\$", "")	// var_value = {"\$abcd", "\$1234", "\$ABCD", "\$"}
var_value = GetAllTokens(var_n, "", "\$", 1)	// var_value = {"", "abcd", "1234", "ABCD"}

```

string n = "#abcd$1234#ABCD$5678#"

value = GetAllTokens(n, "", "", 0)    // value = {"$abcd$1234$ABCD$"}
value = GetAllTokens(n, "$", "", 0)   // value = {"$1234#ABCD", "$5678#"}
value = GetAllTokens(n, "$", "", 1)   // value = {"1234#ABCD", "5678#"}
value = GetAllTokens(n, "$", "", 2)   // value = {}
// $ is not in the start start of the input string. Returns an empty array.

value = GetAllTokens(n, "$", "", 3)   // value = {}
// $ is not in the start start of the input string. Returns an empty array.

value = GetAllTokens(n, "$", "$", 0)  // value = {"$1234#ABCD$"}
value = GetAllTokens(n, "$", "$", 1)  // value = {"1234#ABCD"}
value = GetAllTokens(n, "$", "$", 2)  // value = {}
// $ is not in the start start of the input string. Returns an empty array.

value = GetAllTokens(n, "$", "$", 3)  // value = {}
// $ is not in the start start of the input string. Returns an empty array.

value = GetAllTokens(n, "#", "", 0)   // value = {"#abcd$1234", "#ABCD$5678", "#"}
value = GetAllTokens(n, "#", "", 1)   // value = {"abcd$1234", "ABCD$5678", ""}
value = GetAllTokens(n, "#", "", 2)   // value = {"#abcd$1234", "#ABCD$5678", "#"}
value = GetAllTokens(n, "#", "", 3)   // value = {"abcd$1234", "ABCD$5678", ""}
value = GetAllTokens(n, "#", "$", 0)  // value = {"#abcd$", "#ABCD$"}
value = GetAllTokens(n, "#", "$", 1)  // value = {"abcd", "ABCD"}
value = GetAllTokens(n, "#", "$", 2)  // value = {"#abcd$", "#ABCD$"}
value = GetAllTokens(n, "#", "$", 3)  // value = {"abcd", "ABCD"}
value = GetAllTokens(n, "", "$", 0)   // value = {"#abcd$", "1234#ABCD$"}
value = GetAllTokens(n, "", "$", 1)   // value = {"#abcd", "1234#ABCD"}
value = GetAllTokens(n, "", "$", 2)   // value = {"#abcd$", "1234#ABCD$"}
value = GetAllTokens(n, "", "$", 3)   // value = {"#abcd", "1234#ABCD"}
value = GetAllTokens(n, "", "#", 0)   // value = {"#", "abcd$1234#", "ABCD$5678#"}
value = GetAllTokens(n, "", "#", 1)   // value = {"", "abcd$1234", "ABCD$5678"}
value = GetAllTokens(n, "", "#", 2)   // value = {"#", "abcd$1234#", "ABCD$5678#"}
value = GetAllTokens(n, "", "#", 3)   // value = {"", "abcd$1234", "ABCD$5678"}

```

2.37 GetNow()

Get the current system time

Syntax 1

```

string GetNow(
    string
)

```

Parameters

string The date and time format strings defining the text representation of a date and time value. The definition of each specifier is listed below. The strings not included will remain the same.

d	The day of the month, from 1 through 31.
dd	The day of the month, from 01 through 31.
ddd	The abbreviated name of the day of the week.
dddd	The full name of the day of the week.
f	The tenths of a second in a date and time value.
ff	The hundredths of a second in a date and time value.
fff	The milliseconds in a date and time value.
ffff	The ten thousandths of a second in a date and time value.
h	The hour, using a 12-hour clock from 1 to 12.
hh	The hour, using a 12-hour clock from 01 to 12.
H	The hour, using a 24-hour clock from 0 to 23.
HH	The hour, using a 24-hour clock from 00 to 23.
m	The minute, from 0 through 59.
mm	The minute, from 00 through 59.
M	The month, from 1 through 12.
MM	The month, from 01 through 12.
MMM	The abbreviated name of the month.
MMMM	The full name of the month.
s	The second, from 0 through 59.
ss	The second, from 00 through 59.
t	The first character of the AM/PM designator.
tt	The AM/PM designator.
y	The year, from 0 to 99.
yy	The year, from 00 to 99.
yyyy	The year as a four-digit number.
/	The date separator.

Return

string Current date and time. If there is an error in format setting, the default format will be applied as MM/dd/yyyy HH:mm:ss.

Note

<code>var_value = GetNow("MM/dd/yyyy HH:mm:ss")</code>	<code>// var_value = 08/15/2017 13:40:30</code>
<code>var_value = GetNow("yyyy/MM/dd HH:mm:ss.ffff")</code>	<code>// var_value = 2017/08/15 13:40:30.123</code>
<code>var_value = GetNow("yyyy-MM-dd hh:mm:ss tt")</code>	<code>// var_value = 2017-08-15 01:40:30 PM</code>

Syntax 2

string **GetNow** (

)

Parameters

`void` No format defined. Default format "MM/dd/yyyy HH:mm:ss" will be applied

Return

`string` Current date and time.

Note

```
var_value = GetNow()           // var_value = 08/15/2017 13:40:30
```

2.38 GetNowStamp()

Get the total run time or difference in total run time

Syntax 1

```
int GetNowStamp (
)
```

Parameters

`void` No parameter

Return

`int` The total run time of the current project in ms. The upper limit is 2147483647 ms
`< 0` Over flow, invalid total run time

Note

```
var_value = GetNowStamp()       // var_value = 2147483647
... others ...
var_value = GetNowStamp()       // var_value = -1 // Over flow
```

Syntax 2

```
double GetNowStamp (
    bool
)
```

Parameters

`bool` Use double format to record project's total run time or not?
`true` Use double type, the upper limit is 9223372036854775807 ms
`false` Use int32 type, the upper limit is 2147483647 ms

Return

`double` The total run time of the current project
`< 0` Over flow. Invalid total run time.

Note

```

var_value = GetNowStamp(false)           // var_value = 2147483647
... others ...
var_value = GetNowStamp(false)           // var_value = -1           // Over flow
var_value = GetNowStamp(true)            // var_value = 3147483647

```

Syntax 3

```

int GetNowStamp (
    int
)

```

Parameters

int Previous recorded run time in ms

Return

int The difference between the current run time and the input run time in ms.
 Run time difference = current run time – input run time
 < 0 Invalid run time difference, caused by input run time larger than current run time, or over flow.

Note

```

var_value = GetNowStamp()                // var_value = 2147483546
... others ... (After 100ms)
diff = GetNowStamp(var_value) // diff = 100
... others ... (After 200ms)
diff = GetNowStamp(var_value) // diff = -1    // Value is over 2147483647

```

Syntax 4

```

double GetNowStamp (
    double
)

```

Parameters

double Previous recorded run time in ms

Return

double The difference between the current run time and the input run time in ms.
 Run time difference = current run time – input run time
 < 0 Invalid run time difference, caused by input run time larger than current run time, or over flow.

Note

```

var_value = GetNowStamp()                // var_value = 2147483546
... others ... (After 100ms)
diff = GetNowStamp(var_value) // diff = 100

```

... others ... (After 200ms)

```
diff = GetNowStamp(var_value) // diff = 200
```

Syntax 5

```
bool GetNowStamp (  
    int,  
    int  
)
```

Parameters

int Previous recorded run time in ms
int The expected run time difference

Return

bool The time difference between current run time and input run time is larger than the expected run time difference or not.

true (Current run time – input run time) >= expected run time
 Or Time difference smaller than zero or over flow
false (Current run time – input run time) < expected run time

Note

```
var_value = GetNowStamp()                      // var_value = 41730494  
... others ... (After 60ms)  
var_flag = GetNowStamp(var_value, 100) // diff = 60              // var_flag = false  
... others ... (After 60ms)  
var_flag = GetNowStamp(var_value, 100) // diff = 120             // var_flag = true
```

Syntax 6

```
bool GetNowStamp (  
    double,  
    double  
)
```

Parameters

double Previous recorded run time in ms
double The expected run time difference

Return

bool The time difference between current run time and input run time is larger than the expected run time difference or not.

true (Current run time – input run time) >= expected run time
 Or Time difference smaller than zero or over flow
false (Current run time – input run time) < expected run time

Note

```
var_value = GetNowStamp()           // var_value = 41730494
... others ... (After 60ms)
var_flag = GetNowStamp(var_value, 100) // diff = 60      // var_flag = false
... others ... (After 60ms)
var_flag = GetNowStamp(var_value, 100) // diff = 120     // var_flag = true
```

2.39 GetVarValue()

Retrieve the value of the variable value. Users can use the string to combine the variable names, and then retrieve the combined value of the variables.

Syntax 1

```
? GetVarValue (
    string
)
```

Parameters

`string` The name of the variable

Return

? Return the value of the variable. The returned type goes by the definition of the variable.
Returns an **error** if the variable is not existed.

Note

```
string var_s1 = "Hello World"
string var_s2 = "Hi TM Robot"
string var_h = "var_s1"
string var_t = "var_s"

string var_re = var_t           // var_re = "var_s"
var_re = var_t + "1"           // var_re = "var_s" + "1" = "var_s1"
var_re = GetVarValue("var_h")  // var_re = "var_s1"
var_re = GetVarValue(var_h)    // var_re = "Hello Wrold"    // var_h = "var_s1"
                                // Retrieve the value of var_s1
var_re = GetVarValue(var_t + "1") // var_re = "Hello World"    // var_b + "1" = "var_s1"
                                // Retrieve the value of var_s1
var_re = GetVarValue(var_t + "2") // var_re = "Hi TM Robot"    // var_b + "2" = "var_s2"
                                // Retrieve the value of var_s2
var_re = GetVarValue(var_t)      // Error    // var_t = "var_s"
```

//Retrieve the value of var_s, but the variable is not existed.

2.40 Length()

Acquire the number of byte of input data, length of string or length of array (number of elements in array)

Syntax 1

```
int Length (  
    ?  
)
```

Parameters

? The input data. The available data types are integer, floating-point, boolean, string, or array.

Return

int Length of data
For input as integer, floating-point number, and boolean, returns the number of byte.
For input as string, returns the length of string.
For input as array, returns the number of element in array

Note

```
? byte var_n = 100  
var_value = Length(var_n)    // var_value = 1  
var_value = Length(100)      // var_value = 1  
  
? int var_n = 400  
var_value = Length(var_n)    // var_value = 4  
var_value = Length(400)      // var_value = 4  
  
? float var_n = 1.234  
var_value = Length(var_n)    // var_value = 4  
var_value = Length(1.234)    // var_value = 4  
  
? double var_n = 1.234  
var_value = Length(var_n)    // var_value = 8  
var_value = Length(1.234)    // var_value = 4  
                                // float // Numbers would be stored as the smaller data type first.  
  
? bool var_n = true  
var_value = Length(var_n)    // var_value = 1  
var_value = Length(false)    // var_value = 1  
  
? string var_n = "A"BC"  
var_value = Length(var_n)    // var_value = 4
```

```

// The string is A"BC. Two double quotation marks represent " in string
var_value = Length("") // var_value = 0
var_value = Length("123") // var_value = 3
var_value = Length(empty) // var_value = 0
? byte[] var_n = {100, 200, 30}
var_value = Length(var_n) // var_value = 3
? int[] var_n = {}
var_value = Length(var_n) // var_value = 0
var_n = {400, 500, 600}
var_value = Length(var_n) // var_value = 3
? float[] var_n = {1.234}
var_value = Length(var_n) // var_value = 1
? double[] var_n = {1.234, 200, -100, +300}
var_value = Length(var_n) // var_value = 4
? bool[] var_n = {true, false, true, true, true, true, false}
var_value = Length(var_n) // var_value = 7
? string[] var_n = {"A"BC", "123", "456", "ABC"}
var_value = Length(var_n) // var_value = 4

```

2.41 Ctrl()

Change the integer or string to control characters

Syntax 1

```

string Ctrl(
    int
)

```

Parameters

int The input integer, which follows the Big Endian format. 4 characters could be transformed at most. 0x00 will not be transformed.

Return

string The string formed by input integer (contains the control character)

Note

```

var_b = 0x0D0A
var_value = Ctrl(var_b) // var_value = \r\n
var_value = Ctrl(0x0D0A) // var_value = \r\n
var_value = Ctrl(0x0D000A09) // var_value = \r\n\t // 0x00 will not be transformed

```

```
var_value = Ctrl(0x0D300A09) // var_value = \r0\n\t // 0x30 is transformed to 0
var_value = Ctrl(0x00) // var_value = "" // empty string does not equal to NULL. For NULL, the code is
Ctrl("\0")
```

Syntax 2

```
string Ctrl(
    string
)
```

Parameters

string Input string. The following rules will be applied. For string not on the list, it will remain the same.

\0	0x00 null
\a	0x07 bell
\b	0x08 backspace
\t	0x09 horizontal tab
\r	0x0D carriage return
\v	0x0B vertical tab
\f	0x0C form feed
\n	0x0A line feed

Return

string The string formed by input integer (contains the control character)

Note

```
var_b = "\r\n"
var_value = Ctrl(var_b) // var_value = \r\n
var_value = Ctrl("\r\n") // var_value = \r\n
var_value = Ctrl("\r\n\t") // var_value = \r\n\t
var_value = Ctrl("\r0\n\t") // var_value = \r0\n\t
var_value = Ctrl("\0") // var_value = \0 // NULL
```

Syntax 3

```
string Ctrl(
    byte[]
)
```

Parameters

byte[] The input byte array, the transfer will start from index [0] to the end of the array. (0x00 will be transferred also)

Return

`string` The string formed by input integer (contains the control character)

Note

```
byte[] var_bb1 = {0xFF,0x55,0x31,0x32,0x33,0x00,0x35,0x36,0x0D,0x0A}  
var_value = Ctrl(var_bb1)      // var_value = U123 56\r\n  
byte[] var_bb2 = {}  
var_value = Ctrl(var_bb2)      // var_value = ""
```

2.42 XOR8()

Utilize XOR 8 bits algorithm to computes the checksum

Syntax 1

```
byte XOR8 (  
    byte[],  
    int,  
    int  
)
```

Parameters

`byte[]` The input byte array

`int` The starting index
`0..(array size-1)` Valid
`<0` Invalid. Returns the initial value 0
`>=array size` Invalid. Returns the initial value 0

`int` The number of elements to be computed.
If the number of elements `<0`, the calculation ends at the last element of the array
If the sum of starting index and number of element exceeds the array size, the calculation ends at the last element of the array.

Return

`byte` Checksum.

Note

```
byte[] var_bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}  
var_value = XOR8(var_bb1,0,Length(var_bb1)) // var_value = 0x6F  
var_value = XOR8(var_bb1,0,-1)              // var_value = 0x6F  
var_value = XOR8(var_bb1,1,-1)              // var_value = 0x7F  
var_value = XOR8(var_bb1,-1,-1)             // var_value = 0
```

Syntax 2

```
byte XOR8 (  
    byte[]
```

```

    byte[],
    int
)

```

Note

Similar to Syntax1 with computing to the last element of the array

XOR8(byte[], int) => **XOR8**(byte[], int, Length(byte[]))

Syntax 3

```

byte XOR8 (
    byte[]
)

```

Note

Similar to Syntax1 with computing all the elements of the array

XOR8(byte[]) => **XOR8**(byte[], 0, Length(byte[]))

```

byte[] var_bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}
var_value = XOR8(var_bb1,0,Length(var_bb1))      // var_value = 0x6F
var_value = XOR8(var_bb1,0)                      // var_value = 0x6F
var_value = XOR8(var_bb1)                        // var_value = 0x6F
var_bb1 = Byte_Concat(var_bb1, XOR(var_bb1))
// var_bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF, 0x6F}

```

2.43 SUM8()

Utilize SUM 8 bits algorithm to computes the checksum

Syntax 1

```

byte SUM8 (
    byte[],
    int,
    int
)

```

Parameters

byte[]	The input byte array
int	The starting index
0..array size-1	Valid
<0	Invalid. Returns the initial value 0

`>=array size` Invalid. Returns the initial value 0

`int` The number of elements to be computed.

If the number of elements <0, the calculation ends at the last element of the array

If the sum of starting index and number of element exceeds the array size, the calculation ends at the last element of the array.

Return

`byte` Checksum.

Note

```
byte[] var_bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}
var_value = SUM8(var_bb1,0,Length(var_bb1)) // var_value = 0x6D
var_value = SUM8(var_bb1,0,-1)              // var_value = 0x6D
var_value = SUM8(var_bb1,1,-1)              // var_value = 0x5D
var_value = SUM8(var_bb1,-1,-1)             // var_value = 0
```

Syntax 2

```
byte SUM8 (
    byte[],
    int
)
```

Note

Similar to Syntax1 with computing to the last element of the array

`SUM8(byte[], int) => SUM8(byte[], int, Length(byte[]))`

Syntax 3

```
byte SUM8 (
    byte[]
)
```

Note

Similar to Syntax1 with computing all the elements of the array

`SUM8(byte[]) => SUM8(byte[], 0, Length(byte[]))`

```
byte[] var_bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}
var_value = SUM8(var_bb1,0,Length(var_bb1)) // var_value = 0x6D
var_value = SUM8(var_bb1,0)                 // var_value = 0x6D
var_value = SUM8(var_bb1)                   // var_value = 0x6D
var_bb1 = Byte_Concat(var_bb1, SUM8(var_bb1)) // var_bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF,
0xFF, 0x6D}
```

2.44 SUM16()

Utilize SUM 16 bits algorithm to computes the checksum

Syntax 1

```
byte[] SUM16 (
    byte[],
    int,
    int
)
```

Parameters

`byte[]` The input byte array

`int` The starting index

`0..array size-1` Valid

`<0` Invalid. Returns the initial value 0

`>=array size` Invalid. Returns the initial value 0

`int` The number of elements to be computed.

If the number of elements `<0`, the calculation ends at the last element of the array

If the sum of starting index and number of element exceeds the array size, the calculation ends at the last element of the array.

Return

`byte[]` Checksum. The length is 16bits 2 bytes (The Checksum follows Big Endian)

Note

```
byte[] var_bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}
```

```
var_value = SUM16(var_bb1,0,Length(var_bb1)) // var_value = {0x04, 0x6D}
```

```
var_value = SUM16(var_bb1,0,-1) // var_value = {0x04, 0x6D}
```

```
var_value = SUM16(var_bb1,1,-1) // var_value = {0x04, 0x5D}
```

```
var_value = SUM16(var_bb1,-1,-1) // var_value = {0x00, 0x00}
```

Syntax 2

```
byte[] SUM16 (
    byte[],
    int
)
```

Note

Similar to Syntax1 with computing to the last element of the array

SUM16(byte[], int) => **SUM16**(byte[], int, Length(byte[]))

Syntax 3

```
byte[] SUM16 (  
    byte[]  
)
```

Note

Similar to Syntax1 with computing all the elements of the array

SUM16(byte[]) => **SUM16**(byte[], 0, Length(byte[]))

byte[] var_bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}

var_value = **SUM16**(var_bb1,0,Length(var_bb1)) // var_value = {0x04, 0x6D}

var_value = **SUM16**(var_bb1,0) // var_value = {0x04, 0x6D}

var_value = **SUM16**(var_bb1) // var_value = {0x04, 0x6D}

var_bb1 = **Byte_Concat**(var_bb1, **SUM16**(var_bb1)) // var_bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF,
0x04, 0x6D}

2.45 SUM32()

Utilize SUM 32 bits algorithm to computes the checksum

Syntax 1

```
byte[] SUM32 (  
    byte[],  
    int,  
    int  
)
```

Parameters

`byte[]` The input byte array

`int` The starting index

`0..array size-1` Valid

`<0` Invalid. Returns the initial value 0

`>=array size` Invalid. Returns the initial value 0

`int` The number of elements to be computed.

If the number of elements `<0`, the calculation ends at the last element of the array

If the sum of starting index and number of element exceeds the array size, the calculation ends at the last element of the array.

Return

`byte[]` Checksum. The length is 32bits 4 bytes (The Checksum follows Big Endian)

Note

```
byte[] var_bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}
```

```
var_value = SUM32(var_bb1,0,Length(var_bb1)) // var_value = {0x00, 0x00, 0x04, 0x6D}
```

```
var_value = SUM32(var_bb1,0,-1) // var_value = {0x00, 0x00, 0x04, 0x6D}
```

```
var_value = SUM32(var_bb1,1,-1) // var_value = {0x00, 0x00, 0x04, 0x5D}
```

```
var_value = SUM32(var_bb1,-1,-1) // var_value = {0x00, 0x00, 0x00, 0x00}
```

Syntax 2

```
byte[] SUM32 (  
    byte[],  
    int  
)
```

Note

Similar to Syntax1 with computing to the last element of the array

```
SUM32(byte[], int) => SUM32(byte[], int, Length(byte[]))
```

Syntax 3

```
byte[] SUM32 (  
    byte[]  
)
```

Note

Similar to Syntax1 with computing all the elements of the array

SUM32(byte[]) => **SUM32**(byte[], 0, Length(byte[]))

byte[] var_bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}

var_value = **SUM32**(var_bb1,0,Length(var_bb1)) // var_value = {0x00, 0x00, 0x04, 0x6D}

var_value = **SUM32**(var_bb1,0) // var_value = {0x00, 0x00, 0x04, 0x6D}

var_value = **SUM32**(var_bb1) // var_value = {0x00, 0x00, 0x04, 0x6D}

var_bb1 = **Byte_Concat**(var_bb1, **SUM32**(var_bb1))

// var_bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x04, 0x6D}

2.46 CRC16()

Utilize CRC 16 bits algorithm to computes the checksum

Syntax 1

```
byte[] CRC16 (  
    int,  
    byte[],  
    int,  
    int  
)
```

Parameters

int CRC16 algorithm (Reference <https://www.lammertbies.nl/comm/info/crc-calculation.html>)

0	CRC16	// initial value 0x0000	// Polynomial 0xA001
1	CRC16 (Modbus)	// initial value 0xFFFF	// Polynomial 0xA001
2	CRC16 (Sick)	// initial value 0x0000	// Polynomial 0x8005
3	CRC16-CCITT (0x1D0F)	// initial value 0x1D0F	// Polynomial 0x1021
4	CRC16-CCITT (0xFFFF)	// initial value 0xFFFF	// Polynomial 0x1021
5	CRC16-CCITT (XModem)	// initial value 0x0000	// Polynomial 0x1021
6	CRC16-CCITT (Kermit)	// initial value 0x0000	// Polynomial 0x8408
7	CRC16 Schunk Gripper	// initial value 0xFFFF	// Polynomial 0x1021

byte[] The input byte array

int The starting index

0..array size-1 Valid

<0 Invalid. Returns the initial value

>=array size Invalid. Returns the initial value

int The number of elements to be computed.

If the number of elements <0, the calculation ends at the last element of the array

If the sum of starting index and number of element exceeds the array size, the calculation ends at the last element of the array.

Return

byte[] Checksum. The length is 16bits 2 bytes (The checksum follows Big Endian)

Note

```
byte[] var_bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}
var_value = CRC16(0, var_bb1,0,Length(var_bb1))           // var_value = {0x2D, 0xD4}
var_value = CRC16(0, var_bb1,0,-1)                         // var_value = {0x2D, 0xD4}
var_value = CRC16(0, var_bb1,1,-1)                         // var_value = {0xEC, 0xC5}
var_value = CRC16(0, var_bb1,-1,-1)                        // var_value = {0x00, 0x00}
var_value = CRC16(3, var_bb1,0,Length(var_bb1))           // var_value = {0x42, 0x12}
var_value = CRC16(4, var_bb1,0,Length(var_bb1))           // var_value = {0xAB, 0xAE}
```

Syntax 2

```
byte[] CRC16 (
    int,
    byte[],
    int
)
```

Note

Similar to Syntax1 with computing to the last element of the array

CRC16(int, byte[], int) => CRC16(int, byte[], int, Length(byte[]))

Syntax 3

```
byte[] CRC16 (
    int,
    byte[]
)
```

Note

Similar to Syntax1 with computing all the elements of the array

CRC16(int, byte[]) => CRC16(int, byte[], 0, Length(byte[]))

```
byte[] var_bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}
```

```

var_value = CRC16(0, var_bb1, 0, Length(var_bb1))           // var_value = {0x2D, 0xD4}
var_value = CRC16(0, var_bb1, 0)                           // var_value = {0x2D, 0xD4}
var_value = CRC16(0, var_bb1)                               // var_value = {0x2D, 0xD4}
var_bb1 = Byte_Concat(var_bb1, CRC16(0, var_bb1))
// var_bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF, 0x2D, 0xD4}

```

Syntax 4

```

byte[] CRC16 (
    byte[],
    int,
    int
)

```

Note

Similar to Syntax1 with CRC16 algorithm as 0 CRC16

CRC16(byte[], int, int) => **CRC16**(0, byte[], int, int)

Syntax 5

```

byte[] CRC16 (
    byte[],
    int
)

```

Note

Similar to Syntax1 with CRC16 algorithm as 0 CRC16 and computing to the last element of the array

CRC16(byte[], int) => **CRC16**(0, byte[], int, Length(byte[]))

Syntax 6

```

byte[] CRC16 (
    byte[]
)

```

Note

Similar to Syntax1 with CRC16 algorithm as 0 CRC16 and computing all the elements of the array

CRC16(byte[]) => **CRC16**(0, byte[], 0, Length(byte[]))

2.47 CRC32()

Utilize CRC 32 bits algorithm to computes the checksum

Syntax 1

```
byte[] CRC32 (  
    byte[],  
    int,  
    int  
)
```

Parameters

`byte[]` The input byte array

`int` The starting index

`0..array size-1` Valid

`<0` Invalid. Returns the initial value 0

`>=array size` Invalid. Returns the initial value 0

`int` The number of elements to be computed.

If the number of elements `<0`, the calculation ends at the last element of the array

If the sum of starting index and number of element exceeds the array size, the calculation ends at the last element of the array.

Return

`byte[]` Checksum. The checksum length is 32bits 4 bytes (The checksum follows Big Endian)

Note

```
byte[] var_bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}
```

```
var_value = CRC32(var_bb1,0,Length(var_bb1)) // var_value = {0x43, 0xD5, 0xB9, 0xF8}
```

```
var_value = CRC32(var_bb1,0,-1) // var_value = {0x43, 0xD5, 0xB9, 0xF8}
```

```
var_value = CRC32(var_bb1,1,-1) // var_value = {0x08, 0xA5, 0x5B, 0xEB}
```

```
var_value = CRC32(var_bb1,-1,-1) // var_value = {0x00, 0x00, 0x00, 0x00}
```

Syntax 2

```
byte[] CRC32 (  
    byte[],  
    int  
)
```

Note

Similar to Syntax1 with computing to the last element of the array

```
CRC32(byte[], int) => CRC32(byte[], int, Length(byte[]))
```

Syntax 3

```
byte[] CRC32 (  
    byte[],  
    int,  
    int  
)
```

```
byte[]
```

```
)
```

Note

Similar to Syntax1 with computing all the elements of the array

```
CRC32(byte[]) => CRC32(byte[], 0, Length(byte[]))
```

```
byte[] var_bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}
```

```
var_value = CRC32(var_bb1,0,Length(var_bb1)) // var_value = {0x43, 0xD5, 0xB9, 0xF8}
```

```
var_value = CRC32(var_bb1,0) // var_value = {0x43, 0xD5, 0xB9, 0xF8}
```

```
var_value = CRC32(var_bb1) // var_value = {0x43, 0xD5, 0xB9, 0xF8}
```

```
var_bb1 = Byte_Concat(var_bb1, CRC32(var_bb1))
```

```
// var_bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF, 0x43, 0xD5, 0xB9, 0xF8}
```

2.48 ListenPacket()

Pack the string contents as the compatible protocol for the Listen Node (External Script Control Mode)

Syntax 1

```
string ListenPacket(  
    string,  
    string  
)
```

Parameters

string User defined Header. For empty string, Default string "TMSCT" will be applied

string The data section in Listen Node communication format

Return

string Packed data (Including header, data length and check sum)

Note

```
string var_data1 = "1,var_i++"
```

```
string var_data2 = "Hello World"
```

```
var_value = ListenPacket("TMSCT", var_data1) // $TMSCT,9,1,var_i++,*06\r\n
```

```
var_value = ListenPacket("", var_data2)
```

```
// $TMSCT,11,Hello World,*51\r\n // Error for TMSCT
```

```
var_value = ListenPacket("", "2,Techman Robot") // $TMSCT,15,2,Techman Robot,*57\r\n
```

```
var_value = ListenPacket("TMSTA", var_data2)
```

```
// $TMSTA,11,Hello World,*53\r\n // Error for TMSTA
```

```
var_value = ListenPacket("TMSTA", "00")
```

```
// $TMSTA,2,00,*41\r\n
```

Syntax 2

```
string ListenPacket(  
    string  
)
```

Parameters

string The data section in Listen Node communication format (With TMSCT header)

Return

string Packed data (Including header, data length and check sum)

Note

```
string var_data1 = "1,var_counter++"
```

```
// ScriptID, ScriptLanguage
```

```
var_value = ListenPacket(var_data1)
```

```
// $TMSCT,15,1,var_counter++,*26\r\n
```

2.49 ListenSend()

Send TMSTA, the communication protocol of Listen node, to the client devices connected to the Listen Server currently.

Syntax1

```
int ListenSend(  
    string,  
    int,  
    ?  
)
```

Parameters

string Target IP filtering such as 127.0.0.1 meaning to send to all client devices connecting from 127.0.0.1.

int TMSTA SubCmd numbering for sending self-defined data message only **90 .. 99**

? The value to send. Available types: byte, int, float, double, bool, and string.
Numeric values will be conversed in Little Endian, and string values will be converse in UTF8.

Return

int Return the result

0 sent successfully

-1 error. Listen Server is not starting.

-2 error. SubCmd must be between 90 and 99.

Syntax2

```
int ListenSend (
    int,
    ?
)
```

Parameters

int TMSTA SubCmd numbering for sending self-defined data message only 90 .. 99

? The value to send. Available types: byte, int, float, double, bool, and string.

Numeric values will be conversed in Little Endian, and string values will be converse in UTF8.

Return

int Return the result

0 sent successfully

-1 error. Listen Server is not starting.

-2 error. SubCmd must be between 90 and 99.

Note

No target IP filtering will result in sending data messages to all connected client devices.

Note

```
string var_ip = "127.0.0.1"

byte var_b = 100

var_value = ListenSend(var_ip, 10, var_b)
    // send 0x64 to ipfilter "127.0.0.1"    // var_value = -2 // SubCmd must be between 90 and 99.

var_value = ListenSend(var_ip, 90, var_b)
    // send 0x64 to ipfilter "127.0.0.1"    // var_value = -1 // Supposedly Listen Server is not starting.

var_value = ListenSend(var_ip, 90, var_b)
    // send 0x64 to ipfilter "127.0.0.1"    // var_value = 0    // sent successfully
    // IP filtering 127.0.0.1 and send to the devices connected to Listen Server via the IP.

    // $TMSTA,4,90,d,*06    // The value of 100 is conversed to 0x64.

var_value = ListenSend(var_ip, 90, 123456)
    // send 0x40 0xE2 0x01 0x00 to ipfilter "127.0.0.1"

    // $TMSTA,7,90,@?,*C2

    // The value of 123456 is conversed to 0x40 0xE2 0x01 0x00 (int, Little Endian)

var_value = ListenSend(90, "123.456")
```

```

// send 0x31 0x32 0x33 0x2E 0x34 0x35 0x36
// No target IP filtering will result in sending data messages to all connected client devices.
// $TMSTA,10,90,123.456,*7E
// The value of "123.456" is conversed to 0x31 0x32 0x33 0x2E 0x34 0x35 0x36 (string, UTF8).

byte[] var_bb = {100, 200}
var_value = ListenSend(90, var_bb)
// send 0x64 0xC8
// $TMSTA,5,90,d◆,*CF // The value of {100, 200} is conversed to 0x64 0xC8

string[] var_ss = {"T", "M", "達明機器人" }
var_value = ListenSend(90, var_ss)
// send 0x54 0x4D 0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA
// $TMSTA,20,90,TM達明機器人,*A1

```

2.50 VarSync()

Send the Variable object to TMManager (Robot Management System)

* When performing this function, the flow will not go on until the object is sent out successfully or the maximum retry times is reached.

Syntax 1

```

int VarSync (
    int,
    int,
    ?
)

```

Parameters

- | | |
|----------------|---|
| int | The maximum times to retry |
| <= 0 | Keep retrying as error occurred. |
| int | The time duration between two retries (millisecond) |
| < 0 | Invalid time duration. The default value, 1000ms, will be applied |
| ? | The string or string array. The name of variables to be sent. |
- Multiple items can be listed. If there are indefinite variables, they will be not be sent; other definite variables will be sent.
- * The item is the name of the variable, not what the variable equals to such that var_i goes with "var_i".
- * If the variable is listed, the value of the variable will be used to send the matched object.

Return

<code>int</code>	Sending times
<code>> 0</code>	Send success. The return value returns the sending times
<code>0</code>	Send failed
<code>-1</code>	TM Manager function is not enabled
<code>-9</code>	Invalid Parameters

Note

```
string var_s = "ABC"
string var_s1 = "var_s"
string[] var_ss = {"ABC", "var_s", "var_s1"}
var_value = VarSync(1, 1000, "var_s") // Send var_s variable object
var_value = VarSync(2, 2000, var_s)   // Send ABC variable object (Because the value of var_s is "ABC")
var_value = VarSync(3, 2000, var_ss)  // Send ABC, var_s, var_s1 variable object (From the value of var_ss string array)
var_value = VarSync(3, 2000, "var_ss") // Send var_ss variable object
var_value = VarSync(4, 2000, "var_ss", "var_s1", "ABC") // Send var_ss, var_s1, ABC variable object
```

Syntax 2

```
int VarSync (
    int,
    ?
)
```

Note

Same as Syntax 1 with the time between two retries defaults to 1000 ms.

VarSync(int, ?) => VarSync(int, 1000, ?)

Syntax 3

```
int VarSync (
    ?
)
```

Note

Same as Syntax 1 with the time between two retries defaults to 1000 ms without limit of times to retry

VarSync(?) => VarSync(0, 1000, ?)

3. Math Functions

3.1 abs()

Return the absolute value of the designate number

Syntax 1

```
int abs (  
    int  
)
```

Parameter

`int` Input number in integer

Return

`int` Return the absolute value of the input number in integer

Note

```
int var_i = 10  
var_value = abs(var_i)      // 10  
var_i = -10  
var_value = abs(var_i)      // 10
```

Syntax 2

```
float abs (  
    float  
)
```

Parameter

`float` Input number in float

Return

`float` Return the absolute value of the input number in float

Note

```
float var_f = 10.1  
var_value = abs(var_f)      // 10.1  
var_f = -10.1  
var_value = abs(var_f)      // 10.1
```

Syntax 3

```
double abs (  
    double
```

)

Parameter

`double` Input number in double

Return

`double` Return the absolute value of the input number in double

Note

```
double var_d = 10.8
var_value = abs(var_d)    // 10.8
var_d = -10.8
var_value = abs(var_d)    // 10.8
```

3.2 pow()

Return the power of the designate base and exponent

Syntax 1

```
int pow(
    int,
    double
)
```

Parameter

`int` Input base in integer
`double` Exponent

Return

`int` Return the power in integer

Syntax 2

```
float pow(
    float,
    double
)
```

Parameter

`float` Input base in float
`double` Exponent

Return

`float` Return the power in float

Syntax 3

```
double pow(  
    double,  
    double  
)
```

Parameter

`double` Input base in double
`double` Exponent

Return

`double` Return the power in double

Note

```
? int var_b = 100  
  
var_value = pow(var_b, 2) // 10000  
var_value = pow(var_b, -2) // 0 // 0.0001, but int type  
var_value = pow(var_b, 0.1) // 1 // 1.5848931924611136, but int type  
var_value = pow(var_b, 2.1) // 15848 // 15848.931924611141, but int type  
var_value = pow(var_b, -2.1) // 0 // 0.000063095734448019293, but int type  
  
? float var_b = -100  
  
var_value = pow(var_b, 2) // 10000  
var_value = pow(var_b, -2) // 0.0001  
var_value = pow(var_b, 0.2) // Error // NaN  
var_value = pow(var_b, 2.2) // Error // NaN  
var_value = pow(var_b, -2.2) // Error // NaN  
  
? double var_b = 100  
  
var_value = pow(var_b, 2) // 10000  
var_value = pow(var_b, -2) // 0.0001  
var_value = pow(var_b, 0.31) // 4.16869383470335  
var_value = pow(var_b, 2.31) // 41686.9383470336  
var_value = pow(var_b, -2.31) // 0.0000239883291901949
```

3.3 sqrt()

Return the square root of the designate number

Syntax 1

```
float sqrt (  
    float  
)
```

Parameter

`float` Input number in float

Return

`float` Return the square root in float

Syntax 2

```
double sqrt (  
    double  
)
```

Parameter

`double` Input number in double

Return

`double` Return the square root in double

Note

```
var_value = sqrt(100)      // 10  
var_value = sqrt(100.1234) // 10.005  
var_value = sqrt(0.1234)   // 0.3162278  
var_value = sqrt(-100)     // Error // NaN  
var_value = sqrt(-100.1234) // Error // NaN  
var_value = sqrt(-0.1234)  // Error // NaN
```

3.4 ceil()

Return a number rounded upward to its nearest integer.

Syntax 1

```
float ceil (  
    float  
)
```

Parameter

`float` input number in float

Return

`float` Return a number in float rounded upward to its nearest integer

Syntax 2

```
double ceil(  
    double  
)
```

Parameter

`double` input number in double

Return

`double` Return a number in double rounded upward to its nearest integer

Note

```
var_value = ceil(100)      // 100  
var_value = ceil(100.1234) // 101  
var_value = ceil(0.1234)   // 1  
var_value = ceil(-100)     // -100  
var_value = ceil(-100.1234) // -100  
var_value = ceil(-0.1234)  // 0
```

3.5 floor()

Return a number rounded downward to its nearest integer.

Syntax 1

```
float floor(  
    float  
)
```

Parameter

`float` input number in float

Return

`float` Return a number in float rounded downward to its nearest integer

Syntax 2

```
double floor(  
    double  
)
```

```
double
```

```
)
```

Parameter

`double` input number in double

Return

`double` Return a number in double rounded downward to its nearest integer

Note

```
var_value = floor(100)           // 100
var_value = floor(100.1234)      // 100
var_value = floor(0.1234)        // 0
var_value = floor(-100)          // -100
var_value = floor(-100.1234)     // -101
var_value = floor(-0.1234)      // -1
```

3.6 round()

Return a number rounded to its nearest integer.

Syntax 1

```
float round(
    float,
    int
)
```

Parameter

`float` input number in float
`int` digits after the returned decimal point (0 by default meaning the number is rounded to integer)

<code>0..15</code>	valid values
<code>< 0</code>	value invalid, will use 0 by default
<code>> 15</code>	value invalid, will use 0 by default

Return

`float` Return a number in float rounded to its nearest integer.

Syntax 2

```
float round(
    float
)
```

Note

Same as syntax 1. Obtain 0 digit after the decimal point by default.

round(float) => **round**(float, 0)

Syntax 3

```
double round(  
    double,  
    int  
)
```

Parameter

double	input number in double
int	digits after the returned decimal point (0 by default meaning the number is rounded to integer)
0..15	valid values
< 0	value invalid, will use 0 by default
> 15	value invalid, will use 0 by default

Return

double Return a number in double rounded to its nearest integer.

Syntax 4

```
double round(  
    double  
)
```

Note

Same as syntax 3. Obtain 0 digit after the decimal point by default.

round(double) => **round**(double, 0)

```
var_value = round(100)           // 100  
var_value = round(100.456)       // 100  
var_value = round(0.567)         // 1  
var_value = round(-100)          // -100  
var_value = round(-100.456)      // -100  
var_value = round(-0.567)        // -1  
var_value = round(100.345, 1)     // 100.3  
var_value = round(100.345, 2)     // 100.35  
var_value = round(-100.345, 1)    // -100.3  
var_value = round(-100.345, 2)    // -100.35  
var_value = round(-100.345, 16)   // -100
```

3.7 random()

Return a random number in float between 0 and 1 or in integer between the lower bound and the upper bound.

Syntax 1

```
float random(  
)
```

Parameter

`void` No input value required.

Return

`float` Return a random number in float between 0 and 1.

Note

```
var_value = random()      // 0.9473762  
var_value = random()      // 0.7764986  
var_value = random()      // 0.9911129
```

Syntax 2

```
int random(  
    int  
)
```

Parameter

`int` The upper bound of the random number

Return

`int` Return a random number in integer between 0 and the upper bound

Note

```
var_value = random(10)    // 8  
var_value = random(10)    // 1  
var_value = random(10)    // 5  
var_value = random(-10)   // 0 // The value of the upper bound must be larger than 0.
```

Syntax 3

```
int random(  
    int,  
    int  
)
```

Parameter

`int` The lower bound of the random number
`int` The upper bound of the random number must be larger than the lower bound, or it will return the value of the lower bound in integer.

Return

`int` Return a random number in integer between the lower bound and the upper bound.

Note

```
var_value = random(5, 10) // 8  
var_value = random(5, 10) // 8  
var_value = random(5, 10) // 6  
var_value = random(5, -1) // 5 // The upper bound is smaller than the lower bound. Returned the value of the  
lower bound in integer.
```

3.8 d2r()

Convert the value of degree to radian

Syntax 1

```
float d2r(  
    float  
)
```

Parameter

`float` Input the value of degree in float

Return

`float` Return the value of radian in float

Syntax 2

```
double d2r(  
    double  
)
```

Parameter

`double` Input the value of degree in double

Return

`double` Return the value of radian in double

Note

```
var_value = d2r(1)      // 0.01745329
```

3.9 r2d()

Convert the value of degree to radian to degree

Syntax 1

```
float r2d(  
    float  
)
```

Parameter

`float` Input the value of radian in float

Return

`float` Return the value of degree in float

Syntax 2

```
double r2d(  
    double  
)
```

Parameter

`double` Input the value of radian in double

Return

`double` Return the value of degree in double

Note

```
var_value = r2d(1)      // 57.29578
```

3.10 sin()

Return the sine of the input value of degree

Syntax 1

```
float sin(  
    float  
)
```

Parameter

`float` Input the value of degree in float

Return

`float` Return the sine of the input value of degree in float

Syntax 2

```
double sin (  
    double  
)
```

Parameter

`double` Input the value of degree in double

Return

`double` Return the sine of the input value of degree in double

Note

```
var_value = sin(0)           // 0  
var_value = sin(15)          // 0.258819  
var_value = sin(30)          // 0.5  
var_value = sin(60)          // 0.8660254  
var_value = sin(90)          // 1
```

3.11 cos()

Return the cosine of the input value of degree

Syntax 1

```
float cos (  
    float  
)
```

Parameter

`float` Input the value of degree in float

Return

`float` Return the cosine of the input value of degree in float

Syntax 2

```
double cos (  
    double  
)
```

Parameter

`double` Input the value of degree in double

Return

`double` Return the cosine of the input value of degree in double

Note

```
var_value = cos(0)           // 1
var_value = cos(15)          // 0.9659258
var_value = cos(30)          // 0.8660254
var_value = cos(45)          // 0.7071068
var_value = cos(60)          // 0.5
```

3.12 tan()

Return the tangent of the input value of degree

Syntax 1

```
float tan (
    float
)
```

Parameter

`float` Input the value of degree in float

Return

`float` Return the tangent of the input value of degree in float

Syntax 2

```
double tan (
    double
)
```

Parameter

`double` Input the value of degree in double

Return

`double` Return the tangent of the input value of degree in double

Note

```
var_value = tan(0)           // 0
var_value = tan(15)          // 0.2679492
var_value = tan(30)          // 0.5773503
var_value = tan(45)          // 1
var_value = tan(60)          // 1.732051
```

3.13 asin()

Return the arcsine of the input value in degree

Syntax 1

```
float asin(  
    float  
)
```

Parameter

`float` Input the sine value in float between -1 and 1

Return

`float` Return the arcsine of the input value of degree in float

Syntax 2

```
double asin(  
    double  
)
```

Parameter

`double` Input the sine value in double between -1 and 1

Return

`double` Return the arcsine of the input value of degree in double

Note

```
var_value = asin(0)           // 0  
var_value = asin(0.258819)    // 15  
var_value = asin(0.5)         // 30  
var_value = asin(0.8660254)   // 60  
var_value = asin(1)           // 90
```

3.14 acos()

Return the arccosine of the input value in degree

Syntax 1

```
float acos(  
    float  
)
```

Parameter

`float` Input the cosine value in float between -1 and 1

Return

`float` Return the degree value in float

Syntax 2

```
double acos (  
    double  
)
```

Parameter

`double` Input the cosine value in double between -1 and 1

Return

`double` Return the degree value in double

Note

```
var_value = acos(1)           // 0  
var_value = acos(0.9659258)   // 15  
var_value = acos(0.8660254)   // 30  
var_value = acos(0.7071068)   // 45  
var_value = acos(0.5)         // 60
```

3.15 atan()

Return the arctangent of the input value in degree

Syntax 1

```
float atan (  
    float  
)
```

Parameter

`float` Input the arctangent value in float

Return

`float` Return the degree value in float

Syntax 2

```
double atan (  
    double  
)
```

Parameter

`double` Input the arctangent value in double

Return

`double` Return the degree value in double

Note

```
var_value = atan(0)           // 0
var_value = atan(0.2679492)   // 15
var_value = atan(0.5773503)   // 30
var_value = atan(1)           // 45
var_value = atan(1.732051)    // 60
```

3.16 atan2()

Return the arctangent of the quotient of its arguments

Syntax 1

```
float atan2 (
    float,
    float
)
```

Parameter

`float` Input a number in float representing the Y coordinate
`float` Input a number in float representing the X coordinate

Return

`float` Return the degree value in float

Syntax 2

```
double atan2 (
    double,
    double
)
```

Parameter

`double` Input a number in double representing the Y coordinate
`double` Input a number in double representing the X coordinate

Return

`double` Return the degree value in double

Note

```
var_value = atan2(2, 1)       // 63.43495
```

```
var_value = atan2(1, 1)           // 45
var_value = atan2(-1, -1)        // -135
var_value = atan2(4, -3)         // 126.8699
```

3.17 log()

Return the natural logarithm of the input value

Syntax 1

```
float log(  
    float,  
    double  
)
```

Parameter

`float` Input value in float
`double` The base of the logarithm

Return

`float` Return the logarithm of the input value and the base in float

Syntax 2

```
double log(  
    double,  
    double  
)
```

Parameter

`double` Input value in double
`double` The base of the logarithm

Return

`double` Return the logarithm of the input value and the base in double

Note

```
var_value = log(16, 2)           // 4
var_value = log(16, 8)           // 1.333333
var_value = log(16, 10)          // 1.20412
var_value = log(16, 16)          // 1
```

Syntax 3

```
float log(  
    float  
)
```

Parameter

`float` Input value in float

Return

`float` Return the natural logarithm of the input value and the base e in float

Syntax 4

```
double log(  
    double  
)
```

Parameter

`double` Input value in double

Return

`double` Return the natural logarithm of the input value and the base e in double

Note

```
var_value = log(16, 2)           // 4  
var_value = log(16)              // 2.772589  
var_value = log(2)               // 0.6931472  
var_value = log(16)/log(2) // 2.772589/ 0.6931472 = 4.000000288539
```

3.18 log10()

Return the logarithm of the input value with the base 10

Syntax 1

```
float log10(  
    float  
)
```

Parameter

`float` Input value in float

Return

`float` Return the logarithm of the input value with the base 10 in float

Syntax 2

```
double log10 (
    double
)
```

Parameter

`double` Input value in double

Return

`double` Return the logarithm of the input value with the base 10 in double

Note

```
var_value = log(16, 10)           // 1.20412
var_value = log10(16)             // 1.20412
var_value = log(500, 10)          // 2.69897
var_value = log10(500)            // 2.69897
```

3.19 norm2()

Return the second norm of a specified vector.

Syntax 1

```
float norm2 (
    float[]
)
```

Parameter

`float[]` A vector whose second norm (or called Euclidean norm, vector magnitude) is to be found.

Return

`float` the second norm (or called Euclidean norm, vector magnitude) of a specified vector

Note

$$\|v\| = \sqrt{\sum_{i=1}^{i=N} |v_i|^2}$$

```
float[] var_vector1 = {3,4}
float[] var_vector2 = {3,4,5}
float[] var_vector3 = {3,4,5,6,8}
var_value = norm2(var_vector1)    // 5
var_value = norm2(var_vector2)    // 7.071068
var_value = norm2(var_vector3)    // 12.24745
```

3.20 dist()

Return the distance between the two coordinates.

Syntax 1

```
float dist(  
    float[],  
    float[]  
)
```

Parameter

float[]	The first coordinate $\{X_{1(mm)} \ Y_{1(mm)} \ Z_{1(mm)} \ RX_{1(^{\circ})} \ RY_{1(^{\circ})} \ RZ_{1(^{\circ})}\}$
float[]	The second coordinate $\{X_{2(mm)} \ Y_{2(mm)} \ Z_{2(mm)} \ RX_{2(^{\circ})} \ RY_{2(^{\circ})} \ RZ_{2(^{\circ})}\}$

Return

float	The distance between the two coordinates
-------	--

Note

```
float[] var_c1 = {100,200,100,30,50,20}  
float[] var_c2 = {100,100,100,50,50,10}  
var_value = dist(var_c1, var_c2)    // 100
```

3.21 trans()

Return the displacement and rotation angle from one specified point to another point.

Syntax 1

```
float[] trans(  
    float[],  
    float[],  
    bool  
)
```

Parameter

float[]	First Point	$\{X_{1(mm)} \ Y_{1(mm)} \ Z_{1(mm)} \ RX_{1(^{\circ})} \ RY_{1(^{\circ})} \ RZ_{1(^{\circ})}\}$
float[]	Second Point	$\{X_{2(mm)} \ Y_{2(mm)} \ Z_{2(mm)} \ RX_{2(^{\circ})} \ RY_{2(^{\circ})} \ RZ_{2(^{\circ})}\}$
bool	The reference coordinate	
false	Refer to the robot's base(default)	
true	Refer to the first point	

Return

float[]	The displacement and rotation angle from first point to second point
---------	--

$\{X_{trans} \ Y_{trans} \ Z_{trans} \ RX_{trans} \ RY_{trans} \ RZ_{trans}\}$

Return an empty array if unable to calculate.

Syntax 2

```
float[] trans(
    float[],
    float[]
)
```

Note

Same as syntax 1. By default, the reference coordinate is set to false for the robot's base.

Original Transformation Matrix = $\begin{bmatrix} R & P \\ 0 & 1 \end{bmatrix}$

$$R_n = \begin{bmatrix} \cos(RZ_n)\cos(RY_n) & -\sin(RZ_n)\cos(RX_n) + \cos(RZ_n)\sin(RY_n)\sin(RX_n) & \sin(RZ_n)\sin(RX_n) + \cos(RZ_n)\sin(RY_n)\cos(RX_n) \\ \sin(RZ_n)\cos(RY_n) & \cos(RZ_n)\cos(RX_n) + \sin(RZ_n)\sin(RY_n)\sin(RX_n) & -\cos(RZ_n)\sin(RX_n) + \sin(RZ_n)\sin(RY_n)\cos(RX_n) \\ -\sin(RY_n) & \cos(RY_n)\sin(RX_n) & \cos(RY_n)\cos(RX_n) \end{bmatrix}$$

$$\text{First Point} = \begin{bmatrix} R_1 & P_1 \\ 0 & 1 \end{bmatrix}$$

$$\text{Second Point} = \begin{bmatrix} R_2 & P_2 \\ 0 & 1 \end{bmatrix}$$

If *reference coordinate* is **false** (reference coordinate is Robot Base)

$$P_{trans} = P_2 - P_1$$

$$R_{trans} = R_2 * R_1^{-1}$$

If *reference coordinate* is **true** (reference coordinate is First Point)

$$P_{trans} = R_1^{-1} * (P_2 - P_1)$$

$$R_{trans} = R_1^{-1} * R_2$$

```
float[] var_P1 = {100, -200, 300, 10, 20, 60}
```

```
float[] var_P2 = {-400, 200, 50, -20, 30, -45}
```

```
float[] var_trans_RB = trans(var_P1, var_P2)
```

```
// {-500, 400, -250, -24.61587, -15.56518, -88.61369}
```

```
float[] var_trans_i = trans(var_P1, var_P2, true)
```

```
// {176.101, 588.3278, -308.8024, 3.745925, 23.13792, -92.46916}
```

3.22 inversetrans()

Return the displacement and rotation angle {x, y, z, rx, ry, rz} opposite to the input displacement and

rotation angle {x, y, z, rx, ry, rz}.

Syntax 1

```
float[] inversetrans (  
    float[],  
    bool  
)
```

Parameter

`float[]` The input displacement and rotation angle $\{X_o \ Y_o \ Z_o \ RX_o \ RY_o \ RZ_o\}$
`bool` The reference coordinate
`false` Refer to the robot's base(default)
`true` Refer to the input displacement and the rotation angle

Return

`float[]` The displacement and rotation angle $\{X_{inv} \ Y_{inv} \ Z_{inv} \ RX_{inv} \ RY_{inv} \ RZ_{inv}\}$
opposite to the input displacement and rotation angle $\{X_o \ Y_o \ Z_o \ RX_o \ RY_o \ RZ_o\}$
Return an empty array if unable to calculate.

Syntax 2

```
float[] inversetrans (  
    float[]  
)
```

Note

Same as syntax 1. By default, the reference coordinate is set to false for the robot's base.

Original Transformation Matrix = $\begin{bmatrix} R & P \\ 0 & 1 \end{bmatrix}$

$$R_n = \begin{bmatrix} \cos(RZ_n)\cos(RY_n) & -\sin(RZ_n)\cos(RX_n) + \cos(RZ_n)\sin(RY_n)\sin(RX_n) & \sin(RZ_n)\sin(RX_n) + \cos(RZ_n)\sin(RY_n)\cos(RX_n) \\ \sin(RZ_n)\cos(RY_n) & \cos(RZ_n)\cos(RX_n) + \sin(RZ_n)\sin(RY_n)\sin(RX_n) & -\cos(RZ_n)\sin(RX_n) + \sin(RZ_n)\sin(RY_n)\cos(RX_n) \\ -\sin(RY_n) & \cos(RY_n)\sin(RX_n) & \cos(RY_n)\cos(RX_n) \end{bmatrix}$$

Initial Point = $\begin{bmatrix} R_i & P_i \\ 0 & 1 \end{bmatrix}$

If *reference coordinate* is `false` (reference coordinate is Robot Base)

$$P_{inv} = -P_i$$
$$R_{inv} = R_i^{-1}$$

If *reference coordinate* is `true` (reference coordinate is the input {x, y, z, rx, ry, rz}, which is equivalent to inverse of Transformation Matrix)

$$P_{inv} = R_i^{-1} * (-P_i)$$

$$R_{inv} = R_i^{-1}$$

```
float[] var_P1 = {100, -200, 300, 10, 20, 60}
```

```
float[] var_inv_RB = inversetrans(var_P1)
```

```
// {-100, 200, -300, 12.48313, -18.59011, -60.28337}
```

```
float[] var_inv_i = inversetrans(var_P1, true)
```

```
// {218.381, 142.1322, -268.5297, 12.48313, -18.59011, -60.28337}
```

3.23 applytrans()

Return the terminal point computed by applied the displacement and rotation angle to the specified point.

Syntax 1

```
float[] applytrans (
    float[],
    float[],
    bool
)
```

Parameter

float[]	Initial point {X _i Y _i Z _i RX _i RY _i RZ _i }
float[]	the displacement and rotation angle {X _o Y _o Z _o RX _o RY _o RZ _o }
bool	The reference coordinate
false	Refer to the robot's base(default)
true	Refer to the initial point

Return

float[] the terminal point {X_t Y_t Z_t RX_t RY_t RZ_t} computed by applied the displacement and rotation angle to the initial point
Return an empty array if unable to calculate.

Syntax 2

```
float[] applytrans (
    float[],
    float[]
)
```

Note

Same as syntax 1. By default, the reference coordinate is set to false for the robot's base.

$$\text{Original Transformation Matrix} = \begin{bmatrix} R & P \\ 0 & 1 \end{bmatrix}$$

$$R_n = \begin{bmatrix} \cos(RZ_n)\cos(RY_n) & -\sin(RZ_n)\cos(RX_n) + \cos(RZ_n)\sin(RY_n)\sin(RX_n) & \sin(RZ_n)\sin(RX_n) + \cos(RZ_n)\sin(RY_n)\cos(RX_n) \\ \sin(RZ_n)\cos(RY_n) & \cos(RZ_n)\cos(RX_n) + \sin(RZ_n)\sin(RY_n)\sin(RX_n) & -\cos(RZ_n)\sin(RX_n) + \sin(RZ_n)\sin(RY_n)\cos(RX_n) \\ -\sin(RY_n) & \cos(RY_n)\sin(RX_n) & \cos(RY_n)\cos(RX_n) \end{bmatrix}$$

$$\text{Initial Point} = \begin{bmatrix} R_i & P_i \\ 0 & 1 \end{bmatrix}$$

If *reference coordinate* is **false** (reference coordinate is Robot Base)

$$P_t = P_i + P_{trans}$$

$$R_t = R_{trans} * R_i$$

If *reference coordinate* is **true** (reference coordinate is Initial point)

$$P_t = P_i + R_i * P_{trans}$$

$$R_t = R_i * R_{trans}$$

```
float[] var_P1 = {100, -200, 300, 10, 20, 60}
```

```
float[] var_P2 = {-400, 200, 50, -20, 30, -45}
```

```
float[] var_trans_RB = trans(var_P1, var_P2)
```

```
// {-500, 400, -250, -24.61587, -15.56518, -88.61369}
```

```
float[] var_trans_i = trans(var_P1, var_P2, true)
```

```
// {176.101, 588.3278, -308.8024, 3.745925, 23.13792, -92.46916}
```

```
float[] var_apply_RB = applytrans(var_P1, var_trans_RB)
```

```
// {-400, 200, 50, -20, 30, -45.00001}
```

```
float[] var_apply_i = applytrans(var_P1, var_trans_i, true)
```

```
// {-400, 200.0001, 49.99998, -20, 30, -45}
```

```
float[] var_inv_RB = inversetrans(var_P1)
```

```
// {-100, 200, -300, 12.48313, -18.59011, -60.28337}
```

```
float[] var_inv_i = inversetrans(var_P1, true)
```

```
// {218.381, 142.1322, -268.5297, 12.48313, -18.59011, -60.28337}
```

```
float[] var_apply_1 = applytrans(var_P1, var_inv_RB)
```

```
// {0, 0, 0, 1.488326E-06, 4.389056E-06, -4.243126E-06}
```

```
float[] var_apply_2 = applytrans(var_P1, var_inv_i, true)
```

```
// {-1.029038E-05, 7.456403E-05, 2.154642E-05, -4.266358E-06, 2.728826E-06, -3.719508E-06}
```

3.24 interpoint()

Return the interpolate point between two points according to the specified points and ratio

Syntax 1

```
float[] interpoint(  
    float[],  
    float[],  
    float  
)
```

Parameter

float[]	First Point $\{X_{1(\text{mm})} \ Y_{1(\text{mm})} \ Z_{1(\text{mm})} \ RX_{1(^{\circ})} \ RY_{1(^{\circ})} \ RZ_{1(^{\circ})}\}$
float[]	Second Point $\{X_{2(\text{mm})} \ Y_{2(\text{mm})} \ Z_{2(\text{mm})} \ RX_{2(^{\circ})} \ RY_{2(^{\circ})} \ RZ_{2(^{\circ})}\}$
float	Ratio

Return

float[] the linear interpolate point $\{X_i \ Y_i \ Z_i \ RX_i \ RY_i \ RZ_i\}$ between initial point and endpoint according to the ratio.
Return an empty array if unable to calculate.

Note

$$\begin{aligned} &\{X_i \ Y_i \ Z_i \ RX_i \ RY_i \ RZ_i\} \\ &= (\{X_2 \ Y_2 \ Z_2 \ RX_2 \ RY_2 \ RZ_2\} - \{X_1 \ Y_1 \ Z_1 \ RX_1 \ RY_1 \ RZ_1\}) \times Ratio \\ &+ \{X_1 \ Y_1 \ Z_1 \ RX_1 \ RY_1 \ RZ_1\} \end{aligned}$$

```
float[] var_P1 = {-388.3831,-199.8061,367.0702,177.4319,1.717448,-46.02005}  
float[] var_P2 = {-436.9584,115.7343,371.4378,179.4419,-42.86601,-96.91867}  
float[] var_interp = interpoint(var_P1, var_P2, 0.5)  
// {-412.6707,-42.0359,369.254,172.919,-20.6906,-69.3384}
```

3.25 changeref()

Return the new coordinate value described with the new coordinate system converted from the original coordinate value through the coordinate system conversion. In the process of the conversion, the physical position of the original point in the world of the coordinates will remain the same, the change takes effects on its descriptions of the reference coordinates and the corresponding

coordinate values.

Syntax 1

```
float[] changeref (  
    float[],  
    float[],  
    float[]  
)
```

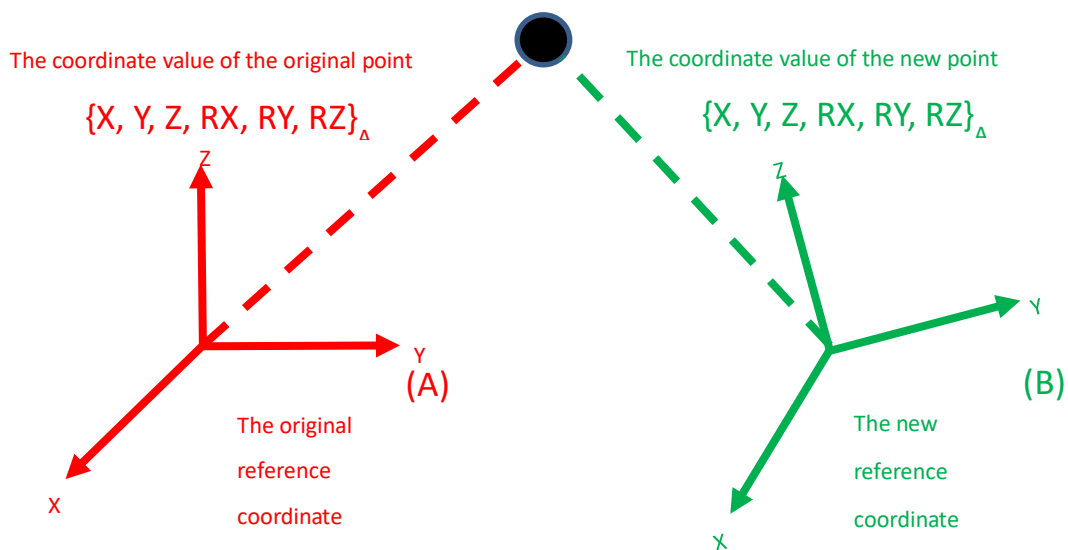
Parameter

`float[]` The coordinate value of the original point $\{X_o \ Y_o \ Z_o \ RX_o \ RY_o \ RZ_o\}_A$
`float[]` The original reference coordinate system $\{X_{oa} \ Y_{oa} \ Z_{oa} \ RX_{oa} \ RY_{oa} \ RZ_{oa}\}_A$
`float[]` The new reference coordinate system $\{X_n \ Y_n \ Z_n \ RX_n \ RY_n \ RZ_n\}_B$

Return

`float[]` The coordinate value of the new point $\{X_{nb} \ Y_{nb} \ Z_{nb} \ RX_{nb} \ RY_{nb} \ RZ_{nb}\}_B$
Return an empty array if unable to calculate.

Note



```
P1 = {-431.927, -140.6103, 368.7306, -179.288, -0.6893783, -105.8449}  
RobotBase = {0, 0, 0, 0, 0, 0}  
base1 = {-431.93, -140.61, 368.73, -57.70, -44.98, 33.62}  
float[] f0 = changeref(Point["P1"].Value, Base["RobotBase"].Value, Base["base1"].Value)  
// var_f0 = {0.002052, 0.000020, -0.002272, 113.9423, 14.9346, -123.1989}  
// Convert the value of "P1" in the coordinate system "RobotBase" to the value of a point in the coordinate system  
"base1"
```

Syntax 2

```
float[] changeref(  
    float[],  
    float[]  
)
```

Parameter

`float[]` The coordinate value of the original point $\{X_o \ Y_o \ Z_o \ RX_o \ RY_o \ RZ_o\}_A$
`float[]` The original reference coordinate system $\{X_{oa} \ Y_{oa} \ Z_{oa} \ RX_{oa} \ RY_{oa} \ RZ_{oa}\}_A$

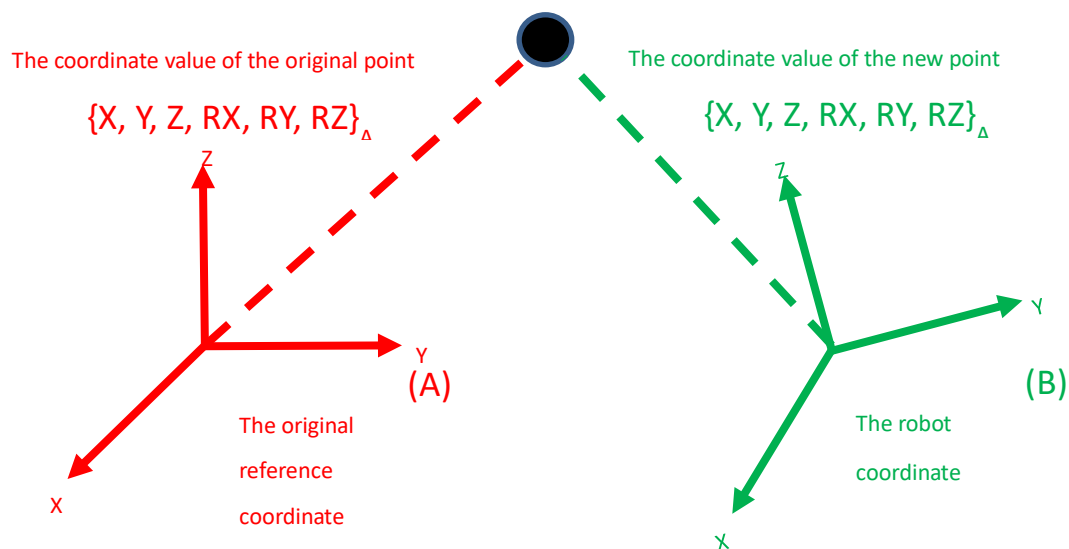
Return

`float[]` The coordinate value of the new point $\{X_{nr} \ Y_{nr} \ Z_{nr} \ RX_{nr} \ RY_{nr} \ RZ_{nr}\}_R$
Return an empty array if unable to calculate.

Note

The usage is the same as Syntax1's except assuming the robot coordinate system

$\{0 \ 0 \ 0 \ 0 \ 0 \ 0\}_R$ as the default new reference coordinate system.



```
base1 = {-431.93, -140.61, 368.73, -57.70, -44.98, 33.62}  
f0 = {0.002052, 0.000020, -0.002272, 113.9423, 14.9346, -123.1989}  
float[] f1 = changeref(f0, Base["base1"].Value)  
// f1 = {-431.927, -140.6103, 368.7306, -179.288, -0.6893424, -105.8449}
```

3.26 points2coord()

Based on the input points, calculate the coordinate plane of the input points and return the values of the plane parameters converted by the three points on the plane.

Syntax 1

```
float[] points2coord(  
    float[],  
    float[],  
    float[]  
)
```

Parameters

`float[]` The origin coordinates in the coordinate plane
 $X_{1(mm)} \ Y_{1(mm)} \ Z_{1(mm)} \ RX_{1(^\circ)} \ RY_{1(^\circ)} \ RZ_{1(^\circ)}$

`float[]` Any point on the X axis in the plane $X_{2(mm)} \ Y_{2(mm)} \ Z_{2(mm)} \ RX_{2(^\circ)} \ RY_{2(^\circ)} \ RZ_{2(^\circ)}$

`float[]` Any point on the +X and +Y in the plane
 $X_{3(mm)} \ Y_{3(mm)} \ Z_{3(mm)} \ RX_{3(^\circ)} \ RY_{3(^\circ)} \ RZ_{3(^\circ)}$

* The input points above are required to be described in the same coordinate system.

Return

`float[]` The value of the plane parameter formed by the three points on the plane
 $X_p(mm) \ Y_p(mm) \ Z_p(mm) \ RX_p(^\circ) \ RY_p(^\circ) \ RZ_p(^\circ)$
that coincided with the origin of the plane
 $X_{1(mm)} \ Y_{1(mm)} \ Z_{1(mm)}$
and calculated with the corners of
 $X_{2(mm)} \ Y_{2(mm)} \ Z_{2(mm)} \ RX_{2(^\circ)} \ RY_{2(^\circ)} \ RZ_{2(^\circ)}$
and
 $X_{3(mm)} \ Y_{3(mm)} \ Z_{3(mm)} \ RX_{3(^\circ)} \ RY_{3(^\circ)} \ RZ_{3(^\circ)}$

Note

Supposed there are three points: P1, P2, and P3

Point["P1"].Value = {389.9641,-4.797323,416.2175,177.3384,0.9190881,91.07789}

Point["P2"].Value = {365.0222,137.3036,423.2249,177.4598,0.9549707,112.4033}

Point["P3"].Value = {546.7307,94.02614,385.1812,176.3468,0.5975906,95.82078}

Base["points2coord_b1"].Value = **points2coord**(Point["P1"].Value, Point["P2"].Value, Point["P3"].Value)

// {389.9641,-4.7973,416.2175,-168.6551,-2.7807,99.9553}

Point["P4"].Value = {0,0,0,0,0,0}

ChangeBase("points2coord_b1")

PTP("CPP",Point["P4"].Value,10,200,0,false)

Syntax 2

```
float[] points2coord(  
    float, float, float,
```

```
float, float, float,  
float, float, float
```

)

Parameter

<code>float, float, float</code>	The origin coordinates in the coordinate plane to calculate $X_{1(mm)} \ Y_{1(mm)} \ Z_{1(mm)}$
<code>float, float, float</code>	The coordinates of any point on the X axis in the coordinate plane to calculate $X_{2(mm)} \ Y_{2(mm)} \ Z_{2(mm)}$
<code>float, float, float</code>	The coordinates of any point in the coordinate plane to calculate $X_{3(mm)} \ Y_{3(mm)} \ Z_{3(mm)}$

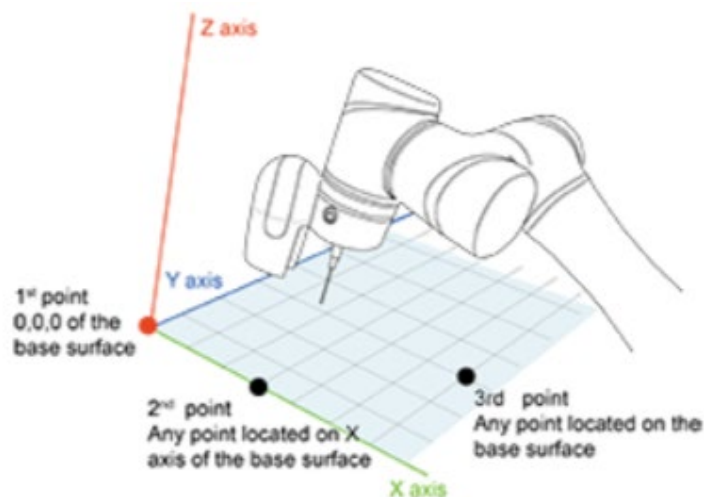
* The input points above are required to be described in the same coordinate system.

Return

`float[]` Definition parameters of the coordinate plane to calculate.
 $X_p(mm) \ Y_p(mm) \ Z_p(mm) \ RX_p(^{\circ}) \ RY_p(^{\circ}) \ RZ_p(^{\circ})$

Note

```
Base["points2coord_2"].Value = points2coord(260,0,360,260,100,360,100,0,360)  
// {260,0,360,0,0,90}  
Point["P1"].Value = {0,0,0,180,0,0}  
ChangeBase("points2coord_2")  
PTP("CPP",Point["P1"].Value,10,200,0,false)
```



3.27 intercoord()

Convert two input planes into a new plane.

Syntax 1

```
float[] intercoord(
    float[],
    float[]
)
```

Parameters

float[] The first input plane $X_{1(mm)}$ $Y_{1(mm)}$ $Z_{1(mm)}$ $RX_{1(^{\circ})}$ $RY_{1(^{\circ})}$ $RZ_{1(^{\circ})}$
 float[] The second input plane $X_{2(mm)}$ $Y_{2(mm)}$ $Z_{2(mm)}$ $RX_{2(^{\circ})}$ $RY_{2(^{\circ})}$ $RZ_{2(^{\circ})}$
 * The input points above are required to be described in the same coordinate system.

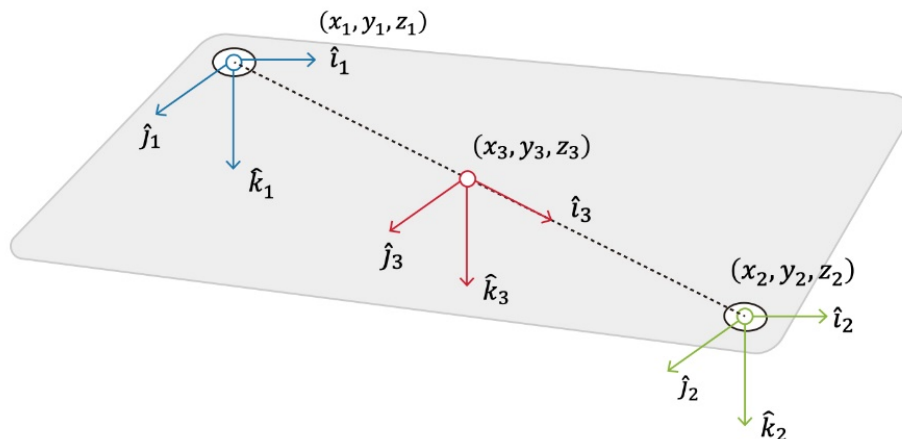
Return

float[] The new plane covered by the two input planes.
 $X_{3(mm)}$ $Y_{3(mm)}$ $Z_{3(mm)}$ $RX_{3(^{\circ})}$ $RY_{3(^{\circ})}$ $RZ_{3(^{\circ})}$

Note

$$\begin{aligned} X_3 &= (X_1 + X_2)/2 \\ Y_3 &= (Y_1 + Y_2)/2 \\ Z_3 &= (Z_1 + Z_2)/2 \\ \hat{i}_3 &= (X_2 - X_1, Y_2 - Y_1, Z_2 - Z_1) \\ \hat{j}_3 &= (\hat{k}_1 \times \hat{i}_3) \\ \hat{k}_3 &= (\hat{i}_1 \times \hat{j}_3) \end{aligned}$$

$$\begin{aligned} x_3 &= (x_1 + x_2)/2 \\ y_3 &= (y_1 + y_2)/2 \\ z_3 &= (z_1 + z_2)/2 \\ \hat{i}_3 &= (x_2 - x_1, y_2 - y_1, z_2 - z_1) \\ \hat{j}_3 &= \hat{k}_1 \times \hat{i}_3 \\ \hat{k}_3 &= \hat{i}_3 \times \hat{j}_3 \end{aligned}$$



Supposed there are vision landmark jobs: vb_1 and vb_2

vb_1 outputs Base["vision_vb_1"].Value = {-69.73,380.02,141.79,-176.11,1.13,-121.27}

vb_2 outputs Base["vision_vb_2"].Value = {58.81,613.03,140.7,171.62,-0.89,0.8}

```
Base["mix_base"].Value = intercoord(Base["vision_vb_1"].Value, Base["vision_vb_2"].Value)  
// {-5.46,496.525,141.245,176.0655,0.2347,61.1167}
```

4. File Functions

- The file functions are capable of operations related to file reading, writing, or inquiry.
- File path

1. Local path. Available in the directories named **TextFiles** or **XmlFiles** only.

<code>FileName.txt</code>	The directory default to <code>.\TextFiles</code> (File path the same as <code>.\TextFiles\FileName.txt</code>)
<code>.\TextFiles\FileName.txt</code>	The file is in the local directory named TextFiles .
<code>.\XmlFiles\FileName.xml</code>	The file is in the local directory named XmlFiles .
<code>.\XmlFiles\folder\file</code>	The subdirector is in the local directory named XmlFiles .
<code>..\folder</code>	Unavailable.
<code>.\TextFiles\..\..\folder</code>	Unavailable.

Void for absolute paths.

<code>C:\file1</code>	Void.
<code>D:\folder\file2</code>	Void.
<code>\TextFiles\FileName.txt</code>	Void.

2. External device path. Available to USB drives or SSD labelled **TMROBOT**.

<code>\USB\TMROBOT</code>	The root directory of the exteranl USB drive.
---------------------------	---

3. Remote path. Available with the Network service in TMflow.

<code>\\127.0.0.1\shared</code>	SMB / CIFS
<code>ftp://127.0.0.1</code>	FTP

- The path is not case sensitive. For example, the paths below all point to the same file.

`.\TextFiles\FileName.txt`
`.\textfiles\fileName.txt`
`.\Textfiles\Filename.TXT`

- The path is avaiable for pointing to subdirectories such as:

`subfolder\file`
`.\TextFiles\subfolder1\subfolder2\file`
`.\XmlFiles\subfolder\file`
`\USB\TMROBOT\subfolder\file`
`\\127.0.0.1\shared\subfolder\file`

- The maximum file size is limited to **2MB (2097152 Bytes)**.

- The type of the array to read or write depends on the definition of the array.

4.1 File_ReadBytes()

Read the file content and return in the type of byte[].

Syntax 1

```
byte[] File_ReadBytes (
    string
)
```

Parameter

string File path

Return

byte[] Return the file content in the type of byte[].

Note

```
.\TextFiles\SampleFile1.txt
1| 1Hello World!
2| 1Hello TM Robot!
```

```
byte[] var_bb1 = File_ReadBytes("sampleFile1.txt")
```

```
// {0x31,0x48,0x65,0x6C,0x6C,0x6F,0x20,0x57,0x6F,0x72,0x6C,0x64,0x21,0x0D,0x0A,
    0x31,0x48,0x65,0x6C,0x6C,0x6F,0x20,0x54,0x4D,0x20,0x52,0x6F,0x62,0x6F,0x74,0x21}
```

```
byte[] var_bb2 = File_ReadBytes(".\TextFiles\SampleFile1.txt")
```

```
// {0x31,0x48,0x65,0x6C,0x6C,0x6F,0x20,0x57,0x6F,0x72,0x6C,0x64,0x21,0x0D,0x0A,
    0x31,0x48,0x65,0x6C,0x6C,0x6F,0x20,0x54,0x4D,0x20,0x52,0x6F,0x62,0x6F,0x74,0x21}
```

```
byte[] var_bb3 = File_ReadBytes("C:\SampleFile1.txt") // Error. Void for absolute paths.
```

```
byte[] var_bb4 = File_ReadBytes(".\SampleFile1.txt") // Error. The file is in the local directory named
                                                    TextFiles or XmlFiles.
```

```
byte[] var_bb5 = File_ReadBytes("SampleFileXX.txt") // Error. The file does not exist.
```

4.2 File_ReadText()

Read the file content and return in the type of string.

Syntax 1

```
string File_ReadText (
    string
```

)

Parameter

`string` File path

Return

`string` Return the file content in the type of string.

Note

```
.\TextFiles\SampleFile1.txt
1| 1Hello World!
2| 1Hello TM Robot!
```

```
string var_s1 = File_ReadText("sampleFile1.txt")           // "1Hello World!\u000A1Hello TM Robot!"
string var_s2 = File_ReadText(".\TextFiles\SampleFile1.txt") // "1Hello World!\u000A1Hello TM Robot!"
* \u000A denotes a new line character but not a string value.
```

```
string var_s3 = File_ReadText("C:\SampleFile1.txt") // Error. Void for absolute paths.
string var_s4 = File_ReadText(".\SampleFile1.txt")  // Error. The file is in the local directory named
                                                    TextFiles or XmlFiles.
```

4.3 File_ReadLines()

Read the file content and return in the type of string separated by new line characters .

Syntax 1

```
string[] File_ReadLines (
    string
)
```

Parameter

`string` File path

Return

`string[]` Return the file content in the type of string separated by new line characters.

Syntax 2

```
string[] File_ReadLines (
    string,
    int,
    int
)
```

Parameter

<code>string</code>	File path
<code>int</code>	The number of the line to start to read
<code>int</code>	The amount of the lines to read

Return

<code>string[]</code>	Return the file content in the type of string separated by new line characters. If <code>the number of the line to start to read <= 0</code> , it returns an empty array. If <code>the number of the line to start to read > the total number of lines</code> , it returns an empty array. If <code>the amount of the lines to read <= 0</code> , it returns from the first line to the last line. If <code>the amount of the lines to read > the total number of lines</code> , it returns from the first line to star to read to the last line.
-----------------------	---

Syntax 3

```
string[] File_ReadLines(  
    string,  
    int  
)
```

Note

Same as Syntax 2 with the parameter of amount of the lines to read defaults to 0 and returns to the last line. `File_ReadLines(string,int,int) => File_ReadLines(string,int,0)`

Note

```
.\TextFiles\SampleFile2.txt  
1| 2Hello World!  
2| 2Hello TM Robot!  
3| 2Hi TM Robot!
```

```
string[] var_ss = {}  
  
var_ss = File_ReadLines("SampleFile2.txt") // {"2Hello World!", "2Hello TM Robot!", "2Hi TM Robot!"}  
var_ss = File_ReadLines("SampleFile2.txt", 1, 2) // {"2Hello World!", "2Hello TM Robot!"}  
var_ss = File_ReadLines("SampleFile2.txt", 2, 2) // {"2Hello TM Robot!", "2Hi TM Robot!"}  
var_ss = File_ReadLines("SampleFile2.txt", 3, 2) // {"2Hi TM Robot!"} // Tops the total number of lines.  
Returns to the last line.  
  
var_ss = File_ReadLines("SampleFile2.txt", 0) // {} // empty array  
var_ss = File_ReadLines("SampleFile2.txt", 4) // {} // empty array
```

```

int var_len = Length(var_ss) // 0

var_ss = File_ReadLines("SampleFile2.txt", 3, 0) // {"2Hi TM Robot!"} // Returns from line 3 to the last line.

.\TextFiles\SampleFile3.txt
1|

var_ss = File_ReadLines("SampleFile3.txt") // {} // var_ss[0] = ""
var_len = Length(var_ss) // 1

```

4.4 File_NextLine()

Record the last read file path, and continue to read the next line of the file content or open the file to read.

Syntax1

```

string File_NextLine (
    string
)

```

Parameter

string File path

Return

string If the same as the the last read file path, it returns the next line of the file content. If different from the last read file path, it opens the file and returns the first line of the file content. If read the end of the file, it returns an empty string.

Syntax 2

```

string File_NextLine (
    string,
    bool
)

```

Parameter

string File path

bool Whether open the file to read or not

false Try the file path. Continue to read the next line if the same. Open the file to read if different.

true Open the file and read the first line.

Return

string Whether open the file to read or not false

If the same as the the last read file path, it returns the next line of the file content.

If different from the last read file path, it opens the file and returns the first line of the file content.

Whether open the file to read or not `true`

It opens the file and returns the first line of the file content.

If read the end of the file, it returns an empty string.

Syntax 3

```
string File_NextLine(  
)
```

Parameter

`void` No input value

Return

`string` Return the next line of the file content in the last record to read or returns an empty string if not read.

Note

```
.\TextFiles\SampleFile4.txt  
1| 4Hello World!  
2|  
3| 4Hello TM Robot!
```

```
.\TextFiles\SampleFile5.txt  
1| 5Hello World!  
2| 5Hello TM Robot!  
3| 5Hi TM Robot!
```

```
string var_s = ""
```

```
var_s = File_NextLine()           // ""           // Not open the file to read.  
var_s = File_NextLine("SampleFile4.txt") // "4Hello World!"  
var_s = File_NextLine("SampleFile4.txt") // ""           // Continue to read the next line.  
var_s = File_NextLine("SampleFile5.txt") // "5Hello World!" // Different file path. Open the file to  
read.  
var_s = File_NextLine("SampleFile4.txt") // "4Hello World!" // Different file path. Open the file to  
read.  
var_s = File_NextLine("SampleFile4.txt") // ""           // Continue to read the next line  
var_s = File_NextLine("SampleFile4.txt") // "4Hello TM Robot!"  
var_s = File_NextLine("SampleFile4.txt") // ""           // Continue to read the next line (to  
the EOF)  
var_s = File_NextLine("SampleFile4.txt", true) // "4Hello World!" // Open the file to read the first line.
```

```

var_s = File_NextLine("SampleFile4.txt", true) // "4Hello World!" // Open the file to read the first line.
var_s = File_NextLine("SampleFile4.txt", false) // "" // Continue to read the next line
var_s = File_NextLine() // "4Hello TM Robot!"

```

- * To determine a blank line or the end of the file, use syntax 4 with the size of string[].
- * Or, use File_NextEOF() to determine the end of the file.

Syntax 4

```

string[] File_NextLine(
    string,
    int
)

```

Parameter

string	File path
int	Parameters to read
0	Try the file path. Continue to read the next line if the same. Open the file to read if different.
1	Open the file and read the first line.
2	Open the file without reading. Returns a empty array.

Return

string[]	Return strings in the next line of the file content in an array. If the array size is 1, it denotes read the strings in the next line line. If the array size is 0, it denotes read the end of the file already.
----------	--

Syntax 5

```

string[] File_NextLine(
    int
)

```

Parameter

int	Parameters to read
0	Try the file path. Continue to read the next line if the same. Open the file to read if different.
1	Open the file and read the first line.
2	Open the file without reading. Returns a empty array.

Return

string[]	Return strings in the next line of the file content in an array in the last record to read or an empty string array if not read. If the array size is 1, it denotes read the strings in the next line line.
----------	--



If the array size is 0, it denotes read the end of the file already.

Note

```
.\TextFiles\SampleFile4.txt
1| 4Hello World!
2|
3| 4Hello TM Robot!
```

```
.\TextFiles\SampleFile5.txt
1| 5Hello World!
2| 5Hello TM Robot!
3| 5Hi TM Robot!
```

```
string[] var_ss = {}

var_ss = File_NextLine(0) // {} // Not open the file to read.

var_ss = File_NextLine("SampleFile4.txt", 0) // {"4Hello World!"}

var_ss = File_NextLine("SampleFile4.txt", 0) // {} // Continue to read the next line.

var_ss = File_NextLine("SampleFile5.txt", 0) // {"5Hello World!"} // Different file path. Open the file to
read.

var_ss = File_NextLine("SampleFile4.txt", 0) // {"4Hello World!"} // Different file path. Open the file to
read.

var_ss = File_NextLine("SampleFile4.txt", 0) // {} // Continue to read the next line.

int var_len = Length(var_ss) // 1

var_ss = File_NextLine("SampleFile4.txt", 0) // {"4Hello TM Robot!"}

var_ss = File_NextLine("SampleFile4.txt", 0) // {} // Continue to read the next line (to
the EOF)

var_len = Length(var_ss) // 0

var_ss = File_NextLine("SampleFile4.txt", 1) // {"4Hello World!"} //Open the file and read the first line.

var_ss = File_NextLine("SampleFile4.txt", 2) // {} //Open the file without reading.

var_len = Length(var_ss) // 0

var_ss = File_NextLine("SampleFile4.txt") // {"4Hello World!"} // Continue to read the next line.

var_ss = File_NextLine(0) // {} // Continue to read the next line.

var_ss = File_NextLine(0) // {"4Hello TM Robot!"}

var_ss = File_NextLine(0) // {}
```

4.5 File_NextEOF()

Try the last read file path for reading to the end of the file already.

Syntax 1

```
bool File_NextEOF(
)
```

Parameter

`void` No input value but needs to use with `File_NextLine()`

Return

`bool` Return true if not read.

`false` Not read to the end of the file.

`true` Not open the file or read to the end of the file.

Note

```
.\TextFiles\SampleFile4.txt
```

```
1| 4Hello World!
```

```
2|
```

```
3| 4Hello TM Robot!
```

```
bool var_eof = File_NextEOF()           // true    // Not open the file to read.
```

```
string var_s = ""
```

```
var_s = File_NextLine("SampleFile4.txt") // "4Hello World!"
```

```
var_eof = File_NextEOF()                // false
```

```
var_s = File_NextLine("SampleFile4.txt") // ""
```

```
var_eof = File_NextEOF()                // false
```

```
var_s = File_NextLine("SampleFile4.txt") // 4Hello TM Robot!"
```

```
var_eof = File_NextEOF()                // true
```

```
var_s = File_NextLine("SampleFile4.txt") // ""
```

```
File_NextLine("SampleFile4.txt", 2)      // Open the file without reading.
```

```
var_eof = File_NextEOF()                // false
```

4.6 File_WriteBytes()

Put the data content into an array in byte and write to a file.

Syntax 1

```
bool File_WriteBytes (  
    string,  
    ?,  
    int,  
    int,  
    int  
)
```

Parameter

<code>string</code>	File path						
<code>?</code>	Values to write. Eligible for integers, floating-point numbers, booleans, strings, or arrays. Values will be converted with Little Endian, and strings will be converted with UTF8.						
<code>int</code>	Overwrite the file or append to the file. <table><tr><td><code>0</code></td><td>Over the file. If not existed, create a new file.</td></tr><tr><td><code>1</code></td><td>Append to the file. If not existed, create a new file.</td></tr></table>	<code>0</code>	Over the file. If not existed, create a new file.	<code>1</code>	Append to the file. If not existed, create a new file.		
<code>0</code>	Over the file. If not existed, create a new file.						
<code>1</code>	Append to the file. If not existed, create a new file.						
<code>int</code>	The starting index of the value to write (eligible for strings and arrays) <table><tr><td><code>0 .. length-1</code></td><td>Legitimate value</td></tr><tr><td><code>< 0</code></td><td>Illegitimate value. The starting index will be set to 0.</td></tr><tr><td><code>>= length</code></td><td>Illegitimate value. The starting index will be set to 0.</td></tr></table>	<code>0 .. length-1</code>	Legitimate value	<code>< 0</code>	Illegitimate value. The starting index will be set to 0.	<code>>= length</code>	Illegitimate value. The starting index will be set to 0.
<code>0 .. length-1</code>	Legitimate value						
<code>< 0</code>	Illegitimate value. The starting index will be set to 0.						
<code>>= length</code>	Illegitimate value. The starting index will be set to 0.						
<code>int</code>	The length of the value to write (eligible for strings and arrays) <table><tr><td><code><= 0</code></td><td>Write from the starting index to the end of the data.</td></tr><tr><td><code>> 0</code></td><td>Write from the starting index for a specified number of the length up to the data ends.</td></tr></table>	<code><= 0</code>	Write from the starting index to the end of the data.	<code>> 0</code>	Write from the starting index for a specified number of the length up to the data ends.		
<code><= 0</code>	Write from the starting index to the end of the data.						
<code>> 0</code>	Write from the starting index for a specified number of the length up to the data ends.						

Return

<code>bool</code>	<code>true</code>	Write successfully.
	<code>false</code>	Write unsuccessfully.

Syntax 2

```
bool File_WriteBytes (  
    string,  
    ?,  
    int,  
    int  
)
```

Note

Same as Syntax 1. Set the length of the value to write to 0 writing up to the data ends.

File_WriteBytes(string,?,int,int) => File_WriteBytes(string,?,int,int,0)

Syntax 3

```
bool File_WriteBytes (  
    string,  
    ?,  
    int  
)
```

Note

Same as Syntax 1. Set the starting index and the length of the value to write to 0 writing up to the data

ends.

File_WriteBytes(string,?,int) => **File_WriteBytes**(string,?,int,0,0)

Syntax 4

```
bool File_WriteBytes (  
    string,  
    ?  
)
```

Note

Same as Syntax 1. Set the starting index and the length of the value to write to 0 overwriting the file up to the data ends.

File_WriteBytes(string,?) => **File_WriteBytes**(string,?,0,0,0)

```
byte[] var_bb1 = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08}
```

```
byte[] var_bb2 = {0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38}
```

```
byte[] var_bb3 = {}
```

```
bool var_flag = false
```

```
var_flag = File_WriteBytes("writebytes.txt", var_bb1) // Overwrite var_bb1 to the file
```

```
writebytes.txt  
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  
00000000 00 01 02 03 04 05 06 07 08
```

```
var_flag = File_WriteBytes("writebytes.txt", var_bb2) // Overwrite var_bb2 to the file
```

```
writebytes.txt  
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  
00000000 30 31 32 33 34 35 36 37 38
```

```
var_flag = File_WriteBytes("writebytes.txt", var_bb3) // Overwrite var_bb3 to the file
```

```
writebytes.txt  
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  
00000000
```

```
File_WriteBytes("writebytes.txt", var_bb1, 1) // Append var_bb1 to the file
```

```
writebytes.txt  
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  
00000000 00 01 02 03 04 05 06 07 08
```

```
File_WriteBytes("writebytes.txt", var_bb2, 1) // Append var_bb2 to the file
```

```
writebytes.txt  
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  
00000000 00 01 02 03 04 05 06 07 08 30 31 32 33 34 35 36  
00000010 37 38
```

```
File_WriteBytes("writebytes.txt", var_bb1, 1, 3) // Append var_bb1 to the file starting from index 3 to  
the end.
```

```
writebytes.txt  
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  
00000000 00 01 02 03 04 05 06 07 08 30 31 32 33 34 35 36  
00000010 37 38 03 04 05 06 07 08
```

```
File_WriteBytes("writebytes.txt", var_bb2, 1, 3, 2) // Append var_bb2 to the file starting from index 3 for
                                                    the length of 2
```

```
writebytes.txt
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 00 01 02 03 04 05 06 07 08 30 31 32 33 34 35 36
00000010 37 38 03 04 05 06 07 08 33 34
```

```
File_WriteBytes("writebytes.txt", var_bb1, 1, -1)
```

```
// -1 illegitimate value // Append var_bb1 to the file starting from index 0 to the end.
```

```
writebytes.txt
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 00 01 02 03 04 05 06 07 08 30 31 32 33 34 35 36
00000010 37 38 03 04 05 06 07 08 33 34 00 01 02 03 04 05
00000020 06 07 08
```

```
File_WriteBytes("writebytes.txt", var_bb2, 1, 0, 100)
```

```
// Append var_bb2 to the file starting from index 0 for the length of 100 or to the end.
```

```
writebytes.txt
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 00 01 02 03 04 05 06 07 08 30 31 32 33 34 35 36
00000010 37 38 03 04 05 06 07 08 33 34 00 01 02 03 04 05
00000020 06 07 08 30 31 32 33 34 35 36 37 38
```

```
? byte var_n1 = 100 // Convert the value with Little Endian.
```

```
File_WriteBytes("writebytes2.txt", var_n1, 1)
```

```
writebytes2.txt
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 64
```

```
? byte[] var_n2 = {100, 200} // Convert every value in the array with Little Endian one after another.
```

```
File_WriteBytes("writebytes2.txt", var_n2, 1)
```

```
writebytes2.txt
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 64 64 C8 10 27 00 00
```

```
? int var_n3 = 10000
```

```
File_WriteBytes("writebytes2.txt", var_n3, 1)
```

```
writebytes2.txt
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 64 64 C8 10 27 00 00
```

```
? int[] var_n4 = {10000, 20000, 80000}
```

```
File_WriteBytes("writebytes2.txt", var_n4, 1)
```

```
writebytes2.txt
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 64 64 C8 10 27 00 00 10 27 00 00 20 4E 00 00 80
00000010 38 01 00
```

? float var_n5 = 1.234

File_WriteBytes("writebytes2.txt", var_n5, 1)

```
writebytes2.txt
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 64 64 C8 10 27 00 00 10 27 00 00 20 4E 00 00 80
00000010 38 01 00 B6 F3 9D 3F
```

? float[] var_n6 = {1.23, 4.56, -7.89}

File_WriteBytes("writebytes2.txt", var_n6, 1)

```
writebytes2.txt
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 64 64 C8 10 27 00 00 10 27 00 00 20 4E 00 00 80
00000010 38 01 00 B6 F3 9D 3F A4 70 9D 3F 85 EB 91 40 E1
00000020 7A FC C0
```

? double var_n7 = -1.2345

File_WriteBytes("writebytes3.txt", var_n7, 1)

```
writebytes3.txt
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 8D 97 6E 12 83 C0 F3 BF
```

? double[] var_n8 = {1.23, -7.89}

File_WriteBytes("writebytes3.txt", var_n8, 1)

```
writebytes3.txt
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 8D 97 6E 12 83 C0 F3 BF AE 47 E1 7A 14 AE F3 3F
00000010 8F C2 F5 28 5C 8F 1F C0
```

? bool var_n9 = true // Convert true to 1 and false to 0.

File_WriteBytes("writebytes3.txt", var_n9, 1)

```
writebytes3.txt
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 8D 97 6E 12 83 C0 F3 BF AE 47 E1 7A 14 AE F3 3F
00000010 8F C2 F5 28 5C 8F 1F C0 01
```

? bool[] var_n10 = {true, false, true, false, false, true, true}

File_WriteBytes("writebytes3.txt", var_n10, 1)

```
writebytes3.txt
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 8D 97 6E 12 83 C0 F3 BF AE 47 E1 7A 14 AE F3 3F
00000010 8F C2 F5 28 5C 8F 1F C0 01 01 00 01 00 00 01 01
```

? string var_n11 = "ABCDEFGH" // Convert the string into UTF8.

File_WriteBytes("writebytes3.txt", var_n11, 1)

```
writebytes3.txt
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 8D 97 6E 12 83 C0 F3 BF AE 47 E1 7A 14 AE F3 3F
00000010 8F C2 F5 28 5C 8F 1F C0 01 01 00 01 00 00 01 01
00000020 41 42 43 44 45 46 47
```

```
? string[] var_n12 = {"ABC", "DEF", "達明機器人"};
```

```
File_WriteBytes("writebytes3.txt", var_n12, 1)
```

```
writebytes3.txt
```

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 8D 97 6E 12 83 C0 F3 BF AE 47 E1 7A 14 AE F3 3F
00000010 8F C2 F5 28 5C 8F 1F C0 01 01 00 01 00 00 01 01
00000020 41 42 43 44 45 46 47 41 42 43 44 45 46 E9 81 94
00000030 E6 98 8E E6 A9 9F E5 99 A8 E4 BA BA
```

4.7 File_WriteText()

Put the data content in a string and write to a file.

Syntax 1

```
bool File_WriteText(
    string,
    ?,
    int,
    int,
    int
)
```

Parameter

string	File path
?	Values to write. Eligible for integers, floating-point numbers, booleans, strings, or arrays.
int	Overwrite the file or append to the file.
0	Over the file. If not existed, create a new file.
1	Append to the file. If not existed, create a new file.
int	The starting index of the value to write (eligible for strings and arrays)
0.. length -1	Legitimate value
< 0	Illegitimate value. The starting index will be set to 0.
>= length	Illegitimate value. The starting index will be set to 0.
int	The length of the value to write (eligible for strings and arrays)
<= 0	Write from the starting index to the end of the data.
> 0	Write from the starting index for a specified number of the length up to the data ends.

Return

bool	true	Write successfully.
	false	Write unsuccessfully.

Syntax 2

```
bool File_WriteText(
    string,
    ?,
    int,
    int
)
```

Note

Same as Syntax 1. Set the length of the value to write to 0 writing up to the data ends.

File_WriteText(string,?,int,int) => File_WriteText(string,?,int,int,0)

Syntax 3

```
bool File_WriteText(
    string,
    ?,
    int
)
```

Note

Same as Syntax 1. Set the starting index and the length of the value to write to 0 writing up to the data ends.

File_WriteText(string,?,int) => File_WriteText(string,?,int,0,0)

Syntax 4

```
bool File_WriteText(
    string,
    ?
)
```

Note

Same as Syntax 1. Set the starting index and the length of the value to write to 0 overwriting the file up to the data ends.

File_WriteText(string,?) => File_WriteText(string,?,0,0,0)

```
string var_s1 = "Hi TM Robot"
```

```
string var_s2 = "達明機器人"
```

```
bool var_flag = false
```

```
var_flag = File_WriteText("writetext.txt", var_s1)    // Overwrite "Hi TM Robot" to the file
```

```
writetext.txt
```

```
1| Hi TM Robot
```

```

var_flag = File_WriteText("writetext.txt", var_s2)    // Overwrite "達明機器人" to the file

writetext.txt
1| 達明機器人
var_flag = File_WriteText("writetext.txt", var_s1, 1) // Append "Hi TM Robot" to the file

writetext.txt
1| 達明機器人 Hi TM Robot

var_flag = File_WriteText("writetext.txt", var_s2, 1, 2, 3) // Append from index 2 for 3 character "機器人"
                                                                to the file.

writetext.txt
1| 達明機器人 Hi TM Robot 機器人

? byte var_n1 = 100 // Convert the value in decimal to a string

File_WriteText("writetext2.txt", var_n1, 1)

writetext2.txt
1| 100

? int[] var_n4 = {10000, 20000, 80000} // The array uses the format {, ,}.

File_WriteText("writetext2.txt", var_n4, 1)

writetext2.txt
1| 100{10000,20000,80000}
// For other formats, use GetString() to convert to string and write.

? float var_n5 = 1.234

File_WriteText("writetext2.txt", var_n5, 1)

writetext2.txt
1| 100{10000,20000,80000}1.234

? double[] var_n8 = {1.23, -7.89}

File_WriteText("writetext2.txt", var_n8, 1)

writetext2.txt
1| 100{10000,20000,80000}1.234{1.23,-7.89}

? bool var_n9 = true

File_WriteText("writetext2.txt", var_n9, 1)

writetext2.txt
1| 100{10000,20000,80000}1.234{1.23,-7.89>true

? string var_n11 = "ABCDEFGH"

File_WriteText("writetext2.txt", var_n11, 1)

writetext2.txt
1| 100{10000,20000,80000}1.234{1.23,-7.89>trueABCDEFGH

? string[] var_n12 = {"ABC", "DEF", "達明機器人"}

```

```
File_WriteText("writetext2.txt", var_n12, 1)
```

```
writetext2.txt
```

```
1| 100{10000,20000,80000}1.234{1.23,-7.89}trueABCDEFGF{ABC,DEF,達明機器人}
```

4.8 File_WriteLine()

Put the data content in a string with newline characters (0x0D 0x0A) in the end and write to a file.

Syntax 1

```
bool File_WriteLine(  
    string,  
    ?,  
    int,  
    int,  
    int  
)
```

Parameter

string	File path
?	Values to write. Eligible for integers, floating-point numbers, booleans, strings, or arrays.
int	Overwrite the file or append to the file. 0 Over the file. If not existed, create a new file. 1 Append to the file. If not existed, create a new file.
int	The starting index of the value to write (eligible for strings and arrays) 0 .. length -1 Legitimate value < 0 Illegitimate value. The starting index will be set to 0. >= length Illegitimate value. The starting index will be set to 0.
int	The length of the value to write (eligible for strings and arrays) <= 0 Write from the starting index to the end of the data. > 0 Write from the starting index for a specified number of the length up to the data ends.

Return

bool	true	Write successfully.
	false	Write unsuccessfully.

Syntax 2

```
bool File_WriteLine(  
    string,  
    ?,  
    int,
```

```
int
```

```
)
```

Note

Same as Syntax 1. Set the length of the value to write to 0 writing up to the data ends.

File_WriteLine(string,?,int,int) => **File_WriteLine**(string,?,int,int,0)

Syntax 3

```
bool File_WriteLine(  
    string,  
    ?,  
    int  
)
```

Note

Same as Syntax 1. Set the starting index and the length of the value to write to 0 writing up to the data ends.

File_WriteLine(string,?,int) => **File_WriteLine**(string,?,int,0,0)

Syntax 4

```
bool File_WriteLine(  
    string,  
    ?  
)
```

Note

Same as Syntax 1. Set the starting index and the length of the value to write to 0 overwriting the file up to the data ends.

File_WriteLine(string,?) => **File_WriteLine**(string,?,0,0,0)

```
string var_s1 = "Hi TM Robot"
```

```
string var_s2 = "達明機器人"
```

```
bool var_flag = false
```

```
var_flag = File_WriteLine("writeline.txt", var_s2) // Overwrite "達明機器人\u00D0A" to the file
```

```
writeline.txt
```

```
1| 達明機器人
```

```
2|
```

```
var_flag = File_WriteLine("writeline.txt", var_s1) // Overwrite "Hi TM Robot\u00D0A" to the file
```

```
writeline.txt
```

```
1| Hi TM Robot
```

```
2|
```

```
var_flag = File_WriteLine("writeline.txt", var_s2, 1) // Append "達明機器人\u000A" to the file
```

writeline.txt

1| Hi TM Robot

2| 達明機器人

3|

```
var_flag = File_WriteLine("writeline.txt", var_s1, 1, 3) // Append "TM Robot\u00D0A" from the starting index 3
                                                    to the end to the file.
```

```
writeline.txt
1| Hi TM Robot
2| 達明機器人
3| TM Robot
4|
```

```
? byte[] var_n2 = {100, 200} // The array uses the format {, ,}.
```

```
File_WriteLine("writeline2.txt", var_n2, 1)
```

```
writeline2.txt
1| {100,200}
2|
```

```
// For other formats, use GetString() to convert to string and write.
```

```
? int var_n3 = 10000 // Convert the value in decimal to a string.
```

```
File_WriteLine("writeline2.txt", var_n3, 1)
```

```
writeline2.txt
1| {100,200}
2| 10000
3|
```

```
? float[] var_n6 = {1.23, 4.56, -7.89}
```

```
File_WriteLine("writeline2.txt", var_n6, 1)
```

```
writeline2.txt
1| {100,200}
2| 10000
3| {1.23,4.56,-7.89}
4|
```

```
? bool var_n9 = true
```

```
File_WriteLine("writeline2.txt", var_n9, 1)
```

```
writeline2.txt
1| {100,200}
2| 10000
3| {1.23,4.56,-7.89}
4| true
5|
```

```
? string var_n11 = "ABCDEFGF"
```

```
File_WriteLine("writeline2.txt", var_n11, 1)
```

```
writeline2.txt
1| {100,200}
2| 10000
3| {1.23,4.56,-7.89}
4| true
5| ABCDEF
6|
```

```
? string[] var_n12 = {"ABC", "DEF", "達明機器人" }
```

```
File_WriteLine("writeline2.txt", var_n12, 1)
```

```
writeline2.txt
```

```
1| {100,200}  
2| 10000  
3| {1.23,4.56,-7.89}  
4| true  
5| ABCDEFG  
6| {ABC,DEF,達明機器人}  
7|
```

4.9 File_WriteLines()

Put the data content in a string array with newline characters (0x0D 0x0A) in the end and write to a file.

Syntax 1

```
bool File_WriteLines (  
    string,  
    ?,  
    int,  
    int,  
    int  
)
```

Parameter

string	File path						
?	Values to write. Eligible for integers, floating-point numbers, booleans, strings, or arrays.						
int	Overwrite the file or append to the file. <table><tr><td>0</td><td>Over the file. If not existed, create a new file.</td></tr><tr><td>1</td><td>Append to the file. If not existed, create a new file.</td></tr></table>	0	Over the file. If not existed, create a new file.	1	Append to the file. If not existed, create a new file.		
0	Over the file. If not existed, create a new file.						
1	Append to the file. If not existed, create a new file.						
int	The starting index of the value to write (eligible for strings and arrays) <table><tr><td>0 .. length-1</td><td>Legitimate value</td></tr><tr><td>< 0</td><td>Illegitimate value. The starting index will be set to 0.</td></tr><tr><td>>= length</td><td>Illegitimate value. The starting index will be set to 0.</td></tr></table>	0 .. length-1	Legitimate value	< 0	Illegitimate value. The starting index will be set to 0.	>= length	Illegitimate value. The starting index will be set to 0.
0 .. length-1	Legitimate value						
< 0	Illegitimate value. The starting index will be set to 0.						
>= length	Illegitimate value. The starting index will be set to 0.						
int	The length of the value to write (eligible for strings and arrays) <table><tr><td><= 0</td><td>Write from the starting index to the end of the data.</td></tr><tr><td>> 0</td><td>Write from the starting index for a specified number of the length up to the data ends.</td></tr></table>	<= 0	Write from the starting index to the end of the data.	> 0	Write from the starting index for a specified number of the length up to the data ends.		
<= 0	Write from the starting index to the end of the data.						
> 0	Write from the starting index for a specified number of the length up to the data ends.						

Return

```
bool true Write successfully.
```

false Write unsuccessfully.

Syntax 2

```
bool File_WriteLines (
    string,
    ?,
    int,
    int
)
```

Note

Same as Syntax 1. Set the length of the value to write to 0 writing up to the data ends.

File_WriteLines(string,?,int,int) => **File_WriteLines**(string,?,int,int,0)

Syntax 3

```
bool File_WriteLines (
    string,
    ?,
    int
)
```

Note

Same as Syntax 1. Set the starting index and the length of the value to write to 0 writing up to the data ends.

File_WriteLines(string,?,int) => **File_WriteLines**(string,?,int,0,0)

Syntax 4

```
bool File_WriteLines (
    string,
    ?
)
```

Note

Same as Syntax 1. Set the starting index and the length of the value to write to 0 overwriting the file up to the data ends.

File_WriteLines(string,?) => **File_WriteLines**(string,?,0,0,0)

- * **File_WriteText()**: convert the data values to write to a string without adding newline characters in the end of the string.
- File_WriteLine()**: convert the data values to write to a string with adding newline characters in the end of the string.

File_WriteLines(): convert the data values to write to a string array with adding newline characters in the end of each element of the array.

```
string var_ss1 = {"Hi TM Robot", "達明機器人"}
```

```
bool var_flag = false
```

```
var_flag = File_WriteLines("writelines.txt", var_ss1) // Overwrite var_ss1 to the file
```

```
writelines.txt
1| Hi TM Robot
2| 達明機器人
3|
```

```
var_flag = File_WriteLines("writelines.txt", var_ss1, 1, 1) // Append var_ss1 from the starting index 1 to the end to the file.
```

```
writelines.txt
1| Hi TM Robot
2| 達明機器人
3| Hi TM Robot
4|
```

```
? byte[] var_n2 = {100, 200}
```

```
File_WriteLines("writelines2.txt", var_n2, 1)
```

```
writelines2.txt
1| 100
2| 200
3|
```

```
? int var_n3 = 10000
```

```
File_WriteLines("writelines2.txt", var_n3, 1)
```

```
writelines2.txt
1| 100
2| 200
3| 10000
4|
```

```
? float[] var_n6 = {1.23, 4.56, -7.89}
```

```
File_WriteLines("writelines2.txt", var_n6, 1)
```

```
writelines2.txt
1| 100
2| 200
3| 10000
4| 1.23
5| 4.56
6| -7.89
7|
```

```
? string var_n11 = "ABCDEFGFG"
```

```
File_WriteLines("writelines2.txt", var_n11, 1)
```

```
writelines2.txt
1| 100
2| 200
```

```
? string[] var_n12 = {"ABC", "DEF", "達明機器人"}
```

```
File_WriteLines("writelines2.txt", var_n12, 1)
```

```
writelines2.txt
```

```
1| 100
2| 200
3| 10000
4| 1.23
5| 4.56
6| -7.89
7| ABCDEFG
8| ABC
9| DEF
10| 達明機器人
11|
```

4.10 File_Exists()

Check the file path for availability.

Syntax 1

```
bool File_Exists (
    string
)
```

Parameter

string File path

Return

bool true File path available
 false File path unavailable

*Return false file path unavailable for Voided file path without errors.

Note

```
bool var_exists = false
```

```
var_exists = File_Exists("sampleFile1.txt")      // true
```

```
var_exists = File_Exists("sampleFileX.txt")      // false      // File path unavailable
```

```
var_exists = File_Exists("C:\SampleFile1.txt")      // false      // Void for absolute paths.
```

4.11 File_Length()

Check the file size.

Syntax 1

```
int File_Length(  
    string  
)
```

Parameter

string File path

Return

int In int32 data type. The maximum file size is limited to 2147483647 bytes.
-1 File path unavailable.
-2 Exceeded the maximum file size limit.
* Return -1 file path unavailable for void file path without errors.

Note

```
Int var_len = 0  
var_len = File_Length("sampleFile1.txt")      // 31  
var_len = File_Length("sampleFileX.txt")      // -1 // File path unavailable  
var_len = File_Length("C:\SampleFile1.txt")      // -1 // Void for absolute paths.
```

4.12 File_Delete()

Delet the file.

Syntax 1

```
bool File_Delete(  
    string  
    ...  
)
```

Parameter

string File path
... Available for multiple strings.

Return

bool true Delete successfully. (Included unavailable or void file paths)
 false Delete unsuccessfully. (Unable to delete the file for occupied)

Syntax 2

```
bool File_Delete (
    string[]
)
```

Parameter

string[] File path

Return

bool	true	Delete successfully. (Included unavailable or void file paths)
	false	Delete unsuccessfully. (Unable to delete the file for occupied)

Note

```
bool var_flag = false

var_flag = File_Delete("sampleFile1.txt")           // true

var_flag = File_Delete("sampleFileX.txt")           // true    // File path unavailable

var_flag = File_Delete("C:\SampleFile1.txt")        // true    // Void for absolute paths.

var_flag = File_Delete("sampleFile1.txt", "sampleFileX.txt") // Available for multiple file paths.

var_flag = File_Delete("sampleFile1.txt", "sampleFileX.txt", "C:\SampleFile1.txt")

                                   // Available for multiple file paths.

string[] var_ss = {"sampleFile1.txt", "sampleFileX.txt", "C:\SampleFile1.txt"}

var_flag = File_Delete(var_ss)

var_flag = File_Delete(var_ss, "sampleFile2.txt") // Error. Void for syntax.
```

4.13 File_Copy()

Copy the file.

Syntax 1

```
bool File_Copy (
    string,
    string,
    string
)
```

Parameter

string	Source file path
string	Target directory path
string	Target file name. Use the source file path equivalent for naming as the default if empty.

Return

bool	true	Copy successfully.
------	------	--------------------

`false` Copy unsuccessfully.
* Overwrite the target file if existed in the target path.

Syntax 2

```
bool File_Copy (  
    string,  
    string  
)
```

Note

Same as Syntax 1. Set the target file name with an empty string and use the source file path equivalent for naming.

`File_Copy(string,string) => File_Copy(string,string,"")`

`File_Copy("sampleFile1.txt", ".\TextFiles", "s1.txt") // copy .\TextFiles\sampleFile1.txt to .\TextFiles\s1.txt`

`File_Copy("sampleFile1.txt", ".\XmlFiles", "") // copy .\TextFiles\sampleFile1.txt to .\XmlFiles\sampleFile1.txt`

`File_Copy("sampleFile1.txt", "\USB\TMROBOT", "s1.txt") // copy .\TextFiles\sampleFile1.txt to USB\s1.txt`

`File_Copy("sampleFile1.txt", "\USB\TMROBOT") // copy .\TextFiles\sampleFile1.txt to USB\sampleFile1.txt`

`bool var_flag = false`

`var_flag = File_Copy("sampleFile1.txt", "C:\folder") // Error. Void for absolute paths.`

`var_flag = File_Copy("sampleFile1.txt", ".") // Error. Neither TextFiles nor XmlFiles is in the file path.`

4.14 File_CopyImage()

Copy the saved vision file.

Syntax 1

```
bool File_CopyImage (  
    string,  
    string,  
    string,  
    int  
)
```

Paramaters

`string` Source saved vision file path

`string` Target directory path

Granted to copy and save images to external paths such as external device path

directories or remote path directories

Not granted to copy and save the image to the local path such as ".\TextFiles" or ".\XmlFiles", and an **error** will be reported.

string Target file name.

If the name is an empty string, the default files name is the same as the one in the source file path.

int Copy options

- 0 No error returned when failed to copy. No relative directory of the sourced image reserved. (default)
- 1 No error returned when failed to copy. The relative directory of the sourced image reserved.
- 2 An **error** returned when failed to copy. No relative directory of the sourced image reserved.
- 3 An **error** returned when failed to copy. The relative directory of the sourced image reserved.

Return

bool **true** Copy successfully.
 false Copy unsuccessfully.

* If the file exists in the destination path, the destination file will be overwritten.

Syntax 2

```
bool File_CopyImage (  
    string,  
    string,  
    string  
)
```

Note

Same as syntax 1. Set the copy option to 0. No error returned when failed to copy. No relative directory of the sourced image reserved.

File_CopyImage(string,string,string) => **File_CopyImage**(string,string,string,0)

Syntax 3

```
bool File_CopyImage (  
    string,  
    string  
)
```

Note

Same as syntax 1. Set the target file name to an empty string and the same as the one in the source file path. Set the copy option to 0. No error returned when failed to copy. No relative directory of the

sourced image reserved.

File_CopyImage(string,string) => **File_CopyImage**(string,string,"",0)

Syntax 4

```
bool File_CopyImage (  
    string,  
    string,  
    int  
)
```

Note

Same as syntax 1. Set the target file name to an empty string and the same as the one in the source file path.

File_CopyImage(string,string,int) => **File_CopyImage**(string,string,"",int)

bool var_flag = false

```
var_flag = File_CopyImage(Job1_ImagePath_TM, ".\TextFiles", "1.png") // false  
                                                                    // Copy to local directories not  
                                                                    supported.
```

```
var_flag = File_CopyImage(Job1_ImagePath_TM, ".\XmlFiles", "1.png") // false  
                                                                    // Copy to local directories not  
                                                                    supported.
```

```
var_flag = File_CopyImage(Job1_ImagePath_TM, ".\XmlFiles", "1.png", 2) // 報錯  
                                                                    // Copy to local directories not  
                                                                    supported.
```

```
var_flag = File_CopyImage(Job1_ImagePath_TM, "\USB\TMROBOT", "1.png")  
// true // Copy Job1_ImagePath_TM (the vision AOI-only variable) to USB\1.png  
var_flag = File_CopyImage(Job1_ImagePath_TM, "\USB\TMROBOT", "1.png", 3)  
// true // Copy Job1_ImagePath_TM (the vision AOI-only variable) to USB\ProjectName\Job1\Date\source\1.png  
// Reserve the directory of the image to save.
```

```
var_flag = File_CopyImage(Job1_ImagePath_TM, "\USB\TMROBOT")  
// true // Copy Job1_ImagePath_TM (the vision AOI-only variable) to USB\15-16-12_423.png  
// Reserve the file name of the image to save.
```

```
var_flag = File_CopyImage(Job1_ImagePath_TM, "\USB\TMROBOT", 3)  
// true // Copy Job1_ImagePath_TM (the vision AOI-only variable) to USB\ProjectName\Job1\Date\source\15-16-12_423.png
```

// Reserve the directory and the file name of the image to save.

4.15 File_Replace()

Replace and overwrite the string in the file with a specified string.

Syntax 1

```
bool File_Replace(  
    string,  
    string,  
    string  
)
```

Parameter

string	File path
string	The string to be replaced
string	The string to replace

Return

bool	true	Success	1. The string to be replaced is empty. 2. The string to be replaced is absent. 3. The string to be replaced is found and overwritten in the file.
	false	Unsuccess	

Note

```
.\TextFiles\SampleFile6.txt  
1| 6Hello World!  
2| 6Hello TM Robot!  
3| 6Hi TM Robot!
```

```
bool var_flag = false
```

```
var_flag = File_Replace("SampleFile6.txt", "Hello", "HI")
```

```
SampleFile6.txt  
1| 6HI World!  
2| 6HI TM Robot!  
3| 6Hi TM Robot!
```

```
var_flag = File_Replace("SampleFile6.txt", "TM", "Techman")
```

```
SampleFile6.txt  
1| 6HI World!  
2| 6HI Techman Robot!  
3| 6Hi Techman Robot!
```

```
var_flag = File_Replace("SampleFile6.txt", "6", "")
```

```
SampleFile6.txt  
1| HI World!  
2| HI Techman Robot!  
3| Hi Techman Robot!
```

4.16 File_GetToken()

Read the file by the string pattern and retrieve the substring in the string.

Syntax 1

```
string File_GetToken (  
    string,  
    string,  
    string,  
    int,  
    int  
)
```

Parameter

string	File path
string	The prefix of the string to retrieve
string	The suffix of the string to retrieve
int	The number of the matched substring to retrieve
>=1	Retrieve the n th matched substring
-1	Retrieve the last matched substring
int	Remove options
0	The 1 st matched not in the start of the input string, and not remove the prefix and the suffix. (default)
1	The 1 st matched not in the start of the input string, and remove the prefix and the suffix.
2	The 1 st matched in the start of the input string, and not remove the prefix and the suffix.
3	The 1 st matched in the start of the input string, and remove the prefix and the suffix.

Return

string Return the retrieved string.

Return the content of the string in the file if [the prefix and the suffix are empty](#).

Return an empty string if [matching number <=0](#).

If the remove option is 2 or 3, the first match retrieved must be at the start of the input string; otherwise, it returns an empty string.

Syntax 2

```
string File_GetToken (  
    string,  
    string,  
    string,
```

```
int
```

```
)
```

Note

Same as Syntax 1. Fill 0 for not removing the prefix and the suffix as the default.

File_GetToken(string,string,string,int) => File_GetToken(string,string,string,int,0)

Syntax 3

```
string File_GetToken (  
    string,  
    string,  
    string  
)
```

Note

Same as Syntax 1. Fill 1 for the matching and 0 for not removing the prefix and the suffix as the default.

File_GetToken(string,string,string) => File_GetToken(string,string,string,1,0)

```
.\TextFiles\SampleFile7.txt  
1| $Hello World!  
2| $Hello TM Robot!  
3| $Hi TM Robot!$
```

```
string var_n = "SampleFile7.txt"
```

```
string var_s = ""
```

```
var_s = File_GetToken(var_n, "", "", 0)           // "$Hello World!\u000A$Hello TM Robot!\u000A$Hi TM Robot!$"
var_s = File_GetToken(var_n, "$", "$")           // "$Hello World!\u000A$"
var_s = File_GetToken(var_n, "$", "$", 0)         // ""
var_s = File_GetToken(var_n, "$", "$", 1)         // "$Hello World!\u000A$"
var_s = File_GetToken(var_n, "$", "$", 2)         // "$Hi TM Robot!$"
var_s = File_GetToken(var_n, "$", "$", 3)         // ""
var_s = File_GetToken(var_n, "$", "$", 1, 1)      // "Hello World!\u000A"
var_s = File_GetToken(var_n, "$", "$", 2, 1)      // "Hi TM Robot!"
var_s = File_GetToken(var_n, "$", "", 1)          // "$Hello World!\u000A"
var_s = File_GetToken(var_n, "$", "", 2)          // "$Hello TM Robot!\u000A"
var_s = File_GetToken(var_n, "$", "", 3)          // "$Hi TM Robot!"
var_s = File_GetToken(var_n, "$", "", 4)          // "$"
var_s = File_GetToken(var_n, "", "$", 1)          // "$"
var_s = File_GetToken(var_n, "", "$", 2)          // "Hello World!\u000A$"
var_s = File_GetToken(var_n, "", "$", 3)          // "Hello TM Robot!\u000A$"
var_s = File_GetToken(var_n, "", "$", 4)          // "Hi TM Robot!$"
var_s = File_GetToken(var_n, "$", Ctrl("\r\n"), 1) // "$Hello World!\u000A"
var_s = File_GetToken(var_n, "$", newline, 2)     // "$Hello TM Robot!\u000A"
```

```

var_s = File_GetToken(var_n, "$", NewLine, 1, 1) // "Hello World!" // Remove the prefix and the suffix
var_s = File_GetToken(var_n, Ctrl("\r\n"), "$", 1) // "\u000A$"
var_s = File_GetToken(var_n, newline, "$", 2) // "\u000A$"
var_s = File_GetToken(var_n, NewLine, "$", 1, 1) // ""

* \u000A denotes a new line character but not a string value.

```

.\TextFiles\SampleFile9.txt

```

1| #Hello World!
2| $Hello TM Robot!
3| $Hi TM Robot!$

```

```

string var_n = "SampleFile9.txt"

string var_s = ""

var_s = File_GetToken(var_n, "", "") // "#Hello World!\u000A$Hello TM Robot!\u000A$Hi TM Robot!$"
var_s = File_GetToken(var_n, "#", "") // "#Hello World!\u000A$Hello TM Robot!\u000A$Hi TM Robot!$"
var_s = File_GetToken(var_n, "", "$") // "#Hello World!\u000A$"
var_s = File_GetToken(var_n, "#", newline, 1, 0) // "#Hello World!\u000A"
var_s = File_GetToken(var_n, "#", newline, 1, 1) // "Hello World!"
var_s = File_GetToken(var_n, "#", newline, 1, 2) // "#Hello World!\u000A"
var_s = File_GetToken(var_n, "#", newline, 1, 3) // "Hello World!"
var_s = File_GetToken(var_n, "$", newline, 1, 0) // "$Hello TM Robot!\u000A"
var_s = File_GetToken(var_n, "$", newline, 1, 2) // ""
// $ not in the start of the file. Return an empty string.

var_s = File_GetToken(var_n, "$", "", 1, 2) // ""
// $ not in the start of the file. Return an empty string.

var_s = File_GetToken(var_n, "#", "", 1, 2) // "#Hello World!\u000A$Hello TM Robot!\u000A$Hi TM Robot!$"
var_s = File_GetToken(var_n, "", "$", 1, 2) // "#Hello World!\u000A$"
var_s = File_GetToken(var_n, "", "$", -1, 2) // "Hi TM Robot!$"
var_s = File_GetToken(var_n, "", "$", 100, 2) // "" // Exceeded the matching number
var_s = File_GetToken(var_n, "", "#", 1, 2) // "#"

```

Syntax 4

```

string File_GetToken (
    string,
    byte[],
    byte[],
    int,
    int
)

```

Parameter

string File path

<code>byte[]</code>	The prefix of the string to retrieve in the byte array
<code>byte[]</code>	The suffix of the string to retrieve in the byte array
<code>int</code>	The number of the matched substring to retrieve
<code>>=1</code>	Retrieve the n th matched substring
<code>-1</code>	Retrieve the last matched substring
<code>int</code>	Remove options
<code>0</code>	The 1 st matched not in the start of the input string, and not remove the prefix and the suffix. (default)
<code>1</code>	The 1 st matched not in the start of the input string, and remove the prefix and the suffix.
<code>2</code>	The 1 st matched in the start of the input string, and not remove the prefix and the suffix.
<code>3</code>	The 1 st matched in the start of the input string, and remove the prefix and the suffix.

Return

`string` Return the retrieved string.

Return the content of the string in the file if [the prefix and the suffix are empty](#).

Return an empty string if [matching number <=0](#).

If the remove option is 2 or 3, the first match retrieved must be at the start of the input string; otherwise, it returns an empty string.

Syntax 5

```
string File_GetToken (
    string,
    byte[],
    byte[],
    int
)
```

Note

Same as Syntax 4. Fill 0 for not removing the prefix and the suffix as the default.

File_GetToken(string,byte[],byte[],int) => File_GetToken(string,byte[],byte[],int,0)

Syntax 6

```
string File_GetToken (
    string,
    byte[],
    byte[]
)
```

Note

Same as Syntax 4. Fill 1 for the matching and 0 for not removing the prefix and the suffix as the default.

File_GetToken(string,byte[],byte[]) => File_GetToken(string,byte[],byte[],1,0)

.\TextFiles\SampleFile8.txt

1| \$Hello World!

2| Hello\$ TM Robot!

3| Hello\$ TM Robot\$

```

string var_n = "SampleFile8.txt", var_s = ""

byte[] var_bb0 = {}, var_bb1 = {0x24}, var_bb2 = {0x0D, 0x0A}    // 0x24 is $ and 0x0D 0x0A is \u000A
var_s = File_GetToken(var_n, bb0, bb0, 0)                        // "$Hello World\u000AHello$ TM Robot!\u000AHi$ TM Robot!$"
var_s = File_GetToken(var_n, bb1, bb1)                          // "$Hello World\u000AHello$"
var_s = File_GetToken(var_n, bb1, bb1, 0)                       // ""
var_s = File_GetToken(var_n, bb1, bb1, 1)                       // "$Hello World\u000AHello$"
var_s = File_GetToken(var_n, bb1, bb1, 2)                       // "$ TM Robot!$"
var_s = File_GetToken(var_n, bb1, bb1, 3)                       // ""
var_s = File_GetToken(var_n, bb1, bb1, 1, 1)                   // "Hello World\u000AHello"
var_s = File_GetToken(var_n, bb1, bb1, 2, 1)                   // " TM Robot!"
var_s = File_GetToken(var_n, bb1, bb0, 1)                       // "$Hello World\u000AHello"
var_s = File_GetToken(var_n, bb1, bb0, 2)                       // "$ TM Robot!\u000AHi"
var_s = File_GetToken(var_n, bb1, bb0, 3)                       // "$ TM Robot!"
var_s = File_GetToken(var_n, bb1, bb0, 4)                       // "$"
var_s = File_GetToken(var_n, bb0, bb1, 1)                       // "$"
var_s = File_GetToken(var_n, bb0, bb1, 2)                       // "Hello World\u000AHello$"
var_s = File_GetToken(var_n, bb0, bb1, 3)                       // " TM Robot!\u000AHi$"
var_s = File_GetToken(var_n, bb0, bb1, 4)                       // " TM Robot!$"
var_s = File_GetToken(var_n, bb1, bb2, 1)                       // "$Hello World\u000A"
var_s = File_GetToken(var_n, bb1, bb2, 2)                       // "$ TM Robot!\u000A"
var_s = File_GetToken(var_n, bb1, bb2, 1, 1)                   // "Hello World" // Remove the prefix and the suffix
var_s = File_GetToken(var_n, bb2, bb1, 1)                       // "\u000AHello$"
var_s = File_GetToken(var_n, bb2, bb1, 2)                       // "\u000AHi$"
var_s = File_GetToken(var_n, bb2, bb1, 1, 1)                   // "Hello"

```

* \u000A denotes a new line character but not a string value.

4.17 File_GetAllTokens()

Read the file by the string pattern and retrieve all eligible substrings.

Syntax 1

```

string[] File_GetAllTokens (
    string,
    string,
    string,
    int
)

```

Parameter

<code>string</code>	File path
<code>string</code>	The prefix of the string to retrieve
<code>string</code>	The suffix of the string to retrieve
<code>int</code>	Remove options
0	The 1 st matched not in the start of the input string, and not remove the prefix and the suffix. (default)
1	The 1 st matched not in the start of the input string, and remove the prefix and the suffix.
2	The 1 st matched in the start of the input string, and not remove the prefix and the suffix.
3	The 1 st matched in the start of the input string, and remove the prefix and the suffix.

Return

`string[]` Return the eligible string in an array.
Return the content of the string in the file as a string array if [the prefix and the suffix are empty](#).
If the remove option is 2 or 3, the first match retrieved must be at the start of the input string; otherwise, it returns an empty string.

Syntax 2

```
string[] File_GetAllTokens (  
    string,  
    string,  
    string  
)
```

Note

Same as Syntax 1. Fill 0 for not removing the prefix and the suffix as the default.

File_GetAllTokens(string,string,string) => File_GetAllTokens(string,string,string,0)

```
.\TextFiles\SampleFile7.txt  
1| $Hello World!  
2| $Hello TM Robot!  
3| $Hi TM Robot!$  
  
string var_n = "SampleFile7.txt"  
  
string[] var_ss = File_GetAllTokens(var_n, "", "") // {"$Hello World!\u000A$Hello TM Robot!\u000A$Hi TM Robot!$"}  
var_ss = File_GetAllTokens(var_n, "$", "$") // {"$Hello World!\u000A$", "$Hi TM Robot!$"}  
var_ss = File_GetAllTokens(var_n, "$", "$", 1) // {"Hello World!\u000A", "Hi TM Robot!"}  
var_ss = File_GetAllTokens(var_n, "$", "", 1) // {"Hello World!\u000A", "Hello TM Robot!\u000A", "Hi TM Robot!", ""}  
  
.\TextFiles\SampleFile9.txt  
1| #Hello World!  
2| $Hello TM Robot!  
3| $Hi TM Robot!$
```

```

string var_n = "SampleFile9.txt"
string[] var_ss = {}
var_ss = File_GetAllTokens(var_n, "", "") // {"#Hello World!\u000A$Hello TM Robot!\u000A$Hi TM Robot!$"}
var_ss = File_GetAllTokens(var_n, "$", "$", 0) // {"$Hello TM Robot!\u000A$"}
var_ss = File_GetAllTokens(var_n, "$", "$", 1) // {"Hello TM Robot!\u000A"}
var_ss = File_GetAllTokens(var_n, "$", "$", 2) // {}
var_ss = File_GetAllTokens(var_n, "$", "$", 3) // {}
var_ss = File_GetAllTokens(var_n, "$", "", 0) // {"$Hello TM Robot!\u000A", "$Hi TM Robot!", "$"}
var_ss = File_GetAllTokens(var_n, "$", "", 2) // {}
var_ss = File_GetAllTokens(var_n, "#", "", 0) // {"#Hello World!\u000A$Hello TM Robot!\u000A$Hi TM Robot!$"}
var_ss = File_GetAllTokens(var_n, "#", "", 1) // {"Hello World!\u000A$Hello TM Robot!\u000A$Hi TM Robot!$"}
var_ss = File_GetAllTokens(var_n, "#", "", 2) // {"#Hello World!\u000A$Hello TM Robot!\u000A$Hi TM Robot!$"}
var_ss = File_GetAllTokens(var_n, "#", "", 3) // {"Hello World!\u000A$Hello TM Robot!\u000A$Hi TM Robot!$"}

```

5. Serial Port Functions

When the project starts running as going from the start node, it opens the serial port for connections and receives the data from the serial port consistently. For data received in the received buffer, users can use the function `com_read` to read data in the buffer. Once the project stops running, it closes the opened serial port and clears the received buffer.

The maximum capacity of the received buffer is 2 MB. If there is data coming to the received buffer and the received buffer is out of space, it removes the earliest data automatically for the latest data coming into the received buffer.

5.1 `com_read()`

Read data in the Serial Port received buffer and return an array `byte[]`.

Syntax 1

```
byte[] com_read(  
    string  
)
```

Parameters

`string` The name of the device on the Serial Port (configure in the Serial Port Device)

Return

`byte[]` Return all the data content. If the content is empty, it returns `byte[0]`.

Note

```
var_ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
```

```
byte[] var_value = com_read("spd")
```

```
// var_value byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
```

```
// var_ReceivedBuffer = {}
```

*This function reads all data in the received buffer and clears the received buffer.

Syntax 2

```
byte[] com_read(  
    string,  
    int,  
    int  
)
```

Parameters

string The name of the device on the Serial Port (configure in the Serial Port Device)

int The number of the elements to read (based on the length of byte[])

- <= 0** Read all elements
- > 0** Read a specified number of the elements (Data is available when the specified number fulfills.)

int The length of time to read in millisecond

- <= 0** Read once only
- > 0** Read many times until there is data or the time is up.

Return

byte[] Return the specified number of the elements with byte[]. If the elements is insufficient, it returns byte[0].

Syntax 3

```
byte[] com_read(  
    string,  
    int  
)
```

Note

The syntax is the same as syntax 2. The default length of time to read is 0.

```
var_ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}  
var_value = com_read("spd", 6)  
    // var_value byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C}  
    // var_ReceivedBuffer = {0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}  
var_value = com_read("spd", 100)  
    // var_value byte[] = {}  
    // Insufficient elements for no more than 100 elements in the received buffer and return byte[0].  
    // var_ReceivedBuffer = {0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}  
var_value = com_read("spd", 0)  
    // var_value byte[] = {0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A} // Read all elements  
    // var_ReceivedBuffer = {}  
var_value = com_read("spd", 4, 100)  
    // value byte[] = {} // Insufficient elements for no more than 4 elements in the received buffer and return
```

byte[0].

// But the length of time to read is set to 100 ms, the process stays in the function until there is data or the time is up and exits the function.

// var_ReceivedBuffer = {0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38} // Supposed it receives data after 50ms,

// var_value byte[] = {0x31,0x32,0x33,0x34} // it reads 4 element and exits the function.

// var_ReceivedBuffer = {0x35,0x36,0x37,0x38}

Syntax 4

```
byte[] com_read(  
    string,  
    byte[] or string,  
    byte[] or string,  
    int,  
    int  
)
```

Parameters

string The name of the device on the Serial Port (configure in the Serial Port Device)

byte[] or string

Terms of the prefix to read. If the input is byte[0] or "", an empty string, it means no prefix terms.

byte[] or string

Terms of the suffix to read. If the input is byte[0] or "", an empty string, it means no suffix terms.

int Remove options

0 The 1st matched not in the start of the input string, and not remove the prefix and the suffix. (default)

1 The 1st matched not in the start of the input string, and remove the prefix and the suffix.

2 The 1st matched in the start of the input string, and not remove the prefix and the suffix.

3 The 1st matched in the start of the input string, and remove the prefix and the suffix.

int The length of time to read in millisecond

<= 0 Read once only

> 0 Read many times until there is data or the time is up.

Return

byte[] Return with byte[] in the first matched terms of the prefix and the suffix.

It retrieves data with the content matches the first of all terms, and the rest will be reserved and not retrieved.

If there is no match, it returns byte[0].

Syntax 5

```
byte[] com_read(  
    string,
```

```

byte[] or string,
byte[] or string,
int
)

```

Note

The syntax is the same as syntax 4. The default length of time to read is 0.

Syntax 6

```

byte[] com_read(
    string,
    byte[] or string,
    byte[] or string
)

```

Note

The syntax is the same as syntax 4. The default is not to remove the prefix and the suffix from the read content and the length of time to read is 0.

```

var_ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
var_value = com_read("spd", "He", newline)           // prefix "He", suffix \u0D0A
// var_value byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A} // Hello,\u0D0A
//vVar_ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A} // retrieve the first match and reserve the
rest
var_value = com_read("spd", "", newline, 1)           // prefix "", suffix \u0D0A. Remove both the prefix and
the suffix.
// var_value byte[] = {0x57,0x6F,0x72,0x6C,0x64} // World
// var_ReceivedBuffer = {}
var_value = com_read("spd", "", newline, 1, 100)      // prefix "", suffix \u0D0A. Remove both the prefix and
the suffix. The length of time to read is 100ms.
// var_value byte[] = {}
// No matched terms to read. Read byte[0]. Wait for 100 ms.
// var_ReceivedBuffer = {}
// var_ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A}
// var_value byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C}
// var_ReceivedBuffer = {}

var_ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
var_value = com_read("spd", "lo", newline)           // prefix "lo", suffix \u0D0A
// var_value byte[] = {0x6C,0x6F,0x2C,0x0D,0x0A} // lo,\u0D0A
// The data before the first matched term, {0x48,0x65,0x6C}, will be removed.

```

```

// var_ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
// retrieve the first match and reserve the rest
byte[] var_bb = {}
var_value = com_read("spd", var_bb, newline)           // prefix byte[0], suffix \u0D0A
// var_value byte[] = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A} // World\u0D0A
// var_ReceivedBuffer = {}
var_value = com_read("spd", bb, newline, 0, 100)       // prefix byte[0], suffix \u0D0A, 100ms
// var_value byte[] = {}
// No matched terms to read. Read byte[0]. Wait for 100 ms.
// var_ReceivedBuffer = {}
// var_ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A}
// var_value byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A}
// var_ReceivedBuffer = {}

ReceivedBuffer = {0x24,0x48,0x65,0x6C,0x6C,0x6F,0x0D,0x0A, // $Hello
                  0x23,0x48,0x69,0x0D,0x0A,                // #Hi
                  0x24,0x54,0x4D,0x0D,0x0A,                // $TM
                  0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A} // #Robot

value = com_read("spd", "#", newline, 2)                // prefix "#" suffix \u0D0A
// value byte[] = {}                                     // # not in the start. Return an empty array.
// ReceivedBuffer = {0x24,0x48,0x65,0x6C,0x6C,0x6F,0x0D,0x0A,0x23,0x48,0x69,0x0D,0x0A,
                    0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

value = com_read("spd", "$", newline, 2)                // prefix "$" suffix \u0D0A
// value byte[] = {0x24,0x48,0x65,0x6C,0x6C,0x6F,0x0D,0x0A} // It must be in the start to retrieve the first match.
// ReceivedBuffer = {0x23,0x48,0x69,0x0D,0x0A,
                    0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

value = com_read("spd", "#", newline, 2)                // prefix "#" suffix \u0D0A
// value byte[] = {0x23,0x48,0x69,0x0D,0x0A}             // It must be in the start to retrieve the first match.
// ReceivedBuffer = {0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

value = com_read("spd", "$", newline, 3)                // prefix "$" suffix \u0D0A
// value byte[] = {0x54,0x4D}
// ReceivedBuffer = {0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

value = com_read("spd", "#", newline, 3)                // prefix "#" suffix \u0D0A
// value byte[] = {0x52,0x6F,0x62,0x6F,0x74}
// ReceivedBuffer = {}

```

Syntax 7

```

byte[] com_read (
    string,

```

```

    byte[] or string,
    int,
    int
)

```

Parameters

string The name of the device on the Serial Port (configure in the Serial Port Device)

byte[] or string Terms of the suffix to read. If the input is byte[0] or "", an empty string, it means no suffix terms.

int Remove options

- 0 The 1st matched not in the start of the input string, and not remove the prefix and the suffix. (default)
- 1 The 1st matched not in the start of the input string, and remove the prefix and the suffix.
- 2 The 1st matched in the start of the input string, and not remove the prefix and the suffix.
- 3 The 1st matched in the start of the input string, and remove the prefix and the suffix.

int The length of time to read in millisecond

- <= 0** Read once only
- > 0** Read many times until there is data or the time is up.

* No terms of the prefix to read.

Return

byte[] Return with byte[] in the first matched terms of the prefix and the suffix.

It retrieves data with the content matches the first of all terms, and the rest will be reserved and not retrieved.

If there is no match, it returns byte[0].

Syntax 8

```

byte[] com_read(
    string,
    byte[] or string,
    int
)

```

Note

The syntax is the same as syntax 7. The default length of time to read is 0.

Syntax 9

```

byte[] com_read(
    string,
    byte[] or string
)

```

Note

The syntax is the same as syntax 7. The default is not to remove the prefix and the suffix from the read

content and the length of time to read is 0.

Note

```
var_ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
var_value = com_read("spd", newline)    // suffix \u0D0A
// var_value byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A} // Hello,\u0D0A
// var_ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
// retrieve the first match and reserve the rest
var_value = com_read("spd", newline)    // suffix \u0D0A
// var_value byte[] = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A} // World\u0D0A
// var_ReceivedBuffer = {}

var_ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
var_value = com_read("spd", newline, 1)    // suffix \u0D0A
// var_value byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C} // Hello,
// var_ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
// retrieve the first match and reserve the rest
var_value = com_read("spd", newline, 1)    // suffix \u0D0A
// var_value byte[] = {0x57,0x6F,0x72,0x6C,0x64} // World
// var_ReceivedBuffer = {}
var_value = com_read("spd", newline, 1, 100) // suffix \u0D0A, 100ms
// var_value byte[] = {} // No matched terms to read. Read byte[0]. Wait for 100 ms.
// var_ReceivedBuffer = {}
// var_ReceivedBuffer = {0x31,0x32,0x33,0x34,0x35,0x36,0x0D,0x0A}
// var_value byte[] = {0x31,0x32,0x33,0x34,0x35,0x36}
// var_ReceivedBuffer = {}
```

5.2 com_read_string()

Read the data in the Serial Port buffer, and convert the data to a UTF8 string.

Syntax 1

```
string com_read_string(
    string
)
```

Parameters

string The name of the device on the Serial Port (configure in the Serial Port Device)

Return

`string` Return all the data content. If the content is empty, it returns an empty string.

Syntax 2

```
string com_read_string(  
    string,  
    int,  
    int  
)
```

Parameters

`string` The name of the device on the Serial Port (configure in the Serial Port Device)

`int` The number of characters to read (based on the number of characters of the string)

`<= 0` Read all characters

`> 0` Read a specified number of the characters (Data is available when the specified number fulfills.)

`int` The length of time to read in millisecond

`<= 0` Read once only

`> 0` Read many times until there is data or the time is up.

Return

`string` Returns the specified number of characters as a string. If the characters are insufficient, it returns an empty string.

Syntax 3

```
string com_read_string(  
    string,  
    int  
)
```

Note

The syntax is the same as syntax 2. The default length of time to read is 0.

Syntax 4

```
string com_read_string(  
    string,  
    byte[] or string,  
    byte[] or string,  
    int,  
    int  
)
```

Parameters

`string` The name of the device on the Serial Port (configure in the Serial Port Device)

`byte[]` or `string`

Terms of the prefix to read. If the input is `byte[0]` or `""`, an empty string, it means no prefix terms.

`byte[]` or `string`

Terms of the suffix to read. If the input is `byte[0]` or `""`, an empty string, it means no suffix terms.

`int` Remove options

0 The 1st matched not in the start of the input string, and not remove the prefix and the suffix. (default)

1 The 1st matched not in the start of the input string, and remove the prefix and the suffix.

2 The 1st matched in the start of the input string, and not remove the prefix and the suffix.

3 The 1st matched in the start of the input string, and remove the prefix and the suffix.

`int` The length of time to read in millisecond

`<= 0` Read once only

`> 0` Read many times until there is data or the time is up.

Return

`string` It retrieves data with the content matches the first of all terms

It retrieves data with the content matches the first of all terms, and the rest will be reserved and not retrieved.

If there is no match, it returns an empty string.

Syntax 5

```
string com_read_string(  
    string,  
    byte[] or string,  
    byte[] or string,  
    int  
)
```

Note

The syntax is the same as syntax 4. The default length of time to read is 0.

Syntax 6

```
string com_read_string(  
    string,  
    byte[] or string,  
    byte[] or string  
)
```

Note

The syntax is the same as syntax 4. The default is not to remove the prefix and the suffix from the read

content and the length of time to read is 0.

Syntax 7

```
string com_read_string(  
    string,  
    byte[] or string,  
    int,  
    int  
)
```

Parameters

- string** The name of the device on the Serial Port (configure in the Serial Port Device)
- byte[] or string** Terms of the suffix to read. If the input is byte[0] or "", an empty string, it means no suffix terms.
- int** Remove options
- 0 The 1st matched not in the start of the input string, and not remove the prefix and the suffix. (default)
 - 1 The 1st matched not in the start of the input string, and remove the prefix and the suffix.
 - 2 The 1st matched in the start of the input string, and not remove the prefix and the suffix.
 - 3 The 1st matched in the start of the input string, and remove the prefix and the suffix.
- int** The length of time to read in millisecond
- <= 0** Read once only
 - > 0** Read many times until there is data or the time is up.
- * No terms of the prefix to read.

Return

- string** Returns as a string with the first matched terms of the prefix and the suffix.
- Retrieves data in the content matches to the first of all terms, and the rest will be reserved and not retrieved.
- If there is no match, it returns an empty string.

Syntax 8

```
string com_read_string(  
    string,  
    byte[] or string,  
    int  
)
```

Note

The syntax is the same as syntax 7. The default length of time to read is 0.

Syntax 9

```
string com_read_string(  
    string,  
    byte[] or string  
)
```

Note

The syntax is the same as syntax 7. The default is not to remove the prefix and the suffix from the read content and the length of time to read is 0.

Note

```
var_ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}  
string var_value = com_read_string("spd")  
    // var_value string = "Hello, World\u00D0A"  
    // var_ReceivedBuffer = {}  
  
var_ReceivedBuffer =  
{0x54,0x4D,0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA}  
var_value = com_read_string("spd", 4)  
    // var_value string = "TM 達明" // {0x54,0x4D,0xE9,0x81,0x94,0xE6,0x98,0x8E}  
    //Retrieve 4 characters based on the length of the string.  
    // var_ReceivedBuffer = {0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA}  
var_value = com_read_string("spd", 5, 100)  
    // var_value string = ""  
    // Insufficient characters for no more than 5 characters in the received buffer based on the length of the string.  
    Wait for 100 ms.  
    // var_ReceivedBuffer = {0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA}  
    // var_ReceivedBuffer = {0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA, 0x0D, 0x0A}  
    // var_value string = "機器人\u00D0A"  
    // var_ReceivedBuffer = {}  
  
Var_ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}  
var_value = com_read_string("spd", "He", newline) // prefix "He", suffix \u00D0A  
    // value string = "Hello,\u00D0A"  
    // var_ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A} // retrieve the first match and reserve the  
rest  
var_value = com_read_string("spd", "", newline, 1)  
    // prefix "", suffix \u00D0A. Remove both the prefix and the suffix.
```

```

// var_value string = "World"
// var_ReceivedBuffer = {}
var_value = com_read_string("spd", "", newline, 1, 100)
// var_value string = "" // No matched terms to read. Read an empty string. Wait for 100 ms.
// var_ReceivedBuffer = {}
// var_ReceivedBuffer = {0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA, 0x0D, 0x0A}
// var_value string = "機器人"

var_ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
var_value = com_read_string("spd", "lo", newline) // prefix "lo", suffix \u0D0A
// var_value string = "lo,\u0D0A"
// The data before the first matched term, "Hel", will be removed.
// var_ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
// retrieve the first match and reserve the rest
var_value = com_read_string("spd", newline, 1) // suffix \u0D0A
// var_value string = "World"
// var_ReceivedBuffer = {}

ReceivedBuffer = {0x24,0x48,0x65,0x6C,0x6C,0x6F,0x0D,0x0A, // $Hello
                  0x23,0x48,0x69,0x0D,0x0A, // #Hi
                  0x24,0x54,0x4D,0x0D,0x0A, // $TM
                  0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A} // #Robot

value = com_read_string("spd", "#", newline, 2) // prefix "#" suffix \u0D0A
// value string = "" // # not in the start. Return an empty array.
// ReceivedBuffer = {0x24,0x48,0x65,0x6C,0x6C,0x6F,0x0D,0x0A,0x23,0x48,0x69,0x0D,0x0A,
                    0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

value = com_read_string("spd", "$", newline, 2) // prefix "$" suffix \u0D0A
// value string = "$Hello\u0D0A" // It must be in the start to retrieve the first match.
// ReceivedBuffer = {0x23,0x48,0x69,0x0D,0x0A,
                    0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

value = com_read_string("spd", "#", newline, 2) // prefix "#" suffix \u0D0A
// value string = "#Hi\u0D0A" // It must be in the start to retrieve the first match.
// ReceivedBuffer = {0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

value = com_read_string("spd", "$", newline, 3) // prefix "$" suffix \u0D0A
// value string = "TM"
// ReceivedBuffer = {0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

value = com_read_string("spd", "#", newline, 3) // prefix "#" suffix \u0D0A

```

```

// value string = "Robot"
// ReceivedBuffer = {}

var_ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64}
var_value = com_read_string("spd", newline) // suffix \u0D0A
// var_value string = "Hello,\u0D0A"
// var_ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64} // retrieve the first match and reserve the rest
var_value = com_read_string("spd", newline, 0) // suffix \u0D0A
// var_value string = "" // No matched terms to read. Read an empty string.
// var_ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64}
var_value = com_read_string("spd", newline, 1, 100) // suffix \u0D0A
// var_value string = ""
// No matched terms to read. Read an empty string. Wait for 100 ms.
// var_ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64}
// var_ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A,0x31,0x32,0x33,0x0D,0x0A}
// var_value string = "World"
// var_ReceivedBuffer = {0x31,0x32,0x33,0x0D,0x0A} // retrieve the first match and reserve the rest

```

5.3 com_write()

Write data to the Serial Port

Syntax 1

```

bool com_write (
    string,
    ?,
    int,
    int
)

```

Parameters

<code>string</code>	The name of the device on the Serial Port (configure in the Serial Port Device)
<code>?</code>	The value to write. Available types: int, float, , bool, string, and array. Numeric values will be conversed in Little Endian, and string values will be converse in UTF8.
<code>int</code>	The starting index of the value to write (eligible for strings and arrays) <code>0 .. length-1</code> Legitimate value <code>< 0</code> Illegitimate value. The starting index will be set to 0. <code>>= length</code> Illegitimate value. The starting index will be set to 0.
<code>int</code>	The length of the value to write (eligible for strings and arrays) <code><= 0</code> Write from the starting index to the end of the data.

> 0

Write from the starting index for a specified number of the length up to the data ends.

Return

bool	True	write successfully	
	False	write unsuccessfully	1. The value to write is an empty string or an empty array. 2. Unable to send to the serial port correctly.

Syntax 2

```
bool com_write(  
    string,  
    ?,  
    int,  
    int  
)
```

Parameters

string	The name of the device on the Serial Port (configure in the Serial Port Device)		
?	The value to write. Available types: int, float, , bool, string, and array. Numeric values will be converted in Little Endian, and string values will be converted in UTF8.		
int	The starting index of the data to write. (valid for strings or arrays)		
	0	The length of the string - 1	Legal value
	< 0		Illegal value, and the starting index will be 0.
	>=	The length of the string	Illegal value, and the starting index will be 0.
int	The length of the data to write. (valid for strings or arrays)		
	<= 0		From the start of the index to the end of the data
	> 0		From the start of the index, write the specified length of the data until the data ends.

Return

bool	True	write successfully	
	False	write unsuccessfully	1. The value to write is an empty string or an empty array. 2. Unable to send to the serial port correctly.

Syntax 3

```
bool com_write(  
    string,  
    ?  
)
```

Note

The syntax is the same as syntax 2. The default length of data to write is 0.

```

var_flag = com_write("spd", 100)           // write 0x64
var_flag = com_write("spd", 1000)          // write 0xE8 0x03 0x00 0x00 (int, Little Endian)
var_flag = com_write("spd", (float)1.234)   // write 0xB6 0xF3 0x9D 0x3F (float, Little Endian)
var_flag = com_write("spd", (double)123.456)
// write 0x77 0xBE 0x9F 0x1A 0x2F 0xDD 0x5E 0x40 (double, Little Endian)
var_flag = com_write("spd", "Hello, World"+newline)
// write 0x48 0x65 0x6C 0x6C 0x6F 0x2C 0x20 0x57 0x6F 0x72 0x6C 0x64 0x0D 0x0A (string, UTF8)
var_flag = com_write("spd", 1000, 1, 2)      //Invalid in the value, the starting index, and the length
// write 0xE8 0x03 0x00 0x00 (int, Little Endian)

byte[] var_bb = {100, 200}
var_flag = com_write("spd", var_bb)          // write 0x64 0xC8
var_flag = com_write("spd", var_bb, 1, 1)    // write 0xC8
// Array. Retrieve 1 element from the index 1. [1]=200
var_flag = com_write("spd", var_bb, -1, 1)    // write 0x64
// Array. Retrieve 1 element from the index 0. [0]=100

var_flag = com_write("spd", "達明機器人", 2)

// String. Retrieve from the index 2 until the index ends. "機器人"
// write 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA (string, UTF8)

string[] var_ss = {"TM", "", "達明機器人" }

var_flag = com_write("spd", var_ss)
// write 0x54 0x4D 0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA
var_flag = com_write("spd", Byte_Concat(GetBytes(var_ss), GetBytes(newline)))
// write 0x54 0x4D 0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA 0x0D 0x0A
var_flag = com_write("spd", var_ss, 2, 100)
// Array. Retrieve 100 elements (to the end) from the index 2. [2]=達明機器人
// write 0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA

```

5.4 com_writeline()

Write data to the Serial Port and add line break symbols, 0x0D 0x0A, in the end of the data automatically

Syntax 1

```

bool com_writeline (
    string,

```

```

    ?,
    int,
    int
)

```

Parameters

<code>string</code>	The name of the device on the Serial Port (configure in the Serial Port Device)	
<code>?</code>	The value to write. Available types: int, float, , bool, string, and array.	
	Numeric values will be conversed in Little Endian, and string values will be converse in UTF8.	
<code>int</code>	The starting index of the value to write (eligible for strings and arrays)	
	<code>0 .. length-1</code>	Legitimate value
	<code>< 0</code>	Illegitimate value. The starting index will be set to 0.
	<code>>= length</code>	Illegitimate value. The starting index will be set to 0.
<code>int</code>	The length of the value to write (eligible for strings and arrays)	
	<code><= 0</code>	Write from the starting index to the end of the data.
	<code>> 0</code>	Write from the staring index for a specified nubmer of the length up to the data ends.

Return

<code>bool</code>	<code>True</code>	write successfully	
	<code>False</code>	write unsuccessfully	1. The value to write is an empty string or an empty array. 2. Unable to send to the serial port correctly.

Syntax 2

```

bool com_writeline(
    string,
    ?,
    int,
)

```

Note

The syntax is the same as syntax 1. The default length of data to write is 0.

Syntax 3

```

bool com_writeline(
    string,
    ?,
)

```

Note

The syntax is the same as syntax 1. The default starting index of the data to write is 0. The default length of data to write is 0.

```
var_flag = com_writeline("spd", 100)           // write 0x64 0x0D 0x0A
var_flag = com_writeline("spd", 1000)          // write 0xE8 0x03 0x00 0x00 0x0D 0x0A (int, Little Endian)
var_flag = com_writeline("spd", (float)1.234)   // write 0xB6 0xF3 0x9D 0x3F 0x0D 0x0A (float, Little Endian)
var_flag = com_writeline("spd", (double)123.456)
           // write 0x77 0xBE 0x9F 0x1A 0x2F 0xDD 0x5E 0x40 0x0D 0x0A (double, Little Endian)
var_flag = com_write("spd", "Hello, World"+newline)
           // write 0x48 0x65 0x6C 0x6C 0x6F 0x2C 0x20 0x57 0x6F 0x72 0x6C 0x64 0x0D 0x0A (string, UTF8)
var_flag = com_writeline("spd", "Hello, World")
           // write 0x48 0x65 0x6C 0x6C 0x6F 0x2C 0x20 0x57 0x6F 0x72 0x6C 0x64 0x0D 0x0A (string, UTF8)
var_flag = com_writeline("spd", 1000, 1, 2)     // Invalid in the value, the starting index, and the length
           // write 0xE8 0x03 0x00 0x00 0x0D 0x0A (int, Little Endian)

byte[] var_bb = {100, 200}
var_flag = com_writeline("spd", var_bb)         // write 0x64 0xC8 0x0D 0x0A
var_flag = com_writeline("spd", var_bb, 1, 1)   // write 0xC8 0x0D 0x0A
           // Array. Retrieve 1 element from the index 1. [1]=200
var_flag = com_writeline("spd", var_bb, -1, 1)  // write 0x64 0x0D 0x0A
           // Array. Retrieve 1 element from the index 0. [0]=100

var_flag = com_writeline("spd", "達明機器人", 2)
           // String. Retrieve from the index 2 until the index ends. "機器人"
           // write 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA 0x0D 0x0A (string, UTF8)

string[] var_ss = {"TM", "", "達明機器人"}

var_flag = com_writeline("spd", var_ss)
           // write 0x54 0x4D 0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA 0x0D 0x0A
var_flag = com_write("spd", Byte_Concat(GetBytes(var_ss), GetBytes(newline)))
           // write 0x54 0x4D 0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA 0x0D 0x0A
var_flag = com_writeline("spd", var_ss)
           // write 0x54 0x4D 0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA 0x0D 0x0A
var_flag = com_writeline("spd", var_ss, 2, 100)
           // Array. Retrieve 100 elements (to the end) from the index 2. [2]=達明機器人
           // write 0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA 0x0D 0x0A
```

6. Socket Functions

When the project starts running as the Start node initiates, it launches the TCP/IP Socket Client to connect to the specified IP and port. It keeps receiving the data in the connection, brings the received data to the Received Buffer, and uses respective functions such as `Socket_read` to read the data. When the project stops running, the existing TCP/IP Socket connection will be closed with the Received Buffer cleared.

The Received Buffer comes with a capacity limit. If the buffer capacity is insufficient when the data comes in, it automatically deletes the oldest data and adds the latest.

6.1 `socket_read()`

Read data in the Received Buffer and return in a byte array.

Syntax 1

```
byte[] socket_read(  
    string  
)
```

Parameters

`string` Network device name (configured in Network Device)

Return

`byte[]` Return all data in the Received Buffer. Return `byte[0]` if buffer empty.

Note

ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}

`byte[] var_value = socket_read("ntd_a")`

`// byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}`

`// ReceivedBuffer = {}`

*This function reads all data in the Received Buffer and clear the buffer.

Syntax 2

```
byte[] socket_read(  
    string,  
    int,  
    int  
)
```

Parameters

`string` Network device name (configured in Network Device)

<code>int</code>	Retrieve the fixed amount of byte (by the length of byte[])
<code><= 0</code>	Get all
<code>> 0</code>	Get a specified amount (the specified amount required to get the data)
<code>int</code>	Read time (millisecond)
<code><= 0</code>	Read once
<code>> 0</code>	Read multiple times until there is data or the times fulfill.

Return

`byte[]` Return the specified amount of data in the Received Buffer with a byte array. Return `byte[0]` if insufficient amount.

Syntax 3

```
byte[] socket_read (
    string,
    int
)
```

Note

Same as syntax 3. Fill 0 as the read time by default.

ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}

```
byte[] var_value = socket_read("ntd_a", 6)
```

```
// byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C}
```

```
// ReceivedBuffer = {0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
```

```
var_value = socket_read("ntd_a", 100)
```

```
// byte[] = {} // The number is insufficient for less than 100 bytes in the Received Buffer. Byte[0] will return.
```

```
// ReceivedBuffer = {0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
```

```
var_value = socket_read("ntd_a", 0)
```

```
// byte[] = {0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A} // Retrieve all data
```

```
// ReceivedBuffer = {}
```

```
var_value = socket_read("ntd_a", 4, 100)
```

```
// byte[] = {} // The number is insufficient for less than 4 bytes in the Received Buffer. Byte[0] will return.
```

```
//But the read time is set to 100 ms, it stays in the function still waiting for data or the read times fulfilled
before exiting.
```

```
// ReceivedBuffer = {0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38} //Supposed data received in 50 ms later
```

```
// byte[] = {0x31,0x32,0x33,0x34} // Retrieve 4 bytes and exit the function.
```

```
// ReceivedBuffer = {0x35,0x36,0x37,0x38}
```

Syntax 4

```
byte[] socket_read (
    string,
```

```

byte[] or string,
byte[] or string,
int,
int

```

)

Parameters

string Network device name (configured in Network Device)

byte[] or string

The prefix condition to read. Input byte[0] or "" as the empty string to denote no prefix condition.

byte[] or string

The suffix condition to read. Input byte[0] or "" as the empty string to denote no suffix condition.

int Fetch option

0 The 1st match not required in the beginning and not removing the prefix and the suffix (default)

1 The 1st match not required in the beginning and removing the prefix and the suffix

2 The 1st match required in the beginning and removing the prefix and the suffix

3 The 1st match required in the beginning and removing the prefix and the suffix

int Read time (millisecond)

<= 0 Read once

> 0 Read multiple times until there is data or the times fulfill.

Return

byte[] Return the 1st array in byte that matches the prefix condition and the suffix condition.

Fetch the first data that matches the conditions in the Received Buffer. The following data remains in the Received Buffer.

If the prefix condition and the suffix condition are byte[0] or an empty string, it fetches all data in the Received Buffer.

If unable to find the data matches the condition, it returns byte[0].

Syntax 5

```

byte[] socket_read(
    string,
    byte[] or string,
    byte[] or string,
    int
)

```

Note

Same as syntax 4. Fill 0 as the read time by default.

Syntax 6

```
byte[] socket_read(  
    string,  
    byte[] or string,  
    byte[] or string  
)
```

Note

Same as syntax 4. Fill 0 as the fetch option and the read time by default.

ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}

```
byte[] var_value = socket_read("ntd_a", "", "") // No prefix. No suffix. Fetch all data.  
// byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}  
// ReceivedBuffer = {}
```

ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}

```
byte[] var_value = socket_read("ntd_a", "He", newline) // Prefix "He" Suffix \u0D0A  
// byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A} // Hello,\u0D0A  
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A} // Fetch the 1st match. Remain the following data.  
var_value = socket_read("ntd_a", "", newline, 1) // Prefix "" Suffix \u0D0A Remove the prefix and the  
suffix.  
// byte[] = {0x57,0x6F,0x72,0x6C,0x64} // World  
// ReceivedBuffer = {}  
var_value = socket_read("ntd_a", "", newline, 1, 100) // Prefix "" Suffix \u0D0A Remove the prefix and the suffix.  
100ms  
// byte[] = {} // Read byte[0] for fetch option unfulfilled. Wait for 100ms.  
// ReceivedBuffer = {}  
// ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A} // Supposed data received in 50ms later  
// byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C} // Hello,  
// ReceivedBuffer = {}
```

ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}

```
byte[] var_value = socket_read("ntd_a", "lo", newline) // Prefix "lo" Suffix \u0D0A  
// byte[] = {0x6C,0x6F,0x2C,0x0D,0x0A} // lo,\u0D0A  
// The data before the 1st match, {0x48,0x65,0x6C}, will be removed.  
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}  
byte[] var_bb = {}  
var_value = socket_read("ntd_a", var_bb, newline) // Prefix byte[0] Suffix \u0D0A
```

```

// byte[] = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}           // World\u0D0A
// ReceivedBuffer = {}

var_value = socket_read("ntd_a", var_bb, newline, 0, 100) // Prefix byte[0] Suffix \u0D0A · 100ms
// byte[] = {}                                             //Read byte[0] for fetch option unfulfilled. Wait for 100ms.
// ReceivedBuffer = {}

// ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A} // Supposed data received in 50ms later
// byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A}
// ReceivedBuffer = {}

ReceivedBuffer = {0x24,0x48,0x65,0x6C,0x6C,0x6F,0x0D,0x0A, // $Hello
                  0x23,0x48,0x69,0x0D,0x0A,                // #Hi
                  0x24,0x54,0x4D,0x0D,0x0A,                // $TM
                  0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A} // #Robot

byte[] var_value = socket_read("ntd_a", "#", newline, 2) // Prefix "#" Suffix \u0D0A
// byte[] = {}                                           //Return an empty string for # not in the prefix.
// ReceivedBuffer = {0x24,0x48,0x65,0x6C,0x6C,0x6F,0x0D,0x0A,0x23,0x48,0x69,0x0D,0x0A,
                    0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

var_value = socket_read("ntd_a", "$", newline, 2) // Prefix "$" Suffix \u0D0A
// byte[] = {0x24,0x48,0x65,0x6C,0x6C,0x6F,0x0D,0x0A} // Fetch the 1st match in the prefix required.
// ReceivedBuffer = {0x23,0x48,0x69,0x0D,0x0A,
                    0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

var_value = socket_read("ntd_a", "#", newline, 2) // Prefix "#" Suffix \u0D0A
// byte[] = {0x23,0x48,0x69,0x0D,0x0A} // Fetch the 1st match in the prefix required.
// ReceivedBuffer = {0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

var_value = socket_read("ntd_a", "$", newline, 3) // Prefix "$" Suffix \u0D0A
// byte[] = {0x54,0x4D}
// ReceivedBuffer = {0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

var_value = socket_read("ntd_a", "#", newline, 3) // Prefix "#" Suffix \u0D0A
// byte[] = {0x52,0x6F,0x62,0x6F,0x74}
// ReceivedBuffer = {}

```

Syntax 7

```

byte[] socket_read(
    string,
    byte[] or string,
    int,
    int
)

```

Parameters

`string` Network device name (configured in Network Device)

`byte[] or string` The suffix condition to read. Input `byte[0]` or `""` as the empty string to denote no suffix condition.

`int` Fetch option

- `0` The 1st match not required in the beginning and not removing the prefix and the suffix (default)
- `1` The 1st match not required in the beginning and removing the prefix and the suffix
- `2` The 1st match required in the beginning and removing the prefix and the suffix
- `3` The 1st match required in the beginning and removing the prefix and the suffix

`int` Read time (millisecond)

- `<= 0` Read once
- `> 0` Read multiple times until there is data or the times fulfill.

Return

`byte[]` Return the 1st array in byte that matches the suffix condition. (No prefix condition restricted)

Fetch the first data that matches the condition in the Received Buffer only. The following data remains in the Received Buffer.

If the suffix condition is `byte[0]` or an empty string, it fetches all data in the Received Buffer.

If unable to find the data matches the condition, it returns `byte[0]`.

Syntax 8

```
byte[] socket_read(
    string,
    byte[] or string,
    int
)
```

Note

Same as syntax 7. Fill 0 as the read time by default.

Syntax 9

```
byte[] socket_read(
    string,
    byte[] or string
)
```

Note

Same as syntax 7. Fill 0 as the fetch option and the read time by default.

```

ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}

byte[] var_value = socket_read("ntd_a", "") // Empty Suffix. Fetch all data.

// byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}

// ReceivedBuffer = {}

```

```

ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}

byte[] var_value = socket_read("ntd_a", newline) // Suffix \u0D0A
// byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A} // Hello,\u0D0A
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A} // Fetch the 1st match. Remain the following data.

var_value = socket_read("ntd_a", newline) // Suffix \u0D0A
// byte[] = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A} // World\u0D0A
// ReceivedBuffer = {}

```

```

ReceivedBuffer = {0x24,0x48,0x65,0x6C,0x6C,0x6F,0x0D,0x0A, // $Hello
                  0x23,0x48,0x69,0x0D,0x0A, // #Hi
                  0x24,0x54,0x4D,0x0D,0x0A, // $TM
                  0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A} // #Robot

byte[] var_value = socket_read("ntd_a", newline, 0) // Suffix \u0D0A
// byte[] = {0x24,0x48,0x65,0x6C,0x6C,0x6F,0x0D,0x0A} // $Hello\u0D0A
// ReceivedBuffer = {0x23,0x48,0x69,0x0D,0x0A,
                    0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

var_value = socket_read("ntd_a", newline, 1) // Suffix \u0D0A
// byte[] = {0x23,0x48,0x69} // #Hi
// ReceivedBuffer = {0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

var_value = socket_read("ntd_a", newline, 2) // Suffix \u0D0A
// byte[] = {0x24,0x54,0x4D,0x0D,0x0A} // $TM\u0D0A
// ReceivedBuffer = {0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

var_value = socket_read("ntd_a", newline, 3) // Suffix \u0D0A
// byte[] = {0x23,0x52,0x6F,0x62,0x6F,0x74} // #Robot
// ReceivedBuffer = {}

```

6.2 socket_read_string()

Read data in the Received Buffer and convert the byte array to string text in UTF8

Syntax 1

```
string socket_read_string(
```

```
string
```

```
)
```

Parameters

`string` Network device name (configured in Network Device)

Return

`string` Return all data in the Received Buffer. Return an empty string if buffer empty.

Syntax 2

```
string socket_read_string(  
    string,  
    int,  
    int  
)
```

Parameters

`string` Network device name (configured in Network Device)

`int` Retrieve the fixed amount of string (by the length of string)

`<= 0` Get all

`> 0` Get a specified amount (the specified amount required to get the data)

`int` Read time (millisecond)

`<= 0` Read once

`> 0` Read multiple times until there is data or the times fulfill.

Return

`string` Return the specified amount of data in the Received Buffer with a string. Return an empty string if insufficient amount.

Syntax 3

```
string socket_read_string(  
    string,  
    int  
)
```

Note

Same as syntax 2. Fill 0 as the read time by default.

Syntax 4

```
string socket_read_string(  
    string,  
    byte[] or string,  
    byte[] or string,  
    int,
```

```
int
```

```
)
```

Parameters

`string` Network device name (configured in Network Device)

`byte[]` or `string`

The prefix condition to read. Input `byte[0]` or `""` as the empty string to denote no prefix condition.

`byte[]` or `string`

The suffix condition to read. Input `byte[0]` or `""` as the empty string to denote no suffix condition.

`int` Retrieve the fixed amount of string (by the length of string)

`<= 0` Get all

`> 0` Get a specified amount (the specified amount required to get the data)

`int` Read time (millisecond)

`<= 0` Read once

`> 0` Read multiple times until there is data or the times fulfill.

Return

`string` Return the 1st string that matches the prefix condition and the suffix condition.

Fetch the first data that matches the conditions in the Received Buffer. The following data remains in the Received Buffer.

If the prefix condition and the suffix condition are `byte[0]` or an empty string, it fetches all data in the Received Buffer.

If unable to find the data matches the condition, it returns an empty string.

Syntax 5

```
string socket_read_string(  
    string,  
    byte[] or string,  
    byte[] or string,  
    int  
)
```

Note

Same as syntax 4. Fill 0 as the read time by default.

Syntax 6

```
string socket_read_string(  
    string,  
    byte[] or string,  
    byte[] or string
```

)

Note

Same as syntax 4. Fill 0 as the fetch option and the read time by default.

Syntax 7

```
string socket_read_string(  
    string,  
    byte[] or string,  
    int,  
    int  
)
```

Parameters

string	Network device name (configured in Network Device)
byte[] or string	The suffix condition to read. Input byte[0] or "" as the empty string to denote no suffix condition.
int	Retrieve the fixed amount of string (by the length of string) <= 0 Get all > 0 Get a specified amount (the specified amount required to get the data)
int	Read time (millisecond) <= 0 Read once > 0 Read multiple times until there is data or the times fulfill.

Return

string Return the 1st string that matches the suffix condition. (No prefix condition restricted)
Fetch the first data that matches the condition in the Received Buffer. The following data remains in the Received Buffer.
If the suffix condition is byte[0] or an empty string, it fetches all data in the Received Buffer.
If unable to find the data matches the condition, it returns an empty string.

Syntax 8

```
string socket_read_string(  
    string,  
    byte[] or string,  
    int  
)
```

Note

Same as syntax 7. Fill 0 as the read time by default.

Syntax 9

```
string socket_read_string(  
    string,  
    byte[] or string  
)
```

Note

Same as syntax 7. Fill 0 as the fetch option and the read time by default.

```
ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
```

```
string var_value = socket_read_string("ntd_a")
```

```
// string = "Hello, World\u00D0A"
```

```
// ReceivedBuffer = {}
```

```
ReceivedBuffer = {0x54,0x4D,0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA}
```

```
string var_value = socket_read_string("ntd_a", 4)
```

```
// string = "TM 達明" // {0x54,0x4D,0xE9,0x81,0x94,0xE6,0x98,0x8E} // Fetch 4 by the string length.
```

```
// ReceivedBuffer = {0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA}
```

```
var_value = socket_read_string("ntd_a", 5, 100)
```

```
// string = "" // The amount to read data in the Received Buffer is less than 5. (by the string length). Wait for 100 ms.
```

```
// ReceivedBuffer = {0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA}
```

```
// ReceivedBuffer = {0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA, 0x0D, 0x0A}
```

```
// string = "機器人\u00D0A"
```

```
// ReceivedBuffer = {}
```

```
ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
```

```
string var_value = socket_read_string("ntd_a", "", "")
```

```
// string = "Hello,\u00D0AWorld\u00D0A"
```

```
// ReceivedBuffer = {}
```

```
ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
```

```
string var_value = socket_read_string("ntd_a", "He", newline) // Prefix "He" Suffix \u00D0A
```

```
// string = "Hello,\u00D0A"
```

```
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
```

```
var_value = socket_read_string("ntd_a", "", newline, 1) // Prefix "" Suffix \u00D0A Remove the prefix and the suffix.
```

```
// string = "World"
```

```
// ReceivedBuffer = {}
```

```

var_value = socket_read_string("ntd_a", "", newline, 1, 100)
// string = "" // Read a empty string for the fetch option unfulfilled. Wait for 100 ms.
// ReceivedBuffer = {}
// ReceivedBuffer = {0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA,0x0D,0x0A}
// string = "機器人"

ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
string var_value = socket_read_string("ntd_a", "lo", newline) // Prefix "lo" Suffix \u0D0A
// string = "lo,\u0D0A"
// The data before the 1st match, "Hel", will be removed.
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
var_value = socket_read_string("ntd_a", newline, 1) // Suffix \u0D0A
// string = "World"
// ReceivedBuffer = {}

ReceivedBuffer = {0x24,0x48,0x65,0x6C,0x6C,0x6F,0x0D,0x0A, // $Hello
                  0x23,0x48,0x69,0x0D,0x0A, // #Hi
                  0x24,0x54,0x4D,0x0D,0x0A, // $TM
                  0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A} // #Robot
string var_value = socket_read_string("ntd_a", "#", newline, 2) // Prefix "#" Suffix \u0D0A
// string = "" // Return an empty string for # not in the prefix.
// ReceivedBuffer = {0x24,0x48,0x65,0x6C,0x6C,0x6F,0x0D,0x0A,0x23,0x48,0x69,0x0D,0x0A,
                    0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}
var_value = socket_read_string("ntd_a", "$", newline, 2) // Prefix "$" Suffix \u0D0A
// string = "$Hello\u0D0A" // Fetch the 1st match in the prefix required.
// ReceivedBuffer = {0x23,0x48,0x69,0x0D,0x0A,
                    0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}
var_value = socket_read_string("ntd_a", "#", newline, 2) // Prefix "#" Suffix \u0D0A
// string = "#Hi\u0D0A" // Fetch the 1st match in the prefix required.
// ReceivedBuffer = {0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

var_value = socket_read_string("ntd_a", "$", newline, 3) // Prefix "$" Suffix \u0D0A
// string = "TM"
// ReceivedBuffer = {0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}
var_value = socket_read_string("ntd_a", "#", newline, 3) // Prefix "#" Suffix \u0D0A
// string = "Robot"
// ReceivedBuffer = {}

```

```

ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64}

string var_value = socket_read_string("ntd_a", newline) // Suffix \u0D0A
// string = "Hello,\u0D0A"
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64}

var_value = socket_read_string("ntd_a", newline, 0) // Suffix \u0D0A
// string = "" // Read a empty string for the fetch option unfulfilled.
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64}

var_value = socket_read_string("ntd_a", newline, 1, 100) // Suffix \u0D0A
// string = "" // Read a empty string for the fetch option unfulfilled. Wait for 100ms.
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64}
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A,0x31,0x32,0x33,0x0D,0x0A}
// Supposed the data comes in and fulfills the fetch option.
// string = "World"
// ReceivedBuffer = {0x31,0x32,0x33,0x0D,0x0A}

var_value = socket_read_string("ntd_a", newline, 2) // Suffix \u0D0A
// string = "123\u0D0A"
// ReceivedBuffer = {}

```

6.3 socket_send()

Send the data value to a remote device.

Syntax 1

```

int socket_send(
    string,
    ?,
    int,
    int
)

```

Parameters

- | | | | | | | | |
|----------------------|--|----------------------|---|---------------|--|---------------------|--|
| string | Network device name (configured in Network Device) | | | | | | |
| ? | The value to write. Available types: int, float, , bool, string, and array.
Numeric values will be conversed in Little Endian, and string values will be converse in UTF8. | | | | | | |
| int | The starting index of the value to write (eligible for strings and arrays) <table border="0" style="margin-left: 20px;"> <tr> <td>0 .. length-1</td> <td>Legitimate value</td> </tr> <tr> <td>< 0</td> <td>Illegitimate value. The starting index will be set to 0.</td> </tr> <tr> <td>>= length</td> <td>Illegitimate value. The starting index will be set to 0.</td> </tr> </table> | 0 .. length-1 | Legitimate value | < 0 | Illegitimate value. The starting index will be set to 0. | >= length | Illegitimate value. The starting index will be set to 0. |
| 0 .. length-1 | Legitimate value | | | | | | |
| < 0 | Illegitimate value. The starting index will be set to 0. | | | | | | |
| >= length | Illegitimate value. The starting index will be set to 0. | | | | | | |
| int | The length of the value to write (eligible for strings and arrays) <table border="0" style="margin-left: 20px;"> <tr> <td><= 0</td> <td>Write from the starting index to the end of the data.</td> </tr> </table> | <= 0 | Write from the starting index to the end of the data. | | | | |
| <= 0 | Write from the starting index to the end of the data. | | | | | | |

> 0

Write from the starting index for a specified number of the length up to the data ends.

Return

<code>int</code>	Send result
<code>1</code>	Sent successfully.
<code>0</code>	Unable to send the data value as an empty string or an empty array.
<code>-1</code>	Socket exception occurred during sending.
<code>-2</code>	Unable to connect to the remote device.
<code>-3</code>	The device name does not exist, or IP or port is incorrect.

Syntax 2

```
int socket_send (
    string,
    ?,
    int
)
```

Note

Same as syntax 1. Fill 0 as the length of the value to write by default.

Syntax 3

```
int socket_send (
    string,
    ?
)
```

Note

Same as syntax 1. Fill 0 as the starting index and the length of the value to write by default.

```
int var_re = socket_send("ntd_a", 100)           // send 0x64
var_re = socket_send("ntd_a", 1000)              // send 0xE8,0x03,0x00,0x00 (int, Little Endian)
var_re = socket_send("ntd_a", (float)1.234)      // send 0xB6,0xF3,0x9D,0x3F (float, Little Endian)
var_re = socket_send("ntd_a", (double)123.456)   // send 0x77,0xBE,0x9F,0x1A,0x2F,0xDD,0x5E,0x40 (double, Little Endian)
var_re = socket_send("ntd_a", "Hello, World"+newline)
// send 0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A (string, UTF8)
int[] var_ii = {100, 200, 300, 400}
var_re = socket_send("ntd_a", var_ii)
// send 0x64,0x00,0x00,0x00,0xC8,0x00,0x00,0x00,0x2C,0x01,0x00,0x00,0x90,0x01,0x00,0x00 (int[], Little Endian)
string[] var_ss = {"TM", "", "Robot"}
var_re = socket_send("ntd_a", var_ss)
```

```

// send 0x54,0x4D,0x52,0x6F,0x62,0x6F,0x74 (string[], UTF8)
// var_ss[1] is an empty string. The conversion value is still empty.

var_re = socket_send("ntd_a", 1000, 1, 2) // Invalid in the value, the starting index, and the length
// send 0xE8,0x03,0x00,0x00 (int, Little Endian)
var_re = socket_send("ntd_a", "Hello, World"+newline, 0, 7)
// send 0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x20 (string, UTF8)

byte[] var_bb = {100, 200}
var_re = socket_send("ntd_a", var_bb) // send 0x64,0xC8
var_re = socket_send("ntd_a", var_bb, 1, 1) // send 0xC8 // Array. Read 1 from address 1,
[1]=200
var_re = socket_send("ntd_a", var_bb, -1, 1) // send 0x64 // Array. Read 1 from address 0.
[0]=100

var_re = socket_send("ntd_a", "達明機器人", 2) // Array. Read from address 2 till the end. "機器人"

// send 0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA (string, UTF8)

var_ss = {"TM", "", "達明機器人"}

var_re = socket_send("ntd_a", var_ss)
// send 0x54,0x4D,0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA
var_re = socket_send("ntd_a", Byte_Concat(GetBytes(var_ss), GetBytes(newline)))
// send 0x54,0x4D,0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA,0x0D,0x0A
var_re = socket_send("ntd_a", var_ss, 2, 100)
// Array. Read 100 from address 2. (till the end) [2]=達明機器人
// send 0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA

```

6.4 socket_sendline()

Send the data value and add line break symbols, 0x0D 0x0A, to a remote device.

Syntax 1

```

int socket_sendline (
    string,
    ?,
    int,
    int
)

```

Parameters

string Network device name (configured in Network Device)

`?` The value to write. Available types: int, float, , bool, string, and array.
 Numeric values will be conversed in Little Endian, and string values will be converse in UTF8.

`int` The starting index of the value to write (eligible for strings and arrays)

- `0 .. length-1` Legitimate value
- `< 0` Illegitimate value. The starting index will be set to 0.
- `>= length` Illegitimate value. The starting index will be set to 0.

`int` The length of the value to write (eligible for strings and arrays)

- `<= 0` Write from the starting index to the end of the data.
- `> 0` Write from the staring index for a specified nubmer of the length up to the data ends.

Return

`int` Send result

- `1` Sent successfully.
- `0` Unable to send the data value as an empty string or an empty array.
- `-1` Socket exception occurred during sending.
- `-2` Unable to connect to the remote device.
- `-3` The device name does not exist, or IP or port is incorrect.

Syntax 2

```
int socket_sendline (
    string,
    ?,
    int
)
```

Note

Same as syntax 1. Fill 0 as the length of the value to write by default.

Syntax 3

```
int socket_sendline (
    string,
    ?
)
```

Note

Same as syntax 1. Fill 0 as the starting index and the length of the value to write by default.

```
int var_re = socket_sendline("ntd_a", 200) // send 0xC8,0x0D,0x0A
var_re = socket_sendline("ntd_a", 2000) // send 0xD0,0x07,0x00,0x00,0x0D,0x0A (int, Little Endian)
var_re = socket_sendline("ntd_a", (float)0.234) // send 0xB2,0x9D,0x6F,0x3E,0x0D,0x0A (float, Little Endian)
var_re = socket_sendline("ntd_a", (double)0.234)
```

```

// send 0xC1,0xCA,0xA1,0x45,0xB6,0xF3,0xCD,0x3F,0x0D,0x0A (double, Little Endian)
var_re = socket_sendline("ntd_a", "Hello, World"+newline)
// send 0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A,0x0D,0x0A (string, UTF8)
int[] var_ii = {100, 200, 300}
var_re = socket_sendline("ntd_a", var_ii)
// send 0x64,0x00,0x00,0x00,0xC8,0x00,0x00,0x00,0x2C,0x01,0x00,0x00,0x0D,0x0A (int[], Little Endian)
string[] var_ss = {"TM", "", "Robot"}
var_re = socket_sendline("ntd_a", var_ss)
// send 0x54,0x4D,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A (string[], UTF8)
//var_ss[1] is an empty string. The conversion value is still empty.

var_re = socket_sendline("ntd_a", 1000, 1, 2) // Invalid in the value, the starting index, and the length
// send 0xE8,0x03,0x00,0x00,0x0D,0x0A (int, Little Endian)
var_re = socket_sendline("ntd_a", "Hello, World"+newline, 0, 7)
// send 0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x20,0x0D,0x0A (string, UTF8)
byte[] var_bb = {123, 234}
var_re = socket_sendline("ntd_a", var_bb) // send 0x7B,0xEA,0x0D,0x0A
var_re = socket_sendline("ntd_a", var_bb, 1, 1) // send 0xEA,0x0D,0x0A //Array. Read 1 from address 1.
var_re = socket_sendline("ntd_a", var_bb, -1, 1) // send 0x7B,0x0D,0x0A // Array. Read 1 from address 0.
var_re = socket_sendline("ntd_a", "達明機器人", 2) // Array. Read from address 2 till the end. "機器人"
// send 0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA,0x0D,0x0A (string, UTF8)
var_ss = {"TM", "", "達明機器人"}
var_re = socket_sendline("ntd_a", var_ss)
// send 0x54,0x4D,0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA,0x0D,0x0A
var_re = socket_sendline("ntd_a", Byte_Concat(GetBytes(var_ss), GetBytes(newline)))
// send 0x54,0x4D,0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA,0x0D,0x0A,0x0D,0x0A
var_re = socket_sendline("ntd_a", var_ss, 2, -1)
// Array. Read from address 2 till the end. [2]=達明機器人
// send 0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA,0x0D,0x0A

```

7. Parameterized objects

Using parameterized objects is the same as using user defined variables. Parameterized objects can be used without declarations to get or modify point data through the syntaxes in the project operations and make the robot go with more flexibility. The expression comes with 3 parts, item, index, and attribute, and the syntax is shown as below.

`parameterized item[index].attribute`

The supported parameterized items include:

1. Point
2. Base
3. TCP
4. VPoint
5. IO
6. Robot
7. FT

Definitions of the indexes and the attributes vary from parameterized items.

Take the reading and writing of the `coordinate` (attribute) of the `Point` (item) `"P1"` (index) as a example. The index is defined as the name of the point, and the attribute, as the data type of float (the same usage as the array's) with modes of reading and writing.

Value	<code>float[]</code>	R/W	point coordinate{X,Y,Z,RX,RY,RZ}
-------	----------------------	-----	----------------------------------

Read values

<code>float[] var_f = Point["P1"].Value</code>	<code>// In the item Point, the index is defined as the name of the point and the data type of string.</code>
<code>float var_f1 = Point["P1"].Value[0]</code>	<code>// The x value of "P1" can be obtained solely</code>

Write values

<code>Point["P1"].Value = {0, 0, 90, 0, 90, 0}</code>	<code>// Replace to the coordinate of "P1" with {0,0,90,0,90,0}</code>
<code>Point["P1"].Value[2] = 120</code>	<code>// or replace the z value of "P1" with 12 solely</code>

7.1 Point

Syntax

`Point[string].attribute`

Item

`Point`

Index

`string` The name of the point in the point manager

Attribute

Name	Type	Mode	Description	Format
Value	float[]	R/W	The coordiniate of the point	{X, Y, Z, RX, RY, RZ}, Size = 6
Pose	int[]	R/W	The pose of the robot	{Config1, Config2, Config3}, Size = 3
Flange	float[]	R	The coordinate of the flange'scenter	{X, Y, Z, RX, RY, RZ}, Size = 6
BaseName	string	R	The name of the base	"Base Name"
TCPName	string	R	The name of the TCP	"TCP Name"
TeachValue	float[]	R	The original coordiniate of the teaching point	{X, Y, Z, RX, RY, RZ}, Size = 6
TeachPose	int[]	R	The original pose of the roboton the teaching point	{Config1, Config2, Config3}, Size = 3

Note

```
// Read values
float[] var_f = Point["P1"].Value           // Obatin the coordinate {X, Y, Z, RX, RY, RZ} of "P1"
float var_f1 = Point["P1"].Value[0]         // or retrieve the x value of "P1" solely
float var_f1 = Point["P1"].Value[6]         // Return error, exceeding the array's access range
string var_s =Point["P1"].BaseName          // var_s ="RobotBase"

// Write values
Point["P1"].Value = {0, 0, 90, 0, 90, 0}    // Replace the coordinate of "P1" with {0,0,90,0,90,0}
Point["P1"].Value[2] = 120                  // or replace the z value of "P1" with 120 solely
Point["P1"].Flange = {0, 0, 90, 0, 90, 0}   // Read only, invalid operation
Point["P1"].Value = {0, 0, 90, 0, 90}       // Return error, writing elements to the array do not match to 6
                                           // (writing 5 elements)
Point["P1"].Pose = {1, 2, 4, 0}             // Return error, writing elements to the array do not match to 3
                                           // (writing 4 elements)
```

7.2 Base

Syntax

```
Base[string].attribute
```

```
Base[string, int].attribute
```

Item

Base

Index

string The name of the base in the base manager
*The name of the base comes with the attribute of the mode in reading without writing only.
 "RobotBase"

int The index of the base, available to assign with multiple bases built by vision one shot get all, ranging from 0 as the default to N.

Attribute

Name	Type	Mode	Description	Format
Value	float[]	R/W	The value of the base	{X, Y, Z, RX, RY, RZ}, Size = 6
Type	string	R	The type of the base	"R": Robot Base "V": Vision Base "C": Custom Base
TeachValue	float[]	R	The original teaching value of the base	{X, Y, Z, RX, RY, RZ}, Size = 6

Note

```
// Read values
float[] var_f = Base["RobotBase"].Value           // Obtain the base value {0,0,0,0,0,0} of the base "RobotBase"
float var_f1 = Base["base1"].Value[0]             // or retrieve the x value of "base1" solely
string var_s = Base["base1"].Type                 // var_s = "C"
var_s = Base[Point["P1"].BaseName].Type           // var_s = "R"           // Given the type of "P1" is
                                                    "RobotBase"

float[] var_f = Base["vision_osga",1].Value        // Obtain the 2nd value of the "vision_osga"

// Write values
Base["RobotBase"].Value = {0, 0, 90, 0, 90, 0}     // Read only, invalid operation, because "RobotBase" is the
                                                    system coordinate system
Base["base1"].Value = {0, 90, 0, 0, 90, 0}         // Replace the value of "base1" with {0,90,0,0,90,0}
Base["base1"].Value[4] = 120                      // or replace the RY value of "base1" with 120 solely
Base["base1"].Value[6] = 120                      // Return error, exceeding the array's access range
```

```

Base["base1"].Type = "C" // Read only, invalid operation
Base["base1"].Value = {0, 0, 90, 0, 90} // Return error, writing elements to the array do not match to
                                         6 (writing 5 elements)
Base["base1"].Value = {0, 0, 90, 0, 90, 0, 100} // Return error, writing elements to the array do not match to
                                                  6 (writing 7 elements)

```

7.3 TCP

Syntax

`TCP[string].attribute`

Item

`TCP`

Index

`string` The name of the TCP in the TCP list

*The name of the TCP comes with the attribute of the mode in reading without writing only.

"NOTOOL"

"HandCamera"

Attribute

Name	Type	Mode	Description	Format
Value	float[]	R/W	The value of the TCP	{X, Y, Z, RX, RY, RZ}, Size = 6
Mass	float	R/W	The value of mass	Mass in kg
MOI	float[]	R/W	The value of the Principal Moments of Inertia	{Ixx, Iyy, Izz}, Size = 3
MCF	float[]	R/W	The value of Mass center frame with principle axes w.r.t tool frame	{X, Y, Z, RX, RY, RZ}, Size = 6
TeachValue	float[]	R	The original value of the TCP	{X, Y, Z, RX, RY, RZ}, Size = 6
TeachMass	float	R	The original value of mass	Mass in kg
TeachMOI	float[]	R	The original value of the Principal Moments of Inertia	{Ixx, Iyy, Izz}, Size = 3
TeachMCF	float[]	R	The original value of Mass center frame with principle axes w.r.t tool frame	{X, Y, Z, RX, RY, RZ}, Size = 6

Note

```

// Read values
float[] var_f = TCP["NOTOOL"].Value // Obtain the value {0,0,0,0,0,0} of the TCP "NOTOOL"
float var_f1 = TCP["NOTOOL"].Value[0] // or retrieve the x value of "NOTOOL" solely

```

```

float var_mass = TCP["T1"].Mass           // var_mass = 2.0
float[] var_moi = TCP["T1"].MOI           // var_moi = {0,0,0}
float[] var_mcf = TCP["T1"].MCF           // var_mcf = {0,0,0,0,0,0}
// Write values
TCP["NOTOOL"].Value = {0, -10, 0, 0, 0, 0} // Read only, invalid operation, because "NOTOOL" is the
                                           // system TCP
TCP["T1"].Value = {0, -10, 0, 0, 0, 0}     // Replace the value of "T1" with {0,-10,0,0,0,0}
TCP["T1"].Value[0] = 10                   // or replace the X value of "T1" with 10 solely
TCP["T1"].Mass = 2.4                      // Replace the mass value of "T1" with 2.4 kg
TCP["T1"].MOI = {0, 0, 0, 1, 2}           // Return error, writing elements to the array do not match to
                                           // 3 (writing 5 elements)
TCP["T1"].MCF = {0, -20, 0, 0, 0, 0, 0}   // Return error, writing elements to the array do not match to
                                           // 6 (writing 7 elements)

```

7.4 VPoint

Syntax

VPoint[string].attribute

Item

VPoint Initial position of the vision job

Index

string The name of the VPoint

Attribute

Name	Type	Mode	Description	Format
Value	float[]	R/W	The initial coordinate of VPoint	{X, Y, Z, RX, RY, RZ}, Size = 6
BaseName	string	R	The name of the VPoint	"Base Name"
TeachValue	float[]	R	The original job initial coordinate of VPoint	{X, Y, Z, RX, RY, RZ}, Size = 6

Note

```

// Read values
float[] var_f = VPoint["Job1"].Value     // Obtain the initial coordinate {X, Y, Z, RX, RY, RZ} of VPoint
"Job1"
float var_f1 = VPoint["Job1"].Value[0]   // or retrieve the x value of "Job1"
float var_f1 = VPoint["Job1"].Value[6]   // Return error, exceeding the array's access range
string var_s = VPoint["Job1"].BaseName   // var_s = "RobotBase"
// Write values
VPoint["Job1"].Value = {0, 0, 90, 0, 90, 0} // Replace the initial coordinate of VPoint "Job1" with

```

```

VPoint["Job1"].Value[2] = 120           {0,0,90,0,90,0}
VPoint["Job1"].BaseName = "base1"      // or replace the Z value of "Job1" with 120 solely
VPoint["Job1"].Value = {0, 0, 90, 0, 90} // Read only, invalid operation
VPoint["Job1"].Value = {0, 0, 90, 0, 90} // Return error, writing elements to the array do not match to
VPoint["Job1"].Value = {0, 0, 90, 0, 90, 0, 100} // Return error, writing elements to the array do not match to

```

7.5 IO

Syntax

IO[string].attribute

Item

IO

Index

string The name of the control module
ControlBox
EndModule
ExtModuleN (N = 0 .. n)
Safety

Attribute

ControlBox / EndModule / ExtModuleN

Name	Type	Mode	Description	Format
DI	byte[]	R	Digital input	[0] = DI0 0: Low, 1: High [1] = DI1 [n] = DI n
DO	byte[]	R/W	Digital output	[0] = DO0 0: Low, 1: High [1] = DO1 [n] = DO n
AI	float[]	R	Analog input	-10.24V .. +10.24V (Voltage)
AO	float[]	R/W	Analog output	-10.00V .. + 10.00V (Voltage)
InstantDO	byte[]	R/W	Digital output (Instant Command)	[0] = DO0 0: Low, 1: High [1] = DO1 [n] = DO n
InstantAO	float[]	R/W	Analog output (Instant Command)	-10.00V .. + 10.00V (Voltage)

* The set of DI[n]/DO[n]/AI[n]/AO[n] vary from actual hardware device identification.



Safety

Name	Type	Mode	Description	Format
SI	byte[]	R	Safety function input	0: Low, 1: High SI[0] = SF1 User Connected ESTOP input SI[1] = SF3 User Connected External Safeguard Input SI[2] = SF9 User Connected External Safeguard Input for Human-Machine Safety Setting SI[3] = SF15 User Connected Enabling Device Input SI[4] = SF16 User Connected ESTOP Input without Robot ESTOP Output
SO	byte[]	R	Safety function output	0: Low, 1: High SO[0] = SF10 Robot ESTOP Output SO[1] = SF11 User Connected External Safeguard Output SO[2] = SF12 User Connected External Safeguard Output for Human-Machine Safety Settings SO[3] = SF13 Robot Internal Protective Stop Output SO[4] = SF14 Robot Encoder Standstill Output

The difference between Do and InstantDO

DO is the queue command with reservations in the main flow of the project. If a DO is after the robot motion function such as a point node with the mixture of trajectories, the DO will be operated after the point node is finished. If using an InstantDO command, the DO will be operated while the point node is on the run and without the wait for the point node finishes, and the result is the same as using DO in a thread page.

Note

// Read values

```
byte[] var_di = IO["ControlBox"].DI           // Obtain the digital input status of ControlBox
int var_dilen = Length(di)                   // Obtain the amount of digital PINs with the size of the array
byte var_di0 = IO["ControlBox"].DI[0]        // Obtain the status of ControlBox DI[0]
byte var_di32 = IO["ControlBox"].DI[32]      // Return error, exceeding the array's access range (given DI is an
                                              array with the length of 16 where the indexes start with 0 and end with
                                              15.

float[] var_ai = IO["ControlBox"].AI          // Obtain the analog input status of ControlBox
float[] var_ao = IO["ControlBox"].AO         // Obtain the analog output status of ControlBox
byte si0 = IO["Safety"].SI[0]                // Obtain the safety input status of Safety SI[0]
byte so4 = IO["Safety"].SO[4]                // Obtain the safety output status of Safety SO[4]
byte si1 = IO["ControlBox"].SI[1]            // Return error, ControlBox does not support SI attribute.
byte di2 = IO["Safety"].DI[2]                // Return error, Safety does not support DI attribute.
```

```

// Write values
IO["ControlBox"].DI = {1,1,0,0} // Read only, invalid operation
IO["ControlBox"].DI[0] = 0 // Read only, invalid operation
IO["ControlBox"].DO[2] = 1 // Set DO 2 to High
IO["ControlBox"].AO[0] = 3.3 // Set AO0 to 3.3V
IO["ControlBox"].DO = {1,1,0,0} // Return error, elements to write mismatch to array's size (given DI is an
// array with the length of 16 which covers 16 elements)

IO["ControlBox"].InstantDO[0] = 1 // Set DO 0 to High (Instant Execution)
IO["Safety"].SI[0] = 0 // Read only, invalid operation
IO["Safety"].SO[4] = 1 // Read only, invalid operation
IO["ControlBox"].SO[1] = 1 // Return error, ControlBox does not support SO attribute.
IO["Safety"].DO[2] = 1 // Return error, Safety does not support DO attribute.

```

7.6 Robot

Syntax

`Robot[int].attribute`

Item

`Robot`

Index

`int` The index of the robot fixed at 0

Attribute

Name	Type	Mode	Description	Format
<code>CoordRobot</code>	<code>float[]</code>	R	The TCP coordinate of the robot end point opposite to the RobotBaseof the robot	{X, Y, Z, RX, RY, RZ}, Size = 6
<code>CoordBase</code>	<code>float[]</code>	R	The TCP coordinate of the robot end point opposite to the current base fo the robot.	{X, Y, Z, RX, RY, RZ}, Size = 6
<code>Joint</code>	<code>float[]</code>	R	The current robot joint angle	{J1, J2, J3, J4, J5, J6}, Size = 6
<code>BaseName</code>	<code>string</code>	R	The name of the current base	"Base Name"
<code>TCPName</code>	<code>string</code>	R	The name of the current TCP	"TCP Name"
<code>CameraLight</code>	<code>byte</code>	R/W	The lighting of the robot's camera	0: Low (Off), 1: High (On)
<code>TCPForce3D</code>	<code>float</code>	R	The current TCP force as the composite force of the robot base x,	N

			y, and z.	
TCPSpeed3D	float	R	The current TCP speed as a composite speed of the robot base x, y, and z.	mm/s

Note

```
// Read values

float[] var_rtool = Robot[0].CoordRobot // Obtain the current TCP coordinate of the robot end point
                                         // opposite to the RobotBaseof the robot

float[] var_ftool = Robot[0].CoordBase  // Obtain the current TCP coordinate of the robot end point
                                         // opposite to the current base fo the robot.

float var_f = Robot[0]. CoordBase[0]    // or retrieve the X value of the current TCP coordinate of the robot
                                         // end point opposite to the current base fo the robot solely.

var_f = Robot[0]. CoordBase[6]          // Return error, exceeding the array's access range

float[] var_joint = Robot[0].Joint       // Obtain the current robot joint angle

float var_j = Robot[0].Joint[0]         // or retrieve the current angle of the robot's 1st joint solely

string var_b = Robot[0].BaseName        // var_b = "RobotBase"

string var_t = Robot[0].TCPName         // var_t = "NOTOOL"

byte var_light = Robot[0].CameraLight   // var_light = 0 (OFF)

float var_tf3d = Robot[0].TCPForce3D    // var_tf3d = 1.234

float var_ts3d = Robot[0].TCPSpeed3D    // var_ts3d = 1.234

// Write values

Robot[0].CoordRobot = {0, 90, 0, 0, 0, 0} // Read only, invalid operation
Robot[0].CoordBase = {0, 0, 90, 0, 90, 0} // Read only, invalid operation
Robot[0].BaseName = "Base1"              // Read only, invalid operation
Robot[0].TCPName = "Tool1"              // Read only, invalid operation
Robot[0].CameraLight = 1                 // Turn on the lighting of the robot's camera
Robot[0].CameraLight = 0                 // Turn off the lighting of the robot's camera
Robot[0].TCPSpeed3D = 1.234              // Read only, invalid operation
```

7.7 FT

Syntax

FT[string].attribute

Item

FT Force Torque sensor status

Index

string The name of F/T sensor in the F/T sensor list

Attribute

Name	Type	Mode	Description	Format
X	float	R	The strength value of the X axis	
Y	float	R	The strength value of the y axis	
Z	float	R	The strength value of the z axis	
TX	float	R	The torque value of the X axis	
TY	float	R	The torque value of the y axis	
TZ	float	R	The torque value of the z axis	
F3D	float	R	The XYZ force strength value	
T3D	float	R	The XYZ force torque value	
ForceValue	float[]	R	The XYZ force strength value array	{X, Y, Z}, Size = 3
TorqueValue	float[]	R	The XYZ force torque value array	{TX, TY, TZ}, Size = 3
RefCoordX	float	R	The X-axis strength value measured based on the reference coordinate system set in the node	
RefCoordY	float	R	The Y-axis strength value measured based on the reference coordinate system set in the node	
RefCoordZ	float	R	The Z-axis strength value measured based on the reference coordinate system set in the node	
RefCoordTX	float	R	The X-axis torque value measured based on the reference coordinate system set in the node	
RefCoordTY	float	R	The Y-axis torque value measured based on the reference coordinate system set in the node	
RefCoordTZ	float	R	The Z-axis torque value measured based on the reference coordinate system set in the node	
RefCoordF3D	float	R	The XYZ strength measured based on the reference coordinate system set in the node	
RefCoordT3D	float	R	The XYZ torque measured based on the reference coordinate system set in the node	
RefCoordForceValue	float[]	R	The XYZ strength value matrix measured based on the reference coordinate system set in the node	{RefCoordX, RefCoordY, RefCoordZ}, Size = 3

RefCoordTorqueValue	float[]	R	The XYZ torque value matrix measured based on the reference coordinate system set in the node	{RefCoordTX, RefCoordTY, RefCoordTZ}, Size = 3
Model	string	R	The Model name of the F/T sensor	
Zero	byte	R/W	Turn on or off F/T sensor offset	0: Zero OFF, 1: Zero ON

Note

// Read values

float var_x = FT["fts1"].X

// Obtain the current X-axis strength value of F/T sensor "fts1"

float var_tx = FT["fts1"].TX

// Obtain the current X-axis torque value of F/T sensor "fts1"

float var_f3d = FT["fts1"].F3D

// Obtain the current XYZ force value of F/T sensor "fts1"

float[] var_force = FT["fts1"].ForceValue

// Obtain the current XYZ force strength value array of F/T sensor "fts1"

string var_mode = FT["fts1"].Model

// Obtain the model name of F/T sensor "fts1"

// Write values

FT["fts1"].Y = 3.14

// Read only, invalid operation

FT["fts1"].TY = 1.34

// Read only, invalid operation

FT["fts1"].T3D = 4.13

// Read only, invalid operation

FT["fts1"].TorqueValue = {1.1, 2.2, 3.3}

// Read only, invalid operation

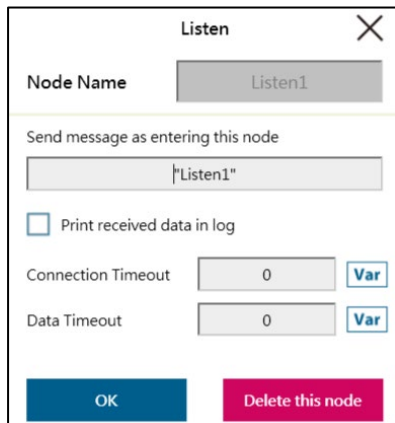
FT["fts1"].Zero = 1

// Book the current offset of F/T sensor

8. External Script

8.1 Listen Node

Users can establish a socket TCPListener (server site) in the listen node to connect to external devices and communicate based on the packet format. All features available in TM_Robot_Function can also be operated in the listen node.



The screenshot shows the 'Listen' configuration dialog box. It has a title bar with a close button. Inside, there's a 'Node Name' field with 'Listen1'. Below it, a text area for 'Send message as entering this node' contains 'Listen1'. There's a checkbox for 'Print received data in log'. Two numeric input fields for 'Connection Timeout' and 'Data Timeout' both have '0' and a 'Var' button. At the bottom are 'OK' and 'Delete this node' buttons.

- **Send Message:** When entering this node, it will initiate a message
- **Print Log:** Enable Communication Log (shown on the right)
- **Connection Timeout:** When entering this node, if more than the time (milliseconds) is not connected, it will be overtime.
If ≤ 0 , no timeout
- **Data Timeout:** When connected, the timeout will be exceeded when there is no communication packet
If ≤ 0 , no timeout

Socket TCPListener is started up after the project being executed, and closed as the project stopped. The IP and listen port will be shown on the Notice Log window on the right, after the Socket TCPListener is started up.

IP HMI → System → Network → IP Address

Port 5890

When entering the Listen Node, the flow will keep at Listen Node until either of the the two exit conditions is fulfilled.

Pass: *ScriptExit()* is executed or the project is stopped

Fail: 1. Connection Timeout

2. Data Timeout

3. Before the TCP Listener is started up, the flow has entered this Listen Node

The command received by listen node will be executed in order. If the command is not valid, an error message will be returned carrying the line number with errors. If the command is valid, it will be executed.

The command can be divided into two categories. The first category is commands which can be accomplished in instance, like assigning variable value. The second category is commands needs to be executed in sequence, like motion command and IO value assigning. The second category command will be placed in queue and executed in order.

<i>Header</i>	<i>X</i>			Header for Communication
Separator	1	,	0x2C	Separator between Header and Length
<i>Length</i>	<i>Y</i>			Length of Data
Separator	1	,	0x2C	Separator between Length and Data
<i>Data</i>	<i>Z</i>			Communication Data
Separator	1	,	0x2C	Separator between Data and Checksum
Sign	1	*	0x2A	Begin Sign of Checksum
<i>Checksum</i>	<i>2</i>			Checksum of Communication
End Byte 1	1	\r	0x0D	
End Byte 2	1	\n	0x0A	End Byte of Communication

1. Header

Defines the purpose of the communication package. The data definition could be different with different Header.

- *TMSCT* External Script
- *TMSTA* Acquiring status or properties
- *CPERR* Communication data error (E.g. Packet error, checksum error, header error, etc.)

2. Length

Length defines the length in UTF8 byte. It can be represented in decimal, hexadecimal or binary, the upper limit is int 32bits

Example:

```
$TMSCT,100,Data,*CS\r\n    // Decimal 100, that is the data length is 100 bytes
$TMSCT,0x100,Data,*CS\r\n  // Hexadecimal 0x100, that is the data length is 256 bytes
$TMSCT,0b100,Data,*CS\r\n  // Binary 0b100, that is the data length is 4 bytes
$TMSCT,8,1,達明,*58\r\n    // The Data length 1,達明 is 8 bytes (UTF8)
```

3. Data

The content of the communication package. Arbitrary characters are supported (including 0x00 .. 0xFF in UTF8).

The data length is defined in Length and the purpose is defined in Header

4. Checksum

The checksum of the communication package. The checksum is calculated with XOR(exclusive OR), and the range for checksum computation starts from \$ to * (\$ and * are excluded) as shown below:

```
$TMSCT,100,Data,*CS\r\n
```

Checksum = Byte[1] ^ Byte[2] ... ^ Byte[N-6]

The representation of checksum is fixed to 2 bytes in hexadecimal format (without 0x).

For example:

`$TMSCT,5,10,OK,*6D`

$CS = 0x54 \wedge 0x4D \wedge 0x53 \wedge 0x43 \wedge 0x54 \wedge 0x2C \wedge 0x35 \wedge 0x2C \wedge 0x31 \wedge 0x30 \wedge 0x2C \wedge 0x4F \wedge 0x4B \wedge 0x2C = 0x6D$

$CS = 6D (0x36 \ 0x44)$

8.4 TMSCT

Start Byte	Hdr		Len		Data			Checksum	End Byte1	End Byte2
\$	TMSCT	,	Length	,	Data	,	*	Checksum	\r	\n

ID		SCRIPT
Script ID	,	Script Language

TMSCT defines the communication package as External Script Language. In External Script Language, the data contains two parts and is separated by comma. One is ID and the other is SCRIPT

- ID** Script ID, can be arbitrary English character or number (a CPERR 04 error will be reported when encountering non-alphanumeric byte). The ID is used as specifying the target SCRIPT of return message.
- ,** Separator
- SCRIPT** The content defined in Script Language. In a communication package, multi-line scripts can fit into the SCRIPT section with separator (0x0D 0x0A)

Note

TMSCT is available only when in the external script control mode, otherwise CPEER error packets will be replied.

Return (Robot→External Device)

- When it enters Listen Node, the robot will send a message to all the connected device. The ID is set to 0.

`$TMSCT,9,0,Listen1,*4C\r\n`

- 9** The length of `0,Listen1` is 9 bytes
- 0** The Script ID is 0
- ,** Separator
- Listen1** The message to send

- The OK or ERROR message is replied according to the Script's content. For message with `;N`, `;N` represents the number of line with error or warning. After the message is received, robot will

execute the message, then send back the return message, if the Script is valid. For invalid Script, the return message will be sent back immediately without executed.

```
$TMSCT,4,1,OK,*5C\r\n      // Response to ID 1
                             // OK means valid Script.

$TMSCT,8,2,OK;2;3,*52\r\n  // Response to ID 2
                             // OK;2;3 means valid Script with warnings in line 2 and 3.

$TMSCT,13,3,ERROR;1;2;3,*3F\r\n // Response to ID 3
                             // ERROR;1;2;3 means invalid Script with errors in line 1, 2 and 3.
```

Receive (Robot←External Device)

1. When it enters the listen node, the robot will start to receive, check, and execute the external script. If the robot did not enter the listen node (not in the external script control mode), the Script received will be disposed and CPEER error packets will be replied.

2. The message from external device should define the Script ID as a ID used in messages replied by robot.

```
<  $TMSCT,25,1,ChangeBase("RobotBase"),*08\r\n  // Defined as ID 1
>  $TMSCT,4,1,OK,*5C\r\n      // Response to ID 1
```

3. In a communication package, multi-line scripts can fit into the SCRIPT section with separator \r\n

```
<  $TMSCT,64,2,ChangeBase("RobotBase")\r\n
    ChangeTCP("NOTOOL")\r\n
    ChangeLoad(10.1),*68\r\n  // Three lines Script in a communication package (Lines are
    separated by \r\n)
>  $TMSCT,4,2,OK,*5F\r\n
```

4. In Listen Node, local variables are supported and valid before quitting the Listen Node.

```
<  $TMSCT,40,3,int var_i = 100\r\n
    var_i = 1000\r\n
    var_i++,*5A\r\n
>  $TMSCT,4,3,OK,*5E\r\n

<  $TMSCT,42,4,int var_i = 100\r\n
    var_i = 1000\r\n
    var_i++\r\n
    ,*58\r\n
```

> `$TMSCT,9,4,ERROR;1,*02\r\n` // Because int var_i has been declared, an error occurred.

5. In the listen node, it is possible to access or modify the project's variables, but no new variable can be declared since the variables created in the listen node are local variables.

8.5 TMSTA

Start Byte	Hdr		Len		Data			Checksum	End Byte1	End Byte2
\$	TMSTA	,	Length	,	Data	,	*	Checksum	\r	\n

SubCmd		
SubCmd	...	...

(Based on SubCmd)

TMSTA defines the communication package as acquiring status or properties. The data section of the package contains different sub command (SubCMD). The package format could be different according to different SubCMD.

The definitions are listed below.

SubCmd

- 00 In external script control mode or not
- 01 Complete the configured QueueTag numbering or not
- 90..99 Date message to send (the format of data is self-definable)

Note

TMSTA could be executed without entering the Listen Node

SubCmd 00 In external script control mode or not

Format

Response (Robot→External Device)

SubCmd		Entry		Message
00	,	false	,	
00	,	true	,	message

Receive (Robot←External Device)

SubCmd
00

Response (Robot→External Device)

1. If not in external script control mode, it will reply false.

\$TMSTA,9,00,false,,*37\r\n

- 9** Indicates the length of 00,false, is 9 bytes
- 00** Indicates SubCmd as 00
- ,** Separator
- false** The flow has not entered Listen Node
- ,** Separator
- Empty string (Have not entered Listen Node)

2. If in external script control mode, it will reply true.

\$TMSTA,15,00,true,Listen1,*79\r\n

- 15** Indicates the length of 00,true,Listen1 is 15 bytes
- 00** Indicates SubCmd as 00
- ,** Separator
- true** The flow has entered the Listen Node
- ,** Separator
- Listen1** The message to be sent as in Listen Node (It indicates the flow is in Listen1)

Receive (Robot←External Device)

1. Send to the robot from the external device

\$TMSTA,2,00,*41\r\n

- 2** Indicates the length of 00 is 2 bytes.
- 00** Indicates the SubCmd is 00 whether in external script control mode or not.

SubCmd 01 Complete the configured QueueTag numbering or not

Format

Send (Robot→External Device)

SubCmd		Tag Number		Status
01	,	01 .. 15	,	true/false/none

Receive (Robot←External Device)

SubCmd		Tag Number
01	,	01 .. 15

Note

When inquiring with TMSTA 01, users can look up to the status of the last 4 tag numbers.

Send (Robot→External Device)

1. Send to the external device from the robot. Spontaneously sending after QueueTag numbering completed.

`$TMSTA,10,01,08,true,*6D\r\n`

10	Indicates the length of 01,00,true is 10 bytes
01	Indicates SubCmd as 01 to send the status of Tag Number
,	Separation symbol
08	Tag Number 08
,	Separation symbol
true	<i>true</i> Indicates Tag Number complete
	<i>false</i> Indicates Tag Number incomplete
	<i>none</i> Indicates Tag Number not existed

Receive (Robot←External Device)

1. Send from the external device to the robot. Users can look up to the status of the last 4 tag numbers.

`$TMSTA,5,01,15,*6F\r\n`

5	Indicates the length of 01,88 is 5 bytes
01	Indicates SubCmd as 01 to send the status of Tag Number
,	Separation symbol
15	Tag Number 15

> `$TMSTA,10,01,15,none,*7D\r\n` // TagNumber 15 not existed

2. Tag Number uses the value of integers between 1 and 15. If the value is invalid, it relies none for not existed.

`$TMSTA,5,01,88,*6B\r\n`

> `$TMSTA,10,01,88,none,*79\r\n` // TagNumber 88 not existed

SubCmd 90..99 Send data message

Format

Send (Robot→External Device)

SubCmd		Data
90 .. 99	,	...

Receive (Robot←External Device)

None

Note

1. When sending with TMSTA 90 .. 99, users can use their self-defined formats.
2. Self-defined formats denote the formats are defined by both the project flow and the external device.
3. To enhance the flexibility of usages, users can various SubCmd of 90 .. 99 to define different formats to send such as

SubCmd 90 defined as string ;

SubCmd 91 defined as float[] ;

SubCmd 92 defined as byte[]

...

and so on for the external device to analyze and resolve based on the SubCmd with different methods.

Send (Robot→External Device)

1. Send to the external device from the robot. When the external script executes the ListenSend() function, it will send data.

```
string var_s = "Hello World"
```

```
float[] var_f = {1,2,3,4}
```

```
byte[] var_b = {0x10, 0x11, 0x12, 0x13}
```

```
ListenSend(90, var_s)
```

```
// the content of communication $TMSTA,14,90,Hello World,*73\r\n
```

```
// 0x39,0x30,0x2C,0x48,0x65,0x6C,0x6C,0x6F,0x20,0x57,0x6F,0x72,0x6C,0x64
```

```
ListenSend(91, var_f)
```

```
// the content of communication $TMSTA,19,91,...,*60\r\n
```

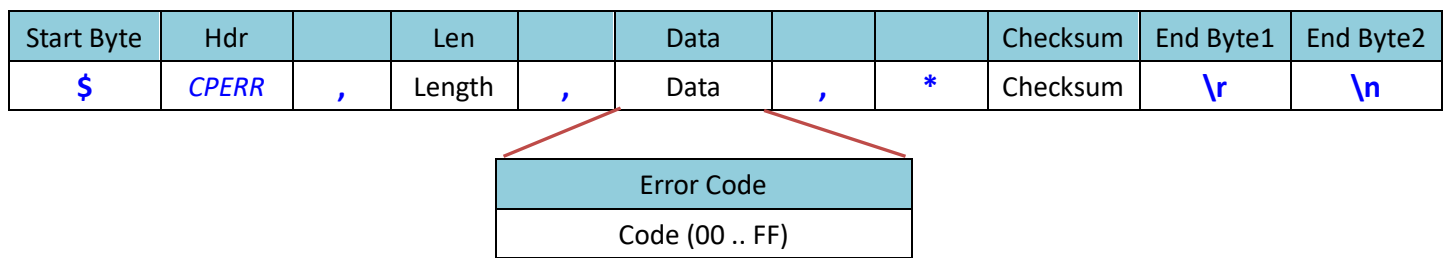
```
// 0x39,0x31,0x2C,0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40,0x00,0x00,0x80,0x40
```

```
ListenSend(92, var_b)
```

```
// the content of communication $TMSTA,7,92,...,*63\r\n
```

```
// 0x39,0x32,0x2C,0x10,0x11,0x12,0x13
```

8.6 CPERR



CPERR defines the communication package as sending the Communication Protocol Error. The data section is defined as Error Code.

Error Code	Error code, presented in 2 bytes hexadecimal format (without 0x)
00	Packet correct. No error. (The return message usually reply to the content of packet instead of returning no error)
01	Packet Error.
02	Checksum Error.
03	Header Error.
04	Packet Data Error.
F1	Have not entered Listen Node

Note

Used by robot to response to external device

Response (Robot→External Device)

01	Packet Error	
	<	\$TMSCT,-100,1,ChangeBase("RobotBase"),*13\r\n // Length cannot be negative
	>	\$CPERR,2,01,*49\r\n // CPERR Error Code 01
02	Checksum Error	
	<	\$TMSCT,25,1,ChangeBase("RobotBase"),*09\r\n // 09 is not a correct Checksum
	>	\$CPERR,2,02,*4A\r\n // CPERR Error Code 02
03	Header Error	
	<	\$TMscT,25,1,ChangeBase("RobotBase"),*28\r\n // TMscT is not a correct Header
	>	\$CPERR,2,03,*4B\r\n // CPERR Error Code 03
04	Packet Data Error	
	<	\$TMSTA,4,XXXX,*47\r\n // There is no XXXX SubCmd under TMSTA

> \$CPERR,2,04,*4C\r\n // CPERR Error Code 04

F1 No External Script Mode

< \$TMSCT,25,1,ChangeBase("RobotBase"),*0D\r\n
// Suppose currently not in external script control mode

> \$CPERR,2,F1,*3F\r\n // CPERR Error Code F1

8.7 Priority Commands

Due to the serial execution nature of the expression syntax, if using the queue syntax such as `QueueTag(1, 1)` or `WaitQueueTag(1)` to wait for the tag number to arrive, the program will stay put until the conditions are satisfied before going on execution. Therefore, if received the `ScriptExit()` syntax sent from the outside while waiting, it is impossible to exit the external Script control mode since the program is still waiting for the condition to meet.

When the program comes to the Listen node (the external Script control mode), other than the serial execution Expression syntax, there are also priority commands to use. In Listen nodes, the priority commands go with a higher execution priority than the syntax execution, and the priority commands will run at once as defined below.

1. *ScriptExit(0)*

Stop the robot motion immediately, clear the robot motion instructions in the buffer, and exit the external Script control mode. Go to the Fail path after leaving the Listen node.

2. *ScriptExit(1)*

Stop the robot motion immediately, clear the robot motion instructions in the buffer, and exit the external Script control mode. Go to the Pass path after leaving the Listen node.

3. *StopAndClearBuffer(0)*

Stop the robot motion immediately and clear the robot motion instructions in the buffer.

4. *StopAndClearBuffer(1)*

Stop the robot motion immediately, clear the robot motion instructions in the buffer, exit the current Script program in execution, and continue to the next Script program.

5. *StopAndClearBuffer(2)*

Stop the robot motion immediately, clear the robot motion instructions in the buffer, exit the current Script program in execution, and clear all the script programs in the Received Script buffer.

- Priority commands support the general use of the command definitions only but not the use with variables and functions such as

```
< $TMSCT,42,1,int var_st=2\r\n
    StopAndClearBuffer(var_st),*3A\r\n
> $TMSCT,9,1,ERROR;2,*04\r\n    // Invalid syntax StopAndClearBuffer(var_st)
```

- Using priority commands with expression syntax leads to priority command executions only but not syntax executions.

```
< $TMSCT,94,2,float[] targetP1= {0,0,90,0,90,0}\r\n    // Will not execute.
    PTP("JPP",targetP1,10,200,0,false)\r\n    // Will not execute.
    StopAndClearBuffer(0),*75\r\n    // Priority execution StopAndClearBuffer(0)
> $TMSCT,4,2,OK,*5F\r\n
```

- The system will handle one priority command only in one external Script packet. If the packet comes with numerous priority commands, the system will handle ScriptExit(0/1) before StopAndClearBuffer(0/1/2). If there are numerous ScriptExit and StopAndClearBuffer respectively, the system will handle the first only.

```
< $TMSCT,46,3,StopAndClearBuffer(2)\r\n    // Will execute.
    // There are many of StopAndClearBuffer and the system will
    // handle the 1st of them.
    StopAndClearBuffer(1),*68\r\n    // Will not execute.
> $TMSCT,4,3,OK,*5E\r\n
< $TMSCT,61,4,StopAndClearBuffer(2)\r\n    // Will not execute.
    StopAndClearBuffer(1)\r\n    // Will not execute.
    ScriptExit(1),*52\r\n    // Will execute.
    // Due to the higher priority, the system handles ScriptExit before S
    topAndClearBuffer.
> $TMSCT,4,4,OK,*59\r\n
```

1. < \$TMSCT,86,1,float[] targetP1= {0,0,90,0,90,0}\r\n
 PTP("JPP",targetP1,10,200,0,false)\r\n
 QueueTag(1,1),*60\r\n // QueueTag(1,1) waits.
 The program will stay put with the command until tag 1 finishes.
 < \$TMSCT,15,2,ScriptExit(0),*55\r\n // When receiving the command ScriptExit(0), if tag 1 does not finish,
 > \$TMSCT,4,1,OK,*5C\r\n // Respond 1 OK
 // It does not send out TMSTA tag number for finishing due to the clearing of the

motion command and QueueTag().

> \$TMSCT,4,2,OK,*5F\r\n // Respond 2 OK

It will exit the exteraIn Script control mode and go to the Fail path after leaving the Listen node.

2. < \$TMSCT,86,1,float[] targetP1= {0,0,90,0,90,0}\r\n

PTP("JPP",targetP1,10,200,0,false)\r\n

QueueTag(1,1)*60\r\n // QueueTag(1,1) waits.

The program will stay put with the command until tag 1 finishes.

< \$TMSCT,23,2,StopAndClearBuffer(0)*55\r\n

// When receiving the command StopAndClearBuffer(0), if tag 1 does not finish

> \$TMSCT,4,1,OK,*5C\r\n // Respond 1 OK

// It does not send out TMSTA tag number for finishing due to the clearing of the motion command and QueueTag().

> \$TMSCT,4,2,OK,*5F\r\n // Respond 2 OK

It does not exit the exteraIn Script control mode and is still in the Listen node.

3. < \$TMSCT,145,1,float[] targetP1= {0,0,90,0,90,0}\r\n

PTP("JPP",targetP1,10,200,0,false)\r\n

QueueTag(1,0)\r\n

WaitQueueTag(0,60000)\r\n // Tag 0 will wait for the 60-seconds timeout.

PTP("JPP",targetP1,40,200,0,false)*64\r\n // It takes a 60-seconds wait before the execution.

< \$TMSCT,23,2,StopAndClearBuffer(0)*55\r\n

// If receiving the command StopAndClearBuffer(0) in 60 seconds, it will clear the 1st PTP() function.

> \$TMSCT,4,2,OK,*5F\r\n // Respond 2 OK

It waits for the 60-seconds timeout before the 2nd PTP() function execution and responds 1 OK.

> \$TMSCT,4,1,OK,*5C\r\n // Respond 1 OK

* StopAndClearBuffer(0) clears the motion commands only but not the function syntax or the logic operations.

While WaitQueueTag(0) sets the number 0, it waits for the timeout and is impossible to exit with the motion command clearing. It is required to use the command StopAndClearBuffer(1/2) to exit from the current Script program in execution.

4. < \$TMSCT,86,1,float[] targetP1= {0,0,90,0,90,0}\r\n

PTP("JPP",targetP1,10,200,0,false)\r\n

QueueTag(1,1)*60\r\n // QueueTag(1,1) waits.

The program will stay put with the command until tag 1 finishes.

```
< $TMSCST,86,2,float[] targetP2= {90,0,0,90,0,0}\r\n
PTP("JPP",targetP2,10,200,0,false)\r\n
QueueTag(2,1),*60\r\n           // The last packet still waits, and this packet does not execute.
< $TMSCST,23,3,StopAndClearBuffer(1),*55\r\n
                                   // When receiving the command StopAndClearBuffer(1), if tag 1 does not finish
> $TMSCST,4,1,OK,*5C\r\n           // Respond 1 OK
                                   // It does not send out TMSTA tag number for finishing due to the clearing of the
                                   // motion command and QueueTag().
> $TMSCST,4,3,OK,*5E\r\n           // Respond 3 OK
It clears and exits the current Script 1 in execution, and it continues to the next, Script 2.
> $TMSCST,4,2,OK,*5F\r\n           // Respond 2 OK
> $TMSTA,10,01,02,true,*67\r\n     // Tag 2 finished
```

```
5. < $TMSCST,86,1,float[] targetP1= {0,0,90,0,90,0}\r\n
PTP("JPP",targetP1,10,200,0,false)\r\n
QueueTag(1,1),*60\r\n           // QueueTag(1,1) waits.
                                   The program will stay put with the command until tag 1 finishes.
< $TMSCST,86,2,float[] targetP2= {90,0,0,90,0,0}\r\n
PTP("JPP",targetP2,10,200,0,false)\r\n
QueueTag(2,1),*60\r\n           // The last packet still waits, and this packet does not execute.
< $TMSCST,23,3,StopAndClearBuffer(2),*56\r\n
                                   // When receiving the command StopAndClearBuffer(2), if tag 1 does not finish
> $TMSCST,4,1,OK,*5C\r\n           // Respond 1 OK
                                   // It does not send out TMSTA tag number for finishing due to the clearing of the
                                   // motion command and QueueTag().
> $TMSCST,4,3,OK,*5E\r\n           // Respond 3 OK
```

It clears and exits the current Script 1 in execution, and clears all the Script programmes in the Received Script buffer. Therefore, it clears Script 2 without the response to Script 2.

9. Robot Motion Functions

Robot motion functions and vision job functions can only be performed with external scripts, meaning the project flow must be in the listen node and in external script control mode to use robot motion functions and vision job functions

All the robot motion functions will be queued in the buffer and performed in sequence.

9.1 QueueTag()

Set robot motions with Queue Tag Numbers to denote the current robot motion in process. The status of each queue tag can be monitored using TMSTA SubCmd 01.

Syntax 1

```
bool QueueTag (  
    int,  
    int  
)
```

Parameters

int	The tag number. Valid for integers between 1 and 15.
int	Wait for the tagging to continue processing or not.
0	Not wait (default)
1	Wait

When the value is set to 1, the process stays in the function and waits for the tagging to complete and continue processing.

Return

bool Return **True** when tagged successfully. Return **False** when tagged unsuccessfully.

Syntax 2

```
bool QueueTag (  
    int  
)
```

Note

The syntax is the same as syntax 1. The default is not to wait for the tagging to continue processing.

QueueTag(int, int) => QueueTag(int, 0)

9.2 WaitQueueTag()

Wait for the Queue Tag Number of the robot motion to complete.

Syntax 1

```
int WaitQueueTag (  
    int,  
    int  
)
```

Parameters

int	The tag number waiting in queue.
1..15	Valid tag numbers.
0	Invalid tag number, but waits for the timeout. (No waiting for the time out if set the timeout to infinite.)
<0	Unavailable tag number. No waiting for the time out.
>15	Unavailable tag number. No waiting for the time out.
int	Set the time to the timeout
< 0	Wait infinitely. Valid when the tag number is between 1 to 15 legitimately. (default)
= 0	Wait to check once
> 0	Wait in milliseconds before timeout

When using waiting in queue, the process stays in the function until the tagging is completed, the tagging is not existed, or timeout, and then continues processing.

Return

int	Return the result of waiting
1	The tagging is completed
0	The tagging is incomplete or timeout
-1	The tagging is not existed

* Tag numbers can be reused.

* The tag number will retain the status of the last four tags. If the tag number waiting is not occurred or it is over the last four tags, it returns not existed.

Syntax 2

```
int WaitQueueTag (  
    int  
)
```

Note

The syntax is the same as syntax 1. The default is no timeout and required to wait for the tagging to complete (or not existed)

WaitQueueTag(int, int) => WaitQueueTag(int, 0)

● Motion Function Queue Tag

Motion function queue tags are used to cooperate with the robot motion functions. Since all motion functions are queued in the buffer and executed in order, use the cooperative queue tags, it is possible to know which motion function is in execution currently.

```
1. < $TMSCT,172,2,float[] targetP1= {0,0,90,0,90,0}\r\n
    PTP("JPP",targetP1,10,200,0,false)\r\n
    QueueTag(1)\r\n                // QueueTag(1) not wait and continue processing
    float[] targetP2 = {0,90,0,90,0,0}\r\n
    PTP("JPP",targetP2,10,200,10,false)\r\n
    QueueTag(2)\r\n                // QueueTag(2) not wait and continue processing
    ,*49\r\n
```

When executed the script content, since QueueTag() did not wait, after execution, the process returned

```
> $TMSCT,4,2,OK,*5F\r\n
```

When robot motion executed PTP() targetP1, because of QueueTag(1), it will return

```
> $TMSTA,10,01,01,true,*64\r\n    // TMSTA SubCmd 01, TagNumber 01, completed
```

When robot motion executed PTP() targetP2, because of QueueTag(2) , it will return

```
> $TMSTA,10,01,02,true,*67\r\n    // TMSTA SubCmd 01, TagNumber 02, completed
```

```
2. < $TMSCT,174,2,float[] targetP1= {0,0,90,0,90,0}\r\n
    PTP("JPP",targetP1,10,200,0,false)\r\n
    QueueTag(3,1)\r\n                // QueueTag(3) wait and stay in the function until the tagging completed
    float[] targetP2 = {0,90,0,90,0,0}\r\n
    PTP("JPP",targetP2,10,200,10,false)\r\n
    QueueTag(4)\r\n                // QueueTag(4) not wait and continue processing
    ,*56\r\n
```

When executed the script content, since QueueTag(3,1) is set to wait, after tagging completed, the process returned

```
> $TMSTA,10,01,03,true,*66\r\n    // TMSTA SubCmd 01, TagNumber 03, completed
```

When QueueTag(3) completed, the process continues, since QueueTag(4) is not set to wait, after execution, the process returned

```
> $TMSCT,4,2,OK,*5F\r\n
```

When robot motion executed PTP() targetP2, because of QueueTag(4) , it will return

```
> $TMSTA,10,01,04,true,*61\r\n      // TMSTA SubCmd 01, TagNumber 04, completed
```

* \$TMSCT,4,2,OK is returned when the process executed the script. Therefore, if using QueueTag to wait or WaitQueueTag to wait, it will return after the execution as well.

9.3 StopAndClearBuffer()

Stop the motion of the robot and clear existing commands of the robot in the buffer.

Syntax

```
bool StopAndClearBuffer (
)
```

Parameter

void No input values required

Return

bool True Command accepted ; False Command rejected

Note

StopAndClearBuffer()

9.4 Pause()

Pause the project and the motion of the robot other than non-paused threads and external script.

Use Resume() or press the Play button on the robot stick to resume.

Syntax1

```
bool Pause (
)
```

Parameter

void No input values required

Return

bool True Command accepted ; False Command rejected

Note



Pause()

9.5 Resume()

Resume the project and the motion of the robot.

Syntax1

```
bool Resume (
)
```

Parameter

`void` No input values required

Return

`bool` `True` Command accepted ; `False` Command rejected

Note

`Resume()`

9.6 PTP()

Define and send PTP motion command into buffer for execution.

Syntax 1

```
bool PTP (
    string,
    float[],
    int,
    int,
    int,
    bool
)
```

Parameters

`string` Definition of data format, combines three letters

#1: Motion target format:

`"J"` expressed in joint angles

`"C"` expressed in Cartesian coordinate

#2: Speed format:

`"P"` expressed as a percentage

#3: Blending format

`"P"` expressed as a percentage

`float[]` Motion target degree. If defined with joint angle, it includes the angles of six joints: Joint1(°), Joint 2(°), Joint 3(°), Joint 4(°), Joint 5(°), Joint 6(°) ; If defined with Cartesian coordinate, it includes the Cartesian coordinate of tool center point: X (mm), Y (mm), Z (mm), RX(°), RY(°), RZ(°)

`int` The speed setting, expressed as a percentage (%)

`int` The time interval to accelerate to top speed (ms)

`int` Blending value, expressed as a percentage (%)

`bool` Disable precise positioning

`true` Disable precise positioning

`false` Enable precise positioning

Return

`bool` `True` Command accepted; `False` Command rejected (format error)

Note

Data format parameter includes: (1) `"JPP"`, (2) `"CPP"`

```
float[] var_targetP1= {0,0,90,0,90,0}
```

```
// Declare a float array to store the target coordinate
```

```
PTP("JPP", var_targetP1,10,200,0,false)
```

```
// Move to var_targetP1 with PTP, speed = 10%, time to  
top speed = 200ms.
```

Syntax 2

```
bool PTP (  
    string,  
    float[],  
    int,  
    int,  
    int,  
    bool,  
    int[]  
)
```

Parameters

`string` Definition of data format, combines three letters

#1: Motion target format:

`"C"` expressed in Cartesian coordinate

#2: Speed format:

`"P"` expressed as a percentage

#3: Blending format

"P" expressed as a percentage

`float[]` Motion target. If defined with Cartesian coordinate, it includes the Cartesian coordinate

of tool center point: X (mm), Y (mm), Z (mm), RX(°), RY(°), RZ(°)

`int` The speed setting, expressed as a percentage (%)

`int` The time interval to accelerate to top speed (ms)

`int` Blending value, expressed as a percentage (%)

`bool` Disable precise positioning

`true` Disable precise positioning

`false` Enable precise positioning

`int[]` The pose of robot : [Config1, Config2, Config3], please find more information in appendix

Return

`bool` `True` Command accepted; `False` Command rejected (format error)

Note

Data format parameter includes: (1) "CPP"

```
float[] var_targetP1 = {417.50,-122.30,343.90,180.00,0.00,90.00}
```

```
// Declare a float array to store the target coordinate.
```

```
float[] var_pose = {0,2,4}
```

```
// Declare a float array to store pose.
```

```
PTP("CPP", var_targetP1,50,200,0,false, var_pose)
```

```
// Move to var_targetP1 with PTP, speed =  
50%, time to top speed = 200ms.
```

Syntax 3

```
bool PTP (  
    string,  
    float, float, float, float, float, float,  
    int,  
    int,  
    int,  
    bool  
)
```

Parameters

`string` Definition of data format, combines three letters

#1: Motion target format:

"J" expressed in joint angles

"C" expressed in Cartesian coordinate

#2: Speed format:

"P" expressed as a percentage

#3: Blending format:

"P" expressed as a percentage

float, float, float, float, float, float

Motion target. If expressed in joint angles, it includes the angles of six joints: Joint1(°), Joint

2(°), Joint 3(°), Joint 4(°), Joint 5(°), Joint 6(°) ; If expressed in Cartesian coordinate, it includes

the Cartesian coordinate of tool center point: X (mm), Y (mm), Z (mm), RX(°), RY(°), RZ(°)

int The speed setting, expressed as a percentage (%)

int The time interval to accelerate to top speed (ms)

int Blending value, expressed as a percentage (%)

bool Disable precise positioning

true Disable precise positioning

false Enable precise positioning

Return

bool True Command accepted; False Command rejected (format error)

Note

Data format parameter includes: (1) "JPP" and (2) "CPP"

PTP("JPP",0,0,90,0,90,0,35,200,0,false)

// Move to joint angle 0,0,90,0,90,0 with PTP,
speed = 35%, time to top speed = 200ms.

Syntax 4

```
bool PTP (  
    string,  
    float, float, float, float, float, float,  
    int,  
    int,  
    int,  
    bool,  
    int, int, int  
)
```

Parameters

string Definition of data format, combines three letters

#1: Motion target format:

"C" expressed in Cartesian coordinate

#2: Speed format:

"P" expressed as a percentage

#3: Blending format:

"P" expressed as a percentage

float, float, float, float, float, float

Motion target. It includes the Cartesian coordinate of tool center point: X (mm), Y (mm), Z (mm), RX(°), RY(°), RZ(°)

int The speed setting, expressed as a percentage (%)

int The time interval to accelerate to top speed (ms)

int Blending value, expressed as a percentage (%)

bool Disable precise positioning

 true Disable precise positioning

 false Enable precise positioning

int, int, int

The pose of robot : Config1, Config2, Config3, please find more information in appendix

Return

bool True Command accepted ; False Command rejected (format error)

Note

Data format parameter includes: (1) "CPP"

```
PTP("CPP",417.50,-122.30,343.90,180.00,0.00,90.00,10,200,0,false,0,2,4) // Move to coordinate
417.50,-122.30,343.90,180.00,0.0
0,90.00, with PTP, speed = 10%,
time to top speed = 200ms, pose
= 024.
```

9.7 Line()

Define and send Line motion command into buffer for execution.

Syntax 1

```
bool Line(
    string,
    float[],
    int,
    int,
    int,
    bool
)
```

Parameters

`string` Definition of data format, combines three letters

#1: Motion target format:

`"C"` expressed in Cartesian coordinate

#2: Speed format:

`"P"` expressed as a percentage

`"A"` expressed in velocity (mm/s)

#3: Blending format:

`"P"` expressed as a percentage

`"R"` expressed in radius

`float[]` Motion target. It includes the Cartesian coordinate of tool center point: X (mm), Y (mm), Z

(mm), RX(°), RY(°), RZ(°)

`int` The speed setting, expressed as a percentage (%) or in velocity (mm/s)

`int` The time interval to accelerate to top speed (ms)

`int` Blending value, expressed as a percentage (%) or in radius (mm)

`bool` Disable precise positioning

`true` Disable precise positioning

`false` Enable precise positioning

Return

`bool` `True` Command accepted; `False` Command rejected (format error)

Note

Data format parameter includes: (1) `"CPP"`, (2) `"CPR"`, (3) `"CAP"`, and (4) `"CAR"`

```
float[] var_Point1 = {417.50,-122.30,343.90,180.00,0.00,90.00}
```

```
// Declare a float array to store the target coordinate
```

```
Line("CAR", var_Point1,100,200,50,false)
```

```
// Move to var_Point1 with Line, speed = 100mm/s,  
time to top speed = 200ms, blending radius =  
50mm
```

Syntax 2

```
bool Line (  
    string,  
    float, float, float, float, float, float,  
    int,  
    int,  
    int,  
    bool  
)
```

Parameters

`string` Definition of data format, combines three letters

#1: Motion target format:

"C" expressed in Cartesian coordinate

#2: Speed format:

"P" expressed as a percentage

"A" expressed in velocity (mm/s)

#3: Blending format:

"P" expressed as a percentage

"R" expressed in radius

float, float, float, float, float, float

Motion target. It includes the Cartesian coordinate of tool center point: X (mm), Y (mm), Z

(mm), RX(°), RY(°), RZ(°)

int The speed setting, expressed as a percentage (%) or in velocity (mm/s)

int The time interval to accelerate to top speed (ms)

int Blending value, expressed as a percentage (%) or in radius (mm)

bool Disable precise positioning

true Disable precise positioning

false Enable precise positioning

Return

bool True Command accepted; False Command rejected (format error)

Note

Data format parameter includes: (1) "CPP", (2) "CPR", (3) "CAP", and (4) "CAR"

```
Line("CAR", 417.50,-122.30,343.90,180.00,0.00,90.00,100,200,50,false) // Move to
                                                                    417.50,-122.30,343.90,180.00,0.0
                                                                    0,90.00 with Line, velocity =
                                                                    100mm/s, time to top speed =
                                                                    200ms, blending radius = 50mm
```

9.8 Circle()

Define and send Circle motion command into buffer for execution.

Syntax 1

```
bool Circle(
    string,
    float[],
    float[],
    int,
    int,
    int,
```

```

    int,
    bool
)

```

Parameters

string Definition of data format, combines three letters

- #1: Motion target format:
 - "C" expressed in Cartesian coordinate
- #2: Speed format:
 - "P" expressed as a percentage
 - "A" expressed in velocity (mm/s)
- #3: Blending format:
 - "P" expressed as a percentage

float[] A point on arc. It includes the Cartesian coordinate of tool center point: X (mm), Y (mm), Z (mm), RX(°), RY(°), RZ(°)

float[] The end point of arc, it includes the Cartesian coordinate of tool center point: X (mm), Y (mm), Z (mm), RX(°), RY(°), RZ(°)

int The speed setting, expressed as a percentage (%) or in velocity (mm/s)

int The time interval to accelerate to top speed (ms)

int Blending value, expressed as a percentage (%)

int Arc angle(°), If non-zero value is given, the TCP will keep the same pose and move from current point to the assigned arc angle via the given point and end point on arc; If zero is given, the TCP will move from current point and pose to end point and pose via the point on arc with linear interpolation on pose.

bool Disable precise positioning

- true** Disable precise positioning
- false** Enable precise positioning

Return

bool **True** Command accepted; **False** Command rejected (format error)

Note

Data format parameter includes: (1) "CPP" and (2) "CAP"

```

float[] var_PassP = {417.50,-122.30,343.90,180.00,0.00,90.00} // Declare a float array to store the via point
                                                             value
float[] var_EndP = {381.70,208.74,343.90,180.00,0.00,135.00} // Declare a float array to store the end point
                                                             value

```

```
Circle("CAP", var_PassP, var_EndP,100,200,50,270,false)
```

```
// Move on 270° arc, velocity = 100mm/s,  
time to top speed = 200ms, blending value =  
50%
```

Syntax 2

```
bool Circle(  
    string,  
    float, float, float, float, float, float,  
    float, float, float, float, float, float,  
    int,  
    int,  
    int,  
    int,  
    bool  
)
```

Parameters

string Definition of data format, combines three letters

#1: Motion target format:

"C" expressed in Cartesian coordinate

#2: Speed format:

"P" expressed as a percentage

"A" expressed in velocity (mm/s)

#3: Blending format:

"P" expressed as a percentage

float, float, float, float, float, float

A point on arc. It includes the Cartesian coordinate of tool center point: X (mm), Y (mm), Z

(mm), RX(°), RY(°), RZ(°)

float, float, float, float, float, float

The end point of arc. It includes the Cartesian coordinate of tool center point: X (mm), Y (mm),

Z (mm), RX(°), RY(°), RZ(°)

int The speed setting, expressed as a percentage (%) or in velocity (mm/s)

int The time interval to accelerate to top speed (ms)

int Blending value, expressed as a percentage (%)

int Arc angle(°), If non-zero value is given, the TCP will keep the same pose and move from current point to the assigned arc angle via the given point and end point on arc; If zero is given, the TCP will move from current point and pose to end point and pose via the point on arc with linear interpolation on pose.

```
bool    Disable precise positioning
true    Disable precise positioning
false   Enable precise positioning
```

Return

```
bool    True Command accepted; False Command rejected (format error)
```

Note

Data format parameter includes: (1) "CPP" and (2) "CAP"

```
Circle("CAP", 417.50,-122.30,343.90,180.00,0.00,90.00,
381.70,208.74,343.90,180.00,0.00,135.00,100,200,50,270,false)
// Move on 270° arc, velocity = 100mm/s, time to top speed = 200ms, blending value = 50%, via point =
417.50,-122.30,343.90,180.00,0.00,90.00, end point = 381.70,208.74,343.90,180.00,0.00,135
```

9.9 PLine()

Define and send PLine motion command into buffer for execution.

Syntax 1

```
bool PLine (
    string,
    float[],
    int,
    int,
    int
)
```

Parameters

string Definition of data format, combines three letters

#1: Motion target format:

"J": expressed in joint angles

"C": expressed in Cartesian coordinate

#2: Speed format:

"A": expressed in velocity (mm/s)

#3: Blending format:

"P": expressed as a percentage

float[] Motion target. If expressed in joint angles, it includes the angles of six joints: Joint1(°), Joint

2(°), Joint 3(°), Joint 4(°), Joint 5(°), Joint 6(°) ; If expressed in Cartesian coordinate, it includes

the Cartesian coordinate of tool center point: X (mm), Y (mm), Z (mm), RX(°), RY(°), RZ(°)

`int` The speed setting, expressed in velocity (mm/s)

`int` The time interval to accelerate to top speed (ms)

`int` Blending value, expressed as a percentage (%)

Return

`bool` True Command accepted ; False Command rejected (format error)

Note

Data format parameter includes: (1) "JAP" and (2) "CAP"

```
float[] targetP1 = {417.50,-122.30,343.90,180.00,0.00,90.00} // Declare a float array to store the
                                                             target coordinate
PLine("CAP",targetP1,100,200,50) // Move to targetP1 with PLine,
                                  velocity = 100mm/s, time to top speed
                                  = 200ms, blending value = 50%
```

Syntax 2

```
bool PLine (
    string,
    float, float, float, float, float, float,
    int,
    int,
    int
)
```

Parameters

`string` Definition of data format, combines three letters

#1: Motion target format:

"J": expressed in joint angles

"C": expressed in Cartesian coordinate

#2: Speed format:

"A": expressed in velocity (mm/s)

#3: Blending format:

"P": expressed as a percentage

`float, float, float, float, float, float,`

Motion target. If expressed in joint angles, it includes the angles of six joints: Joint1(°), Joint

2(°), Joint 3(°), Joint 4(°), Joint 5(°), Joint 6(°) ; If expressed in Cartesian coordinate, it includes

the Cartesian coordinate of tool center point: X (mm), Y (mm), Z (mm), RX(°), RY(°), RZ(°)

`int` The speed setting, expressed in velocity (mm/s)

`int` The time interval to accelerate to top speed (ms)
`int` Blending value, expressed as a percentage (%)

Return

`bool` `True` Command accepted; `False` Command rejected (format error)

Note

Data format parameter includes: (1) `"JAP"` and (2) `"CAP"`

```
PLine("CAP", 417.50,-122.30,343.90,180.00,0.00,90.00,100,200,50)
```

```
// Move to 417.50,-122.30,343.90,180.00,0.00,90.00 with PLine, velocity = 100mm/s, time to top speed =  
200ms, Blending value = 50%
```

9.10 Move_PTP()

Define and send PTP relative motion commands for execution.

Syntax 1

```
bool Move_PTP(  
    string,  
    float[],  
    int,  
    int,  
    int,  
    bool  
)
```

Parameters

`string` Definition of data format made of three letters

 #1: Relative motion target format:

`"C"`: expressed w.r.t. current base

`"T"`: expressed w.r.t. tool coordinate

`"J"`: expressed in joint angles

 #2: Speed format:

`"P"`: expressed as a percentage

 #3: Blending format:

`"P"`: expressed as a percentage

`float[]` relative motion parameters. If expressed in coordinate (w.r.t. current base or tool coordinate), it includes the tool end TCP relative motion value with respect to the specified coordinate: X (mm), Y (mm), Z (mm), RX(°), RY(°), RZ(°); If defined with joint angle, it includes the angles of six joints: Joint1(°), Joint 2(°), Joint 3(°), Joint 4(°), Joint 5(°), Joint 6(°)

`int` The speed setting, expressed as a percentage (%)

<code>int</code>	The time interval to accelerate to top speed (ms)
<code>int</code>	Blending value, expressed as a percentage (%)
<code>bool</code>	Disable precise positioning
<code>true</code>	Disable precise positioning
<code>false</code>	Enable

Return

`bool` `True` Command accepted; `False` Command rejected (format error)

Note

Motion command parameter includes: (1) `"CPP"`, (2) `"TPP"` or (3) `"JPP"`

```
float[] var_relmove = {0,0,10,45,0,0}           // Declare a float array to store the relative motion
                                                target
Move_PTP("TPP", var_relmove,10,200,0,false)     // Move to relative motion target with PTP, velocity
                                                = 10%, time to top speed = 200ms
```

Syntax 2

```
bool Move_PTP(
    string,
    float, float, float, float, float, float,
    int,
    int,
    int,
    bool
)
```

Parameters

`string` Definition of data format, combines three letters

#1: Relative motion target format:

`"C"`: expressed w.r.t. current base

`"T"`: expressed w.r.t. tool coordinate

`"J"`: expressed in joint angles

#2: Speed format:

`"P"`: expressed as a percentage

#3: Blending format:

`"P"`: expressed as a percentage

`float, float, float, float, float, float`

relative motion parameters. If expressed in coordinate (w.r.t. current base or tool coordinate), it includes the tool end TCP relative motion value with respect to the specified coordinate: X (mm), Y (mm), Z (mm), RX(°), RY(°), RZ(°); If defined with joint angle, it includes the angles of six joints: Joint1(°), Joint 2(°), Joint 3(°), Joint 4(°), Joint 5(°), Joint 6(°)

`int` The speed setting, expressed as a percentage (%)

<code>int</code>	The time interval to accelerate to top speed (ms)
<code>int</code>	Blending value, expressed as a percentage (%)
<code>bool</code>	Disable precise positioning
<code>true</code>	Disable precise positioning
<code>false</code>	Enable

Return

`bool` `True` Command accepted; `False` Command rejected (format error)

Note

Motion command parameter includes: (1) `"CPP"`, (2) `"TPP"` and (3) `"JPP"`

```
Move_PTP("TPP",0,0,10,45,0,0,10,200,0,false)           // Move 0,0,10,45,0,0, with respect to tool
                                                         coordinate, with PTP, velocity = 10%, time to top
                                                         speed = 200ms
```

9.11 Move_Line()

Define and send Line relative motion commands for execution.

Syntax 1

```
bool Move_Line (
    string,
    float[],
    int,
    int,
    int,
    bool
)
```

Parameters

`string` Definition of data format made of three letters

#1: Relative motion target format:

`"C"`: expressed w.r.t. current base

`"T"`: expressed w.r.t. tool coordinate

#2: Speed format:

`"P"`: expressed as a percentage

`"A"`: expressed in velocity (mm/s)

#3: Blending format:

`"P"`: expressed as a percentage

`"R"`: expressed in radius

`float[]` Relative motion parameter. It includes the tool end TCP relative motion value with respect to

the specified coordinate (current base or tool coordinate): X (mm), Y (mm), Z (mm), RX(°), RY(°), RZ(°).

`int` The speed setting, expressed as a percentage (%) or in velocity (mm/s)
`int` The time interval to accelerate to top speed (ms)
`int` Blending value, expressed as a percentage (%) or in radius (mm)
`bool` Disable precise positioning
 `true` Disable precise positioning
 `false` Enable

Return

`bool` `True` Command accepted; `False` Command rejected (format error)

Note

Motion command parameter includes: (1) `"CPP"`, (2) `"CPR"`, (3) `"CAP"`, (4) `"CAR"`, (5) `"TPP"`, (6) `"TPR"`, (7) `"TAP"` and (8) `"TAR"`

```
float[] var_relmove = {0,0,10,45,0,0}           //Declare a float array to store the relative motion
                                                target
Move_Line("TAP", var_relmove,125,200,0,false)    // Move to relative motion target, with Line,
                                                velocity = 125mm/s, time to top speed = 200ms
```

Syntax 2

```
bool Move_Line (  
    string,  
    float, float, float, float, float, float,  
    int,  
    int,  
    int,  
    bool  
)
```

Parameters

`string` Definition of data format made of three letters

#1: Relative motion target format:

`"C"`: expressed w.r.t. current base

`"T"`: expressed w.r.t. tool coordinate

#2: Speed format:

`"P"`: expressed as a percentage

`"A"`: expressed in velocity (mm/s)

#3: Blending format:

`"P"`: expressed as a percentage

`"R"`: expressed in radius

`float, float, float, float, float, float`

Relative motion parameter. It includes the tool end TCP relative motion value with respect to the specified coordinate (current base or tool coordinate): X (mm), Y (mm), Z (mm), RX(°), RY(°), RZ(°).

`int` The speed setting, expressed as a percentage (%) or in velocity (mm/s)
`int` The time interval to accelerate to top speed (ms)
`int` Blending value, expressed as a percentage (%) or in radius (mm)
`bool` Disable precise positioning
 `true` Disable precise positioning
 `false` Enable

Return

`bool` True Command accepted ; False Command rejected (format error)

Note

Motion command parameter includes: (1) "`CPP`", (2) "`CPR`", (3) "`CAP`", (4) "`CAR`", (5) "`TPP`", (6) "`TPR`", (7) "`TAP`" and (8) "`TAR`"

```
Move_Line("TAP", 0,0,10,45,0,0,125,200,0,false)      // Move to relative motion target 0,0,10,45,0,0
                                                         with Line, velocity = 125mm/s, time to top speed =
                                                         200ms.
```

9.12 Move_PLine()

Define and send PLine relative motion commands for execution.

Syntax 1

```
bool Move_PLine (  
    string,  
    float[],  
    int,  
    int,  
    int  
)
```

Parameters

`string` Definition of data format made of three letters
 #1: Relative motion target format:
 "`C`": expressed w.r.t. current base
 "`T`": expressed w.r.t. tool coordinate

"J": expressed in joint angles

#2: Speed format:

"A": expressed in velocity (mm/s)

#3: Blending format:

"P": expressed as a percentage

`float[]` If expressed in coordinate (w.r.t. current base or tool coordinate), it includes the tool end TCP relative motion value with respect to the specified coordinate: X (mm), Y (mm), Z (mm), RX(°), RY(°), RZ(°); If defined with joint angle, it includes the angles of six joints: Joint1(°), Joint 2(°), Joint 3(°), Joint 4(°), Joint 5(°), Joint 6(°)

`int` The speed setting, expressed in velocity (mm/s)

`int` The time interval to accelerate to top speed (ms)

`int` Blending value, expressed as a percentage (%)

Return

`bool` `True` Command accepted; `False` Command rejected (format error)

Note

Motion command parameter includes: (1) "CAP", (2) "TAP" and (3) "JAP"

```
float[] var_target = {0,0,10,45,0,0}
```

// Declare a float array to store the relative motion target

```
Move_PLine("CAP", var_target,125,200,0)
```

//Move to relative motion target, with PLine, velocity = 125mm/s, time to top speed = 200ms.

Syntax 2

```
bool Move_PLine(  
    string,  
    float, float, float, float, float, float,  
    int,  
    int,  
    int,  
)
```

Parameters

`string` Definition of data format made of three letters

#1: Relative motion target format:

"C": expressed w.r.t. current base

"T": expressed w.r.t. tool coordinate

"J": expressed in joint angles

#2: Speed format:

"A": expressed in velocity (mm/s)

#3: Blending format:

"P": expressed as a percentage

float, float, float, float, float, float

Relative motion parameters. If expressed in coordinate (w.r.t. current base or tool coordinate), it includes the tool end TCP relative motion value with respect to the specified coordinate: X (mm), Y (mm), Z (mm), RX(°), RY(°), RZ(°); If defined with joint angle, it includes the angles of six joints: Joint1(°), Joint 2(°), Joint 3(°), Joint 4(°), Joint 5(°), Joint 6(°)

int The speed setting, expressed in velocity (mm/s)

int The time interval to accelerate to top speed (ms)

int Blending value, expressed as a percentage (%)

Return

bool True Command accepted ; False Command rejected (format error)

Note

Motion command parameter includes: (1) "CAP", (2) "TAP" and (3) "JAP"

```
Move_PLine("CAP",0,0,10,45,0,0,125,200,0) // Move 0,0,10,45,0,0, with PLine, velocity = 125mm/s,  
time to top speed = 200ms
```

9.13 ChangeBase()

Send the command of changing the base of the follow-up motions into buffer for execution.

Syntax 1

```
bool ChangeBase (  
    string  
)
```

Parameters

string Base Name

Return

bool True Command accepted; False Command rejected (format error)

Note

```
ChangeBase("RobotBase") // Change the base to "RobotBase", a base listed on the base  
list in TMflow.
```

Syntax 2

```
bool ChangeBase (  
    float[]
```

)

Parameters

`float[]` Base parameters, combines X, Y, Z, RX, RY, RZ

Return

`True` Command accepted; `False` Command rejected (format error)

Note

```
float[] var_Base1 = {20,30,10,0,0,90}           // Declare a float array to store the base value
ChangeBase(var_Base1)                           // Change the base value to var_Base1
```

Syntax 3

```
bool ChangeBase (
    float, float, float, float, float, float
)
```

Parameters

`float, float, float, float, float, float`
Base parameters, combines X, Y, Z, RX, RY, RZ

Return

`bool` `True` Command accepted; `False` Command rejected (format error)

Note

```
ChangeBase(20,30,10,0,0,90)                     // Change the base value to {20,30,10,0,0,90}
```

9.14 ChangeTCP()

Send the command of changing the TCP of the follow-up motions into buffer for execution.

Syntax 1

```
bool ChangeTCP (
    string
)
```

Parameters

`string` TCP name

Return

`bool` `True` Command accepted; `False` Command rejected (format error)

Note

```
ChangeTCP("NOTOOL")                             // Change the TCP to "NOTOOL", a TCP listed on the base list in TMflow.
```

Syntax 2

```
bool ChangeTCP (  
    float[]  
)
```

Parameters

`float[]` TCP Parameter, combines X, Y, Z, RX, RY, RZ

Return

`bool` `True` Command accepted; `False` Command rejected (format error)

Note

```
float[] var_Tool1 = {0,0,150,0,0,90}           // Declare a float array to store the TCP value  
  
ChangeTCP(var_Tool1)                          // Change the TCP value to var_Tool1
```

Syntax 3

```
bool ChangeTCP (  
    float[],  
    float  
)
```

Parameters

`float[]` TCP Parameter, combines X, Y, Z, RX, RY, RZ

`float` Tool's weight

Return

`bool` `True` Command accepted; `False` Command rejected (format error)

Note

```
float[] var_Tool1 = {0,0,150,0,0,90}           // Declare a float array to store the TCP value  
  
ChangeTCP(var_Tool1,2)                        // Change the TCP value to var_Tool1 with weight = 2kg
```

Syntax 4

```
bool ChangeTCP (  
    float[],  
    float,  
    float[]  
)
```

Parameters

`float[]` TCP Parameter, combines X, Y, Z, RX, RY, RZ

`float` Tool's weight

`float[]` Tool's moment of inertia: (1)Ixx, (2)Iyy, (3)Izz and its frame of reference: (4)X, (5)Y, (6)Z, (7)RX,

(8)RY, (9)RZ

Return

`bool` `True` Command accepted; `False` Command rejected (format error)

Note

```
float[] var_Tool1 = {0,0,150,0,0,90}           // Declare a float array to store the TCP value
float[] var_COM1 = {2,0.5,0.5,0,0,-80,0,0,0}    // Declare a float array to store the moment of inertia
                                                and its frame of reference

ChangeTCP(var_Tool1,2, var_COM1)                // Change the TCP value to var_Tool1 with weight = 2kg
                                                and moment of inertia to var_COM1
```

Syntax 5

```
bool ChangeTCP (
    float, float, float, float, float, float
)
```

Parameters

```
float, float, float, float, float, float
TCP Parameter, combines X, Y, Z, RX, RY, RZ
```

Return

`bool` `True` Command accepted; `False` Command rejected (format error)

Note

```
ChangeTCP(0,0,150,0,0,90)                    // Change the TCP value to {0,0,150,0,0,90}
```

Syntax 6

```
bool ChangeTCP (
    float, float, float, float, float, float,
    float
)
```

Parameters

```
float, float, float, float, float, float
TCP Parameter, combines X, Y, Z, RX, RY, RZ

float TCP weight
```

Return

`bool` `True` Command accepted; `False` Command rejected (format error)

Note

```
ChangeTCP(0,0,150,0,0,90,2)                  // Change the TCP value to {0,0,150,0,0,90}, weight =
                                                2kg
```

Syntax 7

```
bool ChangeTCP (
    float, float, float, float, float, float,
    float,
    float, float, float, float, float, float, float, float
)
```

Parameters

float, float, float, float, float, float

TCP Parameter, combines X, Y, Z, RX, RY, RZ

float Tool's weight

float, float, float, float, float, float, float, float, float

Tool's moment of inertia: (1)Ixx, (2)Iyy, (3)Izz and its frame of reference: (4)X, (5)Y, (6)Z, (7)RX, (8)RY, (9)RZ

Return

bool True Command accepted; False Command rejected (format error)

Note

```
ChangeTCP(0,0,150,0,0,90,2, 2,0.5,0.5,0,0,-80,0,0,0) // Change the TCP value to {0,0,150,0,0,90}, weight =
                                                       2kg, moment of inertia = {2,0.5,0.5} and frame of
                                                       reference = {0,0,-80,0,0,0}
```

9.15 ChangeLoad()

Send the command of changing the payload value of the follow-up motions into buffer for execution.

Syntax 1

```
bool ChangeLoad (
    float
)
```

Parameters

float Payload(kg)

Return

bool True Command accepted; False Command rejected (format error)

Note

```
ChangeLoad(5.3) //Set payload to 5.3kg
```

9.16 PVTEnter()

Set PVT mode to start with Joint/Cartesian command

Syntax1

```
bool PVTEnter(  
    int  
)
```

Parameter

int	PVT Mode
0	Joint
1	Cartesian

Return

bool True Command accepted; False Command rejected

Note

Syntax2

```
bool PVTEnter(  
)
```

Parameter

void	No input values required. Use PVT mode with Joint Command by default.
------	---

Return

bool True Command accepted; False Command rejected

Note

PVTEnter(1)

9.17 PVTExit()

Set PVT mode motion to exit

Syntax1

```
bool PVTExit(  
)
```

Parameter

`void` No input values required

Return

`bool` `True` Command accepted; `False` Command rejected

Note

`PVTExit()`

9.18 PVTPoint()

Set the PVT mode parameters of motion in position, velocity, and duration.

Syntax1

```
bool PVTPoint(  
    float[],  
    float[],  
    float  
)
```

Parameter

`float[]` Target position
 {J1, J2, J3, J4, J5, J6} in PVT mode with Joint
 {X, Y, Z, RX, RY, RZ} in PVT mode with Cartesian

`float[]` Target velocity
 {J1, J2, J3, J4, J5, J6}' in PVT mode with Joint
 {X, Y, Z, RX, RY, RZ}' in PVT mode with Cartesian

`float` Duration (second)

Return

`bool` `True` Command accepted; `False` Command rejected

Note

Syntax2

```
bool PVTPoint(  
    float, float, float, float, float, float,  
    float, float, float, float, float, float,  
    float  
)
```

Parameter

```
float, float, float, float, float, float,  
    Target Position.  
    {J1, J2, J3, J4, J5, J6} in PVT mode with Joint  
    {X, Y, Z, RX, RY, RZ} in PVT mode with Cartesian  
float, float, float, float, float, float,  
    Target Velocity.  
    {J1, J2, J3, J4, J5, J6}' in PVT mode with Joint  
    {X, Y, Z, RX, RY, RZ}' in PVT mode with Cartesian  
float    Duration (second)
```

Return

```
bool    True Command accepted; False Command rejected
```

Note

```
PVTEnter(1)  
PVTPoint(467.5,-122.2,359.7,180,0,90,50,50,0,0,0,0.5)  
PVTPoint(467.5,-72.2,359.7,180,0,90,-50,50,0,0,0,0.5)  
PVTPoint(417.5,-72.2,359.7,180,0,90,0,0,0,0,0,0.5)  
PVTPoint(417.5,-122.2,359.7,180,0,90,50,50,0,0,0,0.5)  
PVTPoint(417.5,-122.2,359.7,180,0,60,50,50,0,0,0,0.3)  
PVTPoint(417.5,-122.2,359.7,180,0,90,50,50,0,0,0,0.3)  
PVTExit()
```

9.19 PVTPause()

Set PVT mode motion to pause

Syntax1

```
bool PVTPause(  
)
```

Parameter

```
void    No input values required
```

Return

`bool` `True` Command accepted; `False` Command rejected

Note

`PVTPause()`

9.20 PVTResume()

Set PVT mode motion to resume

Syntax1

```
bool PVTResume (  
)
```

Parameter

`void` No input values required

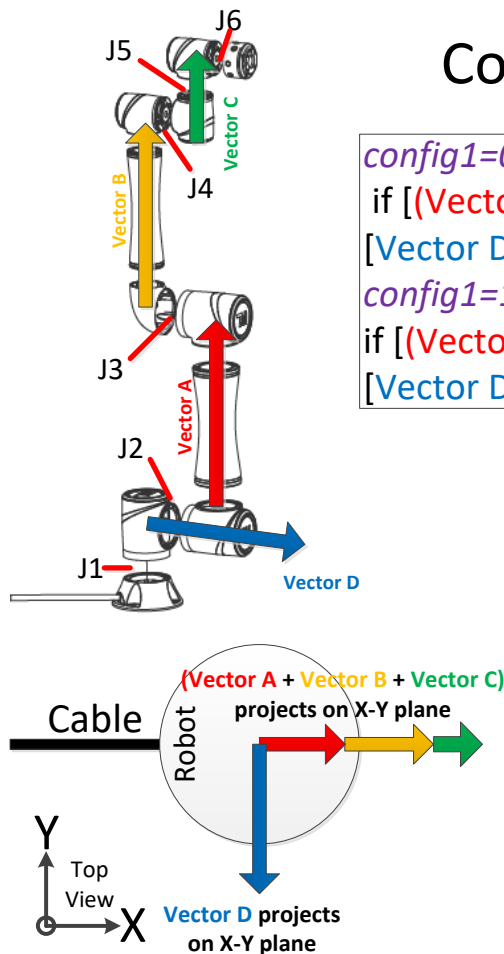
Return

`bool` `True` Command accepted; `False` Command rejected

Note

`PVTResume()`

Pose Configuration Parameters: [Config1, Config2, Config3]



Config: config1, config2, config3

config1=0:

if [(Vector A + Vector B + Vector C) projects on X-Y plane] cross [Vector D projects on X-Y plane] is on negative-Z

config1=1:

if [(Vector A + Vector B + Vector C) projects on X-Y plane] cross [Vector D projects on X-Y plane] is on positive-Z

config2=2:

if (M=0 and J3 is positive) or (M=1 and J3 is negative)

config2=3:

if (M=0 and J3 is negative) or (M=1 and J3 is positive)

config3=4:

if (M=0 and J5 is positive) or (M=1 and J5 is negative)

config3=5:

if (M=0 and J5 is negative) or (M=1 and J5 is positive)

9.21 Vision_DoJob()

Set the present vision job in the current project in execution to queue up with motion commands.

Syntax 1

```
bool Vision_DoJob (
    string
)
```

Parameter

string Vision job name. Cannot use vision jobs with the initial point.

Return

bool **True** Vision job completed and the result is pass.
 False Vision job failed. 1. The result is fail.

2. Fail to execute.
3. Vision job comes with the initial point.

Note

The called vision job needs to be created first in the TMflow. There is a check box **Start at Initial Position** in the Initial of the vision job, it is required to uncheck th box to use this function.

The respective vision base or variables will be updated along with the visual job execution.

```
bool var_result = Vision_DoJob("job1")
```

9.22 Vision_DoJob_PTP()

Set the present vision job in the current project in execution, by moving to the initial position in PTP, to queue up with motion commands.

Syntax 1

```
bool Vision_DoJob_PTP(
    string,
    int,
    int,
    bool
)
```

Parameters

string	Vision job name.
int	The speed setting, expressed as a percentage (%)
int	The time interval to accelerate to top speed (ms)
bool	Whether to use the smart pose of robot selection mode
true	Use the pose of robot by the system.
false	Use the pose of robot recorded teaching the vision job.

Return

bool	True	Vision job completed and the result is pass.
	False	Vsion job failed. 1. The result is fail.
		2. Fail to execute.

Note

The called vision job needs to be created first in the TMflow. There is a check box **Start at Initial Position** in the Initial of the vision job, it is required to uncheck th box to use this function.

The respective vision base or variables will be updated along with the visual job execution.

```
bool var_result = Vision_DoJob_PTP("job1", 100, 250, true)
```

9.23 Vision_DoJob_Line()

Set the present vision job in the current project in execution, by moving to the initial position in Line, to queue up with motion commands.

Syntax 1

```
bool Vision_DoJob_Line(  
    string,  
    int  
)
```

Parameters

string	Vision job name
int	The speed setting, expressed as a percentage (%)

Return

bool	True	Vision job completed and the result is pass.
	False	Vision job failed.
		1. The result is fail.
		2. Fail to execute.

Note

The called vision job needs to be created first in the TMflow. There is a check box **Start at Initial Position** in the Initial of the vision job, it is required to uncheck the box to use this function.

The respective vision base or variables will be updated along with the visual job execution.

```
bool var_result = Vision_DoJob_Line("job1", 100)
```

Syntax 2

```
bool Vision_DoJob_Line(  
    string,  
    int,  
    int  
)
```

Parameters

string	Vision job name
int	The speed setting, expressed in velocity (mm/s)
int	The time interval to accelerate to top speed (ms)

Return

<code>bool</code>	<code>True</code>	Vision job completed and the result is pass.
	<code>False</code>	Vision job failed. 1. The result is fail. 2. Fail to execute.

Note

The called vision job needs to be created first in the TMflow. There is a check box **Start at Initial Position** in the Initial of the vision job, it is required to uncheck the box to use this function.

The respective vision base or variables will be updated along with the visual job execution.

```
bool var_result = Vision_DoJob_Line("job1", 150, 250)
```

9.24 Vision_IsJobAvailable()

Check if the vision job in the current project present and valid

Syntax 1

```
bool Vision_IsJobAvailable (  
    string  
)
```

Parameter

<code>string</code>	Vision job name
---------------------	-----------------

Return

<code>bool</code>	<code>True</code>	the vision present and valid
	<code>False</code>	the vision absent and invalid

Note

Use this function to check the presence and validity of the visual job before calling to avoid errors caused by absence or invalidity while calling.

```
bool var_result = Vision_IsJobAvailable("job1")
```

10. Modbus Functions

10.1 modbus_read()

Modbus TCP/RTU read function

Syntax 1 (TCP/RTU)

```
? modbus_read(  
    string,  
    string  
)
```

Parameters

string TCP/RTU device name (Set in Modbus Device setting)

string The predefined parameters belong to TCP/RTU device (Set in Modbus Device setting)

Return

? The return data type is decided by the predefined parameters

Signal Type	Function Code	Type	Num Of Addr	Return data type
Digital Output	01	byte	1	byte (H: 1)(L: 0)
		bool	1	bool (H: true)(L: false)
Digital Input	02	byte	1	byte (H: 1)(L: 0)
		bool	1	bool (H: true)(L: false)
Register Output	03	byte	1	byte
		int16	1	int
		int32	2	int
		float	2	float
		double	4	double
		string	?	string
		bool	1	bool
Register Input	04	byte	1	byte
		int16	1	int
		int32	2	int
		float	2	float
		double	4	double
		string	?	string
		bool	1	bool

* According to the Little Endian (CD AB) or Big Endian (AB CD) setting, the int32, float, double data will transformed automatically.

* string will follow the UTF8 data format transformation (Stop at 0x00)

Note

Modbus Address data size

Digital	1 address = 1 bit size
Register	1 address = 2 bytes size

If the default values are applied in Preset Setting

preset_800	DO	800	byte	
preset_7202	DI	7202	bool	
preset_9000	RO	9000	string	4
preset_7001	RI	7001	float	Big-Endian (AB CD)

```
var_value = modbus_read("TCP_1", "preset_800") // var_value = 1 // DO 1 address = 1 bit
var_value = modbus_read("TCP_1", "preset_7202") // var_value = true // DI 1 address = 1 bit
var_value = modbus_read("TCP_1", "preset_9000") // var_value = ab1234cd // RO 4 address = 8 bytes
size
var_value = modbus_read("TCP_1", "preset_7001") // var_value = -314.1593 // RI 2 address = 4 bytes
size (float)
```

Syntax 2 (TCP/RTU)

```
byte[] modbus_read(
    string,
    byte,
    string,
    int,
    int
)
```

Parameters

string	TCP/RTU Device Name (Set in Modbus Device setting)		
byte	Slave ID		
string	Read type		
DO	Digital Output	(Function Code : 01)	
DI	Digital Input	(Function Code : 02)	
RO	Register Output	(Function Code : 03)	
RI	Register Input	(Function Code : 04)	

`int` Starting address
`int` Data length

Return

`byte[]` The returned byte array from modbus server
*User defined `modbus_read` only follows [Big-Endian \(AB CD\)](#) format to read `byte[]`

Note

Modbus Address data size

Digital	1 address = 1 bit size
Register	1 address = 2 bytes size

If the user defined values are applied to User Setting as

TCP device	0	DO	800	4
TCP device	0	DI	7202	3
TCP device	0	RO	9000	6
TCP device	0	RI	7001	12
TCP device	0	RI	7301	6

```
var_value = modbus_read("TCP_1", 0, "DO", 800, 4)
    // var_value = {0,0,0,0} // DO 4 address = 4 bit to byte array
var_value = modbus_read("TCP_1", 0, "DI", 7202, 3)
    // var_value = {1,0,0} // DI 3 address = 3 bit to byte array
var_value = modbus_read("TCP_1", 0, "RO", 9000, 6)
    // var_value = {0x54,0x65,0x63,0x68,0x6D,0x61,0x6E,0xE9,0x81,0x94,0xE6,0x98}
    // RO 6 address = 12 bytes size
var_value = modbus_read("TCP_1", 0, "RI", 7001, 12)
    // var_value = {0x29,0x30,0x9F,0x4C,0xC3,0x7C,0x99,0x9A,0x44,0x5E,0xEC,0xCD,0x42,0xB4,0x00,0x00,
    // 0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00} // RI 12 address = 24 bytes size
var_value = modbus_read("TCP_1", 0, "RI", 7301, 6)
    // var_value = {0x07,0xE2,0x00,0x05,0x00,0x12,0x00,0x0F,0x00,0x0A,0x00,0x39}
    // RI 6 address = 12 bytes size
```

10.2 modbus_read_int16()

Modbus TCP/RTU read function, and transform modbus address data array to int16 array

Syntax 1 (TCP/RTU)

```
int[] modbus_read_int16(  
    string,  
    byte,  
    string,  
    int,  
    int,  
    int  
)
```

Parameters

string	TCP/RTU Device Name (Set in Modbus Device setting)		
byte	Slave ID		
string	Read type		
	DO	Digital Output	(Function Code : 01)
	DI	Digital Input	(Function Code : 02)
	RO	Register Output	(Function Code : 03)
	RI	Register Input	(Function Code : 04)
int	Starting address		
int	Data length		
int	Follows Little Endian (CD AB) or Big Endian (AB CD) to transform the address data to int16 array.		
	*Invalid Parameter. Only support int32, float, double		
	0	Little Endian	
	1	Big Endian (Default)	

Return

int[] The returned int array from modbus server

Syntax 2 (TCP/RTU)

```
int[] modbus_read_int16(  
    string,  
    byte,  
    string,  
    int,  
    int  
)
```

Note

Similar to Syntax1 with Big Endian (AB CD) setting

modbus_read_int16("TCP_1", 0, "DI", 7200, 2) => **modbus_read_int16**("TCP_1", 0, "DI", 7200, 2, **1**)

Modbus Address data size

Digital	1 address = 1 bit size
Register	1 address = 2 bytes size

If the user defined values are applied to User Setting as

TCP device	0	DO	800	4
TCP device	0	DI	7202	3
TCP device	0	RO	9000	6
TCP device	0	RI	7001	12
TCP device	0	RI	7301	6

```
var_value = modbus_read_int16("TCP_1", 0, "DO", 800, 4)
```

```
// byte[] = {0,0,0,0}    to int16[]  var_value = {0,0} // byte[0][1] , byte[2][3]
```

```
var_value = modbus_read_int16("TCP_1", 0, "DI", 7202, 3)
```

```
// byte[] = {1,0,0}      to int16[]  var_value = {256,0}
```

```
// byte[0][1] , byte[2][3] // Fill up to [3] automatically
```

```
var_value = modbus_read_int16("TCP_1", 0, "RO", 9000, 6)
```

```
// byte[] = {0x54,0x65,0x63,0x68,0x6D,0x61,0x6E,0xE9,0x81,0x94,0xE6,0x98}
```

```
// to int16[]    var_value = {21605,25448,28001,28393,-32364,-6504}
```

```
var_value = modbus_read_int16("TCP_1", 0, "RI", 7001, 12)
```

```
// byte[] = {0x29,0x30,0x9F,0x4C,0xC3,0x7C,0x99,0x9A,0x44,0x5E,0xEC,0xCD,0x42,0xB4,0x00,0x00,  
            0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
```

```
// to int16[]    var_value = {10544,-24756,-15492,-26214,17502,-4915,17076,0,-32768,0,0,0}
```

```
var_value = modbus_read_int16("TCP_1", 0, "RI", 7301, 6)
```

```
// byte[] = {0x07,0xE2,0x00,0x05,0x00,0x12,0x00,0x0F,0x00,0x31,0x00,0x23}
```

```
// to int16[]    var_value = {2018,5,18,15,49,35}
```

```
var_value = modbus_read_int16("TCP_1", 0, "RI", 7301, 6, 0)
```

```
// byte[] = {0x07,0xE2,0x00,0x05,0x00,0x12,0x00,0x0F,0x00,0x31,0x00,0x23}
```

```
// to int16[]    var_value = {2018,5,18,15,49,35}
```

10.3 modbus_read_int32()

Modbus TCP/RTU read function, and transform modbus address data array to int32 array

Syntax 1 (TCP/RTU)

```
int[] modbus_read_int32(  
    string,  
    byte,  
    string,  
    int,  
    int,  
    int  
)
```

Parameters

string	TCP/RTU DEVICE NAME (Set in Modbus Device setting)		
byte	Slave ID		
string	Read type		
	DO	Digital Output	(Function Code : 01)
	DI	Digital Input	(Function Code : 02)
	RO	Register Output	(Function Code : 03)
	RI	Register Input	(Function Code : 04)
int	Starting address		
int	Data length		
int	Follows Little Endian (CD AB) or Big Endian (AB CD) to transform the address data to int32 array.		
	0	Little Endian	
	1	Big Endian (Default)	

Return

int[] The returned int array from modbus server

Syntax 2 (TCP/RTU)

```
int[] modbus_read_int32(  
    string,  
    byte,  
    string,  
    int,  
    int  
)
```

Note

Similar to Syntax1 with Big Endian (AB CD) setting.

modbus_read_int32("TCP_1", 0, "DI", 7200, 2) => **modbus_read_int32**("TCP_1", 0, "DI", 7200, 2, **1**)

Modbus Address data size

Digital	1 address = 1 bit size
Register	1 address = 2 bytes size

If the user defined values are applied to User Setting as

TCP device	0	DO	800	4
TCP device	0	DI	7202	3
TCP device	0	RO	9000	6
TCP device	0	RI	7001	12
TCP device	0	RI	7301	6

```
var_value = modbus_read_int32("TCP_1", 0, "DO", 800, 4)
```

```
// byte[] = {0,0,0,0} to int32[] var_value = {0} // byte[0][1][2][3]
```

```
var_value = modbus_read_int32("TCP_1", 0, "DI", 7202, 3)
```

```
// byte[] = {1,0,0} to int32[] var_value = {16777216} // byte[0][1][2][3] // Fill up to [3] automatically.
```

```
var_value = modbus_read_int32("TCP_1", 0, "RO", 9000, 6)
```

```
// byte[] = {0x54,0x65,0x63,0x68,0x6D,0x61,0x6E,0xE9,0x81,0x94,0xE6,0x98}
```

```
// to int32[] var_value = {1415930728,1835101929,-2120948072}
```

```
var_value = modbus_read_int32("TCP_1", 0, "RI", 7001, 12)
```

```
// byte[] = {0x29,0x30,0x9F,0x4C,0xC3,0x7C,0x99,0x9A,0x44,0x5E,0xEC,0xCD,0x42,0xB4,0x00,0x00,  
0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
```

```
// to int32[] var_value = {691052364,-1015244390,1147071693,1119092736,-2147483648,0}
```

```
var_value = modbus_read_int32("TCP_1", 0, "RI", 7301, 6)
```

```
// byte[] = {0x07,0xE2,0x00,0x05,0x00,0x12,0x00,0x0F,0x00,0x31,0x00,0x23}
```

```
// to int32[] var_value = {132251653,1179663,3211299}
```

```
var_value = modbus_read_int32("TCP_1", 0, "RI", 7301, 6, 0) // byte[2][3][0][1]
```

```
// byte[] = {0x07,0xE2,0x00,0x05,0x00,0x12,0x00,0x0F,0x00,0x31,0x00,0x23}
```

```
// to int32[] value = {0x000507E2,0x000F0012,0x00230031} = {329698,983058,2293809}
```

10.4 modbus_read_float()

Modbus TCP/RTU read function, and transform modbus address data array to float array

Syntax 1 (TCP/RTU)

```
float[] modbus_read_float(  
    string,  
    byte,  
    string,  
    int,  
    int,  
    int  
)
```

Parameters

string	TCP/RTU DEVICE NAME (Set in Modbus Device setting)		
byte	Slave ID		
string	Read type		
	DO	Digital Output	(Function Code : 01)
	DI	Digital Input	(Function Code : 02)
	RO	Register Output	(Function Code : 03)
	RI	Register Input	(Function Code : 04)
int	Starting address		
int	Data length		
int	Follows Little Endian (CD AB) or Big Endian (AB CD) to transform the address data to float array.		
	0	Little Endian	
	1	Big Endian (Default)	

Return

float[] The returned float array from modbus server

Syntax 2 (TCP/RTU)

```
float[] modbus_read_float(  
    string,  
    byte,  
    string,  
    int,  
    int  
)
```

Note

Similar to Syntax1 with Big Endian (AB CD) setting.

modbus_read_float("TCP_1", 0, "DI", 7200, 2) => **modbus_read_float**("TCP_1", 0, "DI", 7200, 2, **1**)

Modbus Address data size

Digital	1 address = 1 bit size
Register	1 address = 2 bytes size

If the user defined values are applied to User Setting as

TCP device	0	DO	800	4
TCP device	0	DI	7202	3
TCP device	0	RO	9000	6
TCP device	0	RI	7001	12
TCP device	0	RI	7301	6

var_value = **modbus_read_float**("TCP_1", 0, "DO", 800, 4)

// byte[] = {0,0,0,0} to float[] var_value = {0} // byte[0][1][2][3]

var_value = **modbus_read_float**("TCP_1", 0, "DI", 7202, 3)

// byte[] = {1,0,0} to float[] var_value = {2.350989E-38} // byte[0][1][2][3]

// Fill up to [3] automatically.

var_value = **modbus_read_float**("TCP_1", 0, "RO", 9000, 6)

// byte[] = {0x54,0x65,0x63,0x68,0x6D,0x61,0x6E,0xE9,0x81,0x94,0xE6,0x98}

// to float[] var_value = {3.940861E+12,4.360513E+27,-5.46975E-38}

var_value = **modbus_read_float**("TCP_1", 0, "RI", 7001, 12)

// byte[] = {0x29,0x30,0x9F,0x4C,0xC3,0x7C,0x99,0x9A,0x44,0x5E,0xEC,0xCD,0x42,0xB4,0x00,0x00,
0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}

// to float[] var_value = {3.921802E-14,-252.6,891.7,90,0,0}

var_value = **modbus_read_float**("TCP_1", 0, "RI", 7001, 12, **0**) // byte[2][3][0][1]

// to float[] var_value = {0x9F4C2930,0x999AC37C,0xECCD445E,0x000042B4,0x00008000,0x00000000}
= {-4.323275E-20,-1.600218E-23,-1.985221E+27,2.392857E-41,4.591775E-41,0}

var_value = **modbus_read_float**("TCP_1", 0, "RI", 7301, 6)

// byte[] = {0x07,0xE2,0x00,0x05,0x00,0x12,0x00,0x0F,0x00,0x3A,0x00,0x26}

// to float[] var_value = {3.400471E-34,1.65306E-39,5.326512E-39}

10.5 modbus_read_double()

Modbus TCP/RTU read function, and transform modbus address data array to double array

Syntax 1 (TCP/RTU)

```
double[] modbus_read_double(  
    string,  
    byte,  
    string,  
    int,  
    int,  
    int  
)
```

Parameters

string	TCP/RTU DEVICE NAME (Set in Modbus Device setting)		
byte	Slave ID		
string	Read type		
	DO	Digital Output	(Function Code : 01)
	DI	Digital Input	(Function Code : 02)
	RO	Register Output	(Function Code : 03)
	RI	Register Input	(Function Code : 04)
int	Starting address		
int	Data length		
int	Follows Little Endian (CD AB) or Big Endian (AB CD) to transform the address data to double array.		
	0	Little Endian	
	1	Big Endian (Default)	

Return

double[] The returned double array from modbus server

Syntax 2 (TCP/RTU)

```
double[] modbus_read_double(  
    string,  
    byte,  
    string,  
    int,  
    int  
)
```

Note

Similar to Syntax1 with Big Endian (AB CD) setting.

modbus_read_double("TCP_1", 0, "DI", 7200, 2) => **modbus_read_double**("TCP_1", 0, "DI", 7200, 2, **1**)

Modbus Address data size

Digital	1 address = 1 bit size
Register	1 address = 2 bytes size

If the user defined values are applied to User Setting as

TCP device	0	DO	800	4
TCP device	0	DI	7202	3
TCP device	0	RO	9000	6
TCP device	0	RI	7001	12
TCP device	0	RI	7301	6

var_value = **modbus_read_double**("TCP_1", 0, "DO", 800, 4)

// byte[] = {0,0,0,0} to double[] var_value = {0} // byte[0][1][2][3][4][5][6][7]

var_value = **modbus_read_double**("TCP_1", 0, "DI", 7202, 3)

// byte[] = {1,0,0} to double[] var_value = {7.2911220195564E-304} //

byte[0][1][2][3][4][5][6][7]

var_value = **modbus_read_double**("TCP_1", 0, "RO", 9000, 6)

// byte[] = {0x54,0x65,0x63,0x68,0x6D,0x61,0x6E,0xE9,0x81,0x94,0xE6,0x98}

// to double[] var_value = {3.65481260356117E+98,-4.87647898854073E-301}

var_value = **modbus_read_double**("TCP_1", 0, "RI", 7001, 12)

// byte[] = {0x29,0x30,0x9F,0x4C,0xC3,0x7C,0x99,0x9A,0x44,0x5E,0xEC,0xCD,0x42,0xB4,0x00,0x00,
0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00}

// to double[] var_value = {2.76472410615396E-110,2.2818627604613E+21,0}

var_value = **modbus_read_double**("TCP_1", 0, "RI", 7001, 12, **0**) // byte[6][7][4][5][2][3][0][1]

// to double[] var_value = {0x999AC37C9F4C2930,0x000042B4ECCD445E,0x0000000000008000}
= {-2.4604103205376E-185,3.62371629877526E-310,1.6189543082926E-319}

var_value = **modbus_read_double**("TCP_1", 0, "RI", 7301, 6)

// byte[] = {0x07,0xE2,0x00,0x05,0x00,0x12,0x00,0x10,0x00,0x0B,0x00,0x29}

```
// to double[]    var_value = {1.06475148078395E-270,1.52982527955113E-308}
```

10.6 modbus_read_string()

Modbus TCP/RTU read function, and convert modbus address data array to string text in UTF8

Syntax 1 (TCP/RTU)

```
string modbus_read_string(  
    string,  
    byte,  
    string,  
    int,  
    int,  
    int  
)
```

Parameters

string	TCP/RTU DEVICE NAME (Set in Modbus Device setting)	
byte	Slave ID	
string	Read type	
	DO	Digital Output (Function Code : 01)
	DI	Digital Input (Function Code : 02)
	RO	Register Output (Function Code : 03)
	RI	Register Input (Function Code : 04)
int	Starting address	
int	Data length	
int	Follows Little Endian (CD AB) or Big Endian (AB CD) to transform the address data to string.	
	*Invalid Parameter. Only support int32, float, double. String follows UTF8 and is sequentially transferred according to address.	
0	Little Endian	
1	Big Endian (Default)	

Return

string The returned UTF8 string from modbus server (Stop at 0x00)

Syntax 2 (TCP/RTU)

```
string modbus_read_string(  
    string,  
    byte,  
    string,
```

```

    int,
    int
)

```

Note

Similar to Syntax1 with Big Endian (AB CD) setting.

modbus_read_string("TCP_1", 0, "RO", 9000, 2) => **modbus_read_string**("TCP_1", 0, "RO", 9000, 2, 1)

Modbus Address data size

Digital	1 address = 1 bit size
Register	1 address = 2 bytes size

If the user defined values are applied to User Setting as

TCP device	0	RO	9000	12
------------	---	----	------	----

modbus_write("TCP_1", 0, "RO", 9000) = "1234 達明機器手臂"

// Undefined numbers of addresses to write, the default value 0 denotes to write the complete data length of 22 bytes.

// Write byte[] = {0x31,0x32,0x33,0x34,0xE9,0x81,0x94,0xE6,0x98,0x8E,
0xE6,0x9C,0xBA,0xE5,0x99,0xA8,0xE6,0x89,0x8B,0xE8,0x87,0x82}

var_value = **modbus_read_string**("TCP_1", 0, "RO", 9000, 3)

// byte[] = {0x31,0x32,0x33,0x34,0xE9,0x81} // RO 3 address = 6 bytes size

// to string = 1234

var_value = **modbus_read_string**("TCP_1", 0, "RO", 9000, 6)

// byte[] = {0x31,0x32,0x33,0x34,0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0x9C}

// to string = 1234 達明

var_value = **modbus_read_string**("TCP_1", 0, "RO", 9000, 12)

// byte[] = {0x31,0x32,0x33,0x34,0xE9,0x81,0x94,0xE6,0x98,0x8E,

0xE6,0x9C,0xBA,0xE5,0x99,0xA8,0xE6,0x89,0x8B,0xE8,0x87,0x82, 0x41,0x42}

// to string = 1234 達明機器手臂 AB // UTF8 format conversion

// The ending, 0x00, will not be included when writing data. When reading 12 addresses, it will read beyond the range.

modbus_write("TCP_1", 0, "RO", 9000) = "1234"+Ctrl("\0")

```

// Write byte[] = {0x31,0x32,0x33,0x34,0x00} // Needs to write 3 Register address

var_value = modbus_read_string("TCP_1", 0, "RO", 9000, 5)

// byte[] = {0x31,0x32,0x33,0x34,0x00,0x00, 0x94,0xE6,0x98,0x8E} // The last 4 values are the original
data at those addresses

// to string = 1234 // UTF8 format conversion stops at 0x00

```

10.7 modbus_write()

Modbus TCP/RTU write function

Syntax 1 (TCP/RTU)

```

bool modbus_write(
    string,
    string,
    ?,
    int
)

```

Parameters

string TCP/RTU Device Name (Set in Modbus Device setting)

string TCP/RTU The predefined parameters belong to TCP/RTU device (Set in Modbus Device setting)

? The input data. The predefined parameters will be applied according to the table below.

Signal Type	Function Code	Type	Input type	Input value
Digital Output	05	byte	byte	(H: 1)(L: 0)
		bool	bool	(H: true)(L: false)
Register Output	06	byte	byte	
		bool	bool	
		int16	int	
Register Output	16	int32	int	
		float	float	
		double	double	
		string	string	

* int32, float, double will be transferred with Little Endian (CD AB) or Big Endian (AB CD) according to user's setting.

* string will be transferred with UTF8 format

* Writing array value is not supported with predefined parameters. To write with the array value, user defined method should be applied (Syntax 3/4)

4, is applied. That is 4 RO = 8 bytes size can be written.

```
modbus_write("TCP_1", "preset_9000", var_ss) // write ABCDEFGH    // The exceeding part will be skipped
// With no number of address, the predefined address length,
4, is applied. That is 4 RO = 8 bytes size can be written.
```

```
modbus_write("TCP_1", "preset_9000", "1234567")    // write 1234567\0    // Use 0x00 to fill up 4
address
// With address length = 0, write all the data.
"09AB123" needs 4 addresses.
```

```
modbus_write("TCP_1", "preset_9000", "09AB123", 0) // write 09AB123\0    // Use 0x00 to fill up 4
address
// With address length = 5, write data in 5
addresses. That is 5 RO = 10 bytes size can be
wrote.
```

```
modbus_write("TCP_1", "preset_9000", "09AB1234", 5) // write 09AB1234    // The input data needs
only 4 addresses.
```

Syntax 3 (TCP/RTU)

```
bool modbus_write(
    string,
    byte,
    string,
    int,
    ?,
    int
)
```

Parameters

string TCP/RTU DEVICE NAME (Set in Modbus Device setting)

byte Slave ID

string Write type

DO Digital Output (Function Code : 05/15)

RO Register Output (Function Code : 06/16)

int Starting address

? Input data

Signal Type	Function Code	Input ? type	Input value
Digital Output	05	byte	(H: 1)(L: 0)

		bool	(H: true)(L: false)
Digital Output	15	byte[]	(H: 1)(L: 0)
		bool[]	(H: true)(L: false)
Register Output	06	byte	
		bool	
Register Output	16	int	
		float	
		double	
		string	
		byte[]	
		int[]	
		float[]	
		double[]	
		string[]	
		bool[]	

*User defined modbus_write will follows **Big-Endian (AB CD)** format to write

* Here int means int32. For int16 type data, GetBytes() needs to be applied first to change int16 to byte[]

int The maximum number of addresses to be write, only effective to string type data

> 0 Valid address length. Write with defined address length

<= 0 Invalid address length. Write all the data

Return

bool **True** Write success

False Write failed

1. If the input data ? is empty string or array
2. If an error occurred in Modbus communication

Syntax 4 (TCP/RTU)

```
bool modbus_write(
    string,
    byte,
    string,
    int,
    ?
)
```

Note

Similar to Syntax3 with address length <= 0, it will write all the data.

modbus_write("TCP_1", 0, "RO", 9000, var_bb) => modbus_write("TCP_1", 0, "RO", 9000, var_bb, 0)

Modbus Address data size

Digital	1 address = 1 bit size
Register	1 address = 2 bytes size

If the user defined values are applied to User Setting as

TCP device	0	DO	800	4
TCP device	0	RO	9000	12

```
byte[] var_bb = {10, 20, 30}
```

```
modbus_write("TCP_1", 0, "DO", 800, var_bb) // write 1,1,1
```

// Zero value, write 0. Non-zero value, write 1.

```
modbus_write("TCP_1", 0, "DO", 800, var_bb, 2) // write 1,1
```

// Address number = 2, only write 2 addresses.

```
modbus_write("TCP_1", 0, "DO", 800, true) // write 1
```

```
int var_i = 10000
```

```
modbus_write("TCP_1", 0, "RO", 9000, var_i) // write 0x00,0x00,0x27,0x10
```

// with int32 BigEndian (AB CD) default

```
var_bb = GetBytes(var_i, 0, 1)
```

// var_bb = {0x10,0x27}

// transfer to int16 LittleEndian (CD AB)

```
modbus_write("TCP_1", 0, "RO", 9000, var_bb) // write 0x10,0x27
```

```
string[] var_n = {"ABC", "12", "34"}
```

```
modbus_write("TCP_1", 0, "RO", 9000, var_n, 2) // write ABC1
```

// Only 2 addresses available, the exceeding values cannot be applied.

```
modbus_write("TCP_1", 0, "RO", 9000, var_n, 5) // write ABC12340
```

// The data needs 4 addresses (0xAB 0xC1 0x23 0x40)

11. TM Ethernet Slave

Ethernet Slave comes with functions established with Socket TCP based on the framework of client/server connections. Once enabled, the robot establishes a Socket TCP Listener Serve to send the robot status and data to all of the connected clients or receive the contents from the clients to execute the respective instructions and update the respective information periodically and promptly without the real-time guarantee.

Like the Modbus Slave, the Ethernet Slave will automatically start on its own after power cycling if it was previously set to **Enable**. The established IP and Port will be shown in the Notice Log on the right.

IP TMflow → System → Network → IP Address
Port 5891
Frequency 100Hz (no real-time guarantee)

11.1 GUI Setting

Modbus Slave

Ethernet Slave

PROFINET IO Device

EtherCAT Server

EtherNet/IP Adapter

Data Table Setting

Enable

STATUS:

Disable

IP Filter:

~

192

168

2

100

~

200

192

168

1

100

~

200

☐ Write Permission

☐ Write Permission

☒ Write Permission

+

-

-

Enable/Disable	Enable or disable Ethernet Slave
IP Filter	IP whitelist Sets ranges for eligible IP addresses that are allowed to connect to the Ethernet Slave. If no filters are set, all devices on the network can connect to the Ethernet Slave.
Write Permission	If checked, allows devices within the corresponding IP range to write to the Ethernet Server with TMSVR commands.

For example, setting IP Filter.

- Group 1 192.168.2.100 ~ 200 denotes IP 192.168.2.100, 192.168.2.101, ... , and 192.168.2.200 are available for connections.
- Group 2 192.168.1.100 ~ 200 denotes IP 192.168.1.100, 192.168.1.101, ... , and 192.168.1.200 are available for connections.
- If the IP address of the client is not in the range of the IPs listed above, it rejects the client to connect.

Group 2 192.168.1.100 ~ 200 has write permission, so clients within the Group 2 IP range can send TMSVR commands and write to the Ethernet Slave. Group 1 192.168.2.100 ~ 200does not have write permission, so clients in Group 1 will receive errors when attempting to write to the Ethernet Slave.

11.2 svr_read()

Read the item value in the communication data table of Ethernet Slave in the Connection Tab of Robot Setting at the local host.

Syntax 1

```
? svr_read(  
    string  
)
```

Parameter

string Item name

Return

? Return value by set data type

Note

For example, checking with TCP_Value float[], Ctrl_DO0 byte, Ctrl_DO1 byte, and g_ss string[] as the communication data table.

Item	Description	Data Type	
TCP_Value	TCP Value	float	6
Ctrl_DO0	Digital Output 0	byte	1
Ctrl_DO1	Digital Output 1	byte	1
g_ss		string[]	6

```
float[] var_fval = svr_read("TCP_Value")           // var_fval = {1, 1, 1, 0.1, 0.2, 0.1}  
byte var_b0 = svr_read("Ctrl_DO0")                 // var_b0 = 0  
byte var_b1 = svr_read("Ctrl_DO1")                 // var_b1 = 1  
string[] var_ss = svr_read("g_ss")                 // var_ss = {"Hi","TM","Robot"}  
                                                    // Use g_ss for global variable naming  
byte var_st = svr_read("Robot_Link")                // var_st = 0 (Robot disconnected) 1 (Robot connected)  
  
float[] var_fval = svr_read("TCP_Value")            // Error. Suppose Ethernet Slave is not launched.  
float[] var_fval = svr_read("TCP_Value1")          // Error. Item name TCP_Value1 does not exist.  
float[] var_fval = svr_read("Coord_Base_Flange")    // Error. Item name Coord_Base_Flange does not exist.  
                                                    // Item name Coord_Base_Flange is predefined but does not exist for not checking as the communication data.
```

11.3 svr_write()

Write the item value into the communication data table of Ethernet Slave in the Connection Tab of Robot Setting at the local host.

Syntax 1

```
bool svr_write(  
    string,  
    ?  
)
```

Parameters

string Item name
? Item value

Return

bool True Write successfully
 False Write failed Possible causes

1. Item name does not exist.
2. Unable to write the read-only item name.
3. Item value to write mismatched with item data type.

Note

For example, checking with TCP_Value float[], Ctrl_DO0 byte, Ctrl_DO1 byte, and g_ss string[] as the communication data table.

Item	Description	Data Type	
TCP_Value	TCP Value	float	€
Ctrl_DO0	Digital Output 0	byte	1
Ctrl_DO1	Digital Output 1	byte	1
g_ss		string[]	€

```
float[] var_tvalue = {1,2,3,0.1,0.2,0.3}
```

```
var_flag = svr_write("TCP_Value", var_tvalue)    // var_flag = false     read-only, invalid process (not an  
error)
```

```
var_flag = svr_write("Ctrl_DO0", 1)             // var_flag = true , Ctrl_DO0 = 1
```

```
var_flag = svr_write("Ctrl_DO1", 0)             // var_flag = true , Ctrl_DO1 = 0
```

```
var_flag = svr_write("TCP_Value", var_tvalue)    // Error. Suppose Ethernet Slave is not launched.
```

```
var_flag = svr_write("TCP_Value1", var_tvalue)   // Error. Item name TCP_Value1 does not exist.
```

```
var_flag = svr_write("Ctrl_DO0", "True")        // Error. Item name Ctrl_DO0 writes value as string (the data  
type is set to byte)
```

11.4 Data Table

Users can use the items listed the Data Table to customize the required data content as well as configure the communication protocol to transmit between the Ethernet Slave and clients, and save the settings as a communication file. When the Ethernet Slave is enabled, the data items in the communication file will be established with the relevant data content to the item to send to the connected clients periodically (no

real-time guarantee). The types of the data format is defined by the settings in the commuicataion file. The client can send data to the server with any type of the supported data formats.

In the protocol, the types of the supported data format are:

- BINARY Binary format, converse in Byte array (Little Endian / UTF8)
- STRING String format, similar to the external command format
- JSON JSON string format

Receive/Send Data Table Settings

Predefined

User-Defined

Global Variable

Item	Description	Data Type
Robot_Error	Error or Not	bool
Project_Run	Project Running or Not	bool
Project_Edit	Project Editing or Not	bool
Project_Pause	Project Pause or Not	bool
Get_Control	Get Control or Not	bool
Safeguard_A	Safety IO (Safeguard Port A trigger)	bool
ESTOP	Emergency Stop	bool
Camera_Light	Light	byte
Robot_Model	Robot Model	string

Transmit File (Data Table)

Transmit File Name:

B

Open

Data Format:

BINARY

Item	Description	Data Type
Project_Edit	Project Editing or Not	bool
Project_Run	Project Running or Not	bool

>>

^

v

Delete All

Save

Item **Robot_Link** is predefined in Ethernet Slave as a type of byte and the attribute of read only to denote whether to connect to the robot.

6. Predefined

Items and settings in this section are defined by TMflow, and the data content of the items is updated by TMflow. The defined items are the general statuses of the robot, such as the coordinates of the robot, the state of the project, the state of the electrical control box, or the IO related statuses, such as digital input / digital output, analog input / analog output.

7. User defined

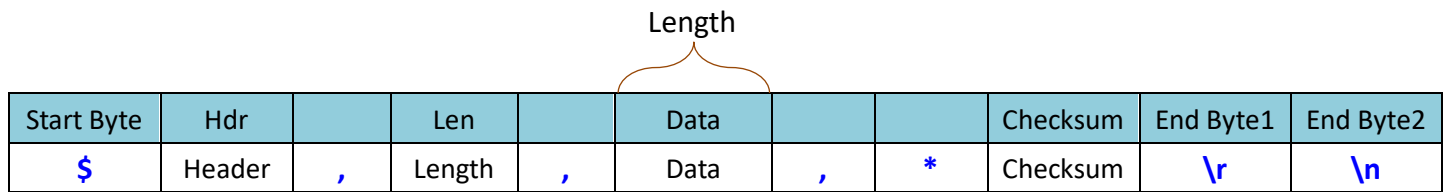
Items and settings in this section are defined by TMflow users for project programs to read / write item data through the Expression Editor or for external users to read / write item data through the TMSVR commands over a TCP/IP connection. With the user defined tab, the project programs can work with external communication devices as a data exchange protocol. The item list in the user-defined tab can be saved as a custom-defined fil to be edited or exchanged data in the future.

8. Global Variable

In the global variable tab, the variable list created by the TMflow users provides a way to directly use the variable name for read / write operations in the project programming, and the external communication devices

can read / write global variables with the communication protocol.

11.5 Communication Protocol



Checksum (XOR of these Bytes)

Name	Size	ASCII	HEX	Description
Start Byte	1	\$	0x24	Start Byte for Communication
Header	X			Header for Communication
Separator	1	,	0x2C	Separator between Header and Length
Length	Y			Length of Data
Separator	1	,	0x2C	Separator between Length and Data
Data	Z			Communication Data
Separator	1	,	0x2C	Separator between Data and Checksum
Sign	1	*	0x2A	Begin Sign of Checksum
Checksum	2			Checksum of Communication
End Byte 1	1	\r	0x0D	
End Byte 2	1	\n	0x0A	End Byte of Communication

*Using the same communication protocol with external commands.

1. Header

Defines the purpose of communication packets. Different headers come with different definitions of communication packets and data.

- **TMSVR** Defines the function of TM Ethernet Slave
- **CPERR** Defines the errors of the communication packets such as packer errors, checksum errors, header errors, and so on.

*Using the same content definitions with external commands.

2. Length

The length indicates the length in the UTF8 bytes occupied by Data. Users can use decimal, hexadecimal, or binary format. The maximum length is 32 bits.

For example,

```
$TMSVR,100,Data,*CS\r\n    // 100 in decimal indicates the data length is 100 bytes
$TMSVR,0x100,Data,*CS\r\n  // 0x100 in hexadecimal indicates the data length is 256 bytes
$TMSVR,0b100,Data,*CS\r\n  // 0b100 in binary indicates the data length is 4 bytes
$TMSVR,8,1,達明,*CS\r\n    // indicates the length of Data, 1,達明, is 8 bytes (UTF8)
```

3. Data

The content of the communication packet can support any character (including 0x00 .. 0xFF and uses UTF8 encoding), and the data length determines by Length. The purpose and description defined in Data must be defined by the header.

4. Checksum

The checksum of the communication packet. The calculation method is XOR (exclusive OR). The calculation range is all Bytes between \$ and * (excluding \$ and *) as shown below.

```
$TMSVR,100,Data,*CS\r\n
```

Checksum = Byte[1] ^ Byte[2] ... ^ Byte[N-6]

The checksum format is set to 2 bytes in hexadecimal (but not 0x), such as

```
$TMSVR,5,10,OK,*7E
```

CS = 0x54 ^ 0x4D ^ 0x53 ^ 0x56 ^ 0x52 ^ 0x2C ^ 0x35 ^ 0x2C ^ 0x31 ^ 0x30 ^ 0x2C ^ 0x4F ^ 0x4B ^ 0x2C = 0x7E

CS = 7E (0x37 0x45)

11.6 TMSVR

Start Byte	Hdr		Len		Data			Checksum	End Byte1	End Byte2
\$	TMSVR	,	Length	,	Data	,	*	Checksum	\r	\n

ID		Mode		Content
Transaction ID	,	0/1/2/3/ 11/12/13	,	Item and Value

TMSVR is defined as the TM Ethernet Slave protocol. The Data section of the packet is further divided into three segments, ID (Transaction ID), Mode (Content Mode), and Content (Item and Value), separated with commas and described below.

ID	The transaction number expressed in any alphanumeric characters. (Reports the CPERR 04 error if a non-alphanumeric byte is encountered.) When used as a communication packet response, it is a transaction number that identifies which group of commands to respond.
,	the symbol to separate
Mode	The mode as the format of the data content
0	Indicates the server responds to the client command in string format.
1	Indicates the content data type in binary format
2	Indicates the content data type in string format
3	Indicates the content data type in JSON format
11	Indicates the content data type in binary format (Request read)
12	Indicates the content data type in string format (Request read)
13	Indicates the content data type in JSON format (Request read)
●	1/2/3 are for client write to server and client read from server. The client read from server is that the server sends contents to the connected client periodically.
●	11/12/13 are for client read from server with request read, which is the client sends read request for the item and the server responds with the item value to the client.
,	the symbol to separate
Content	The data content. Formatted by the mode definition.

Note

TMSVR command is for the client and the server to communicate in both directions. Under normal circumstances, the server will broadcast the data items from the Transmit and User Defined communication files to the connected clients periodically. When the server sends to the client, the data is sent to the client to read from the server with no response to the server required. When the client sends to the server, the data is received by the server from the client to write with response to the client required.

The transaction number, when the server sends data, cycles from 0 to 9 with each iteration. When the client sends data to the server, the transaction number can be in any alphanumeric characters customized at the client side. If the communication packet format is checked and correct , the server will reply the client with the command processing status by the transaction number in the packet.

When the client sends data to the server, the server checks whether all the criteria to write are correct before performing data writing. If there is any error with the write command, no data will be overwritten. The criteria to write for inspection are:

1. The validity of the mode as the format of the data content
2. The connected client's write permissions based on the IP Filter.

3. The data content matches to the mode.
4. The item to write exists in the Transmit or User Defined communication file.
5. The attribute of the item to write is not read only.
6. The robot is in the appropriate mode (M/A).
7. The written data matches the data type of each item.

When the Mode is 11/12/13, the request read method is used. The client sends an item request and the server responds to the item value. The requested items can be the items in the Predefined area without limited to check the periodical delivery, but in the Userdefined area and the GlobalVariable area, for custom definitions, it still need to check the periodical delivery to get the request.

When the client sends an item request, it is not limited to only one item but multiple items can be acquired at once. When the item request is success, the server will send the item value to the client based on the format matching to Mode 11/12/13. However, if the item request is failed such as the item name does not exist, the server will send the command processing status based on the Mode 0 format.

0. Mode = 0 (the status the server responds to the client command processing)

After the server receives and processes a write command from a client, it will respond with another TMSVR command with Mode 0. The details for Mode 0 are as follows.

Data

ID		Mode		Error Code		Error Description
Transaction ID	,	0	,	00 .. 07	,	

Transaction ID Defined while the client sends the command for the server to respond with.

Mode 0 for the server to respond to the client

Error Code Error code definitions. **Fixed as 2 bytes and in hexadecimal (but not 0x)**

- | | |
|----|--|
| 00 | Correct writing. No error. |
| 01 | The communication format or mode is not supported. (Ex. Mode = 99) |
| 02 | The connected client is not permitted to write. (IP filer without write permission) |
| 03 | The communication format and the data content format are mismatched.
(Ex. Mode = 3, but the data content is not in JSON format) |
| 04 | Item to write or read does not exist. |
| 05 | Unable to write to read-only items. |
| 06 | Incorrect M/A mode while writing. |
| 07 | Values to write mismatches with the configured type or the size. |

Error Description	Error description, following the error code.	
00	OK	
01	NotSupport	
02	WritePermission	
03	InvalidData	
04	NotExist;XXX	// ;XXX denotes which data item
05	ReadOnly;XXX	
06	ModeError;XXX	
07	ValueError;XXX	
<	\$TMSVR,15,S0,2,Ctrl_DO0=1,*76\r\n // transcation ID S0, string format, set Ctrl_DO0=1	
>	\$TMSVR,10,S0,0,00,OK,*18\r\n // server responds transcation ID S0, mode 0, error code 00, correct writing	
<	\$TMSVR,16,S1,99,Ctrl_DO0=1,*46\r\n // transcation ID S1, mode 99	
>	\$TMSVR,18,S1,0,01,NotSupport,*0E\r\n // server responds transcation ID S1, mode 0, error code 01, mode not support	
<	\$TMSVR,15,S2,2,Ctrl_DO0=1,*74\r\n // transcation ID S2, string format, set Ctrl_DO0=1	
>	\$TMSVR,23,S2,0,02,WritePermission,*6A\r\n // server responds transcation ID S2, mode 0, error code 02, the connected client is not granted with write permission.	
<	\$TMSVR,15,S3,3,Ctrl_DO0=1,*74\r\n // transcation ID S3, JSON format, set Ctrl_DO0=1	
>	\$TMSVR,19,S3,0,03,InvalidData,*74\r\n // server responds transcation ID S3, mode 0, error code 03, JSON format, data format (JSON) mistached with the content data format (STRING)	
<	\$TMSVR,16,S4,2,Ctrl_DO32=1,*40\r\n // transcation ID S4, string format, set Ctrl_DO32=1	
>	\$TMSVR,26,S4,0,04,NotExist;Ctrl_DO32,*58\r\n // server responds transcation ID S4, mode 0, error ode 04, item Ctrl_DO32 does not exist.	
<	\$TMSVR,17,S5,2,Robot_Link=1,*07\r\n // transcation ID S5, string format, set Robot_Link=1	

> \$TMSVR,27,S5,0,05,ReadOnly;Robot_Link,*1E\r\n

// server responds transction ID S5, mode 0, error code 05, the item Robot_Link is read only.

Supposed the user defined Item: adata, Type: int, Size: 4, and Write: Auto.

< \$TMSVR,20,S6,2,adata={1,2,3,4},*55\r\n // transction ID S6, string format, set adata={1,2,3,4}

> \$TMSVR,23,S6,0,06,ModeError;adata,*2D\r\n

// server responds transction ID S6, mode 0, error code 06, M/A mode mismatched while writing (suppose it it Manual Mode while writing).

< \$TMSVR,18,S7,2,adata={1,2,3},*47\r\n // transction ID S7, string format, set adata={1,2,3}

> \$TMSVR,24,S7,0,07,ValueError;adata,*42\r\n

// server responds transction ID S7, mode 0, error code 07, writing values and data size or type mismatched. (the configured size is 4, but there is only 3 to write.)

1. Mode = 1 BINARY

The data content is transmitted in binary mode by converting the data item name with the Little Endian value and the value with UTF8 to a byte array accordingly. The format is shown as below.

Data

ID		Mode		Content
Transaction ID	,	1	,	Item and Value

Length of Item	Item	Length of Value	Value	...
2 bytes Little Endian	UTF8	2 bytes Little Endian	Little Endian / UTF8	

Length of Item 2 bytes in Little Endian, value from 0 to 65535 indicating the length of the item that follows

Item item name

Length of Value 2 bytes in Little Endian), value from 0 to 65535 indicating the length of the data that follows

Value data value

Check TCP_Value float[] and Ctrl_DO0 byte as the communication data and transmit in binary mode as an example.

Item	Description	Data Type	
TCP_Value	TCP Value	float	4
Ctrl_DO0	Digital Output 0	byte	1

```

> 24 54 4D 53 56 52 2C    // $TMSVR,    // Header
36 39 2C                  // 69,        // Length
30 2C 31 2C              // 0,1,        // transaction ID 0, mode 1, binary
0A 00 52 6F 62 6F 74 5F 4C 69 6E 6B 01 00 00  // Robot_Link=0 // The name occupied 10 bytes, the value, 1
byte
09 00 54 43 50 5F 56 61 6C 75 65 18 00 00 00 80 3F 00 00 80 3F 00 00 80 3F CD CC CC 3D CD CC 4C
3E CD CC CC 3D          // TCP_Value={1,1,1,0.1,0.2,0.1} // The name occupied 9 bytes, the value, 24 bytes
08 00 43 74 72 6C 5F 44 4F 30 01 00 00  // Ctrl_DO0=0 // The name occupied 8 bytes, the value, 1 byte
2C 2A 39 36 0D 0A        // ,*96\r\n    // Checksum

< 24 54 4D 53 56 52 2C    // $TMSVR,    // Header
31 38 2C                  // 18,        // Length
54 31 2C 31 2C          // T1,1,      // transaction ID T1, mode 1, binary
08 00 43 74 72 6C 5F 44 4F 30 01 00 01  // Ctrl_DO0=1 // The name occupied 8 bytes, the value, 1 byte
2C 2A 37 41 0D 0A        // ,*7A\r\n    // Checksum

> $TMSVR,10,T1,0,00,OK,*1E\r\n    // server responds to ID T1, mode 1, error code 00, correct
writing

```

Once the data type of the item to send is string [], two bytes, 0x00 0x00, are inserted between the string elements as the separators.

```

> 24 54 4D 53 56 52 2C    // $TMSVR,    // Header
39 30 2C                  // 90,        // Length
30 2C 31 2C              // 0,1        // transaction ID 0, mode 1, binary
0A 00 52 6F 62 6F 74 5F 4C 69 6E 6B 01 00 00  // Robot_Link=0 //The name occupied 10 bytes, the value, 1
byte
09 00 54 43 50 5F 56 61 6C 75 65 18 00 00 00 80 3F 00 00 80 3F 00 00 80 3F CD CC CC 3D CD CC 4C
3E CD CC CC 3D          // TCP_Value={1,1,1,0.1,0.2,0.1} //The name occupied 9 bytes, the value, 24 bytes
08 00 43 74 72 6C 5F 44 4F 30 01 00 01  // Ctrl_DO0=1 // The name occupied 8 bytes, the value, 1 byte
04 00 67 5F 73 73 0D 00 48 69 00 00 54 4D 00 00 52 6F 62 6F 74
// g_ss={"Hi","TM","Robot"} // The name occupied 4 bytes, the value, 13 bytes

```

2C 2A 44 43 0D 0A //,*DC\r\n // Checksum

Also, if the data type of the item to receive is string [], when converting to a byte array, two bytes, 00 00, are inserted between the string elements as the separators.

```
<  24 54 4D 53 56 52 2C      // $TMSVR,      // Header
    32 35 2C                  // 25,          // Length
    54 32 2C 31 2C           // T2,1,        // transaction ID T2, mode 1, binary
    04 00 67 5F 73 73 0C 00 48 65 6C 6C 6F 00 00 57 6F 72 6C 64
                                // g_ss={"Hello", "World"} // The name occupied 4 bytes, the value, 12 bytes
    2C 2A 30 32 0D 0A        // ,*02\r\n     // Checksum
>  $TMSVR,10,T2,0,00,OK,*1D\r\n //server responds to ID T2, mode 0, error code 00, correct writing
```

2. Mode = 2 STRING

The data content is transmitted as a string with the name and value of the data item in the Script string of an external command. The format is shown as below.

Data

ID		Mode		Content
Transaction ID	,	2	,	Item and Value

<i>Item</i>	=	<i>Value</i>	\r\n	...
-------------	---	--------------	------	-----

Item item name
 = equal
 Value data value
 \r\n symbol of carriage return as required if there is an item up next for separation.

Check TCP_Value float[] and Ctrl_DO0 byte, Ctrl_DO1 byte, g_ss string[] as the communication data and transmit in string mode as an example.

Item	Description	Data Type	
TCP_Value	TCP Value	float	€
Ctrl_DO0	Digital Output 0	byte	1
Ctrl_DO1	Digital Output 1	byte	1
g_ss		string[]	€

```

> $TMSVR,97,9,2,Robot_Link=0\r\n // Robot_Link=0 // transaction ID 9, mode 2, string
TCP_Value={1,1,1,0.1,0.2,0.1}\r\n // TCP_Value={1,1,1,0.1,0.2,0.1}
Ctrl_DO0=1\r\n // Ctrl_DO0=1
Ctrl_DO1=0\r\n // Ctrl_DO1=0
g_ss={"Hi","TM","Robot"},*77\r\n // g_ss={"Hi","TM","Robot"}

< $TMSVR,15,T2,2,Ctrl_DO0=0\r\n // set Ctrl_DO0=0// transaction ID T2, mode 2, string
Ctrl_DO1=1,*34\r\n // set Ctrl_DO1=1

> $TMSVR,10,T2,0,00,OK,*1D\r\n // server responds to ID T2, mode 0, error code 00, correct writing

```

3. Mode = 3 JSON

The data content is transmitted as a JSON string with the name and value of the data item serialized in the JSON format as shown below.

Data

ID		Mode		Content
Transaction ID	,	3	,	Item and Value

Item item name
Value data value

```

public class TMSVRJsonData
{
    public string Item;
    public object Value;
}

```

*[] array is in use when it comes it multiple items.

Check TCP_Value float[] and Ctrl_DO0 byte, Ctrl_DO1 byte, g_ss string[] as the communication data and transmit in JSON mode as an example.

Item	Description	Data Type	
TCP_Value	TCP Value	float	6
Ctrl_DO0	Digital Output 0	byte	1
Ctrl_DO1	Digital Output 1	byte	1
g_ss		string[]	6

```

> $TMSVR,196,5,3,["Item":"Robot_Link","Value":0], // Robot_Link=0

```

```

// transaction ID 5, mode 3, JSON
{"Item":"TCP_Value","Value":[1.0,1.0,1.0,0.1,0.2,0.1]}, // TCP_Value={1,1,1,0.1,0.2,0.1}
{"Item":"Ctrl_DO0","Value":0}, // Ctrl_DO0=0
{"Item":"Ctrl_DO1","Value":0}, // Ctrl_DO1=0
{"Item":"g_ss","Value":["Hi","TM","Robot"]}],*3A\r\n // g_ss={"Hi","TM","Robot"}

< $TMSVR,113,T9,3, [{"Item":"Ctrl_DO0","Value":1}, // Ctrl_DO0=1
{"Item":"Ctrl_DO1","Value":0}, // Ctrl_DO1=0
{"Item":"g_ss","Value":["Hello","TM","Robot"]}],*7C\r\n // g_ss={"Hello","TM","Robot"}
> $TMSVR,10,T9,0,00,OK,*16\r\n // server responds to ID T9, mode 0, error code 0, correct writing

```

11. Mode = 11 BINARY (Request read)

The data content is transmitted in binary mode by converting the data item name with the Little Endian value and the value with UTF8 to a byte array accordingly. The format is shown as below.

Data (client to server)

ID		Mode		Content
Transaction ID	,	11	,	Item

Length of Item 2 bytes Little Endian	Item UTF8	...
--	---------------------	-----

The difference of the read request from Mode = 1 is no value required.

Length of Item 2 bytes in Little Endian, value from 0 to 65535 indicating the length of the item that follows

Item Item name

Check TCP_Value float[] and Ctrl_DO0 byte as the communication data and transmit in binary mode as an example

Item	Description	Data Type	
TCP_Value	TCP Value	float	6
Ctrl_DO0	Digital Output 0	byte	1

server periodical delivery

```

> 24 54 4D 53 56 52 2C // $TMSVR, // Header
36 39 2C // 69, // Length

```

```

30 2C 31 2C          // 0,1,          // transaction ID 0, mode 1, binary

0A 00 52 6F 62 6F 74 5F 4C 69 6E 6B 01 00 00  // Robot_Link=0 // The name occupied 10 bytes, the value, 1
byte

09 00 54 43 50 5F 56 61 6C 75 65 18 00 00 00 80 3F 00 00 80 3F 00 00 80 3F CD CC CC 3D CD CC 4C
3E CD CC CC 3D          // TCP_Value={1,1,1,0.1,0.2,0.1} // The name occupied 9 bytes, the value, 24 bytes

08 00 43 74 72 6C 5F 44 4F 30 01 00 00  // Ctrl_DO0=0 // The name occupied 8 bytes, the value, 1 byte

2C 2A 39 36 0D 0A      // ,*96\r\n // Checksum

```

client requested to read

```

< 24 54 4D 53 56 52 2C  // $TMSVR, // Header

32 36 2C                // 26, // Length

51 31 2C 31 31 2C      // Q1,11, // transaction ID Q1, mode 11 binary (Request read)

08 00 43 74 72 6C 5F 44 4F 30  // Ctrl_DO0 // The name occupied 8 bytes

08 00 54 43 50 5F 4D 61 73 73  // TCP_Mass // The name occupied 8 bytes

2C 2A 37 46 0D 0A      // ,*7F\r\n // Checksum

```

server replied with the item value

```

> 24 54 4D 53 56 52 2C  // $TMSVR, // Header

33 35 2C                // 35, // Length

51 31 2C 31 31 2C      // Q1,11, // server responds to ID Q1, mode 11 binary

08 00 43 74 72 6C 5F 44 4F 30 01 00 00  // Ctrl_DO0=0 // The name occupied 8 bytes, the value, 1 byte

08 00 54 43 50 5F 4D 61 73 73 04 00 00 00 00  // TCP_Mass=0 // The name occupied 8 bytes, the value, 4 byte

2C 2A 37 38 0D 0A      // ,*78\r\n // Checksum

```

* server replied the same content format as Mode = 1 BINARY

client requested to read

```

< 24 54 4D 53 56 52 2C  // $TMSVR, // Header

32 36 2C                // 26, // Length

51 32 2C 31 31 2C      // Q2,11, // transaction ID Q1, mode 11 binary (Request read)

08 00 43 74 72 6C 5F 44 4F 30  // Ctrl_DO0 // The name occupied 8 bytes

08 00 54 43 50 5F 4D 61 58 58  // TCP_MaXX // The name occupied 8 bytes

```

2C 2A 37 43 0D 0A //,*7C\r\n // Checksum

server replied with the item value

```
> $TMSVR,25,Q2,0,04,NotExist;TCP_MaxX,*17\r\n
// server responds to ID Q2, mode 0, error code 04, item not existed
```

12. Mode = 12 STRING (Request read)

The data content is transmitted as a string with the name and value of the data item in the Script string of an external command. The format is shown as below.

Data (client to server)

ID		Mode		Content
Transaction ID	,	12	,	Item and Value

Item	\r\n	...	No Value required.
------	------	-----	--------------------

Item Item name

\r\n The newline characters. Required only as a delimiter if the next item comes.

Check TCP_Value float[] and Ctrl_DO0 byte, Ctrl_DO1 byte, g_ss string[] as the communication data and transmit in STRING mode.

Item	Description	Data Type	
TCP_Value	TCP Value	float	€
Ctrl_DO0	Digital Output 0	byte	1
Ctrl_DO1	Digital Output 1	byte	1
g_ss		string[]	€

server periodical delivery

```
> $TMSVR,97,9,2,Robot_Link=0\r\n      // Robot_Link=0      // transaction ID 9, mode 2
TCP_Value={1,1,1,0.1,0.2,0.1}\r\n      // TCP_Value={1,1,1,0.1,0.2,0.1}
Ctrl_DO0=1\r\n                              // Ctrl_DO0=1
Ctrl_DO1=0\r\n                              // Ctrl_DO1=0
g_ss={"Hi","TM","Robot"},*77\r\n      // g_ss={"Hi","TM","Robot"}
```

client requested to read

```
< $TMSVR,28,Q2,12,Robot_Link\r\n      // Item Robot_Link
```

// transaction ID Q2, mode 12 JSON (Request read)

TCP_Mass,*0E\r\n // Item TCP_Mass

server replied with the item value

> \$TMSVR,30,Q2,12,Robot_Link=0\r\n // server responds to ID Q2, mode 12

TCP_Mass=0,*09\r\n

* server replied the same content format as Mode = 2 STRING

13. Mode = 13 JSON (Request read)

The data content is transmitted as a JSON string with the name and value of the data item serialized in the JSON format as shown below.

Data (client to server)

ID		Mode		Content
Transaction ID	,	13	,	Item and Value

Item Item name
Value Data value

```
public class TMSVRJsonData
{
    public string Item;
    public object Value;
}
```

* [] array is in use when it comes it multiple items.

* Shared with Mode = 3 JSON for using the same class for serialization / deserialization, but the Value attribute may not exist

Check TCP_Value float[] and Ctrl_DO0 byte, Ctrl_DO1 byte, g_ss string[] as the communication data and transmit in JSON mode.

Item	Description	Data Type	
TCP_Value	TCP Value	float	€
Ctrl_DO0	Digital Output 0	byte	1
Ctrl_DO1	Digital Output 1	byte	1
g_ss		string[]	€

server periodical delivery

> \$TMSVR,196,5,3,["Item":"Robot_Link","Value":0], // Robot_Link=0 // transaction ID 5, mode

3

["Item":"TCP_Value","Value":[1.0,1.0,1.0,0.1,0.2,0.1]], // TCP_Value={1,1,1,0.1,0.2,0.1}

```
{ "Item": "Ctrl_DO0", "Value": 0 },           // Ctrl_DO0=0
{ "Item": "Ctrl_DO1", "Value": 0 },           // Ctrl_DO1=0
{ "Item": "g_ss", "Value": ["Hi", "TM", "Robot"] }, *3A\r\n // g_ss={"Hi", "TM", "Robot"}
```

client requested to read

```
< $TMSVR,27,Q3,13,{ "Item": "TCP_Mass" }, *3C\r\n
                                     // transaction ID Q3, mode 13 JSON (Request
                                     read)
```

server replied with the item value

```
> $TMSVR,39,Q3,13,{ "Item": "TCP_Mass", "Value": 0.0 }, *40\r\n // server responds to ID Q3, mode 13
```

* server replied the same content format as Mode = 3 JSON

12. Profinet Functions

The robot communicates with external controllers via the Profinet communication protocol. In the mechanism of the Profinet communication protocol, the robot works as a Profinet IO device for external devices to read and write the robot data. Meanwhile, TMflow monitors the table of data receiving from external devices and the table of data sending to external devices with Profinet functions as well as changes the custom definition section in the table of data sending to external devices.

Communication Data Table

The data table is composed of the input data and the output data. Input Data Table is for external devices posting on the robot, and Output Data Table is for the robot sending to external devices. Both of the data tables come with System Definition Section and Custom Definition Section for data.

1. System Definition Section : Items and settings are defined by the robot, and the data contents are updated by the robot or external devices. The defined items are robot status relevant such as robot bases, project status, control box status, or input/output status relevant such as digital I/Os and analog I/Os. Users can use Profinet functions to read the input data table and the output data table in the system definition section.
2. Custom Definition Section : Items and settings are defined by users, and the data contents are updated by users or external devices. In the meantime of the project editing, users can use Profinet functions to read and write the output data table in the custom definition section or read input data table in the custom definition section as well as use the custom definition section as a data exchange register between the project and external devices.

Communication Data Table (at the robot's viewpoint)	Data Section	TMflow Profinet Function Permissions	External Device Permissions
Input Data Table	System Definition Section	Read	Write
	Custom Definition Section	Read	Write
Output Data Table	System Definition Section	Read	Read
	Custom	Read/Write	Read

	Definition Section		
--	-----------------------	--	--

12.1 profinet_read_input()

Read the input table content.

Syntax 1

```
byte[] profinet_read_input (
    int,
    int
)
```

Parameters

`int` Starting address
`int` The address amount to read

Return

`byte[]` Return data in a byte array.

Note

```
byte[] var_value = profinet_read_input(148, 16)
// {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
```

Syntax 2

```
byte profinet_read_input (
    int,
)
```

Parameters

`int` Starting address

Return

`byte` Return data in byte.

Note

```
byte var_value = profinet_read_input(148)
// 0x30
```

Syntax 3

```
? profinet_read_input (
    string,
    int,
```

```
int
```

```
)
```

Parameters

`string` Item name

`int` The starting shifted address of the item

`int` The amount of the addresses to read

Return

`?` The data type returned by the item definition in the communication data table.

* Data type includes `byte, byte[], int, int[], float, float[], string`

Syntax 4

```
? profinet_read_input(
```

```
    string,
```

```
    int,
```

```
)
```

Parameters

`string` Item name

`int` The starting shifted address of the item

Return

`?` The data type returned by the item definition in the communication data table.

* Data type includes `byte, byte[], int, int[], float, float[], string`

Note

* Same as syntax 3. Read to the end of the item by default.

* Reading data based on the configuration file Little Endian (DCBA) or Big Endian (ABCD).

Syntax 5

```
? profinet_read_input(
```

```
    string
```

```
)
```

Parameter

`string` Item name

Return

`?` The data type returned by the item definition in the communication data table.

* Data type includes `byte, byte[], int, int[], float, float[], string`

Note

* Same as syntax 3. Fill 0 as the starting shifted address of the item and read to the end of the item by

Year	Number of people (millions)
1990	0
1995	20
2000	40
2005	60
2010	80

* Reading data based on the configuration file Little Endian (DCBA) or Big Endian (ABCD).

```
byte var_b = profinet_read_input("StickStatus",0,1)

// 0x02

var_b = profinet_read_input("StickStatus",0)

// 0x02

var_b = profinet_read_input("StickStatus")

// 0x02

byte[] var_ba = profinet_read_input("CtrlBox_DO",0,2)

// {0x00,0x04}

var_ba = profinet_read_input("CtrlBox_DO")

// {0x00,0x04}

int[] var_ia = profinet_read_input("Register_Int",0,12)

// byte[] = {0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x02,0x00,0x00,0x00,0x03} (Little Endian)    to int[]

// int[] = {0x00000001,0x00000002,0x00000003} (Little Endian)

// int[] = {1,2,3}

var_ia = profinet_read_input("Register_Int",12)

// byte[] = {0x00,0x00,0x00,0x04,0x00,0x00,0x00,0x05,0x00,0x00,0x00,0x06,0x00,0x00,0x00,0x07,

            0x00,0x00,0x00,0x08,0x00,0x00,0x00,0x09,0x00,0x00,0x00,0x0A,0x00,0x00,0x00,0x0B,

            0x00,0x00,0x00,0x0C,0x00,0x00,0x00,0x0D,0x00,0x00,0x00,0x0E,0x00,0x00,0x00,0x0F,

            0x00,0x00,0x00,0x10,0x00,0x00,0x00,0x11,0x00,0x00,0x00,0x12,0x00,0x00,0x00,0x13,

            0x00,0x00,0x00,0x14,0x00,0x00,0x00,0x15,0x00,0x00,0x00,0x16,0x00,0x00,0x00,0x17,

            0x00,0x00,0x00,0x18,0x00,0x00,0x00,0x19,0x00,0x00,0x00,0x1A,0x00,0x00,0x00,0x1B,

            0x00,0x00,0x00,0x1C,0x00,0x00,0x00,0x1D,0x00,0x00,0x00,0x1E,0x00,0x00,0x00,0x1F} (Little Endian)    to int[]

// int[] = {0x00000004,0x00000005,0x00000006,0x00000007,0x00000008,0x00000009,

            0x0000000A,0x0000000B,0x0000000C,0x0000000D,0x0000000E,0x0000000F,

            0x00000010,0x00000011,0x00000012,0x00000013,0x00000014,0x00000015,

            0x00000016,0x00000017,0x00000018,0x00000019,0x0000001A,0x0000001B,

            0x0000001C,0x0000001D,0x0000001E,0x0000001F} (Little Endian)

// int[] = {4,5,6,7,8,9,10,11,12,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
```



```
var_fa = profinet_read_input("Register_Float")

// byte[] = {0x3F,0x8C,0xCC,0xCD,0x3F,0x99,0x99,0x9A,0x3F,0xA6,0x66,0x66,0x40,0x06,0x66,0x66,
             0x40,0x0C,0xCC,0xCD,0x40,0x13,0x33,0x33,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
             0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
             0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
             0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
             0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
             0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
             0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Big Endian)    to float[]

// float[] = {0x3F8CCCCD,0x3F99999A,0x3FA66666,0x40066666,0x400CCCCD,0x40133333,
             0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,
             0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,
             0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,
             0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000} (Big Endian)

// float[] = {1.1,1.2,1.3,2.1,2.2,2.3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
```

12.2 profinet_read_input_int()

Read the input table content and convert the data to the 32-bit integer.

Syntax 1

```
int[] profinet_read_input_int(
    int,
    int,
    int
)
```

Parameters

```
int    Starting address
int    The address amount to read
int    The conversion of the read data to an int array based on Little Endian (DCBA) or Big Endian (ABCD).
0      Little-Endian
1      Big-Endian
2      Based on the configuration file.
```

Return

`int[]` Return data in an integer array.

Note

```
int[] var_value = profinet_read_input_int(164, 12, 0)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)     to int[]
// int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)
// int[] = {32767,99999,-32768}

int[] var_value = profinet_read_input_int(164, 11, 0)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF} (Little Endian)     to int[]
// int[] = {0x00007FFF,0x0001869F,0x00FF8000} (Little Endian)
// int[] = {32767,99999,16744448}

int[] var_value = profinet_read_input_int(164, 10, 0)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80} (Little Endian)     to int[]
// int[] = {0x00007FFF,0x0001869F,0x00008000} (Little Endian)
// int[] = {32767,99999,32768}

int[] var_value = profinet_read_input_int(164, 12, 1)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)     to int[]
// int[] = {0xFF7F0000,0x9F860100,0x0080FFFF} (Big Endian)
// int[] = {-8454144,-1618607872,8454143}

int[] var_value = profinet_read_input_int(164, 12, 2)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)     to int[]
// int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)
// int[] = {32767,99999,-32768}
```

Syntax 2

```
int[] profinet_read_input_int(
    int,
    int
)
```

Parameters

`int` Starting address
`int` The address amount to read

Note

Same as Syntax 1 with the parameter of the conversion of the read data defaults to **2**.

* Convert the read data to an int array based on Little Endian (DCBA) or Big Endian (ABCD).

```
int[] var_ia = profinet_read_input_int(164,12,0)
```

```

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]

// int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)

// int[] = {32767,99999,-32768}

var_ia = profinet_read_input_int(164,11,0)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF} (Little Endian)    to int[]

// int[] = {0x00007FFF,0x0001869F,0x00FF8000} (Little Endian)

// int[] = {32767,99999,16744448}

var_ia = profinet_read_input_int(164,10,0)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80} (Little Endian)    to int[]

// int[] = {0x00007FFF,0x0001869F,0x00008000} (Little Endian)

// int[] = {32767,99999,32768}


var_ia = profinet_read_input_int(164,12,1)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Big Endian)to int[]

// int[] = {0xFF7F0000,0x9F860100,0x0080FFFF} (Big Endian)

// int[] = {-8454144,-1618607872,8454143}


var_ia = profinet_read_input_int(164,12,2)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]

// int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)

// int[] = {32767,99999,-32768}


var_ia = profinet_read_input_int(164,12)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]

// int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)

// int[] = {32767,99999,-32768}

```

Syntax 3

```

int profinet_read_input_int (
    int
)

```

Paramaters

int Starting address

* Convert the read data to an int array based on Little Endian (DCBA) or Big Endian (ABCD).

Return

`int` Return data in integer.

Note

```
int var_value = profinet_read_input_int(164)

// byte[] = {0xE4,0x07,0x00,0x00} (Little Endian)    to int
// int = 0x000007E4 (Little Endian)
// int = 2020

int var_value = profinet_read_input_int(164)

// byte[] = {0x00,0x00,0x07,0xE4} (Big Endian)      to int
// int = 0x000007E4 (Big Endian)
// int = 2020
```

12.3 profinet_read_input_float()

Read the input table content and convert the data to the 32-bit floating-point number.

Syntax 1

```
float[] profinet_read_input_float(
    int,
    int,
    int
)
```

Parameters

<code>int</code>	Starting address
<code>int</code>	The address amount to read
<code>int</code>	The conversion of the read data to a float array based on Little Endian (DCBA) or Big Endian (ABCD).
0	Little-Endian
1	Big-Endian
2	Based on the configuration file.

Return

`float[]` Return data in a floating-point number array.

Syntax 2

```
float[] profinet_read_input_float(
    int,
```

```
int
```

```
)
```

Note

Same as Syntax 1 with the parameter of the conversion of the read data defaults to **2**.

* Convert the read data to a float array based on Little Endian (DCBA) or Big Endian (ABCD).

```
float[] var_fa = profinet_read_input_float(284,12,0)
```

```
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian) to float[]
```

```
// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)
```

```
// float[] = {1.0,2.0,3.0}
```

```
var_fa = profinet_read_input_float(284,11,0)
```

```
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40} (Little Endian) to float[]
```

```
// float[] = {0x3F800000,0x40000000,0x00400000} (Little Endian)
```

```
// float[] = {1.0,2.0,5.877472E-39}
```

```
var_fa = profinet_read_input_float(284,10,0)
```

```
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00} (Little Endian) to float[]
```

```
// float[] = {0x3F800000,0x40000000,0x00000000} (Little Endian)
```

```
// float[] = {1.0,2.0,0.0}
```

```
var_fa = profinet_read_input_float(284,12,1)
```

```
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian) to float[]
```

```
// float[] = {0x0000803F,0x00000040,0x00004040} (Big Endian)
```

```
// float[] = {4.600603E-41,8.96831E-44,2.304856E-41}
```

```
var_fa = profinet_read_input_float(284,12,2)
```

```
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian) to float[]
```

```
// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)
```

```
// float[] = {1.0,2.0,3.0}
```

```
var_fa = profinet_read_input_float(284,12)
```

```
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian) to float[]
```

```
// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)
```

```
// float[] = {1.0,2.0,3.0}
```

Syntax 3

```
float profinet_read_input_float(  
    int  
)
```

Parameters

`int` Starting address
* Convert the read data to a float array based on Little Endian (DCBA) or Big Endian (ABCD).

Return

`float` Return data in floating-point

Note

```
float var_value = profinet_read_input_float(284)  
  
    // byte[] = {0x00,0x00,0x80,0x3F} (Little Endian) to float  
    // float = 0x3F800000 (Little Endian)  
    // float = 1.0  
  
float var_value = profinet_read_input_float(284)  
  
    // byte[] = {0x3F,0x80,0x00,0x00} (Big Endian) to float  
    // float = {0x3F800000} (Big Endian)  
    // float = {1.0}
```

12.4 profinet_read_input_string()

Read the input table content and convert the data to the string encoded in UTF8.

Syntax 1

```
string profinet_read_input_string(  
    int,  
    int  
)
```

Parameters

`int` Starting address
`int` The address amount to read

Return

`string` Return data in a UTF8 string (ending with 0x00 encountered).

Note

```
string var_value = profinet_read_input_string(148,16)
```

```

// byte[] = {0x54,0x4D,0x35,0x2D,0x37,0x30,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}

// string = "TM5-700"

string var_value = profinet_read_input_string(148,32)

// byte[] = {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x54,0x4D,0x35,0x2D,0x37,0x30,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}

// string = "01060112"

string var_value = profinet_read_input_string(148,32)

// byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,
            0xE4,0xBA,0xBA,0x31,0x32,0x33,0x34,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}

// string = "abcd 達明機器人 1234"

string var_value = profinet_read_input_string(148,10)

// byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6,0x98,0x8E}

// string = "abcd 達明"

string var_value = profinet_read_input_string(148,8)

// byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6}

// string = "abcd 達💎"

```

12.5 profinet_read_input_bit()

Read the input table content and retrieve the nth bit value of the data byte.

Syntax 1

```

byte profinet_read_input_bit(
    int,
    int
)

```

Parameters

int	Starting address
int	The n th bit value in the data byte

Return

byte	Return data in byte.
	Return 1 for bit value == 1.
	Return 0 for bit value == 0.

Note

```
byte var_value = profinet_read_input_bit(148,0)
```

```
// 0x30    get bit: "0"
```

```
// 0
```

```
byte var_value = profinet_read_input_bit(148,5)
```

```
// 0x30    get bit: "5"
```

```
// 1
```

Syntax 2

```
byte profinet_read_input_bit(  
    string,  
    int  
)
```

Parameters

`string` Item name
`int` The nth bit value

Return

`byte` Return data in byte.
Return 1 for bit value == 1.
Return 0 for bit value == 0.
*Data in bit will return in byte.

Note

```
byte var_b = profinet_read_input_bit("Register_Bit",0)
```

```
// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total
```

```
// 1  get bit: "0"
```

```
var_b = profinet_read_input_bit("Register_Bit",17)
```

```
// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total
```

```
// 0  get bit: "17"
```

Syntax 3

```
byte[] profinet_read_input_bit(  
    int,  
    int,  
    int  
)
```

Parameters

`int` Starting address
`int` Starting bit
`int` The amount of bit to read

Return

`byte[]` Return data in `byte[]`.
Return 1 for bit value == 1.
Return 0 for bit value == 0.
*Data in bit will return in byte such as `bit[0]` for `byte[0]` and `bit[1]` for `byte[1]`.

Note

```
byte[] var_ba = profinet_read_input_bit(148,0,20)

// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total

// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1}

var_ba = profinet_read_input_bit(148,12,8)

// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total

// byte[] = {1,1,0,1,0,0,1,1}
```

Syntax 4

```
byte[] profinet_read_input_bit(
    string,
    int,
    int
)
```

Parameters

`string` Item name
`int` Starting bit
`int` The amount of bit to read

Return

`byte[]` Return data in `byte[]`.
Return 1 for bit value == 1.
Return 0 for bit value == 0.
*Data in bit will return in byte such as `bit[0]` for `byte[0]` and `bit[1]` for `byte[1]`.

Note

```
byte[] var_ba = profinet_read_input_bit("Register_Bit",0,20)

// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total

// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1}

var_ba = profinet_read_input_bit("Register_Bit",12,8)

// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total
```

```
// byte[] = {1,1,0,1,0,0,1,1}
```

12.6 profinet_read_output()

Read the output table content.

Syntax 1

```
byte[] profinet_read_output(  
    int,  
    int  
)
```

Parameters

`int` Starting address
`int` The address length to read

Return

`byte[]` Return data in a byte array.

Note

```
byte[] var_value = profinet_read_output(540, 16)  
// {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
```

Syntax 2

```
byte profinet_read_output(  
    int  
)
```

Parameters

`int` Starting address

Return

`byte` Return data in byte

Note

```
byte var_value = profinet_read_output(540)  
// 0x30
```

Syntax 3

```
? profinet_read_output(  
    string,  
    int,  
    int  
)
```

Parameters

`string` Item name
`int` The starting shifted address of the item
`int` The amount of the addresses to read

Return

? The data type returned by the item defined in the communication data table.
* Data type includes `byte`, `byte[]`, `int`, `int[]`, `float`, `float[]`, `string`

Syntax 4

```
? profinet_read_output(  
    string,  
    int,  
)
```

Parameters

`string` Item name
`int` The starting shifted address of the item

Return

? The data type returned by the item defined in the communication data table.
* Data type includes `byte`, `byte[]`, `int`, `int[]`, `float`, `float[]`, `string`

Note

- * Same as syntax 3. Read to the end of the item by default.
- * Reading data based on the configuration file Little Endian (DCBA) or Big Endian (ABCD).

Syntax 5

```
? profinet_read_output(  
    string  
)
```

Parameter

`string` Item name

Return

? The data type returned by the item defined in the communication data table.
* Data type includes `byte`, `byte[]`, `int`, `int[]`, `float`, `float[]`, `string`

Note

- * Same as syntax 3. Fill 0 as the starting shifted address of the item and read to the end of the item by default.
- * Reading data based on the configuration file Little Endian (DCBA) or Big Endian (ABCD).

```
byte var_b = profinet_read_output("ManualAuto",0,1)
```

```

// 0x02

var_b = profinet_read_output("ManualAuto",0)

// 0x02

var_b = profinet_read_output("ManualAuto")

// 0x02


byte[] var_ba = profinet_read_output("Error_Code",0,4)

// {0x00,0x04,0x80,0x0C}

var_ba = profinet_read_output("Error_Code",0,2)

// {0x00,0x04}

var_ba = profinet_read_output("Error_Code")

// {0x00,0x04,0x80,0x0C}


int var_i = profinet_read_output("Current_Time_YY")

// byte[] = {0x00,0x00,0x07,0xE4} (Little Endian)    to int
// int = 0x000007E4 (Little Endian)
// int = 2020


int[] var_ia = profinet_read_output("Register_Int",0,12)

// byte[] = {0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x02,0x00,0x00,0x00,0x03} (Little Endian)    to int[]
// int[] = { 0x00000001,0x00000002,0x00000003} (Little Endian)
// int[] = {1,2,3}

var_ia = profinet_read_output("Register_Int",12)

// byte[] = {0x00,0x00,0x00,0x04,0x00,0x00,0x00,0x05,0x00,0x00,0x00,0x06,0x00,0x00,0x00,0x07,
//            0x00,0x00,0x00,0x08,0x00,0x00,0x00,0x09,0x00,0x00,0x00,0x0A,0x00,0x00,0x00,0x0B,
//            0x00,0x00,0x00,0x0C,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
//            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
//            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
//            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
//            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)    to int[]
// int[] = {0x00000004,0x00000005,0x00000006,0x00000007,0x00000008,0x00000009,
//            0x0000000A,0x0000000B,0x0000000C,0x00000000,0x00000000,0x00000000,

```

[illegible]

Syntax 1

```
int[] profinet_read_output_int(  
    int,  
    int,  
    int  
)
```

Parameters

int	Starting address
int	The address amount to read
int	The conversion of the read data to an int array based on Little Endian (DCBA) or Big Endian (ABCD).
0	Little-Endian
1	Big-Endian
2	Based on the configuration file.

Return

int[] Return data in an integer array.

Syntax 2

```
int[] profinet_read_output_int(  
    int,  
    int  
)
```

Note

Same as Syntax 1 with the parameter of the conversion of the read data defaults to **2**.

* Convert the read data to an int array based on Little Endian (DCBA) or Big Endian (ABCD).

```
int[] var_value = profinet_read_output_int(556, 12, 0)  
  
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]  
  
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)  
  
    // int[] = {32767,99999,-32768}  
  
int[] var_value = profinet_read_output_int(556, 11, 0)  
  
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF} (Little Endian)    to int[]  
  
    // int[] = {0x00007FFF,0x0001869F,0x00FF8000} (Little Endian)  
  
    // int[] = {32767,99999,16744448}  
  
int[] var_value = profinet_read_output_int(556, 10, 0)  
  
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80} (Little Endian)    to int[]  
  
    // int[] = {0x00007FFF,0x0001869F,0x00008000} (Little Endian)
```

```

// int[] = {32767,99999,32768}

int[] var_value = profinet_read_output_int(556, 12, 1)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]

// int[] = {0xFF7F0000,0x9F860100,0x0080FFFF} (Big Endian)

// int[] = {-8454144,-1618607872,8454143}

int[] var_value = profinet_read_output_int(556, 12, 2)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]

// int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)

// int[] = {32767,99999,-32768}

int[] var_value = profinet_read_output_int(556, 12)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]

// int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)

// int[] = {32767,99999,-32768}

```

Syntax 3

```

int profinet_read_output_int (
    int
)

```

Parameters

int Starting address
 * Convert the read data to an integer based on Little Endian (DCBA) or Big Endian (ABCD).

Return

int Return data in integer

Note

```

int var_value = profinet_read_output_int(556)

// byte[] = {0xE4,0x07,0x00,0x00} (Little Endian)    to int

// int = 0x000007E4 (Little Endian)

// int = 2020

int var_value = profinet_read_output_int(556)

// byte[] = {0x00,0x00,0x07,0xE4} (Big Endian)    to int

// int = 0x000007E4 (Big Endian)

// int = 2020

```

12.8 profinet_read_output_float()

Read the output table content and convert the data to the 32-bit floating-point number.

Syntax 1

```
float[] profinet_read_output_float(  
    int,  
    int,  
    int  
)
```

Parameters

- | | |
|-----|--|
| int | Starting address |
| int | The address amount to read |
| int | The conversion of the read data to a float array based on Little Endian (DCBA) or Big Endian (ABCD). |
| 0 | Little-Endian |
| 1 | Big-Endian |
| 2 | Based on the configuration file. |

Return

float[] Return data in a floating-point number array.

Syntax 2

```
float[] profinet_read_output_float(  
    int,  
    int  
)
```

Note

Same as Syntax 1 with the parameter of the conversion of the read data defaults to **2**.

* Convert the read data to a float array based on Little Endian (DCBA) or Big Endian (ABCD).

```
float[] var_value = profinet_read_output_float(676, 12, 0)
```

```
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian) to float[]
```

```
// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)
```

```
// float[] = {1.0,2.0,3.0}
```

```
float[] var_value = profinet_read_output_float(676, 11, 0)
```

```
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40} (Little Endian) to float[]
```

```
// float[] = {0x3F800000,0x40000000,0x00400000} (Little Endian)
```

```

// float[] = {1.0,2.0,5.877472E-39}

float[] var_value = profinet_read_output_float(676, 10, 0)

// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00} (Little Endian)  to float[]

// float[] = {0x3F800000,0x40000000,0x00000000} (Little Endian)

// float[] = {1.0,2.0,0.0}

float[] var_value = profinet_read_output_float(676, 12, 1)

// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)  to float[]

// float[] = {0x0000803F,0x00000040,0x00004040} (Big Endian)

// float[] = {4.600603E-41,8.96831E-44,2.304856E-41}

float[] var_value = profinet_read_output_float(676, 12, 2)

// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)  to float[]

// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)

// float[] = {1.0,2.0,3.0}

float[] var_value = profinet_read_output_float(676, 12)

// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)  to float[]

// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)

// float[] = {1.0,2.0,3.0}

```

Syntax 3

```

float profinet_read_output_float(
    int
)

```

Parameters

int Starting address

* Convert the read data to a float point number based on Little Endian (DCBA) or Big Endian (ABCD).

Return

float Return data in floating-point

Note

```

float var_value = profinet_read_output_float(676)

// byte[] = {0x00,0x00,0x80,0x3F} (Little Endian) to float

// float = 0x3F800000 (Little Endian)

// float = 1.0

float var_value = profinet_read_output_float(676)

```

```

// byte[] = {0x3F,0x80,0x00,0x00} (Big Endian) to float
// float = 0x3F800000 (Big Endian)
// float = 1.0

```

12.9 profinet_read_output_string()

Read the output table content and convert the data to the string encoded in UTF8.

Syntax 1

```

string profinet_read_output_string(
    int,
    int
)

```

Parameters

int Starting address
int The address amount to read

Return

string Return data in a UTF8 string (ending with 0x00 encountered).

Note

```

string var_value = profinet_read_output_string(556,16)
// byte[] = {0x54,0x4D,0x35,0x2D,0x37,0x30,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
// string = "TM5-700"

string var_value = profinet_read_output_string(540,32)
// byte[] = {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
//           0x54,0x4D,0x35,0x2D,0x37,0x30,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
// string = "01060112"

string var_value = profinet_read_output_string(540,32)
// byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,
//           0xE4,0xBA,0xBA,0x31,0x32,0x33,0x34,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
// string = "abcd 達明機器人 1234"

string var_value = profinet_read_output_string(540,10)
// byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6,0x98,0x8E}
// string = "abcd 達明"

string var_value = profinet_read_output_string(540,8)

```

```
// byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6}
```

```
// string = "abcd 達💎"
```

12.10 profinet_read_output_bit()

Read the output table content and retrieve the nth bit value of the data byte.

Syntax 1

```
byte profinet_read_output_bit(  
    int,  
    int  
)
```

Parameters

`int` Starting address
`int` The nth bit value in the data byte

Return

`byte` Return data in byte.
Return 1 for bit value == 1.
Return 0 for bit value == 0.

Note

```
byte var_value = profinet_read_output_bit(540,0)  
  
// 0x30    get bit: "0"  
  
// 0  
  
byte var_value = profinet_read_output_bit(540,5)  
  
// 0x30    get bit: "5"  
  
// 1
```

Syntax 2

```
byte profinet_read_output_bit(  
    string,  
    int  
)
```

Parameters

`string` Item name
`int` The nth bit value

Return

`byte` Return data in byte.

Return 1 for bit value == 1.
Return 0 for bit value == 0.
*Data in bit will return in byte.

Note

```
byte[] var_data = {57,184,12}

profinet_write_output(540,var_data,3)

// {00111001,10111000,00001100} (binary)

byte var_b = profinet_read_output_bit("Register_Bit",0)

// 1

var_b = profinet_read_output_bit("Register_Bit",17)

// 0
```

Syntax 3

```
byte[] profinet_read_output_bit(
    int,
    int,
    int
)
```

Parameters

int	Starting address
int	Starting bit
int	The amount of bit to read

Return

byte[] Return data in byte[].
Return 1 for bit value == 1.
Return 0 for bit value == 0.
*Data in bit will return in byte such as bit[0] for byte[0] and bit[1] for byte[1].

Note

```
byte[] var_data = {57,184,12}

profinet_write_output(540,var_data,3)

// {00111001,10111000,00001100} (binary)

byte[] var_ba = profinet_read_output_bit(540,0,20)

// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1}

var_ba = profinet_read_output_bit(540,12,8)

// byte[] = {1,1,0,1,0,0,1,1}
```

Syntax 4

```
byte[] profinet_read_output_bit(  
    string,  
    int,  
    int  
)
```

Parameters

`string` Item name
`int` Starting bit
`int` The amount of bit to read

Return

`byte[]` Return data in byte[].
Return 1 for bit value == 1.
Return 0 for bit value == 0.
*Data in bit will return in byte such as bit[0] for byte[0] and bit[1] for byte[1].

Note

```
byte[] var_data = {57,184,12}  
  
profinet_write_output(540,var_data,3)  
    // {00111001,10111000,00001100} (binary)  
  
byte[] var_ba = profinet_read_output_bit("Register_Bit",0,20)  
    // byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1}  
  
var_ba = profinet_read_output_bit("Register_Bit",12,8)  
    // byte[] = {1,1,0,1,0,0,1,1}
```

12.11 profinet_write_output()

Write data to the output table.

Syntax 1

```
bool profinet_write_output (  
    int,  
    ?,  
    int  
)
```

Paramaters

`int` Starting address

? The data to write
 * Available data types include `byte`, `byte[]`, `int`, `int[]`, `float`, `float[]`, `string`

`int` The maximum amount of the address to write

`> 0` Legitimate data length. Write by the amount of the address.

`<= 0` Illegitimate data length. Write by the complete length of the data to write.

Return

`bool` `True` Write successfully.

`False` Write unsuccessfully.

1. If the data to write is an empty string or an empty array
2. Unable to send and receive correctly.

Note

* Write data based on Little Endian (DCBA) or Big Endian (ABCD) in the configuration file.

Syntax 2

```
bool profinet_write_output (
    int,
    ?
)
```

Parameters

`int` Starting address

? The data to write
 * Available data types include `byte`, `byte[]`, `int`, `int[]`, `float`, `float[]`, `string`

Return

`bool` `True` Write successfully.

`False` Write unsuccessfully.

1. If the data to write is an empty string or an empty array
2. Unable to send and receive correctly.

Note

Same as syntax 1. Write with the full length of the data to write by default.

* Write data based on Little Endian (DCBA) or Big Endian (ABCD) in the configuration file.

Syntax 3

```
bool profinet_write_output (
    int,
    ?,
    int,
    int
)
```

Parameters

`int` Starting address

? The data to write
 * Available data types include **byte**, **byte[]**, **int**, **int[]**, **float**, **float[]**, **string**

int Starting address of the data to write

int The amount of the address to write

Return

bool **True** Write successfully.
 False Write unsuccessfully.

1. If the data to write is an empty string or an empty array
2. Unable to send and receive correctly.

Note

* Write data based on Little Endian (DCBA) or Big Endian (ABCD) in the configuration file.

```
byte var_data = 255

profinet_write_output(540,var_data,1)

byte var_b = profinet_read_output(540)

// 0xFF

byte[] var_data = {1,127,255}

profinet_write_output(540,var_data,3)

byte[] var_ba = profinet_read_output(540,3)

// {0x01,0x7F,0xFF}

profinet_write_output(540,var_data,2)

var_ba = profinet_read_output(540,3)

// {0x01,0x7F,0x00}

profinet_write_output(540,var_data,-1)

var_ba = profinet_read_output(540,3)

// {0x00,0x7F,0xFF}

int var_data = 32767

profinet_write_output(556,var_data,4)

int var_i = profinet_read_output_int(556)

// byte[] = {0xFF,0x7F,0x00,0x00} (Little Endian)      to int

// int = 0x00007FFF (Little Endian)

// int = 32767
```

```
profinet_write_output(556,var_data,1)
```

```
var_i = profinet_read_output_int(556)
```

```
// byte[] = {0xFF,0x00,0x00,0x00} (Little Endian)    to int
```

```
// int = 0x000000FF (Little Endian)
```

```
// int = 255
```

```
int[] var_data = {32767,99999,-32768}
```

```
profinet_write_output(556,var_data,12)
```

```
int[] var_ia = profinet_read_output_int(556,12)
```

```
// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]
```

```
// int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)
```

```
// int[] = {32767,99999,-32768}
```

```
profinet_write_output(556,var_data,3)
```

```
var_ia = profinet_read_output_int(556,12)
```

```
// byte[] = {0xFF,0x7F,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)    to int[]
```

```
// int[] = {0x00007FFF,0x00000000,0x00000000} (Little Endian)
```

```
// int[] = {32767,0,0}
```

```
profinet_write_output(556,var_data,11)
```

```
var_ia = profinet_read_output_int(556,12)
```

```
// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0x00} (Little Endian)    to int[]
```

```
// int[] = {0x00007FFF,0x0001869F,0x00FF8000} (Little Endian)
```

```
// int[] = {32767,99999,16744448}
```

```
profinet_write_output(556,var_data,4,4)
```

```
var_ia = profinet_read_output_int(556,12)
```

```
// byte[] = {0x9F,0x86,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)    to int[]
```

```
// int[] = {0x0001869F,0x00000000,0x00000000} (Little Endian)
```

```
// int[] = {99999,0,0}
```

```
float var_data = -10.0
```

```
profinet_write_output(676,var_data,4)
```

```
float var_f = profinet_read_output_float(676)
```

```
// byte[] = {0x00,0x00,0x20,0xC1} (Little Endian)    to float
```

```

        // float = 0xC1200000 (Little Endian)

        // float = -10.0

    profinet_write_output(676,var_data,1)
    var_f = profinet_read_output_float(676)

        // byte[] = {0x00,0x00,0x00,0x00} (Little Endian)    to float

        // float = 0x00000000 (Little Endian)

        // float = 0

    float[] var_data = {-10.0,3.3,123.45}

    profinet_write_output(676,var_data,12)
    float[] var_fa = profinet_read_output_float(676,12)

        // byte[] = {0x00,0x00,0x20,0xC1,0x33,0x33,0x53,0x40,0x66,0xE6,0xF6,0x42} (Little Endian)    to float[]

        // float[] = {0xC1200000,0x40533333,0x42F6E666} (Little Endian)

        // float[] = {-10,3.3,123.45}

    profinet_write_output(676,var_data,3)
    var_fa = profinet_read_output_float(676,12)

        // byte[] = {0x00,0x00,0x20,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)    to float[]

        // float[] = {0x00200000,0x00000000,0x00000000} (Little Endian)

        // float[] = {2.938736E-39,0,0}

    profinet_write_output(676,var_data,11)
    var_fa = profinet_read_output_float(676,12)

        // byte[] = {0x00,0x00,0x20,0xC1,0x33,0x33,0x53,0x40,0x66,0xE6,0xF6,0x00} (Little Endian)    to float[]

        // float[] = {0xC1200000,0x40533333,0x00F6E666} (Little Endian)

        // float[] = {-10,3.3,2.267418E-38}

    profinet_write_output(676,var_data,4,4)
    var_fa = profinet_read_output_float(676,12)

        // byte[] = {0x33,0x33,0x53,0x40,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)    to float[]

        // float[] = {0x40533333,0x00000000,0x00000000} (Little Endian)

        // float[] = {3.3,0,0}

    string var_data = "abcd 達明機器人 1234"

    profinet_write_output(540,var_data,32)

```

```

string var_s = profinet_read_output_string(540,32)

// byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,
            0xE4,0xBA,0xBA,0x31,0x32,0x33,0x34,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}

// string = "abcd 達明機器人 1234"

profinet_write_output(540,var_data,10)

var_s = profinet_read_output_string(540,32)

// byte[] = { 0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6,0x98,0x8E,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}

// string = "abcd 達明"

profinet_write_output(540,var_data,8)

var_s = profinet_read_output_string(540,32)

// byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}

// string = "abcd 達💎"

profinet_write_output(540,var_data,4,15)

var_s = profinet_read_output_string(540,15)

// byte[] = {0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA}

// string = "達明機器人"

```

Syntax 4

```

bool profinet_write_output (
    string,
    int,
    ?
    int,
    int
)

```

Parameters

string	Item name
int	The starting shifted address of the item
?	The data to write * Available data types include byte , byte[] , int , int[] , float , float[] , string
int	Starting address of the data to write
int	The amount of the address to write

Return

`bool` `True` Write successfully.
 `False` Write unsuccessfully.

1. If the data to write is an empty string or an empty array
2. Unable to send and receive correctly.

Note

* Write data based on Little Endian (DCBA) or Big Endian (ABCD) in the configuration file.

Syntax 5

```
bool profinet_write_output(  
    string,  
    int,  
    ?  
    int  
)
```

Parameters

`string` Item name
`int` The starting shifted address of the item
`?` The data to write
 * Available data types include `byte`, `byte[]`, `int`, `int[]`, `float`, `float[]`, `string`
`int` Starting address of the data to write

Return

`bool` `True` Write successfully.
 `False` Write unsuccessfully.

1. If the data to write is an empty string or an empty array
2. Unable to send and receive correctly.

Note

Same as syntax 4. Write with the full length of the data to write by default.

* Write data based on Little Endian (DCBA) or Big Endian (ABCD) in the configuration file.

Syntax 6

```
bool profinet_write_output(  
    string,  
    int,  
    ?  
)
```

Parameters

`string` Item name
`int` The starting shifted address of the item
`?` The data to write
 * Available data types include `byte`, `byte[]`, `int`, `int[]`, `float`, `float[]`, `string`

Return

`bool` `True` Write successfully.
 `False` Write unsuccessfully.
 1. If the data to write is an empty string or an empty array
 2. Unable to send and receive correctly.

Note

Same as syntax 4. Fill 0 as the starting address to write and write with the full length of the data to write by default.

* Write data based on Little Endian (DCBA) or Big Endian (ABCD) in the configuration file.

Syntax 7

```
bool profinet_write_output (
    string,
    ?
)
```

Parameters

`string` Item name
`?` The data to write
 * Available data types include `byte`, `byte[]`, `int`, `int[]`, `float`, `float[]`, `string`

Return

`bool` `True` Write successfully.
 `False` Write unsuccessfully.
 1. If the data to write is an empty string or an empty array
 2. Unable to send and receive correctly.

Note

Same as syntax 4. Fill 0 as the starting shifted address and the starting address to write as well as write with the full length of the data to write by default.

* Write data based on Little Endian (DCBA) or Big Endian (ABCD) in the configuration file.

```
int[] var_data = {32767,99999,-32768}

profinet_write_output("Register_Int",0,var_data,0,12)

int[] var_ia = profinet_read_output_int(556,12)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]
```

```

        // int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)

        // int[] = {32767,99999,-32768}

    profinet_write_output("Register_Int",4,var_data,4,4)

    var_ia = profinet_read_output_int(556,12)

        // byte[] = {0x00,0x00,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x00,0x00,0x00} (Little Endian)    to int[]

        // int[] = {0x00000000,0x0001869F,0x00000000} (Little Endian)

        // int[] = {0,99999,0}

    profinet_write_output("Register_Int",4,var_data)

    var_ia = profinet_read_output_int(556,20)

        // byte[] = {0x00,0x00,0x00,0x00,0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF,
        //           0x00,0x00,0x00,0x00} (Little Endian)    to int[]

        // int[] = {0x00000000,0x00007FFF,0x0001869F,0xFFFF8000,0x00000000} (Little Endian)

        // int[] = {0,32767,99999,-32768,0}

    float[] var_data = {-10.0,3.3,123.45}

    profinet_write_output("Register_Float",0,var_data,0,12)

    float[] var_fa = profinet_read_output_float(676,12)

        // byte[] = {0x00,0x00,0x20,0xC1,0x33,0x33,0x53,0x40,0x66,0xE6,0xF6,0x42} (Little Endian)    to float[]

        // float[] = {0xC1200000,0x40533333,0x42F6E666} (Little Endian)

        // float[] = {-10,3.3,123.45}

    profinet_write_output("Register_Float",4,var_data,4,8)

    var_fa = profinet_read_output_float(676,12)

        // byte[] = {0x00,0x00,0x00,0x00,0x33,0x33,0x53,0x40,0x66,0xE6,0xF6,0x42} (Little Endian)    to float[]

        // float[] = {0x00000000,0x40533333,0x42F6E666} (Little Endian)

        // float[] = {0,3.3,123.45}

    profinet_write_output("Register_Float",8,var_data)

    var_fa = profinet_read_output_float(676,20)

        // byte[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x20,0xC1,0x33,0x33,0x53,0x40,
        //           0x66,0xE6,0xF6,0x42} (Little Endian)    to float[]

        // float[] = {0x00000000,0x00000000,0xC1200000,0x40533333,0x42F6E666} (Little Endian)

        // float[] = {0,0,-10,3.3,123.45}

```

12.12 profinet_write_output_bit()

Write content to the nth bit value of the data byte in the output table.

Syntax 1

```
bool profinet_write_output_bit(  
    int,  
    int,  
    int  
)
```

Parameters

int	Starting address
int	The n th bit value in the data byte
int	The data to write

Return

bool	True	Write successfully.	
	False	Write unsuccessfully.	1. Unable to send correctly and receive .

Note

```
byte var_data = 240  
  
profinet_write_output(540,var_data)  
  
byte var_value = profinet_read_output(540)  
  
    // 0xF0  
  
profinet_write_output_bit(540,1,1)  
  
byte var_value = profinet_read_output_bit(540,1)  
  
    // 0xF2    get bit: "1"  
  
    // 1  
  
profinet_write_output_bit(540,7,0)  
  
byte var_value = profinet_read_output_bit(540,7)  
  
    // 0x72    get bit: "7"  
  
    // 0
```

Syntax 2

```
bool profient_write_output_bit(  
    string,  
    int,
```

```
int
```

```
)
```

Parameter

`int` Item name

`int` the n^{th} bit value

`int` The data to write

*Data in bit will write in int.

Return

`bool` `True` Write successfully.

`False` Write unsuccessfully.

1. Unable to send correctly and receive .

Note

```
byte var_data = 240
```

```
profinet _write_output(540,var_data)
```

```
byte var_b = profinet _read_output(540)
```

```
// 0xF0
```

```
profinet _write_output_bit("Register_Bit",1,1)
```

```
var_b = profinet _read_output_bit(540,1)
```

```
// 0xF2 get bit: "1"
```

```
// 1
```

```
profinet _write_output_bit("Register_Bit",7,0)
```

```
var_b = profinet _read_output_bit(540,7)
```

```
// 0x72 get bit: "7"
```

```
// 0
```

Syntax 3

```
bool profinet_write_output_bit(
```

```
int,
```

```
int,
```

```
byte[],
```

```
int,
```

```
int
```

```
)
```

Parameters

`int` Starting address

`int` Starting bit

`byte[]` Data to write.

*Data in bit will write in byte such as bit[0] for byte[0] and bit[1] for byte[1].

`int` Starting bit to write data
`int` The amount of bit to write data

Return

`bool` `True` Write successfully.
`False` Write unsuccessfully. 1. Unable to send correctly and receive .

Note

Bit value = 1 for byte value >=1

Bit value = 0 for byte value ==0

Syntax 4

```
bool profinet_write_output_bit(  
    int,  
    int,  
    byte[],  
    int  
)
```

Parameters

`int` Starting address
`int` The nth bit value
`byte[]` The data to write
*Data in bit will return in byte such as bit[0] for byte[0] and bit[1] for byte[1].
`int` The starting bit of the data to write

Return

`bool` `True` Write successfully.
`False` Write unsuccessfully. 1. Unable to send correctly and receive .

Note

*Same as syntax 3. Write with the full length of the rest data to write.

Bit value = 1 for byte value >=1

Bit value = 0 for byte value ==0

Syntax 5

```
bool profinet_write_output_bit(  
    int,  
    int,  
    byte[]  
)
```

Parameters

`int` Starting address

`int` The nth bit value
`byte[]` The data to write
 *Data in bit will return in byte such as bit[0] for byte[0] and bit[1] for byte[1].

Return

`bool` `True` Write successfully.
 `False` Write unsuccessfully. 1. Unable to send correctly and receive .

Note

*Same as syntax 3. Fill 0 as the starting bit to write as well as write with the full length of the data to write by default.

Bit value = 1 for byte value >=1

Bit value = 0 for byte value ==0

```
byte[] var_data = {1,0,0,1,1,1,0,0,0,0,1,1,1,0,1,0,0,1,1}
```

```
profinet_write_output_bit(540,0,var_data,0,20)
```

```
byte[] var_ba = profinet_read_output (540,0,3)
```

```
// byte[] = {0x39,0xB8,0x0C}
```

```
var_ba = profinet_read_output_bit(540,0,20)
```

```
// byte[] = {1,0,0,1,1,1,0,0,0,0,1,1,1,0,1,0,0,1,1}
```

```
profinet_write_output_bit(540,3,var_data,5,10)
```

```
var_ba = profinet_read_output (540,0,3)
```

```
// byte[] = {0x08,0x0E,0x00}
```

```
var_ba = profinet_read_output_bit(540,0,20)
```

```
// byte[] = {0,0,0,1,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0}
```

Syntax 6

```
bool profinet_write_output_bit(  

  string,  

  int,  

  byte[],  

  int,  

  int  

  )
```

Parameters

`string` Item name
`int` Starting bit
`byte[]` The data to write
 *Data in bit will return in byte such as bit[0] for byte[0] and bit[1] for byte[1].
`int` The starting bit to write data

`int` The bit amount of the data to write

Return

`bool` `True` Write successfully.
 `False` Write unsuccessfully. 1. Unable to send correctly and receive .

Note

Bit value = 1 for byte value ≥ 1

Bit value = 0 for byte value ≤ 0

Syntax 7

```
bool profinet_write_output_bit(  
    string,  
    int,  
    byte[],  
    int  
)
```

Parameters

`string` Item name
`int` Starting bit
`byte[]` The data to write
 *Data in bit will return in byte such as bit[0] for byte[0] and bit[1] for byte[1].
`int` The starting bit to write data

Return

`bool` `True` Write successfully.
 `False` Write unsuccessfully. 1. Unable to send correctly and receive .

Note

*Same as syntax 6. Write with the full length of the rest data to write.

Bit value = 1 for byte value ≥ 1

Bit value = 0 for byte value ≤ 0

Syntax 8

```
bool profinet_write_output_bit(  
    string,  
    int,  
    byte[]  
)
```

Parameters

`string` Item name
`int` Starting bit
`byte[]` The data to write

*Data in bit will return in byte such as bit[0] for byte[0] and bit[1] for byte[1].

Return

<code>bool</code>	<code>True</code>	Write successfully.	
	<code>False</code>	Write unsuccessfully.	1. Unable to send correctly and receive .

Note

*Same as syntax 6. Fill 0 as the starting bit to write as well as write with the full length of the data to write by default.

Bit value = 1 for byte value >=1

Bit value = 0 for byte value ==0

```
byte[] var_data = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1}

profinet_write_output_bit("Register_Bit",0,var_data,0,20)

byte[] var_ba = profinet_read_output (540,3)

// byte[] = {0x39,0xB8,0x0C}

var_ba = profinet_read_output_bit(540,0,20)

// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1}

profinet_write_output_bit("Register_Bit",3,var_data,5,10)

var_ba = profinet_read_output (540,3)

// byte[] = {0x08,0x0E,0x00}

var_ba = profinet_read_output_bit(540,0,20)

// byte[] = {0,0,0,1,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0}
```

13. EtherNet/IP Functions

The robot communicates with external controllers via the EtherNet/IP communication protocol. In the mechanism of the EtherNet/IP communication protocol, the robot works as a EtherNet/IP IO device for external devices to read and write the robot data. Meanwhile, TMflow monitors the table of data receiving from external devices and the table of data sending to external devices with EtherNet/IP functions as well as changes the custom definition section in the table of data sending to external devices.

Communication Data Table

The data table is composed of the input data and the output data. Input Data Table is for external devices posting on the robot, and Output Data Table is for the robot sending to external devices. Both of the data tables come with System Definition Section and Custom Definition Section for data.

1. System Definition Section : Items and settings are defined by the robot, and the data contents are updated by the robot or external devices. The defined items are robot status relevant such as robot bases, project status, control box status, or input/output status relevant such as digital I/Os and analog I/Os. Users can use EtherNet/IP functions to read the input data table and the output data table in the system definition section.
2. Custom Definition Section : Items and settings are defined by users, and the data contents are updated by users or external devices. In the meantime of the project editing, users can use EtherNet/IP functions to read and write the output data table in the custom definition section or read input data table in the custom definition section as well as use the custom definition section as a data exchange register between the project and external devices.

Communication Data Table (at the robot's viewpoint)	Data Section	TMflow EtherNet/IP Function Permissions	External Device Permissions
Input Data Table	System Definition Section	Read	Write
	Custom Definition Section	Read	Write
Output Data Table	System Definition Section	Read	Read
	Custom	Read/Write	Read

	Definition Section		
--	-----------------------	--	--

13.1 *eip_read_input()*

Read the input table content.

Syntax 1

```
byte[] eip_read_input(
    int,
    int
)
```

Parameters

`int` Starting address
`int` The address amount to read

Return

`byte[]` Return data in a byte array.

Note

```
byte[] var_ba = eip_read_input(104,8)
// {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32}
```

Syntax 2

```
byte eip_read_input(
    int,
)
```

Parameters

`int` Starting address

Return

`byte` Return data in byte.

Note

```
byte var_b = eip_read_input(104)
// 0x30
```

Syntax 3

```
? eip_read_input(
    string,
    int,
```

```
int
```

```
)
```

Parameters

`string` Item name

`int` The starting shifted address of the item

`int` The amount of the addresses to read

Return

`?` The data type returned by the item definition in the communication data table.

* Data type includes `byte, byte[], int, int[], float, float[], string`

Syntax 4

```
? eip_read_input (
```

```
    string,
```

```
    int,
```

```
)
```

Parameters

`string` Item name

`int` The starting shifted address of the item

Return

`?` The data type returned by the item definition in the communication data table.

* Data type includes `byte, byte[], int, int[], float, float[], string`

Note

* Same as syntax 3. Read to the end of the item by default.

* Reading data based on the configuration file Little Endian (DCBA) or Big Endian (ABCD).

Syntax 5

```
? eip_read_input (
```

```
    string
```

```
)
```

Parameters

`string` Item name

Return

`?` The data type returned by the item definition in the communication data table.

* Data type includes `byte, byte[], int, int[], float, float[], string`

Note

* Same as syntax 3. Fill 0 as the starting shifted address of the item and read to the end of the item by

default.

* Reading data based on the configuration file Little Endian (DCBA) or Big Endian (ABCD).

```
byte var_b = eip_read_input("O2T_StickStatus",0,1)
```

```
// 0x02
```

```
var_b = eip_read_input("O2T_StickStatus",0)
```

```
// 0x02
```

```
var_b = eip_read_input("O2T_StickStatus")
```

```
// 0x02
```

```
byte[] var_ba = eip_read_input("O2T_CtrlBox_DO",0,2)
```

```
// {0x00,0x04}
```

```
var_ba = eip_read_input("O2T_CtrlBox_DO")
```

```
// {0x00,0x04}
```

```
int[] var_ia = eip_read_input("O2T_Register_Int",0,12)
```

```
// byte[] = {0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x02,0x00,0x00,0x00,0x03} (Little Endian)    to int[]
```

```
// int[] = {0x00000001,0x00000002,0x00000003} (Little Endian)
```

```
// int[] = {1,2,3}
```

```
var_ia = eip_read_input("O2T_Register_Int",12)
```

```
// byte[] = {0x00,0x00,0x00,0x04,0x00,0x00,0x00,0x05,0x00,0x00,0x00,0x06,0x00,0x00,0x00,0x07,  
            0x00,0x00,0x00,0x08,0x00,0x00,0x00,0x09,0x00,0x00,0x00,0x0A,0x00,0x00,0x00,0x0B,  
            0x00,0x00,0x00,0x0C,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)  
to int[]
```

```
// int[] = {0x00000004,0x00000005,0x00000006,0x00000007,0x00000008,0x00000009,  
            0x0000000A,0x0000000B,0x0000000C,0x00000000,0x00000000,0x00000000} (Little Endian)
```

```
// int[] = {4,5,6,7,8,9,10,11,12,0,0,0}
```

```
var_ia = eip_read_input("O2T_Register_Int")
```

```
// byte[] = {0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x02,0x00,0x00,0x00,0x03,0x00,0x00,0x00,0x04,  
            0x00,0x00,0x00,0x05,0x00,0x00,0x00,0x06,0x00,0x00,0x00,0x07,0x00,0x00,0x00,0x08,  
            0x00,0x00,0x00,0x09,0x00,0x00,0x00,0x0A,0x00,0x00,0x00,0x0B,0x00,0x00,0x00,0x0C,
```

```

        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)    to int[]

// int[] = {0x00000001,0x00000002,0x00000003,0x00000004,0x00000005,0x00000006,
        0x00000007,0x00000008,0x00000009,0x0000000A,0x0000000B,0x0000000C,
        0x00000000,0x00000000,0x00000000} (Little Endian)

// int[] = {1,2,3,4,5,6,7,8,9,10,11,12,0,0,0}

float[] var_fa = eip_read_input("O2T_Register_Float",4,12)

// byte[] = {0x3F,0x99,0x99,0x9A,0x3F,0xA6,0x66,0x66,0x40,0x06,0x66,0x66} (Big Endian)    to float[]

// float[] = {0x3F99999A,0x3FA66666,0x40066666} (Big Endian)

// float[] = {1.2,1.3,2.1}

var_fa = eip_read_input("O2T_Register_Float",12)

// byte[] = {0x40,0x06,0x66,0x66,0x40,0x0C,0xCC,0xCD,0x40,0x13,0x33,0x33,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Big Endian)
        to float[]

// float[] = {0x40066666,0x400CCCCD,0x40133333,0x00000000,0x00000000,0x00000000,
        0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000} (Big Endian)

// float[] = {2.1,2.2,2.3,0,0,0,0,0,0,0,0,0,0,0,0,0}

var_fa = eip_read_input("O2T_Register_Float")

// byte[] = {0x3F,0x8C,0xCC,0xCD,0x3F,0x99,0x99,0x9A,0x3F,0xA6,0x66,0x66,0x40,0x06,0x66,0x66,
        0x40,0x0C,0xCC,0xCD,0x40,0x13,0x33,0x33,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Big Endian)    to float[]

// float[] = {0x3F8CCCCD,0x3F99999A,0x3FA66666,0x40066666,0x400CCCCD,0x40133333,
        0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,
        0x00000000,0x00000000,0x00000000} (Big Endian)

// float[] = { 1.1,1.2,1.3,2.1,2.2,2.3,0,0,0,0,0,0,0,0,0,0}

```

13.2 eip_read_input_int()

Read the input table content and convert the data to the 32-bit integer.

Syntax 1

```
int[] eip_read_input_int(  
    int,  
    int,  
    int  
)
```

Parameters

int	Starting address
int	The address amount to read
int	The conversion of the read data to an int array based on Little Endian (DCBA) or Big Endian (ABCD).
0	Little-Endian
1	Big-Endian
2	Based on the configuration file.

Return

int[] Return data in an integer array.

Syntax 2

```
int[] eip_read_input_int(  
    int,  
    int  
)
```

Parameters

int	Starting address
int	The address amount to read

Note

Same as Syntax 1 with the parameter of the conversion of the read data defaults to **2**.

* Convert the read data to an int array based on Little Endian (DCBA) or Big Endian.

```
int[] var_ia = eip_read_input_int(112,12,0)  
  
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]  
  
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)  
  
    // int[] = {32767,99999,-32768}  
  
var_ia = eip_read_input_int(112,11,0)  
  
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF} (Little Endian)    to int[]
```

```

        // int[] = {0x00007FFF,0x0001869F,0x00FF8000} (Little Endian)

        // int[] = {32767,99999,16744448}

var_ia = eip_read_input_int(112,10,0)

        // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80} (Little Endian)    to int[]

        // int[] = {0x00007FFF,0x0001869F,0x00008000} (Little Endian)

        // int[] = {32767,99999,32768}


var_ia = eip_read_input_int(112,12,1)

        // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Big Endian) to int[]

        // int[] = {0xFF7F0000,0x9F860100,0x0080FFFF} (Big Endian)

        // int[] = {-8454144,-1618607872,8454143}


var_ia = eip_read_input_int(112,12,2)

        // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]

        // int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)

        // int[] = {32767,99999,-32768}


var_ia = eip_read_input_int(112,12)

        // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]

        // int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)

        // int[] = {32767,99999,-32768}

```

Syntax 3

```

int eip_read_input_int(
    int
)

```

Parameter

int Starting address
 * Convert the read data to an int array based on Little Endian (DCBA) or Big Endian (ABCD).

Return

int Return data in integer.

Note

```

int var_i = eip_read_input_int(112)

        // byte[] = {0xE4,0x07,0x00,0x00} (Little Endian)    to int

```

```

        // int = 0x000007E4 (Little Endian)

        // int = 2020

var_i = eip_read_input_int(112)

        // byte[] = {0x00,0x00,0x07,0xE4} (Big Endian)      to int

        // int = 0x000007E4 (Big Endian)

        // int = 2020

```

13.3 *eip_read_input_float()*

Read the input table content and convert the data to the 32-bit floating-point number.

Syntax 1

```

float[] eip_read_input_float (
    int,
    int,
    int
)

```

Parameter

int	Starting address
int	The address amount to read
int	The conversion of the read data to a float array based on Little Endian (DCBA) or Big Endian (ABCD).
0	Little-Endian
1	Big-Endian
2	Based on the configuration file.

Return

float[] Return data in a floating-point number array.

Syntax 2

```

float[] eip_read_input_float (
    int,
    int
)

```

Parameter

int	Starting address
int	The address amount to read

Note

Same as Syntax 1 with the parameter of the conversion of the read data defaults to **2**.

* Convert the read data to a float array based on Little Endian (DCBA) or Big Endian (ABCD).

```
float[] var_fa = eip_read_input_float(172,12,0)

// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)   to float[]
// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)
// float[] = {1.0,2.0,3.0}

var_fa = eip_read_input_float(172,11,0)

// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40} (Little Endian)   to float[]
// float[] = {0x3F800000,0x40000000,0x00400000} (Little Endian)
// float[] = {1.0,2.0,5.877472E-39}

var_fa = eip_read_input_float(172,10,0)

// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00} (Little Endian)   to float[]
// float[] = {0x3F800000,0x40000000,0x00000000} (Little Endian)
// float[] = {1.0,2.0,0.0}

var_fa = eip_read_input_float(172,12,1)

// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)   to float[]
// float[] = {0x0000803F,0x00000040,0x00004040} (Big Endian)
// float[] = {4.600603E-41,8.96831E-44,2.304856E-41}

var_fa = eip_read_input_float(172,12,2)

// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)   to float[]
// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)
// float[] = {1.0,2.0,3.0}

var_fa = eip_read_input_float(172,12)

// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)   to float[]
// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)
// float[] = {1.0,2.0,3.0}
```

Syntax 3

```
float eip_read_input_float(  
    int  
)
```

Parameter

`int` Starting address
* Convert the read data to a float array based on Little Endian (DCBA) or Big Endian (ABCD).

Return

`float` Return data in floating-point.

Note

```
float var_f = eip_read_input_float(172)  
  
    // byte[] = {0x00,0x00,0x80,0x3F} (Little Endian) to float  
  
    // float = 0x3F800000 (Little Endian)  
  
    // float = 1.0  
  
var_f = eip_read_input_float(172)  
  
    // byte[] = {0x3F,0x80,0x00,0x00} (Big Endian) to float  
  
    // float = 0x3F800000 (Big Endian)  
  
    // float = 1.0
```

13.4 eip_read_input_string()

Read the input table content and convert the data to the string encoded in UTF8.

Syntax 1

```
string eip_read_input_string(  
    int,  
    int  
)
```

Parameter

`int` Starting address
`int` The address amount to read

Return

`string` Return data in a UTF8 string (ending with 0x00 encountered).

Note

```
string var_s = eip_read_input_string(104,16)  
  
    // byte[] = {0x54,0x4D,0x35,0x2D,0x37,0x30,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
```

```

// string = "TM5-700"

var_s = eip_read_input_string(104,32)
// byte[] = {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
//           0x54,0x4D,0x35,0x2D,0x37,0x30,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
// string = "01060112"

var_s = eip_read_input_string(104,32)
// byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,
//           0xE4,0xBA,0xBA,0x31,0x32,0x33,0x34,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
// string = "abcd 達明機器人 1234"

var_s = eip_read_input_string(104,10)
// byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6,0x98,0x8E}
// string = "abcd 達明"

var_s = eip_read_input_string(104,8)
// byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6}
// string = "abcd 達💎"

```

13.5 eip_read_input_bit()

Read the input table content and retrieve the nth bit value of the data byte.

Syntax 1

```

byte eip_read_input_bit(
    int,
    int
)

```

Parameter

int	Starting address
int	The n th bit value in the data byte

Return

byte	Return data in byte.
	Return 1 for bit value == 1.
	Return 0 for bit value == 0.
	*Data in bit will return in byte.

Note

```
byte var_b = eip_read_input_bit(104,0)

// 0x30    get bit: "0"

// 0

var_b = eip_read_input_bit(104,5)

// 0x30    get bit: "5"

// 1
```

Syntax 2

```
byte eip_read_input_bit(
    string,
    int
)
```

Parameter

`string` Item name
`int` The nth bit value

Return

`byte` Return data in byte.
Return 1 for bit value == 1.
Return 0 for bit value == 0.
*Data in bit will return in byte.

Note

```
byte var_b = eip_read_input_bit("O2T_Register_Bit",0)

// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total

// 1    get bit: "0"

var_b = eip_read_input_bit("O2T_Register_Bit",17)

// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total

// 0    get bit: "17"
```

Syntax 3

```
byte[] eip_read_input_bit(
    int,
    int,
    int
)
```

Parameter

`int` Starting address
`int` Starting bit
`int` The amount of bit to read

Return

`byte[]` Return data in byte[].
Return 1 for bit value == 1.
Return 0 for bit value == 0.
*Data in bit will return in byte such as bit[0] for byte[0] and bit[1] for byte[1].

Note

```
byte[] var_ba = eip_read_input_bit(104,0,20)

// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total

// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1}

var_ba = eip_read_input_bit(104,12,8)

// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total

// byte[] = {1,1,0,1,0,0,1,1}
```

Syntax 4

```
byte[] eip_read_input_bit(  
    string,  
    int,  
    int  
)
```

Parameter

`string` Item name
`int` Starting bit
`int` he amount of bit to read

Return

`byte[]` Return data in byte[].
Return 1 for bit value == 1.
Return 0 for bit value == 0.
*Data in bit will return in byte such as bit[0] for byte[0] and bit[1] for byte[1].

Note

```
byte[] var_ba = eip_read_input_bit("O2T_Register_Bit",0,20)

// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total

// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1}

var_ba = eip_read_input_bit("O2T_Register_Bit",12,8)
```

```
// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total  
// byte[] = {1,1,0,1,0,0,1,1}
```

13.6 *eip_read_output()*

Read the output table content.

Syntax 1

```
byte[] eip_read_output(  
    int,  
    int  
)
```

Parameter

`int` Starting address
`int` The address length to read

Return

`byte[]` Return data in a byte array.

Note

```
byte[] var_ba = eip_read_output(300,8)  
// {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32}
```

Syntax 2

```
byte eip_read_output(  
    int  
)
```

Parameter

`int` Starting address

Return

`byte` Return data in byte.

Note

```
byte var_b = eip_read_output(300)  
// 0x30
```

Syntax 3

```
? eip_read_output(  
    int,  
    int  
)
```

```

    string,
    int,
    int
)

```

Parameter

string Item name
int The starting shifted address of the item
int The amount of the addresses to read

Return

? The data type returned by the item defined in the communication data table.
 * Data type includes **byte, byte[], int, int[], float, float[], string**

Syntax 4

```

? eip_read_output (
    string,
    int,
)

```

Parameter

string Item name
int The starting shifted address of the item

Return

? The data type returned by the item defined in the communication data table.
 * Data type includes **byte, byte[], int, int[], float, float[], string**

Note

- * Same as syntax 3. Read to the end of the item by default.
- * Reading data based on the configuration file Little Endian (DCBA) or Big Endian (ABCD).

Syntax 5

```

? eip_read_output (
    string
)

```

Parameter

string Item name

Return

? The data type returned by the item defined in the communication data table.
 * Data type includes **byte, byte[], int, int[], float, float[], string**

Note

- * Same as syntax 3. Fill 0 as the starting shifted address of the item and read to the end of the item by

default.

* Reading data based on the configuration file Little Endian (DCBA) or Big Endian (ABCD).

```
byte var_b = eip_read_output("ManualAuto",0,1)
```

```
// 0x02
```

```
var_b = eip_read_output("ManualAuto",0)
```

```
// 0x02
```

```
var_b = eip_read_output("ManualAuto")
```

```
// 0x02
```

```
byte[] var_ba = eip_read_output("Error_Code",0,4)
```

```
// {0x00,0x04,0x80,0x0C}
```

```
var_ba = eip_read_output("Error_Code",0,2)
```

```
// {0x00,0x04}
```

```
var_ba = eip_read_output("Error_Code")
```

```
// {0x00,0x04,0x80,0x0C}
```

```
int var_i = eip_read_output("Current_Time_Year")
```

```
// byte[] = {0x00,0x00,0x07,0xE4} (Little Endian)    to int
```

```
// int = 0x000007E4 (Little Endian)
```

```
// int = 2020
```

```
int[] var_ia = eip_read_output("T2O_Register_Int",0,12)
```

```
// byte[] = {0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x02,0x00,0x00,0x00,0x03} (Little Endian)    to int[]
```

```
// int[] = { 0x00000001,0x00000002,0x00000003} (Little Endian)
```

```
// int[] = {1,2,3}
```

```
var_ia = eip_read_output("T2O_Register_Int",12)
```

```
// byte[] = {0x00,0x00,0x00,0x04,0x00,0x00,0x00,0x05,0x00,0x00,0x00,0x06,0x00,0x00,0x00,0x07,
```

```
0x00,0x00,0x00,0x08,0x00,0x00,0x00,0x09,0x00,0x00,0x00,0x0A,0x00,0x00,0x00,0x0B,
```

```
0x00,0x00,0x00,0x0C,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)
```

```
to int[]
```

```
// int[] = {0x00000004,0x00000005,0x00000006,0x00000007,0x00000008,0x00000009,
```

```

        0x0000000A,0x0000000B,0x0000000C,0x00000000,0x00000000,0x00000000} (Little Endian)

// int[] = {4,5,6,7,8,9,10,11,12,0,0,0}

var_ia = eip_read_output("T2O_Register_Int")

// byte[] = {0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x02,0x00,0x00,0x00,0x03,0x00,0x00,0x00,0x04,
//           0x00,0x00,0x00,0x05,0x00,0x00,0x00,0x06,0x00,0x00,0x00,0x07,0x00,0x00,0x00,0x08,
//           0x00,0x00,0x00,0x09,0x00,0x00,0x00,0x0A,0x00,0x00,0x00,0x0B,0x00,0x00,0x00,0x0C,
//           0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)   to int[]

// int[] = {0x00000001,0x00000002,0x00000003,0x00000004,0x00000005,0x00000006,
//           0x00000007,0x00000008,0x00000009,0x0000000A,0x0000000B,0x0000000C,
//           0x00000000,0x00000000,0x00000000} (Little Endian)

// int[] = {1,2,3,4,5,6,7,8,9,10,11,12,0,0,0}

float var_f = eip_read_output("Current_TCP_Mass")

// byte[] = {0x40,0x40,0x00,0x00} (Big Endian)           to float
// float = 0x40400000 (Big Endian)
// float = 3.0

float[] var_fa = eip_read_output("Current_TCP_Value",4,12)

// byte[] = {0x3F,0x99,0x99,0x9A,0x3F,0xA6,0x66,0x66,0x40,0x06,0x66,0x66} (Big Endian)   to float[]
// float[] = {0x3F99999A,0x3FA66666,0x40066666} (Big Endian)
// float[] = {1.2,1.3,2.1}

var_fa = eip_read_output("Current_TCP_Value",12)

// byte[] = { 0x40,0x06,0x66,0x66,0x40,0x0C,0xCC,0xCD,0x40,0x13,0x33,0x33} (Big Endian)   to float[]
// float[] = {0x40066666,0x400CCCCD,0x40133333} (Big Endian)
// float[] = {2.1,2.2,2.3}

var_fa = eip_read_output("Current_TCP_Value")

// byte[] = {0x3F,0x8C,0xCC,0xCD,0x3F,0x99,0x99,0x9A,0x3F,0xA6,0x66,0x66,0x40,0x06,0x66,0x66,
//           0x40,0x0C,0xCC,0xCD,0x40,0x13,0x33,0x33} (Big Endian)   to float[]
// float[] = {0x3F8CCCCD,0x3F99999A,0x3FA66666,0x40066666,0x400CCCCD,0x40133333} (Big Endian)
// float[] = { 1.1,1.2,1.3,2.1,2.2,2.3}

```

```

string var_s = eip_read_output ("ControlBoxID",0,16)

    // byte[] = {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}

    // string = "01060112"

var_s = eip_read_output ("ControlBoxID",4,3)

    // byte[] = {0x30,0x31,0x31}

    // string = "011"

var_s = eip_read_output ("ControlBoxID")

    // byte[] = {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}

    // string = "01060112"

```

13.7 eip_read_output_int()

Read the output table content and convert the data to the 32-bit integer.

Syntax 1

```

int[] eip_read_output_int (
    int,
    int,
    int
)

```

Parameter

int	Starting address
int	The address amount to read
int	The conversion of the read data to an int array based on Little Endian (DCBA) or Big Endian (ABCD).
0	Little-Endian
1	Big-Endian
2	Based on the configuration file.

Return

int[] Return data in an integer array.

Syntax 2

```

int[] eip_read_output_int (
    int,
    int
)

```

Parameter

`int` Starting address
`int` The address amount to read

Note

Same as Syntax 1 with the parameter of the conversion of the read data defaults to **2**.

* Convert the read data to an int array based on Little Endian (DCBA) or Big Endian (ABCD).

```
int[] var_ia = eip_read_output_int(308,12,0)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]
// int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)
// int[] = {32767,99999,-32768}

var_ia = eip_read_output_int(308,11,0)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF} (Little Endian)    to int[]
// int[] = {0x00007FFF,0x0001869F,0x00FF8000} (Little Endian)
// int[] = {32767,99999,16744448}

var_ia = eip_read_output_int(308,10,0)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80} (Little Endian)    to int[]
// int[] = {0x00007FFF,0x0001869F,0x00008000} (Little Endian)
// int[] = {32767,99999,32768}

var_ia = eip_read_output_int(308,12,1)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]
// int[] = {0xFF7F0000,0x9F860100,0x0080FFFF} (Big Endian)
// int[] = {-8454144,-1618607872,8454143}

var_ia = eip_read_output_int(308,12,2)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]
// int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)
// int[] = {32767,99999,-32768}

var_ia = eip_read_output_int(308,12)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]
// int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)
// int[] = {32767,99999,-32768}
```

```
int eip_read_output_int(
    int
)
```

Parameter

`int` Starting address
 * Convert the read data to an integer based on Little Endian (DCBA) or Big Endian (ABCD).

Return

`int` Return data in integer

Note

```
int var_i = eip_read_output_int(308)
    // byte[] = {0xE4,0x07,0x00,0x00} (Little Endian)    to int
    // int = 0x000007E4 (Little Endian)
    // int = 2020

var_i = eip_read_output_int(308)
    // byte[] = {0x00,0x00,0x07,0xE4} (Big Endian)    to int
    // int = 0x000007E4 (Big Endian)
    // int = 2020
```

13.8 *eip_read_output_float()*

Read the output table content and convert the data to the 32-bit floating-point number.

Syntax 1

```
float[] eip_read_output_float(
    int,
    int,
    int
)
```

Parameter

`int` Starting address
`int` The address amount to read
`int` The conversion of the read data to a float array based on Little Endian (DCBA) or Big Endian (ABCD).
 0 Little-Endian
 1 Big-Endian
 2 Based on the configuration file.

Return

`float[]` Return data in a floating-point number array.

Syntax 2

```
float[] eip_read_output_float(  
    int,  
    int  
)
```

Parameter

`int` Starting address
`int` The address amount to read

Note

Same as Syntax 1 with the parameter of the conversion of the read data defaults to **2**.

* Convert the read data to a float array based on Little Endian (DCBA) or Big Endian (ABCD).

```
float[] var_fa = eip_read_output_float(368,12,0)
```

```
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian) to float[]
```

```
// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)
```

```
// float[] = {1.0,2.0,3.0}
```

```
var_fa = eip_read_output_float(368,11,0)
```

```
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40} (Little Endian) to float[]
```

```
// float[] = {0x3F800000,0x40000000,0x00400000} (Little Endian)
```

```
// float[] = {1.0,2.0,5.877472E-39}
```

```
var_fa = eip_read_output_float(368,10,0)
```

```
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00} (Little Endian) to float[]
```

```
// float[] = {0x3F800000,0x40000000,0x00000000} (Little Endian)
```

```
// float[] = {1.0,2.0,0.0}
```

```
var_fa = eip_read_output_float(368,12,1)
```

```
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian) to float[]
```

```
// float[] = {0x0000803F,0x00000040,0x00004040} (Big Endian)
```

```
// float[] = {4.600603E-41,8.96831E-44,2.304856E-41}
```

```
var_fa = eip_read_output_float(368,12,2)
```

```
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)    to float[]
// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)
// float[] = {1.0,2.0,3.0}
```

```
var_fa = eip_read_output_float(368,12)
```

```
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)    to float[]
// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)
// float[] = {1.0,2.0,3.0}
```

Syntax 3

```
float eip_read_output_float(
    int
)
```

Parameter

int Starting address
 * Convert the read data to a floating point number based on Little Endian (DCBA) or Big

Return

float Return data in floating-point

Note

```
float var_f = eip_read_output_float(368)

// byte[] = {0x00,0x00,0x80,0x3F} (Little Endian) to float
// float = 0x3F800000 (Little Endian)
// float = 1.0

var_f = eip_read_output_float(368)

// byte[] = {0x3F,0x80,0x00,0x00} (Big Endian) to float
// float = 0x3F800000 (Big Endian)
// float = 1.0
```

13.9 eip_read_output_string()

Read the output table content and convert the data to the string encoded in UTF8.

Syntax 1

```
string eip_read_output_string(
```

```
int,  
int
```

```
)
```

Parameter

```
int    Starting address  
int    The address amount to read
```

Return

```
string Return data in a UTF8 string (ending with 0x00 encountered).
```

Note

```
string var_s = eip_read_output_string(300,16)  
    // byte[] = {0x54,0x4D,0x35,0x2D,0x37,0x30,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}  
    // string = "TM5-700"  
  
var_s = eip_read_output_string(300,32)  
    // byte[] = {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  
                0x54,0x4D,0x35,0x2D,0x37,0x30,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}  
    // string = "01060112"  
  
var_s = eip_read_output_string(300,32)  
    // byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,  
                0xE4,0xBA,0xBA,0x31,0x32,0x33,0x34,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}  
    // string = "abcd 達明機器人 1234"  
  
var_s = eip_read_output_string(300,10)  
    // byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6,0x98,0x8E}  
    // string = "abcd 達明"  
  
var_s = eip_read_output_string(300,8)  
    // byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6}  
    // string = "abcd 達💎"
```

13.10 eip_read_output_bit()

Read the output table content and retrieve the nth bit value of the data byte.

Syntax 1

```
byte eip_read_output_bit(
    int,
    int
)
```

Parameter

`int` Starting address
`int` The nth bit value in the data byte

Return

`byte` Return data in byte.
 Return 1 for bit value == 1.
 Return 0 for bit value == 0.
 *Data in bit will return in byte.

Note

```
byte var_b = eip_read_output_bit(300,0)

// 0x30    get bit: "0"

// 0

var_b = eip_read_output_bit(300,5)

// 0x30    get bit: "5"

// 1
```

Syntax 2

```
byte eip_read_output_bit(
    string,
    int
)
```

Parameter

`string` Item name
`int` The nth bit value

Return

`byte` Return data in byte.
 Return 1 for bit value == 1.
 Return 0 for bit value == 0.
 *Data in bit will return in byte.

Note

```
byte[] var_data = {57,184,12}

eip_write_output(300,var_data,3)

// {00111001,10111000,00001100} (binary)
```

```

byte var_b = eip_read_output_bit("T2O_Register_Bit",0)

// 1

var_b = eip_read_output_bit("T2O_Register_Bit",17)

// 0

```

Syntax 3

```

byte[] eip_read_output_bit (
    int,
    int,
    int
)

```

Parameter

int	Starting address
int	Starting bit
int	The amount of bit to read

Return

byte[] Return data in byte[].

Return 1 for bit value == 1.

Return 0 for bit value == 0.

*Data in bit will return in byte such as bit[0] for byte[0] and bit[1] for byte[1].

Note

```

byte[] var_data = {57,184,12}

eip_write_output(300,var_data,3)

// {00111001,10111000,00001100} (binary)

byte[] var_ba = eip_read_output_bit(300,0,20)

// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1}

var_ba = eip_read_output_bit(300,12,8)

// byte[] = {1,1,0,1,0,0,1,1}

```

Syntax 4

```

byte[] eip_read_output_bit (
    string,
    int,
    int
)

```

Parameter

```

string Item name
int Starting bit
int The amount of bit to read

```

Return

```

byte[] Return data in byte[].
Return 1 for bit value == 1.
Return 0 for bit value == 0.
*Data in bit will return in byte such as bit[0] for byte[0] and bit[1] for byte[1].

```

Note

```

byte[] var_data = {57,184,12}

eip_write_output(300,var_data,3)

// {00111001,10111000,00001100} (binary)

byte[] var_ba = eip_read_output_bit("T2O_Register_Bit",0,20)

// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1}

var_ba = eip_read_output_bit("T2O_Register_Bit",12,8)

// byte[] = {1,1,0,1,0,0,1,1}

```

13.11 eip_write_output()

Write data to the output table.

Syntax 1

```

bool eip_write_output (
    int,
    ?,
    int
)

```

Parameter

```

int Starting address
? The data to write
  * Available data types include byte, byte[], int, int[], float, float[], string
int The maximum amount of the address to write
  > 0 Legitimate data length. Write by the amount of the address.
  <= 0 Illegitimate data length. Write by the complete length of the data to write.

```

Return

```

bool True Write successfully.
False Write unsuccessfully.

```

1. If the data to write is an empty string or an empty array
2. Unable to send and receive correctly.

Note

- * Write data based on Little Endian (DCBA) or Big Endian (ABCD) in the configuration file.

Syntax 2

```
bool eip_write_output(
    int,
    ?
)
```

Parameter

<code>int</code>	Starting address
<code>?</code>	The data to write
* Available data types include <code>byte</code> , <code>byte[]</code> , <code>int</code> , <code>int[]</code> , <code>float</code> , <code>float[]</code> , <code>string</code>	

Return

<code>bool</code>	<code>True</code>	Write successfully.
	<code>False</code>	Write unsuccessfully.

1. If the data to write is an empty string or an empty array
2. Unable to send and receive correctly.

Note

Same as syntax 1. Write with the full length of the data to write by default.

- * Write data based on Little Endian (DCBA) or Big Endian (ABCD) in the configuration file.

Syntax 3

```
bool eip_write_output(
    int,
    ?,
    int,
    int
)
```

Parameter

<code>int</code>	Starting address
<code>?</code>	The data to write
* Available data types include <code>byte</code> , <code>byte[]</code> , <code>int</code> , <code>int[]</code> , <code>float</code> , <code>float[]</code> , <code>string</code>	
<code>int</code>	Starting address of the data to write
<code>int</code>	The amount of the address to write

Return

<code>bool</code>	<code>True</code>	Write successfully.
-------------------	-------------------	---------------------

False Write unsuccessfully.

1. If the data to write is an empty string or an empty array
2. Unable to send and receive correctly.

Note

** Write data based on Little Endian (DCBA) or Big Endian (ABCD) in the configuration file.

```
byte var_data = 255

eip_write_output(300,var_data,1)

byte var_b = eip_read_output(300)

// 0xFF

byte[] var_data = {1,127,255}

eip_write_output(300,var_data,3)

byte[] var_ba = eip_read_output(300,3)

// {0x01,0x7F,0xFF}

eip_write_output(300,var_data,2)

var_ba = eip_read_output(300,3)

// {0x01,0x7F,0x00}

eip_write_output(300,var_data,-1)

var_ba = eip_read_output(300,3)

// {0x00,0x7F,0xFF}

int var_data = 32767

eip_write_output(308,var_data,4)

int var_i = eip_read_output_int(308)

// byte[] = {0xFF,0x7F,0x00,0x00} (Little Endian)    to int
// int = 0x00007FFF (Little Endian)
// int = 32767

eip_write_output(308,var_data,1)

var_i = eip_read_output_int(308)

// byte[] = {0xFF,0x00,0x00,0x00} (Little Endian)    to int
// int = 0x000000FF (Little Endian)
// int = 255
```

```

int[] var_data = {32767,99999,-32768}

eip_write_output(308,var_data,12)

int[] var_ia = eip_read_output_int(308,12)

    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]

    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)

    // int[] = {32767,99999,-32768}

eip_write_output(308,var_data,3)

var_ia = eip_read_output_int(308,12)

    // byte[] = {0xFF,0x7F,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)    to int[]

    // int[] = {0x00007FFF,0x00000000,0x00000000} (Little Endian)

    // int[] = {32767,0,0}

eip_write_output(308,var_data,11)

var_ia = eip_read_output_int(308,12)

    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0x00} (Little Endian)    to int[]

    // int[] = {0x00007FFF,0x0001869F,0x00FF8000} (Little Endian)

    // int[] = {32767,99999,16744448}

eip_write_output(308,var_data,4,4)

var_ia = eip_read_output_int(308,12)

    // byte[] = {0x9F,0x86,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)    to int[]

    // int[] = {0x0001869F,0x00000000,0x00000000} (Little Endian)

    // int[] = {99999,0,0}


float var_data = -10.0

eip_write_output(368,var_data,4)

float var_f = eip_read_output_float(368)

    // byte[] = {0x00,0x00,0x20,0xC1} (Little Endian)    to float

    // float = 0xC1200000 (Little Endian)

    // float = -10.0

eip_write_output(368,var_data,1)

var_f = eip_read_output_float(368)

    // byte[] = {0x00,0x00,0x00,0x00} (Little Endian)    to float

```

```

// float = 0x00000000 (Little Endian)

// float = 0

float[] var_data = {-10.0,3.3,123.45}

eip_write_output(368,var_data,12)

float[] var_fa = eip_read_output_float(368,12)

// byte[] = {0x00,0x00,0x20,0xC1,0x33,0x33,0x53,0x40,0x66,0xE6,0xF6,0x42} (Little Endian)    to float[]

// float[] = {0xC1200000,0x40533333,0x42F6E666} (Little Endian)

// float[] = {-10,3.3,123.45}

eip_write_output(368,var_data,3)

var_fa = eip_read_output_float(368,12)

// byte[] = {0x00,0x00,0x20,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)    to float[]

// float[] = {0x00200000,0x00000000,0x00000000} (Little Endian)

// float[] = {2.938736E-39,0,0}

eip_write_output(368,var_data,11)

var_fa = eip_read_output_float(368,12)

// byte[] = {0x00,0x00,0x20,0xC1,0x33,0x33,0x53,0x40,0x66,0xE6,0xF6,0x00} (Little Endian)    to float[]

// float[] = {0xC1200000,0x40533333,0x00F6E666} (Little Endian)

// float[] = {-10,3.3,2.267418E-38}

eip_write_output(368,var_data,4,4)

var_fa = eip_read_output_float(368,12)

// byte[] = {0x33,0x33,0x53,0x40,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)    to float[]

// float[] = {0x40533333,0x00000000,0x00000000} (Little Endian)

// float[] = {3.3,0,0}

string var_data = "abcd 達明機器人 1234"

eip_write_output(300,var_data,32)

string var_s = eip_read_output_string(300,32)

// byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,
//           0xE4,0xBA,0xBA,0x31,0x32,0x33,0x34,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}

// string = "abcd 達明機器人 1234"

eip_write_output(300,var_data,10)

```

```

var_s = eip_read_output_string(300,32)

// byte[] = { 0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6,0x98,0x8E,0x00,0x00,0x00,0x00,0x00,0x00,
              0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}

// string = "abcd 達明"

eip_write_output(300,var_data,8)

var_s = eip_read_output_string(300,32)

// byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
              0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}

// string = "abcd 達💎"

eip_write_output(300,var_data,4,15)

var_s = eip_read_output_string(300,15)

// byte[] = {0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA}

// string = "達明機器人"

```

Syntax 4

```

bool eip_write_output (
    string,
    int,
    ?,
    int,
    int
)

```

Parameter

string	Item name
int	The starting shifted address of the item
?	The data to write
	* Available data types include byte , byte[] , int , int[] , float , float[] , string
int	Starting address of the data to write
int	The amount of the address to write

Return

bool	True	Write successfully.
	False	Write unsuccessfully.

1. If the data to write is an empty string or an empty array
2. Unable to send and receive correctly.

Note

** Write data based on Little Endian (DCBA) or Big Endian (ABCD) in the configuration file.

Syntax 5

```
bool eip_write_output(  
    string,  
    int,  
    ?  
    int  
)
```

Parameter

string	Item name
int	The starting shifted address of the item
?	The data to write * Available data types include byte , byte[] , int , int[] , float , float[] , string
int	Starting address of the data to write

Return

bool	True	Write successfully.
	False	Write unsuccessfully.

1. If the data to write is an empty string or an empty array
2. Unable to send and receive correctly.

Note

Same as syntax 4. Write with the full length of the data to write by default.

* Write data based on Little Endian (DCBA) or Big Endian (ABCD) in the configuration file.

Syntax 6

```
bool eip_write_output(  
    string,  
    int,  
    ?  
)
```

Parameter

string	Item name
int	The starting shifted address of the item
?	The data to write * Available data types include byte , byte[] , int , int[] , float , float[] , string

Return

bool	True	Write successfully.
	False	Write unsuccessfully.

1. If the data to write is an empty string or an empty array

2. Unable to send and receive correctly.

Note

Same as syntax 4. Fill 0 as the starting address to write and write with the full length of the data to write by default.

* Write data based on Little Endian (DCBA) or Big Endian (ABCD) in the configuration file.

Syntax 7

```
bool eip_write_output(  
    string,  
    ?  
)
```

Parameter

`string` Item name
`?` The data to write
* Available data types include `byte`, `byte[]`, `int`, `int[]`, `float`, `float[]`, `string`

Return

`bool` `True` Write successfully.
 `False` Write unsuccessfully.

1. If the data to write is an empty string or an empty array
2. Unable to send and receive correctly.

Note

Same as syntax 4. Fill 0 as the starting shifted address and the starting address to write as well as write with the full length of the data to write by default.

* Write data based on Little Endian (DCBA) or Big Endian (ABCD) in the configuration file.

```
int[] var_data = {32767,99999,-32768}  
  
eip_write_output("T2O_Register_Int",0,var_data,0,12)  
  
int[] var_ia = eip_read_output_int(308,12)  
  
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)     to int[]  
  
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)  
  
    // int[] = {32767,99999,-32768}  
  
eip_write_output("T2O_Register_Int",4,var_data,4,4)  
  
var_ia = eip_read_output_int(308,12)  
  
    // byte[] = {0x00,0x00,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x00,0x00,0x00} (Little Endian)     to int[]  
  
    // int[] = {0x00000000,0x0001869F,0x00000000} (Little Endian)
```

```

        // int[] = {0,99999,0}

eip_write_output("T2O_Register_Int",4,var_data)
var_ia = eip_read_output_int(308,20)

        // byte[] = {0x00,0x00,0x00,0x00,0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF,
        //           0x00,0x00,0x00,0x00} (Little Endian)      to int[]

        // int[] = {0x00000000,0x00007FFF,0x0001869F,0xFFFF8000,0x00000000} (Little Endian)
        // int[] = {0,32767,99999,-32768,0}

float[] var_data = {-10.0,3.3,123.45}

eip_write_output("T2O_Register_Float",0,var_data,0,12)
float[] var_fa = eip_read_output_float(368,12)

        // byte[] = {0x00,0x00,0x20,0xC1,0x33,0x33,0x53,0x40,0x66,0xE6,0xF6,0x42} (Little Endian)      to float[]
        // float[] = {0xC1200000,0x40533333,0x42F6E666} (Little Endian)
        // float[] = {-10,3.3,123.45}

eip_write_output("T2O_Register_Float",4,var_data,4,8)
var_fa = eip_read_output_float(368,12)

        // byte[] = {0x00,0x00,0x00,0x00,0x33,0x33,0x53,0x40,0x66,0xE6,0xF6,0x42} (Little Endian)      to float[]
        // float[] = {0x00000000,0x40533333,0x42F6E666} (Little Endian)
        // float[] = {0,3.3,123.45}

eip_write_output("T2O_Register_Float",8,var_data)
var_fa = eip_read_output_float(368,20)

        // byte[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x20,0xC1,0x33,0x33,0x53,0x40,
        //           0x66,0xE6,0xF6,0x42} (Little Endian)      to float[]
        // float[] = {0x00000000,0x00000000,0xC1200000,0x40533333,0x42F6E666} (Little Endian)
        // float[] = {0,0,-10,3.3,123.45}

```

13.12 eip_write_output_bit()

Write content to the nth bit value of the data byte in the output table.

Syntax 1

```

bool eip_write_output_bit(
    int,

```

```

    int,
    int
)

```

Parameter

int Starting address
int the n^{th} bit value
int The data to write
 *Data in bit will write in int.

Return

bool **True** Write successfully.
 False Write unsuccessfully. 1. Unable to send correctly and receive .

Note

```

byte var_data = 240

eip_write_output(300,var_data)

byte var_b = eip_read_output(300)

// 0xF0

eip_write_output_bit(300,1,1)

var_b = eip_read_output_bit(300,1)

// 0xF2    get bit: "1"

// 1

eip_write_output_bit(300,7,0)

var_b = eip_read_output_bit(300,7)

// 0x72    get bit: "7"

// 0

```

Syntax 2

```

bool eip_write_output_bit(
    string,
    int,
    int
)

```

Parameter

int Starting address
int The n^{th} bit value
int The data to write
 *Data in bit will write in int.

Return

<code>bool</code>	<code>True</code>	Write successfully.	
	<code>False</code>	Write unsuccessfully.	1. Unable to send correctly and receive .

Note

```
byte var_data = 240

eip_write_output(300,var_data)

byte var_b = eip_read_output(300)

// 0xF0

eip_write_output_bit("T2O_Register_Bit",1,1)

var_b = eip_read_output_bit(300,1)

// 0xF2    get bit: "1"

// 1

eip_write_output_bit("T2O_Register_Bit",7,0)

var_b = eip_read_output_bit(300,7)

// 0x72    get bit: "7"

// 0
```

Syntax 3

```
bool eip_write_output_bit(
    int,
    int,
    byte[],
    int,
    int
)
```

Parameter

<code>int</code>	Starting address
<code>int</code>	The n th bit value
<code>byte[]</code>	The data to write
	*Data in bit will return in byte such as bit[0] for byte[0] and bit[1] for byte[1].
<code>int</code>	The starting bit of the data to write
<code>int</code>	The bit amount of the data to write

Return

<code>bool</code>	<code>True</code>	Write successfully.	
	<code>False</code>	Write unsuccessfully.	1. Unable to send correctly and receive .

Note

Bit value = 1 for byte value >=1

Bit value = 0 for byte value ==0

Syntax 4

```
bool eip_write_output_bit(  
    int,  
    int,  
    byte[],  
    int  
)
```

Parameter

int	Starting address
int	The n th bit value
byte[]	The data to write
	*Data in bit will return in byte such as bit[0] for byte[0] and bit[1] for byte[1].
int	The starting bit of the data to write

Return

bool	True	Write successfully.	
	False	Write unsuccessfully.	1. Unable to send correctly and receive .

Note

*Same as syntax 3. Write with the full length of the rest data to write.

Bit value = 1 for byte value >=1

Bit value = 0 for byte value ==0

Syntax 5

```
bool eip_write_output_bit(  
    int,  
    int,  
    byte[]  
)
```

Parameter

int	Starting address
int	The n th bit value
byte[]	The data to write
	*Data in bit will return in byte such as bit[0] for byte[0] and bit[1] for byte[1].

Return

bool	True	Write successfully.	
	False	Write unsuccessfully.	1. Unable to send correctly and receive .

Note

*Same as syntax 3. Fill 0 as the starting bit to write as well as write with the full length of the data to

write by default.

Bit value = 1 for byte value >=1

Bit value = 0 for byte value ==0

```
byte[] var_data = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1}
```

```
eip_write_output_bit(300,0,var_data,0,20)
```

```
byte[] var_ba = eip_read_output (300,0,3)
```

```
// byte[] = {0x39,0xB8,0x0C}
```

```
var_ba = eip_read_output_bit(300,0,20)
```

```
// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1}
```

```
eip_write_output_bit(300,3,var_data,5,10)
```

```
var_ba = eip_read_output (300,0,3)
```

```
// byte[] = {0x08,0x0E,0x00}
```

```
var_ba = eip_read_output_bit(300,0,20)
```

```
// byte[] = {0,0,0,1,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0}
```

Syntax 6

```
bool eip_write_output_bit(  
    string,  
    int,  
    byte[],  
    int,  
    int  
)
```

Parameter

`string` Item name

`int` Starting bit

`byte[]` The data to write

*Data in bit will return in byte such as bit[0] for byte[0] and bit[1] for byte[1].

`int` The starting bit to write data

`int` The bit amount of the data to write

Return

`bool` `True` Write successfully.

`False` Write unsuccessfully. 1. Unable to send correctly and receive .

Note

Bit value = 1 for byte value >=1

Bit value = 0 for byte value ==0

Syntax 7

```
bool eip_write_output_bit(  
    string,  
    int,  
    byte[],  
    int  
)
```

Parameter

`string` Item name
`int` Starting bit
`byte[]` The data to write
*Data in bit will return in byte such as bit[0] for byte[0] and bit[1] for byte[1].
`int` The starting bit to write data

Return

`bool` `True` Write successfully.
 `False` Write unsuccessfully. 1. Unable to send correctly and receive .

Note

*Same as syntax 6. Write with the full length of the rest data to write.

Bit value = 1 for byte value >=1

Bit value = 0 for byte value ==0

Syntax 8

```
bool eip_write_output_bit(  
    string,  
    int,  
    byte[]  
)
```

Parameter

`string` Item name
`int` Starting bit
`byte[]` The data to write
*Data in bit will return in byte such as bit[0] for byte[0] and bit[1] for byte[1].

Return

`bool` `True` Write successfully.
 `False` Write unsuccessfully. 1. Unable to send correctly and receive .

Note

*Same as syntax 6. Fill 0 as the starting bit to write as well as write with the full length of the data to write by default.

Bit value = 1 for byte value >=1

Bit value = 0 for byte value ==0

```
byte[] var_data = {1,0,0,1,1,1,0,0,0,0,1,1,1,0,1,0,0,1,1}
```

```
eip_write_output_bit("T2O_Register_Bit",0,var_data,0,20)
```

```
byte[] var_ba = eip_read_output (300,3)
```

```
// byte[] = {0x39,0xB8,0x0C}
```

```
var_ba = eip_read_output_bit(300,0,20)
```

```
// byte[] = {1,0,0,1,1,1,0,0,0,0,1,1,1,0,1,0,0,1,1}
```

```
eip_write_output_bit("T2O_Register_Bit",3,var_data,5,10)
```

```
var_ba = eip_read_output (300,3)
```

```
// byte[] = {0x08,0x0E,0x00}
```

```
var_ba = eip_read_output_bit(300,0,20)
```

```
// byte[] = {0,0,0,1,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0}
```

OMRON Corporation Industrial Automation Company

Kyoto, JAPAN

Contact: www.ia.omron.com

Regional Headquarters

OMRON EUROPE B.V.

Wegalaan 67-69, 2132 JD Hoofddorp
The Netherlands
Tel: (31)2356-81-300/Fax: (31)2356-81-388

OMRON ASIA PACIFIC PTE. LTD.

No. 438A Alexandra Road # 05-05/08 (Lobby 2),
Alexandra Technopark,
Singapore 119967
Tel: (65) 6835-3011/Fax: (65) 6835-2711

OMRON ELECTRONICS LLC

2895 Greenspoint Parkway, Suite 200 Hoffman Estates,
IL 60169 U.S.A.
Tel: (1) 847-843-7900/Fax: (1) 847-843-7787

OMRON ROBOTICS AND SAFETY TECHNOLOGIES, INC.

4225 Hacienda Drive, Pleasanton CA 94588 U.S.A.
Tel: (1) 925-245-3400/Fax: (1) 925-960-0590

OMRON (CHINA) CO., LTD.

Room 2211, Bank of China Tower, 200 Yin Cheng Zhong Road,
PuDong New Area, Shanghai, 200120, China
Tel: (86) 21-5037-2222/Fax: (86) 21-5037-2200

Authorized Distributor:

© Techman Robot Inc. 2019-2021 All Rights Reserved
In the interest of product improvement,
specifications are subject to change without notice.

Cat. No. I848-E-04

1021