OMRON

Omron TM Collaborative Robot: TMscript Language Manual

TM ROBOT

Original Instruction

I664-E-02

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.

and logos are registered trademarks of TECHMAN ROBOT INC. and the company reserves the ownership of this manual and its copy and its copyrights.

Terms and Conditions Agreement

Warranty Limitations of Liability

Warranties

- Exclusive Warranty

  Omron's exclusive warranty is that the Products will be free from defects in materials and workmanship for a period of twelve months from the date of sale by Omron (or such other period expressed in writing by Omron). Omron disclaims all other warranties, express or implied.

- Limitations

  OMRON MAKES NO WARRANTY OR REPRESENTATION, EXPRESS OR IMPLIED, ABOUT NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OF THE PRODUCTS. BUYER ACKNOWLEDGES THAT IT ALONE HAS DETERMINED THAT THE PRODUCTS WILL SUITABLY MEET THE REQUIREMENTS OF THEIR INTENDED USE. Omron further disclaims all warranties and responsibility of any type for claims or expenses based on infringement by the Products or otherwise of any intellectual property right.

- Buyer Remedy

  Omron's sole obligation hereunder shall be, at Omron's election, to (i) replace (in the form originally shipped with Buyer responsible for labor charges for removal or replacement thereof) the non-complying Product, (ii) repair the non-complying Product, or (iii) repay or credit Buyer an amount equal to the purchase price of the non-complying Product; provided that in no event shall Omron be responsible for warranty, repair, indemnity or any other claims or expenses regarding the Products unless Omron's analysis confirms that the Products were properly handled, stored, installed and maintained and not subject to contamination, abuse, misuse or inappropriate modification. Return of any Products by Buyer must be approved in writing by Omron before shipment. Omron Companies shall not be liable for the suitability or unsuitability or the results from the use of Products in combination with any electrical or electronic components, circuits, system assemblies or any other materials or substances or environments. Any advice, recommendations or information given orally or in writing, are not to be construed as an amendment or addition to the above warranty.

See http://www.omron.com/global/ or contact your Omron representative for published information.

Limitations of Liability: Etc.

OMRON COMPANIES SHALL NOT BE LIABLE FOR SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, LOSS OF PROFITS OR PRODUCTION OR COMMERCIAL LOSS IN ANY WAY CONNECTED WITH THE PRODUCTS, WHETHER SUCH CLAIM IS BASED IN CONTRACT, WARRANTY, NEGLIGENCE OR STRICT LIABILITY.

Further, in no event shall liability of Omron Companies exceed the individual price of the Product on which liability is asserted.

Application Considerations

Suitability of Use

Omron Companies shall not be responsible for conformity with any standards, codes or regulations which apply to the combination of the Product in the Buyer's application or use of the Product. At Buyer's request, Omron will provide applicable third party certification documents identifying ratings and limitations of use which apply to the Product. This information by itself is not sufficient for a complete determination of the suitability of the Product in combination with the end product, machine, system, or other application or use. Buyer shall be solely responsible for determining appropriateness of the particular Product with respect to Buyer's application, product or system. Buyer shall take application responsibility in all cases.
NEVER USE THE PRODUCT FOR AN APPLICATION INVOLVING SERIOUS RISK TO LIFE OR PROPERTY WITHOUT ENSURING THAT THE SYSTEM AS A WHOLE HAS BEEN DESIGNED TO ADDRESS THE RISKS, AND THAT THE OMRON PRODUCT(S) IS PROPERLY RATED AND INSTALLED FOR THE INTENDED USE WITHIN THE OVERALL EQUIPMENT OR SYSTEM.

Programmable Products

Omron Companies shall not be responsible for the user's programming of a programmable Product, or any consequence thereof.

Disclaimers

Performance Data

Data presented in Omron Company websites, catalogs and other materials is provided as a guide for the user in determining suitability and does not constitute a warranty. It may represent the result of Omron's test conditions and the user must correlate it to actual application requirements. Actual performance is subject to the Omron's Warranty and Limitations of Liability.

- Change in Specifications

Product specifications and accessories may be changed at any time based on improvements and other reasons. It is our practice to change part numbers when published ratings or features are changed, or when significant construction changes are made. However, some specifications of the Product may be changed without any notice. When in doubt, special part numbers may be assigned to fix or establish key specifications for your application. Please consult with your Omron representative at any time to confirm actual specifications of purchased Product.

Errors and Omissions

Information presented by Omron Companies has been checked and is believed to be accurate; however, no responsibility is assumed for clerical, typographical or proofreading errors or omissions.

Statement of Responsibilities for Cyersecurity Threats

To maintain the security and reliability of the system, a robust cybersecurity defense program should be implemented, which may include some or all of the following:

Anti-virus protection

- Install the latest commercial-quality anti-virus software on the computer connected to the control system and keep the software and virus definitions up-to-date.
- Scan USB drives or other external storage devices before connecting them to control systems and equipment.

Security measures to prevent unauthorized network access

- Install physical controls so that only authorized personnel can access control systems and equipment.
- Reduce connections to control systems and equipment via networks to prevent access from untrusted devices.
- Install firewalls to block unused communications ports and limit communication between systems. Limit access between control systems and systems from the IT network.
- Control remote access and adopt multifactor authentication to devices with remote access to control systems and equipment.
- Set strong password policies and monitor for compliance frequently.

Data input and output protection

- Backup data and keep the data up-to-date periodically to prepare for data loss.
- Validate backups and retention policies to cope with unintentional modification of input/output data to control systems and equipment.
- Validate the scope of data protection regularly to accommodate changes.

- Check validity of backups by scheduling test restores to ensure successful recovery from incidents.
- Safety design, such as emergency shutdown and fail-soft operations in case of data tampering and incidents.

Additional recommendations

- When using an external network environment to connect to an unauthorized terminal such as a SCADA, HMI or to an unauthorized server may result in network security issues such as spoofing and tampering.
- You must take sufficient measures such as restricting access to the terminal, using a terminal equipped with a secure function, and locking the installation area by yourself.
- When constructing network infrastructure, communication failure may occur due to cable disconnection or the influence of unauthorized network equipment.
- Take adequate measures, such as restricting physical access to network devices, by means such as locking the installation area.
- When using devices equipped with an SD Memory Card, there is a security risk that a third party may acquire, alter, or replace the files and data in the removable media by removing or unmounting the media.

Contents

Revision History Table

Revision	Date	Description
1.00	January, 2021	Original release
2.00	June, 2023	Added 2.0 features

1. Overview

TMscript is the programming language of Techman Robot applicable to Flow projects and Script projects. Refer to the table below for the main scope of applications.

- Scope of Applications

Application	Flow Projects			Script Projects
	Set Node (& other nodes)	Listen Node (external scripts)	Script Node	
Expression	✓	✓	✓	✓
define/main/closestop/errorstop				✓
Customized Function				✓
Comment		✓	✓	✓
Global variables in the Project	✓	✓	✓	✓
Local variables in the Function		✓	✓	✓
Multiline Input		✓	✓	✓
Conditional Statement		✓	✓	✓
Loop Statement		✓	✓	✓
Branching Statement		✓	✓	✓
Thread Function				✓
General Function	✓	✓	✓	✓
General Function (Script)		✓	✓	✓
Math Function	✓	✓	✓	✓
File Function	✓	✓	✓	✓
Serial Port Class		✓	✓	✓
Serial Port Function	✓	✓	✓	✓
Socket Class				
Socket Function	✓	✓	✓	✓
Parameterized Object	✓	✓	✓	✓
Robot Teach Class		✓	✓	✓
Robot Motion & Vision Job Function		✓	✓	✓
Vision Function	✓	✓	✓	✓
External Script		✓		
Modbus TCP/RTU Class		✓	✓	✓
Modbus Function	✓	✓	✓	✓
TM Ethernet Slave	✓	✓	✓	✓
Profinet Function	✓	✓	✓	✓
EtherNet/IP Function	✓	✓	✓	✓
FTSensor Class		✓	✓	✓
Force Class		✓	✓	✓

2. Expression

2.1 Types

Different data types of variables can be declared in Variable 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	-3.4028235E+38 to 3.4028235E+38	significant digit 7
double	64bit floating-point	signed	-1.7976931348623157E+308 to 1.7976931348623157E+308	significant digit 15
bool	Boolean		true or false	
string	string			

In function terms, the integer type further goes by int16 and int32. The default is int32.

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

2.2 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 i = 0
string s = "ABC"
string s1 = "DEF"
string s2 = "123"
```

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

```
s = s1 + " and " + s2    // s = "DEF and 123"
                         // s, s1, 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 such as var_s1 or g_s2. 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.

- Byte can only present unsigned numbers in 8 bits ranging from 0 to 255. As a result, if a negative sign is assigned to the byte type or through calculation, it will still save the 8-bit unsigned value only.

  For example:

  byte b = -100 // Error // -100 mismatches with the value range of byte.
  byte b = 0 - 100 // b = 156 // 0-100=-100 (0xFFFFFF9C). Saved 8 bits as 0x9C. The value equals to 156.
  b = 0 - 1 // b = 255 // 0-1=-1 (0xFFFFFFFF) Saved 8 bits as 0xFF The value equals to 255.
  b = 255 + 1 // b = 0 // 255+1=256 (0x100) Saved 8 bits as 0x00 The value equals to 0.

- Int can present signed numbers in 32 bits ranging from -2147483648 to 2147483647. If the calculation exceeds the value range, it will still save the 32-bit signed integer value only.

  For example:

  int i = -2147483648 - 1 // i = 2147483647 // -2147483648 - 1 = -2147483649 (0xFFFFFFFF 7FFFFFFF) Saved 32 bits as 0x7FFFFFFF. The value equals to 2147483647.

  i = 2147483647 + 1 // i = -2147483648 // 2147483647 + 1 = 2147483648 (0x80000000) Save 32 bits in the method of signed integer value. The value equals to -2147483648

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 follows the rules below
  1. Numbers will be view as numbers
  2. The combination of numbers and characters will be view as variable as long as the variable does exist.
  3. If the variable does not exist, it will be compiled as string with a warning message.
- The combination of string and variable
  1. Inside double quotation marks, variables will not be combined as variables

     For example:

     s = "TM5" // s = "TM5"
     s1 = "Hi, s Robot" // s1 = "Hi, s Robot"
  2. Standard syntax. Double quotation marks needs to be placed around the string, and plus sign (+) shall be used to link variables and numbers

Example:

s1 = "Hi, " + s + " Robot"   // s1 = "Hi, TM5 Robot"

3. Compatible syntax (not recommended). The single quotation marks can be placed around the variables, but a warning message will be send out

   For example:

   single quotation marks   "Hi, 's' Robot"   // s1 = "Hi, TM5 Robot"

   "Hi, 'x' Robot"   // 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[] ss = {"Techman", "Robot"}

   "Hi, 's' 'ss[0]' Robot"   // s1 = "Hi, TM5 'ss[0]' Robot"   // 'ss[0]' is invalid

   "Hi, " +   s + " " + ss[0] + " Robot"   // 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"   // s1 = "Hi, TM5 Robot"

   // If s1 = "Hi, 's' Robot" is what you want, please use the following syntax.

   "Hi, '" + "s" + "' Robot"   // s1 = "Hi, 's' Robot"

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

  For example

  s1 = "Hi, " + Ctrl("\r\n") + s + " Robot"   or   "Hi, " + NewLine + 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.

Denote true value	true True
Denote false value	false False

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

2.3 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[] i = {0,1,2,3}                      // elements in number data type
string[] s = {"ABC", "DEF", "GHI"}       // elements in string data type
bool[] 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 if accessing beyond the range, such as A[8].
- 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[] ss = {empty, empty, empty}           // The initial size of string array is 3 elements
ss = String_Split("A_B_C_D_F_G_H", "_")       // After splitting string, the string array has 7 elements
len = Length(ss)                              // len = 7
ss = String_Split("A,B", ",")                 // After splitting string, the string array has 2 elements
len = Length(ss)                              // len = 2
```

2.4 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 ➔ (A = (((A * B) / C) % D))

$$\underbrace{A = \underbrace{\underbrace{\underbrace{A * B}_{1} / C}_{2} \% D}_{3}}_{4}$$

  right-to-left associativity

  A -= B += 10 && ! !D ➔ (A -= (B += (10 && (! (!D)))))

$$\underbrace{A \mathrel{-}= \underbrace{B \mathrel{+}= \underbrace{10 \mathbin{\&\&} \underbrace{! \underbrace{!D}_{1}}_{2}}_{3}}_{4}}_{5}$$

- 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 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)
        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		

Precedence High to low	Operator	Name	Example	Requirement	associativity
	-	Unary minus	int i = -1	Numeric variable,	
	!	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 (condition A && condition B)		
5	||	Logical OR	if (condition A || condition B)		
4	c ? t : f	Ternary conditional	int i = a > b ? a : b		right-to-left
3	=	Basic assignment	int a = b	Left side: Numeric variable Right side: Numeric variable, Constant	
	+=	Addition assignment	a += 3		
	-=	Subtraction assignment	b -= 4		
	*=	Multiplication assignment	a *= 5		
	/=	Division assignment	a /= 2		
	%=	Modulo assignment	a %= 3		
	<<=	Bitwise left shift assignment	flags <<= 2	Left side: Integer variable	

Precedence High to low	**Operator**	**Name**	**Example**	**Requirement**	**associativity**
	>>=	Bitwise right shift assignment	flags >>= 2	Right side: Integer variable, Constant	
	&=	Bitwise AND assignment	flags &= new_flags		
	^=	Bitwise XOR assignment	flags ^= new_flags		
	|=	Bitwise OR assignment	flags |= new_flags		

2.5 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 = false (0 equals to false)
	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 (non 0 equals to true)
	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 (non 0 equals to 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" (shown in lower-case)
string	byte	byte b1 = (byte)"1.23" byte b2 = (byte)"XYZ"	1 Error (Unable to convert "XYZ" to value)
	int	int i = (int)"1.23"	1
	float	float f1 = (float)"1.23" float f2 = (float)"A1"	1.23 Error (Unable to convert "A1" to value)
	double	double d = (double)"1.23"	1.23

	bool	bool flag1 = (bool)"true"
		bool flag2 = (bool)"false"
		bool flag3 = (bool)"1.23"
		bool flag4 = (bool)""
			flag1 = true (string "true" as true)
			flag2= false (string "false" as false)
			flag3 = true (non-empty string as true)
			flag4 = false (empty string for false)

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

```
string[] ss = {"1.23", "4.56", "0.789"}
int[] i_array = (int[])ss              // i_array = {1, 4, 0}
float[] f_array = (float[])ss          // 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 value = (int)true        // Error
int value = (int)"XYZ"       // Error
```

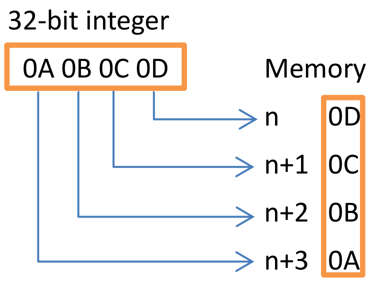
- Invalid floating-point numbers to convert to floats or doubles such as NaN or Infinity.

```
string   dvalue = "1.79769e+308"
float f = (float) dvalue     // Error 1.79769e+308 is a valid double type and unable to convert to the float
                             type.
```

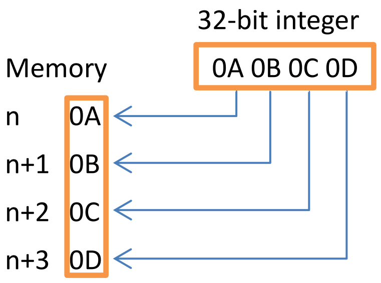
2.6 Endianness and Conversion

Endianness refers to the applications in the memory or the communication networks in which data comes in multiple bytes for the expressions of the order to sort among multiple bytes because the minimal unit comes in bytes.

- Little Endian. Place the low bits of the multibyte at smaller addresses and the high bits at larger ones.



- Big Endian. Place the high bits of the multibyte at smaller addresses and the low bits at larger ones.



It is a factor to consider because different orders to sort may cause different conversion results. For example, an integer type is 32 bits. Namely, it occupies 4 bytes. If the int value is 300, the expression in 16-bit is 0x0000012C. If using Little Endian for memory storing or communication networks, the order is

[0]	[1]	[2]	[3]
2C	01	00	00

It fetches 0x0000012C if adopting Little Endian while 0x2C010000 if Big Endian. These are two completely different values, which may cause different results.

TMscript provides a variety of robot programming functions. Among the functions, the conversions of the numeric types, such as int, float, and double, also follow the general endianness rules, which usually apply to file functions, communication-related functions (such as SerialPort, Socket, Modbus), or value-to-byte conversion-related functions. Unless specified otherwise, the function defaults to go by Little Endian. For the string types, it goes by the UTF-8 encoding format that features ASCII character compatibility and fixes endianness without the byte order distinction. For the bool types, it turns the Boolean value true to 1 and false to 0. Refer to the table organized below.

Type	Bit/Byte	Conversion Method	Example	Result
byte	8 bits/1 byte	Little Endian	byte b = 100	0x64
int	32 bits /4 bytes	Little Endian	int i = 300	0x2C 0x01 0x00 0x00
float	32 bits /4 bytes	IEEE-754, Little Endian	float f = 300	0x00 0x00 0x96 0x43
double	64 bits /8 bytes	IEEE-754, Little Endian	double d = 300	0x00 0x00 0x00 0x00 0x00 0xC0 0x72 0x40
bool	bool value	True:1, False:0	bool bf = true	0x01
string	string value	UTF-8	"TM Robot"	0x54 0x4D 0x20 0x52 0x6F 0x62 0x6F 0x74

The value conversion method is an essential basis for communication network applications. Both sides of the applications should recognize the same conversion method to parse the data content correctly. For example,

- **socket_send**("ntd_a", 9000)
  By the function definition, it sends 0x28,0x23,0x00,0x00 (int, Little Endian) to the device ntd_a.
  It gets various values if the receiving device uses different methods to parse. Such as

int, Big Endian,	0x28 0x23 0x00 0x00 = 673382400
float, Little Endian,	0x00 0x00 0x23 0x28 = 1.2612E-41
string, UTF-8,	0x28 0x23 0x00 0x00 = "(#"
int, Little Endian,	0x00 0x00 0x23 0x28 = 9000

- **socket_send**("ntd_a", (float)9000)
  By the function definition, it sends 0x00,0xA0,0x0C,0x46 (int, Little Endian) to the device ntd_a.
  It gets various values if the receiving device uses different methods to parse. Such as

int, Big Endian,	0x00 0xA0 0x0C 0x46 = 10488902	
int, Little Endia,	0x46 0x0C 0xA0 0x00 = 1175232512	
float, Big Endian,	0x00 0xA0 0x0C 0x46 = 1.4698082E-38	
string, UTF-8,	0x00 0xA0 0x0C 0x46 = ""	// The string ends if encountered 0x00.
float, Little Endian,	0x46 0x0C 0xA0 0x00 = 9000	

2.7 Warning

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

- Double quotation marks does not placed around the string constant.
```
string s = Hello                // warning  Hello
```
- There is single quotation mark inside the string constant.
```
string s0 = "World"
string s1 = "Hello ‘s0’"        // warning  ‘s0’
```
- When assigning float value to integer constant, some digits may get lost such as
```
int i = 0
float f = 1.234
i = f               // warning  i = 1
```
- When assigning value to variables with fewer digits, some digits may get lost such as
```
byte b = 100
int i = 1000
float f = 1.234
double d = 2.345
b = i               // warning  b = 232      //   byte can contain values from 0 to 255
f = d               // warning  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 i = "1234"              // warning  i = 1234
int j = "0x89AB"            // warning  j = 35243
int k = "0b1010"            // warning  k = 10
string s1 = 123             // warning s1 = "123" // Number to string
string s2 = "123"
int x = s2                  // warning  x = 123 // string to number
// The following code can be compiled with warning, but will be stopped by error when executing.
S2 = "XYZ"
x = s2                // warning // Stop executing by error      // s = "XYZ" cannot be converted to number
s2 = ""
x = s2                // warning // Stop executing by error      // 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 s = "Hello"
```
  2. Use + to link the string constant and the string variable
```
string s0 = "World"
string s1 = "Hello " + s0
```
  3. Specify the type clearly for numerical conversions
```
float   f = 1.234
int i = (int) f             // Use (int) for type conversion, i = 1 while processing // It turns the number in
                            floating-point to an integer.
```

3. Script Project Programming

3.1 define

This function defines the global variables in the same project. When the project is running, it prioritizes all the variables in the definition section of the project.

Syntax

```
define
```

For example:

```
define
{
    string text = "Hi TM Robot"
}
```

3.2 main

This function is the first function to call when the project is running, and it is the initial function of the project.

Syntax

```
main
```

For example:

```
define
{
    string text = "Hi TM Robot"
}
main
{
    Display("Hello Techman Robot")
    Display(text)
}
```

The project begins by running the main function as the result of

```
Hello Techman Robot
Hi TM Robot            //Since the dashboard shows the last Display content only, the dashboard content is Hi
                       TM Robot.
```

3.3 closestop

When the project is in the halt state without any error, it goes to this function. Not until this function ends will the project stop for sure.

Syntax

```
closestop
```

For example:

```
closestop
{
        IO["ControlBox"].DO = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}        // Set the control box DO to Low.
}
```

After the project is running, pressing the STOP button results in the project running the closestop function.

Important

This function does not go with motion commands and will close mandatory after 12 seconds of execution at most.

When the closestop function is running, if errors occur (including other errors occur in the system), it will not run the errorstop function but end the project running directly.

For example:

```
main
{
        Exit()        // Stop the project. Suppose errors did not occur; it runs the closestop function consecutively.
}
closestop
{
        int zero = 0
        int k = 100 / zero        // A dividing by zero error occurred in the closestop function. It ends the project
                                  running directly.
        Display("closestop")  // This line does not run due to an error occurring.
```

Note

When the project enters the halt state, the page will remove the running state, but it will continue to run the function. Not until this function ends does the project running stop exactly. Before the project stops running, pressing the Play button to run the project returns errors.

3.4 errorstop

When the project is in the halt state with any error, it goes to this function. Not until this function ends will the project stop for sure.

Syntax

```
errorstop
```

For example:

```
main
{
        int zero = 0
        int k = 100 / zero              // A divide by zero error occurs.
}
errorstop
{
        IO["ControlBox"].DO = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}        // Set the control box DO to Low.
}
```

After the project is running, the project will generate errors by dividing by zero and go to the errorstop function.

Important

This function does not go with motion commands and will close mandatory after 12 seconds of execution at most.

When the errorstop function is running, if errors occur (including other errors occur in the system), it will not run the errorstop function but end the project running directly.

For example:

```
main
{
        int k = 100 / (byte)"0.1"     // A dividing by zero error occurred.
}
errorstop
{
        int k = 100 / (byte)"0.1"     // A dividing by zero error occurred in the errorstop function. It ends the project
                                      running directly.
        Display("errorstop ")         // This line does not run due to an error occurring.
```

Note

When the project enters the halt state, the page will remove the running state, but it will continue to run the function. Not until this function ends does the project running stop exactly. Before the project stops running, pressing the Play button to run the project returns errors.

3.5 Customized Function

Other than the built-in define, main, closetop, and errorstop functions, Script projects provide customized functions for users. As customized to the Show() function below, there is no parameter to input nor the value to return (void).

For example:

```
main
{
        Show()      // Call the customized function Show().
}
void Show()
{
        Display("Hello TM AI Cobot")
}
```

Customized functions support parameter input and output. There are two functions customized in the example below. The Add() can add up the two values users input and return the added up result; the Sub() subtracts the two values users input and return the subtraction result. Note that as long as the returned value is not void, it requires to return the value of the same type as the definition. Such as the Add() adopts the float type, and the returned variable must be the float type as well.

For example:

```
main
{
        string Add_Result = "Add:" + Add(5.9, 3.6)
        string Sub_Result = "Sub:" + Sub(5.9, 3.6)
        Display("Green", "Yellow", "Result", Add_Result + newline + Sub_Result)
}
float Add(float augend, float addend)
{
        float result = augend + addend
        return result
}
float Sub(float minuend, float subtrahend)
{
        float result = minuend - subtrahend
        return result
}
```

IMPORTANT

Important

If the function defines the return type, it must return with the same type. If the definition is void, it is not required to use return.

3.6 Comment

Use part of the code as a comment. The compiler ignores the commented code. Users can comment a single line or multi-lines on the code not to execute.

- Single Line Comment

Uses can comment at the very beginning of the code with // to complete a single-line comment, as shown below. The compiler will ignore a = a + 1.

```
main
{
        int a = 0
        // a = a + 1
}
```

- Multi Line Comment

Users can comment with /* as the start of the comment and */ as the end of the comment. As below, between /* and */, the compiler will ignore int a = 0 and a = a + 1.

```
main
{
        /*
        int a = 0
        a = a + 1
        */
}
```

3.7 Variable

By variable declaration, users can proceed with applications such as calculation, reading and writing, and parameter conveyance. Variables come with the global variables and the local variables.

IMPORTANT

Important
Variables are case sensitive such as TMrobot and TMROBOT are two different variables.

- Global Variable

Once declared a variable in the definition section, the variable is a global variable in the project. This variable can be read and written in functions of the same project. As shown below, if the define declares the variable b, and the main function runs, it calls Test1() first and sets the variable as 20. When it displays the variable value next, it gets b as 20 for the last set value.

```
define
{
        int b = 0
}
main
{
        Test1()
        Display("B Value: " + b)
}
void Test1()
{
        b = 20
}
// B Value: 20
```

IMPORTANT

Important
The global variables in the project are different from the global variables in the global variable manager of the robot. The global variables in the project indicate they can operate across functions in the project. Other than across functions in the project, the global variables in the global variable manager of the robot operate across functions of different projects.

IMPORTANT

Important
Users can create global variables in the global variable manager in the TMflow operation interface only but not with TMscript syntax.

- Local Variable

If variables are declared outside of the definition section such as main, closestop, errorstop, and customized functions, the variables are local variables and good in the functions only.

```
int sum(int s)
{
```

```
          int a = 0
      }
      // Defined two variables int s and int a. When sum() ends, the two local variable are gone.
```

Neither can variable names in the definition section of the same function be duplicated, nor can they replicate with the global variables in the define section and in the global variable manager of the robot.

```
      define
      {
          int b = 0
          string g_s1 = ""  // Suppose the global variable manager defines g_s1; there will be variable
                            duplication.
      }
      void sum(int s)
      {
          int a = 0
          int b = 1         // Duplicated with int b in define.
          float a = 0       // Duplicated with int a.
      }
```

Local variables are independent of different function sections. Such declared a variable a in both Test1() and Test2(). Because of being in disparate functions, the two variables are independent and valid only in each function section.

```
      void Test1()
      {
          int a = 0
      }
      void Test2()
      {
          int a = 2
      }
```

The scope of the local variables can be valid in a top-down coverage. Users can also declare variables in the conditional statement or the loop statement, and the variables are present in the conditional statement or the loop statement. The variables are absent from the outside of the scope. Therefore, once the variables exit from the scope, the variables will release the variable data. For variables declared by the class, such as SerialPort, Socket, and Modbus, the variables will close the devices.

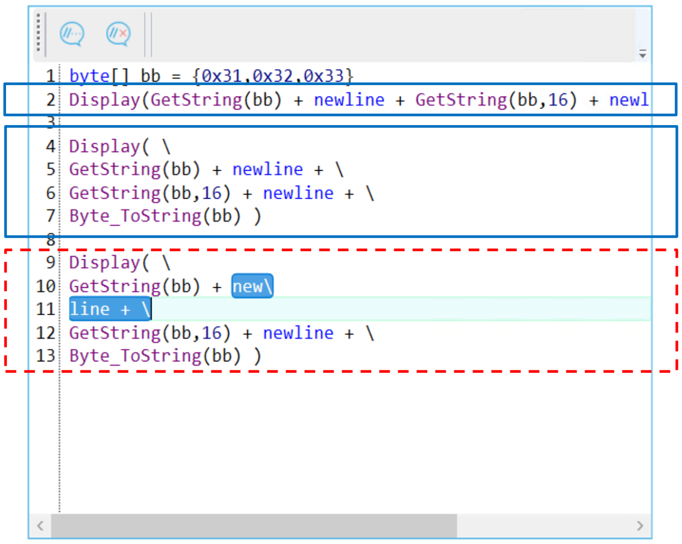
```
    void Test1()
    {
                                         variable a effective scope
        int a = 10
        if (a > 5)
        {                                variable b effective scope
            int b = 20
            Display("A Value: " + a)
        }
        Display("B Value: " + b)         // Warning. b will be taken as a string instead of a variable.
}
    void Test2()
    {
        if (true)
        {                                        variable bbb effective scope
            Socket bbb = "127.0.0.1",12345
            socket_open("bbb")
            Sleep(5000)
        } // After a 5-second waiting, variable bbb will be released for exiting the f conditional statement
        scope and close the connection.
        Sleep(10000)
    }
```

In the Flow project, if users want to create a global variable in the project, they need to do it in the variable manager or device manager. Users cannot create a global variable in the flow project with the Script. Moreover, because each process node is an independent function, including the Script node, if declaring variables in the node, they are all local variables. When exiting the node, the variables in the node will no longer exist, and the variable data will be released.

3.8 Multi-Line Input

Once an expression to input goes with more contents, it is a disadvantage to maintain or debug. Users can add \ at the end of the line to fulfill the multi-line input of consecutive contents. The multi-line input contents will be taken as an expression in the same line until there is no \, and it goes with a new line character at the end.

While using multi-line input, it is still required to maintain the correctness of wording but not to use \ at random for multi-line continuations among words.



```
byte[] bb = {0x31,0x32,0x33}
Display(GetString(bb) + newline + GetString(bb,16) + newl

Display( \
GetString(bb) + newline + \
GetString(bb,16) + newline + \
Byte_ToString(bb) )

Display( \
GetString(bb) + new\
line + \
GetString(bb,16) + newline + \
Byte_ToString(bb) )
```

//Beyond the viewable scope, which requires scrolling to read the content.

//Use multi-line input, \,   to continue for easy content viewing.

//Incorrect multi-line put, where newline is a reserved word and cannot be cut.

3.9 Conditional Statements

During the project running, it might be necessary to consider different approaches, such as task fails, function, and communication errors, to various situations. Users can use conditional statements to adopt different paths to the result values at this time. Two conditional statements, if and switch, are available so far.

3.9.1 if

The if statement can judge the conditional expression in the brackets. It enforces statement 1 if the condition meets.

```
if (conditional expression)
{
        statement 1             // the condition satisfies.
}
```

Or vice versa. If the condition satisfies, it goes to statement 1. If not, it goes statement 2.

```
if (conditional expression)
{
        statement 1             // the condition satisfies.
}
else
{
        statement 2             // the condition not satisfies.
}
```

Users can also check more possibilities with the if .. else if .. else statement to judge the condition 1, 2, 3, and the last otherwise condition in sequence.

```
if (conditional expression 1)
{
        statement 1             // the condition 1 satisfies.
}
else if (conditional expression 2)
{
        statement 2             // the condition 2 satisfies.
}
else if (conditional expression 3)
{
        statement 3             // the condition 3 satisfies.
}
else
{
        statement 4             // the condition 1, 2, and 3 not satisfy at all.
}
```

Take the example below. If the Score is 100, it displays "Full Score." Between 60 and 99, "Excellent"; otherwise, "Failed." Users can write the condition with the if statement.

```
void Test1()
```

```
{
        int Score = 65
        if (Score == 100)
        {
                Display("Green", "Yellow", "Full Score", "")
        }
        else if (Score >= 60)
        {
                Display("Green", "Yellow", "Excellent", "")
        }
        else
        {
                Display("Red", "Yellow", "Failed", "")
        }
}
// Excellent
```

NOTE:
== of if (Score==100) stands for comparison, and = of int Score = 65, assignment.

The criteria to evaluate conditional expressions are as follows, but boolean values have much to recommend for evaluation.

● Boolean values	true or True for the condition satisfies, false or False, not satisfies.
● Numeric values	non-zeros for the condition satisfies, zeros, not satisfies.
● String values	true or True for the condition satisfies, other string values, not satisfies.

3.9.2 switch

The switch statement is similar in condition judgment to the if statement. The switch statement goes with the other way of writing for similar conditions.

```
switch (variable or expression)
{
        case condition 1 satisfies
                statement 1             // the condition 1 satisfies.
                break
        case condition 2 satisfies
                statement 2             // the condition 2 satisfies.
                break
        default
                statement 3             // the condition 1 and 2 both not satisfy.
                break
}
```

As the example below, it can match various results by the content value of the variable.

```
void Test1()
{
        string di_st = (string)IO["ControlBox"].DI[1] + (string)IO["ControlBox"].DI[0]
        switch (di_st)
        {
            case "00"
                Display("DI[0]=0, DI[1]=0")
                break
            case "01"
                Display("DI[0]=1, DI[1]=0")
                break
            case "10"
                Display("DI[0]=0, DI[1]=1")
                break
            default
                Display("DI[0]=1, DI[1]=1")
                break
        }
    }
    // DI[0]=0, DI[1]=0
```

Besides, the switch statement supports expressions. Like the example below, if the Score is 100, it displays "Full Score." Between 60 and 99, "Excellent"; otherwise, "Failed."

```
    void Test1()
    {
        int Score = 65
        switch (Score)
        {
            case >= 100
                Display("Green", "Yellow", "Full Score", "")
                break
            case >= 100-40
                Display("Green", "Yellow", "Excellent", "")
                break
            default
                Display("Red", "Yellow", "Failed", "")
                break
        }
    }
    // Excellent
```

The differences from the if statement is

- The switch statement retrieves the value once and compares the result value multiple times, and the if statement retrieves every time it compares. This makes the switch's judgment on the DI status more accurate.
- As the reason above, the switch statement cannot apply to the conditions of different natures, and the if statement can. Such as

```
void Test1()
{
        int Payload = 4
        int Length = 130
        if (Payload > 4)
        {
                Display("Green", "Yellow", "Payload", "")
        }
        else if (Length > 70)
        {
                Display("Green", "Yellow", "Length", "")
        }
        // This statement of if .. else if .is unable to be written with the switch statement.
}
```

3.10 Loop Statements

During the project running, it may be necessary to repeatedly calculate certain values or check whether the conditions are satisfied. In these cases, it requires the loop statements that go on iterative processing the code in the statement until the condition is satisfied. Three loop statements, for, while, and do-while, are available so far.

3.10.1 for

The for loop syntax comprises four sections, the initialization section, the loop condition section, the statement, and the iterative operation section.

```
for (the initialization section; the loop condition section; the iterative operation section)
{
        statement
}
```

The execution sequence is as below.

1. The initialization section: When it goes to the for syntax, it executes the initialization section one time. It is usually used to declare variables. (The variables are the local variables in the for syntax.)
2. The loop condition section: It decides whether to go on the for loop condition execution. Once satisfied (true) or not existed, it goes on the for loop execution. It exits the for loop until the condition is not satisfied (false).
3. The execution statement: The statement to execute.
4. The iterative operation section: After the statement execution, it executes the iterative operation section once and then goes back to the loop condition section for the condition judgment.

The example is a basic application of the for loop. It adds the number from 0 to the input K value.

```
int sum(int k)
{
        int result = 0
        for (int i = 0; i < k; i++)
        {
                result += i
        }
        return result
}
```

Users can use multiple for loops together. The example presents the multiplication table by the for loop.

```
void Test1()
{
        string result = ""
        for (int i = 1; i <= 9; i++)
        {
                for (int j = 1; j <= 9; j++)
                {
                        result += j + "X" + i + "=" + i * j + " "
                }
                result += newline
        }
        Display("Green", "Yellow", "multiplication table", result)
    }
```

Users can adopt the four sections in for loop optionally.

```
    void Test1()
    {
        int i = 0
        for (i = 3; i < 4; )
            i++
        // i = 3
        // i < 4       // i++        // i = 4
        // i < 4       // false, exit for
        for ( ; i < 5; i++)
        {
        }
        // Since the value of i is not reset, it will continue using i =4.
        // i < 5       //            // i++, i = 5
        // i < 5       // false, exit for

        for ( ; ; )                  // Execution continues because the loop condition section does not exist.
        {
            // ...
        }
    }
```

3.10.2 while

The while loop comes with one boolean conditional expression only. Once the condition satisfies (true), it executes the statement until the condition does not satisfy (false) and exit the while loop. Therefore, the number of the while loop execution is zero or more, and it may exit for the condition does not satisfy at the first time.

```
while  (conditional expression )
{
    statement
}
```

The example below adopts an arithmetic progression to count the number from 0, 1, 2, 3, …, and N until the total is 500500 and gets the value of N.

```
    void Test1()
    {
        int sum = 0
        int N = 0
        while (sum != 500500)
        {
            N += 1
            sum += N
        }
        Display(N)
    }
    // N = 1000
```

3.10.3 do while

The syntax of the do while loop is similar to that of the while loop. The while loop checks whether if the condition satisfies first before executing statements in the code block; however, the do while loop executes the command first and then check whether if the condition satisfies. Accordingly, the do while loop executes the statement one time at least.

```
do
{
    statement
} while  (conditional expression )
```

For example as below, if using the do while loop, it displays Hello TM Robot.

```
    void Test1()
    {
        int result = 0
        do
        {
            Display("Hello TM Robot")
        } while(result > 5)
    }
```

On the contrary, if using the while loop, it does not display any result.

```
    void Test1()
    {
        int result = 0
        while(result > 5)
        {
            Display("Hello TM Robot")
        }
    }
```

3.11 Branching Statements

3.11.1 break

It applies to exit the last loop statement of for, while, or do while without satisfying the loop condition and exit. As the example below, once i is larger than 10, it exits and ends the for loop.

```
int sum(int k)
{
    int result = 0
    for (int i = 0; i < k; i++)
    {
        if (i > 10)
        {
            break                        Exit the closest for i loop directly.
        }
        result += i
    }
    return result
}
void Test1()
{
    string result = ""
    for (int i = 1; i <= 9; i++)
    {
        for (int j = 1; j <= 9; j++)
        {
            result += j + "X" + i + "=" + i * j + " "
            if (j > 4) break             Exit the closest for j loop directly.
        }
        result += newline
    }
    Display("Green", "Yellow", "multiplication table", result)
}
```

3.11.2 continue

It applies to the loop statements of for, while, and do while, but it is different from the break statement in that it ends the closest and current loop and begins the next loop without exiting the loop. As the example below, if it is an even number, it ends the current loop directly and begins the next loop. So, it registers odd numbers only.

```
void Test1()
{
    string result = ""                   Ends the current loop and goes back to the next loop.
    for (int i = 0; i < 10; i++)
    {
        if (i % 2 == 0)
        {
            continue
        }
        result += i + ", "
    }
    Display("Odd value: " + result)
}
// Odd value: 1, 3, 5, 7, 9
```

3.11.3 return

The return statement ends the statement execution in the function and returns the value to the function caller. For example,

```
main
{
    int num = sum(10)                   // num = 3
    Display(num)
}
int sum(int k)
{
    int result = 0
    for (int i = 0; i < k; i++)
    {
        if (i == 3) return result       // Once i == 3, it returns the result directly without further statement
                                        executions.
        result += i
    }
    return result
}
```

3.12 Thread

During the process of project programming, when it comes to asynchronous parallel operations, users need to use threads. By creating new threads for operations, users can make the project go with the multi-tasking concept. However, users also have to pay attention to no priority among threads in execution, and each operates independently when with multiple ones.

The main function to be called first during the project running acts as a thread also. What's different about this thread from others is that the main function called by this thread and other customized functions called in the main function come with the privilege to call robot motion functions. Calling robot motion functions is not granted to other newly created threads. This is to ensure the consistency of the robot motion process because each thread processes independently. It is easy to confuse or interrupt the robot motion processes if granting other threads to use the robot motion functions.

Other than the other threads, the main thread comes with the privilege of the robot motion function call. The other newly created threads are not allowed for the robot motion function call. It is to ensure the consistency of the robot motion process. Since each thread operates independently, if others are for the robot motion function call, it gets the robot motion to be interrupted or confused easily.

In addition to the privilege of the robot motion function call, users can set the thread to whether continuously execute or not, which means if the project pausethread will be influenced by the or not. Once setting the thread to continuous execution, even if the project pauses, this thread continues to operate. Such a setting is suitable for communication applications.

When the thread-to-use works in a loop (with the loop statement), adding the additional sleep function to free up the thread-occupied CPU usage is requisite and recommended while working around threads, which is an attention must because excessive CPU usage may result in poor execution efficiency.

3.12.1 ThreadRun()

Create a new thread and use the new one to execute the statement. The previous one will continue proceeding.

Syntax 1

```
int ThreadRun(
    ?,
    bool
)
```

Parameters

?	statement or customized function	
bool	Whether the thread will continue the execution without being paused	
	true	continue the execution without being paused
	false	not continue the execution and to be paused (default)

Return

int	return the ID of the newly created thread.	
	> 0	created successfully
	<= 0	created unsuccessfully

*The ID of the newly created thread goes by the system, and it is not a fixed value. The thread number in the process will not be repetitive, but the number might be recurring once the thread stops.

Syntax 2

```
int ThreadRun(
    ?
)
```

Note

Same as Syntax 1. The default is to set whether the thread will continue the execution to false.

3.12.2 ThreadID()

Retrieve the ID of the current thread.

Syntax 1

```
int ThreadID(
)
```

Parameter

void	No parameter

Return

int	return the ID of the thread

Note

```
main
{
        int tid = ThreadRun(ThreadTest1(), false)     Create a new thread and and continue executiing ThreadTest1().
        Sleep(1000)                                   The previous thread continues proceeding.
        ThreadRun(ThreadTest2(tid), true)             Create a new thread and and continue executiing ThreadTest2().
}
void ThreadTest1()
{
        Display("Hello ThreadTest1() " + ThreadID())
}
void ThreadTest2(int k)
{
        Display("Hello ThreadTest2() " + ThreadID() + " " + k)
}
// Hello ThreadTest1() 18
// Hello ThreadTest2() 61 18
```

The following description denotes the execution order of creating new threads for running statements.

```
main
{
        int tid = ThreadID()
        ThreadRun(ThreadTest3(tid, ThreadID()), false)
}
void ThreadTest3(int k1, int k2)
{
        Display("Hello ThreadTest3() " + ThreadID() + " " + k1 + " " + k2)
}
// Hello ThreadTest3() 65 47 65
```

- The execution order goes by

1. Retrieve the thread ID in the main thread and assign the value the local variable tid.

2. Create a new thread and execute the statement of ThreadTest3(tid, ThreadID()).
3. As tid is a local variable, the value of tid is available to lead in.
4. Since ThreadID() is a function and it uses a new thread for the function call, the obtained thread ID will be different from the tid.
5. Again, call the ThreadTest3() function, and lead it and the obtained thread ID into k1 and k2 in the function, respectively
6. Call the ThreadID() function in the ThreadTest3() function to get the thread ID. Because it is the same thread as the ThreadTest3() function is in, k2 will go with the same ID and k1 with a different ID.

3.12.3 ThreadState()

Retrieve the status of the assigned thread ID.

Syntax 1

```
int ThreadState(
    int
)
```

Parameter

int	thread ID

Return

int return the status of the dedicate thread

-1	The thread does not exist.
0	The thread is in execution.
1	The thread is requesting to stop.
2	The thread stopped.

Syntax 2

```
int ThreadState(
)
```

Note

Same as Syntax 1. The default is to set the thread ID with the current thread ID of the function call.

3.12.4 ThreadExit()

Request the assigned thread ID to stop execution.

Syntax 1

```
int ThreadExit(
    int
)
```

Parameter

int	thread ID

Return

int return the result of requesting the assigned thread to stop

-1	The thread does not exist.
0	The requested thread stopped execution.

Syntax 2

```
int ThreadExit(
)
```

Note

Same as Syntax 1. The default is to set the thread ID with the current thread ID of the function call.

```
define
{
        int tid = 0
}
main
{
        int st = 0
        int t4 = ThreadRun(ThreadTest4())
        do
        {
                Sleep(100)
                st = ThreadState(t4)
                Display("t4 " + st)             // t4 0
        } while (st != 2)                       // Use the loop statement to wait for the thread t4 to stop.
        // Since there is no loop in ThreadTest4, the thread stops after the execution ends.
        Display("t4 " + st + " " + tid)         // t4 2 51
}
void ThreadTest4()
{
        Sleep(1000)
        tid = ThreadID()
}


define
{
        string title = ""
}
main
{
        int st = 0
        title = "t5 " + st                      // t5 0
        int t5 = ThreadRun(ThreadTest5())
        bool flag = WaitFor((st = ThreadState(t5)) == 2, 1000)
        // Use WaitFor to wait for the thread t5 to stop.
        title = "t5 " + st + " " + flag         // t5 0 false
        // Since there is a loop in ThreadTest5, the thread will not stop.
        // Therefore, WaitFor quits after a 1000 ms timeout. So, flag = false.
        Sleep(1000)
        if (flag == false)
        {
                ThreadExit(t5)                  // Request the thread t5 to quit.
        }
        flag = WaitFor((st = ThreadState(t5)) == 2, 1000)
        // Use WaitFor to wait for the thread t5 to stop.
        // Since the system requested the thread t5 to stop, the WaitFor condition sustains.
        Display("t5 " + st + " " + flag, "Hello main() " + ThreadID())  // t5 2 true
}
void ThreadTest5()
```

```
{
        while (true)        // the loop statement
        {
                Display(title , "Hello ThreadTest5() " + ThreadID())
                Sleep(100)
                // While using the loop statement, applying the additional sleep function is recommended to
                free up the thread-occupied CPU usage.
        }
}
```

4. General Functions

4.1 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

`byte[]`	Byte array
`int`	Byte array follows the 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 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[] bb1 = {0x90, 0x01, 0x05}
byte[] bb2 = {0x01} // Cause bb2[] does not fill 2 bytes. It would be filled to 2 bytes before transforming.

value = Byte_ToInt16(bb1, 0, 0) // 0x0190 value = 400
value = Byte_ToInt16(bb1, 0, 1) // 0x0190 value = 400
value = Byte_ToInt16(bb1, 1, 0) // 0x9001 value = -28671
value = Byte_ToInt16(bb1, 1, 1) // 0x9001 value = 36865
value = Byte_ToInt16(bb2, 0, 0) // 0x0001 value = 1
value = Byte_ToInt16(bb2, 0, 1) // 0x0001 value = 1
value = Byte_ToInt16(bb2, 1, 0) // 0x0100 value = 256
value = Byte_ToInt16(bb2, 1, 1) // 0x0100 value = 256
```

Syntax 2

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

Note

Similar to Syntax 1 with return value as signed int16

Byte_ToInt16(bb1, 0) => **Byte_ToInt16**(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(bb1)   =>   **Byte_ToInt16**(bb1, 0)

4.2 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[] bb1 = {0x01, 0x02, 0x03, 0x4F, 1}
byte[] bb2 = {0x01, 0x02, 0x03}
                                 // Cause bb2[] does not fill 4 bytes. It would be filled to 4 bytes before
                                 transforming.
value = Byte_ToInt32(bb1, 0)     // 0x4F030201 value = 1325597185
value = Byte_ToInt32(bb1, 1)     // 0x0102034F value = 16909135
value = Byte_ToInt32(bb2, 0)     // 0x00030201 value = 197121
value = Byte_ToInt32(bb2, 1)     // 0x01020300 value = 16909056
```

Syntax 2

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

Note

Similar to Syntax 1 with little endian input

Byte_ToInt32(bb1) => **Byte_ToInt32**(bb1, 0)

4.3 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[] bb1 = {0x01, 0x02, 0x03, 0x4F, 1}
byte[] bb2 = {0x01, 0x02, 0x03}  // Cause bb2[] does not fill 4 bytes. It would be filled to 4 bytes before
                                 transforming.
value = Byte_ToFloat(bb1, 0)     // 0x4F030201 value = 2.1979466E+09
value = Byte_ToFloat(bb1, 1)     // 0x0102034F value = 2.3879603E-38
value = Byte_ToFloat(bb2, 0)     // 0x00030201 value = 2.76225E-40
value = Byte_ToFloat(bb2, 1)     // 0x01020300 value = 2.387938E-38
```

Syntax 2

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

Note

Similar to Syntax 1 with little endian input

Byte_ToFloat(bb1) => **Byte_ToFloat**(bb1, 0)

4.4 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[] bb1 = {0x01, 0x02, 0x03, 0x4F, 1}   // Cause bb1[] does not fill 8 bytes. It would be filled to 8 bytes before
                                            transforming.
byte[] bb2 = {0x01, 0x02, 0x03} // Cause bb1[] does not fill 8 bytes. It would be filled to 8 bytes before
                                transforming.
value = Byte_ToDouble(bb1, 0) // 0x000000014F030201 value = 2.7769278203E-314
value = Byte_ToDouble(bb1, 1) // 0x0102034F01000000 value = 8.20840179153173E-304
value = Byte_ToDouble(bb2, 0) // 0x0000000000030201 value = 9.73907E-319
value = Byte_ToDouble(bb2, 1) // 0x0102030000000000 value = 8.207852449261364E-304
```

Syntax 2

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

Note

Similar to Syntax 1 with little endian input

Byte_ToDouble(bb1) => **Byte_ToDouble**(bb1, 0)

4.5 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[] 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[] bb2 = {1, 2, 3, 4}

value = Byte_ToInt16Array(bb1, 0, 0) // {0x0190, 0x0302, 0x0004} value = {400, 770, 4}
value = Byte_ToInt16Array(bb1, 0, 1) // {0x0190, 0x0302, 0x0004} value = {400, 770, 4}
value = Byte_ToInt16Array(bb1, 1, 0) // {0x9001, 0x0203, 0x0400} value = {-28671, 515, 1024}
value = Byte_ToInt16Array(bb1, 1, 1) // {0x9001, 0x0203, 0x0400} value = {36865, 515, 1024}

value = Byte_ToInt16Array(bb2, 0, 0) // {0x0201, 0x0403} value = {513, 1027}
value = Byte_ToInt16Array(bb2, 0, 1) // {0x0201, 0x0403} value = {513, 1027}
value = Byte_ToInt16Array(bb2, 1, 0) // {0x0102, 0x0304} value = {258, 772}
value = Byte_ToInt16Array(bb2, 1, 1) // {0x0102, 0x0304} value = {258, 772}
```

Syntax 2

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

Note

Similar to Syntax 1 with return value as signed int16

Byte_ToInt16Array(bb1, 0) => **Byte_ToInt16Array**(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(bb1) => **Byte_ToInt16Array**(bb1, 0)

4.6 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[] 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[] bb2 = {1, 2, 3, 4}

value = **Byte_ToInt32Array**(bb1, 0) // {0x04030201, 0x00000005} value = {67305985, 5}
value = **Byte_ToInt32Array**(bb1, 1) // {0x01020304, 0x05000000} value = {16909060, 83886080}
value = **Byte_ToInt32Array**(bb2, 0) // {0x04030201} value = {67305985}
value = **Byte_ToInt32Array**(bb2, 1) // {0x01020304} value = {16909060}

Syntax 2

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

Note

Similar to Syntax 1 with little endian input.

Byte_ToInt32Array(bb1) => **Byte_ToInt32Array**(bb1, 0)

4.7 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[] 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[] bb2 = {1, 2, 3, 4}

value = Byte_ToFloatArray(bb1, 0)
            // {0x04030201, 0x00000005} value = {1.5399896E-36,7E-45}
value = Byte_ToFloatArray(bb1, 1)
            // {0x01020304, 0x05000000} value = {2.3879393E-38,6.018531E-36}
value = Byte_ToFloatArray(bb2, 0) // {0x04030201} value = {1.5399896E-36}
value = Byte_ToFloatArray(bb2, 1) // {0x01020304} value = {2.3879393E-38}
```

Syntax 2

```
float[] Byte_ToFloatArray(
    byte[]
)
```

Note

Similar to Syntax 1 with little endian input

Byte_ToFloatArray(bb1) => **Byte_ToFloatArray**(bb1, 0)

4.8 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[] 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[] bb2 = {1, 2, 3, 4} // When the remaining part does not fill 8 byte, it would be filled to 8 bytes before transforming.

value = **Byte_ToDoubleArray**(bb1, 0) // {0x0000000504030201} value = {1.064323253E-313}
value = **Byte_ToDoubleArray**(bb1, 1) // {0x0102030405000000} value = {8.207880398492326E-304}
value = **Byte_ToDoubleArray**(bb2, 0) // {0x0000000004030201} value = {3.3253575E-316}
value = **Byte_ToDoubleArray**(bb2, 1) // {0x0102030400000000} value = {8.207880262684596E-304}

Syntax 2

```
double[] Byte_ToDoubleArray(
    byte[]
)
```

Note

Similar to Syntax 1 with little endian input

Byte_ToDoubleArray(bb1) => **Byte_ToDoubleArray**(bb1, 0)

4.9 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[] bb1 = {0x31, 0x32, 0x33, 0x00, 0x4F, 1}
byte[] bb2 = {0x01, 0x54, 0x4D, 0x35, 0xE6, 0xA9, 0x9F, 0xE5, 0x99, 0xA8, 0xE4, 0xBA, 0xBA}

value = **Byte_ToString**(bb1, 0) // value = "123" (UTF8 stop at 0x00)
value = **Byte_ToString**(bb1, 1) // value = "313233004F01"
value = **Byte_ToString**(bb1, 2) // value = "123" (ASCII stop at 0x00)
value = **Byte_ToString**(bb2, 0) // value = "\u01TM5機器人" (UTF8)
value = **Byte_ToString**(bb2, 1) // value = "01544D35E6A99FE599A8E4BABA"
value = **Byte_ToString**(bb2, 2) // 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(bb1) => **Byte_ToString**(bb1, 0)

4.10 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[] bb1 = {0x31, 0x32, 0x33, 0x00, 0x4F, 1}

value = Byte_Concat(bb1, 12) // 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[] bb1 = {0x31, 0x32, 0x33, 0x00, 0x4F, 1}
byte[] bb2 = {0x01, 0x02, 0x03}

value = Byte_Concat(bb1, bb2)
            // value = {0x31, 0x32, 0x33, 0x00, 0x4F, 0x01, 0x01, 0x02, 0x03}
```

Syntax 3

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

Parameters

`byte[]`	The input byte array1
`byte[]`	The input byte array2, would be concatenated after the end of array1
`int`	The number of element in array2 to be concatenated

0..the length of array2	Valid number
<0	Invalid. Length of array2 will be applied instead.
> the length of array2	Invalid. Length of array2 will be applied instead.

Return

byte[] Byte array formed from concatenating input arrays.

Note

byte[] bb1 = {0x31, 0x32, 0x33, 0x00, 0x4F, 1}
byte[] bb2 = {0x01, 0x02, 0x03}

```
value = Byte_Concat(bb1, bb2, 2)          // value = {0x31, 0x32, 0x33, 0x00, 0x4F, 0x01, 0x01, 0x02} //
                                          Concatenate only 2 elements from array2
value = Byte_Concat(bb1,   bb2, -1)       // value = {0x31, 0x32, 0x33, 0x00, 0x4F, 0x01, 0x01, 0x02, 0x03} // -1
                                          is invalid value
value = Byte_Concat(bb1, bb2, 10)         // value = {0x31, 0x32, 0x33, 0x00, 0x4F, 0x01, 0x01, 0x02, 0x03} // 10
                                          exceeds the array size

                                          // Length() can be utilized to acquire the array size
value = Byte_Concat(bb1, bb2, Length(bb2))
                                          // 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[] bb1 = {0x31, 0x32, 0x33, 0x00, 0x4F, 1}
byte[] bb2 = {0x01, 0x02, 0x03}

value = Byte_Concat(bb1, 1, 3, bb2, 1, 2)    // value = {0x32, 0x33, 0x00, 0x02, 0x03}
value = Byte_Concat(bb1, -1, 3, bb2, 3, -1)  // value = {0x31, 0x32, 0x33}
```

Syntax 5

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

Parameters *(active parameter amount)*

`byte`	The input byte value
`byte[]`	The input byte array

It concatenates the content of each parameter in order. It ignores the parameter if it is not a byte or a byte array and continues to concatenate the next parameter.

Return

`byte[]` Concatenate the parameters in byte and return a new byte array.

Note

```
byte[] bb1 = {0x31, 0x32, 0x00, 0x4F, 1}
byte[] bb2 = {0x01, 0x02, 0x03}
byte bb3 = 0x5A

value = Byte_Concat(bb1, bb2, bb3)          // value = {0x31,0x32,0x00,0x4F,0x01,0x01,0x02,0x03}    // Syntax 3
value = Byte_Concat(bb1, bb2, "", bb3)      // value = {0x31,0x32,0x00,0x4F,0x01,0x01,0x02,0x03,0x5A}
value = Byte_Concat(bb2, 0x10, bb1)         // value = {0x01,0x02,0x03,0x10,0x31,0x32,0x00,0x4F,0x01}
value = Byte_Concat(bb1, "AB", bb2, 10)     // value = {0x31,0x32,0x00,0x4F,0x01,0x01,0x02,0x03,0x0A}
                                            // Parameter "AB" is the string type. Ignored.
value = Byte_Concat(bb1, "AB", bb2, 1000)   // value = {0x31,0x32,0x00,0x4F,0x01,0x01,0x02,0x03}
                                            // Parameter "AB" is the string type. Ignored.
                                            // Parameter 1000 is the integer type. Ignored.
```

4.11 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

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

`string[]`	Input string array	
`int`	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

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

4.12 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

```
value = String_ToFloat("12.34", 10)        // value = 12.34
value = String_ToFloat("12.34", 16)        // value = 0          // Invalid hexadecimal format
value = String_ToFloat("12.34", 2)         // value = 0          // Invalid binary format
value = String_ToFloat("11.00", 2)         // value = 0          // Invalid binary format
value = String_ToFloat("0x1234", 10)       // value = 6.53E-42      // Hexadecimal format by auto detecting
value = String_ToFloat("0x1234", 16)       // value = 6.53E-42
value = String_ToFloat("0x1234", 2)        // value = 0             // Invalid binary format
value = String_ToFloat("0b1100", 10)       // value = 1.7E-44       // Binary format by auto detecting
value = String_ToFloat("0b1100", 16)       // value = 1.016289E-39      // Valid hexadecimal format
value = String_ToFloat("0b1100", 2)        // value = 1.7E-44
value = String_ToFloat("+12.34", 10)       // value = 12.34
value = String_ToFloat("-12.34", 10)       // value = -12.34
value = String_ToFloat("-0x1234", 16)      // value = 0          // Invalid hex format
value = String_ToFloat("-0b1100", 2)       // 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

`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 have to be identical

Return

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

Note

```
ss = {"12.345", "ab", "cc", "dd", "10.111"}
value = String_ToFloat(ss)          // value = {12.345,0,0,0,10.111}
value = String_ToFloat(ss, 2)       // value = {0,0,0,0,0}
value = String_ToFloat(ss, 16)      // value = {0,2.4E-43,2.86E-43,3.1E-43,0}
value = String_ToFloat(ss, 10)      // value = {12.345,0,0,0,10.111}
```

4.13 String_ToDouble()

Transform string to floating-point number (double type)

Syntax 1

```
double String_ToDouble(
    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

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

Note

```
value = String_ToDouble("12.34", 10)      // value = 12.34
value = String_ToDouble("12.34", 16)      // value = 0          // Invalid hexadecimal format
value = String_ToDouble("12.34", 2)       // value = 0          // Invalid binary format
value = String_ToDouble("11.00", 2)       // value = 0          // Invalid binary format
value = String_ToDouble("0x1234", 10)     // value = 2.3023E-320       // Hexadecimal format by auto
                                          detecting
value = String_ToDouble("0x1234", 16)     // value = 2.3023E-320
value = String_ToDouble("0x1234", 2)      // value = 0                 // Invalid binary format
value = String_ToDouble("0b1100", 10)     // value = 6E-323 // Binary format by auto detecting
value = String_ToDouble("0b1100", 16)     // value = 3.5832E-318       // Valid hexadecimal format
value = String_ToDouble("0b1100", 2)      // value = 6E-323
value = String_ToDouble("+12.34", 10)     // value = 12.34
value = String_ToDouble("-12.34", 10)     // value = -12.34
value = String_ToDouble("-0x1234", 16)    // value = 0          // Invalid hex format
value = String_ToDouble("-0b1100", 2)     // 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

```
ss = {"12.345", "ab", "cc", "dd", "10.111"}
value = String_ToDouble(ss)           // value = {12.345, 0, 0, 0, 10.111}
value = String_ToDouble(ss, 2)        // value = {0, 0, 0, 0, 0}
value = String_ToDouble(ss, 16)       // value = {0,8.45E-322,1.01E-321,1.09E-321,0}
value = String_ToDouble(ss, 10)       // value = {12.345, 0, 0, 0, 10.111}
```

4.14 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
2	ASCII	

Return

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

Note

value = **String_ToByte**("12345", 0)	// value = {0x31, 0x32, 0x33, 0x34, 0x35}
value = **String_ToByte**("12345", 1)	// value = {0x12, 0x34, 0x50} // the insufficient part will be filled with 0
value = **String_ToByte**("12345", 2)	// value = {0x31, 0x32, 0x33, 0x34, 0x35}
value = **String_ToByte**("0x12345", 0)	// value = {0x30, 0x78, 0x31, 0x32, 0x33, 0x34, 0x35}
value = **String_ToByte**("0x12345", 1)	// value = {0x00} // Only 0 be transformed, cause x is an invalid Hex value
value = **String_ToByte**("0x12345", 2)	// value = {0x30, 0x78, 0x31, 0x32, 0x33, 0x34, 0x35}
value = **String_ToByte**("TM5機器人", 0)	// value = {0x54, 0x4D, 0x35, 0xE6, 0xA9, 0x9F, 0xE5, 0x99, 0xA8, 0xE4, 0xBA, 0xBA}
value = **String_ToByte**("TM5機器人", 1)	// value = {0x00} // T is an invalid Hex value
value = **String_ToByte**("TM5機器人", 2)	// value = {0x54, 0x4D, 0x35, 0x3F, 0x3F, 0x3F}
value = **String_ToByte**("0123456", 1)	// value = {0x01, 0x23, 0x45, 0x60}
value = **String_ToByte**("01234G5", 1)	// 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)

4.15 String_IndexOf()

Search the address of the first occurrence of a specified strin from left to right.

Syntax 1

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

Parameters

`string`	Input string
`string`	The specified string to search
`int`	The initial address to start searching

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

Syntax 2

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

Note

Same as syntax 1. It defaults 0 to the initial address of the parameter int as searching from the leftmost.

```
int index = String_IndexOf("012314", "1")  // 1
index = String_IndexOf("012314", "")       // 0
index = String_IndexOf("012314", empty)    // 0
index = String_IndexOf("012314", "d")      // -1
index = String_IndexOf("", "d")            // -1
index = String_IndexOf("012314", "1", 1)   // 1   // Start searching with the 1st index number
index = String_IndexOf("012314", "1", 2)   // 4   // Start searching with the 2nd index number
index = String_IndexOf("012314", "1", 10)  // -1
```

Syntax 3

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

Parameters

`string`	The array of the input string
`string`	The specified string to search
`int`	The initial index of the array to start searching

Return

`int`	`0..Array Size -1`	Return the index of the string array if found the string.
	`-1`	Not found

* The array index will start searching with the initial index from left to right.

* Will use String_IndexOf(string, string) to search if the array element available. Return the index number of the array element but not the index number of the string.

Syntax 4

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

Note

Same as syntax 3. It defaults 0 to the initial index of the parameter int as searching from the foremost of the array elements.

```
string[] ss = {"012314", "ABCDEF", "123TM"}
int index = String_IndexOf(ss, "1")      // 0
index = String_IndexOf(ss, "")           // 0   // Since using String_IndexOf to search strings, it is available when
                                         searching for "".
index = String_IndexOf(ss, "d")          // -1
index = String_IndexOf(ss, "1", 1)       // 2
index = String_IndexOf(ss, "1", 10)      // -1
```

4.16 String_LastIndexOf()

Search the address of the last occurrence of a specified strin from right to left.

Syntax 1

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

Parameters

`string`	Input string
`string`	The specified string to search
`int`	The initial address to start searching

Return

`int`	`0..`(Length of string)`-1`	If the specified string is found, returns the index number
	`-1`	Not found
	(Length of string)	The specified string is "" or empty

Syntax 2

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

Note

Same as syntax 1. It defaults 0 to the initial address of the parameter int as searching from the rightmost.

```
int index = String_LastIndexOf("012314", "1")       // 4
index = String_LastIndexOf("012314", "")            // 6
index = String_LastIndexOf("012314", empty)         // 6
index = String_LastIndexOf("012314", "d")           // -1
index = String_LastIndexOf("", "d")                 // -1
index = String_LastIndexOf("012314", "1", 1)        // 1    // Start searching with the 1st index number
index = String_LastIndexOf("012314", "1", 2)        // 1    // Start searching with the 2nd index number
index = String_LastIndexOf("012314", "1", 10)       // 4
index = String_LastIndexOf("012314", "4", 10)       // 5
```

Syntax 3

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

Parameters

`string`	The array of the input string
`string`	The specified string to search
`int`	The initial index of the array to start searching

Return

`int`	`0..Array Size -1`	Return the index of the string array if found the string.
	`-1`	Not found

* The array index will start searching with the initial index from right to left.
* Will use String_IndexOf(string, string) to search if the array element available. Return the index number of the array element but not the index number of the string.

Syntax 4

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

Note

Same as syntax 3. It defaults 0 to the initial index of the parameter int as the array size as searching from the last of the array elements.

```
string[] ss = {"012314", "ABCDEF", "123TM"}
int index = String_LastIndexOf(ss, "1")        // 2
index = String_LastIndexOf(ss, "")             // 2    // Since using String_IndexOf to search strings, it is available
                                                when searching for "".
index = String_LastIndexOf(ss, "d")            // -1
index = String_LastIndexOf(ss, "1", 1)         // 0
index = String_LastIndexOf(ss, "1", 10)        // 2
```

4.17 String_DiffIndexOf()

Compare the address where the first string difference occurs starting from the start address.

Syntax 1

```
int String_DiffIndexOf(
    string,
    string,
    int
)
```

Parameters

`string`	Input string 1
`string`	Input string 2
`int`	The initial address to start comparing

Return

`int`	`0..`(Length of string)`-1`	If a difference is found, returns the index number of the difference.
	`-1`	No difference found. Namely, both strings match.
	`-2`	The initial address exceeds the length of two stings.

Syntax 2

```
int String_DiffIndexOf(
    string,
    string
)
```

Note

Same as syntax 1. It defaults 0 to the initial address of the parameter int as searching from the leftmost.

```
string s1 = "abcdef"
string s2 = "abcDef"
string s3 = "abcdeF123"
int index = String_DiffIndexOf(s1, s2)              // 3
index = String_DiffIndexOf(s1, s3)                  // 5
index = String_DiffIndexOf(s1, s3, 5)               // 5   // Start comparing with index 5, f and F.
index = String_DiffIndexOf(s1, s3, 7)               // 7   // Start comparing with index 7, '\0' and 2.
index = String_DiffIndexOf(s1, "abcdef", 7)         // -1  // Both strings match.
index = String_DiffIndexOf(s1, "ABCDEF", 7)         // -2  // Start comparing with index 7. Exceeded the length of
                                                    two strings.
```

4.18 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

```
value = String_Substring("0x12345", 2, 4)      // value = "1234"
value = String_Substring("0x12345", -1, 4)     // value = ""
value = String_Substring("0x12345", 7, 4)      // value = ""
value = String_Substring("0x12345", 2, -1)     // value = "12345"
value = String_Substring("0x12345", 2, 100)    // 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)

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

```
value = String_Substring("0x12345", "", "", 0)             // value = "0x12345"
value = String_Substring("0x12345", "1", "4", 1)           // value = "1234"
value = String_Substring("0x12345", "1", "4", 2)           // value = ""
value = String_Substring("0x12345", "1", "4", 0)           // value = ""
value = String_Substring("0x123450x12-345", "1", "4", 1)   // value = "1234"
value = String_Substring("0x123450x12-345", "1", "4", 2)   // value = "12-34"
value = String_Substring("0x123450x12-345", "1", "4", 3)   // value = ""
value = String_Substring("0x12345122", "1", "", 1)         // value = "12345122"   // Retrieves what's after
                                                           the matched prefix
value = String_Substring("0x12345122", "1", "", 2)         // value = "122"   // Retrieves what's after the
                                                           matched prefix. The matched amount moves from
```

the front to the back.

value = **String_Substring**("0x12345122", "1", "", 4) // value = ""

value = **String_Substring**("0x12345433", "", "4", 1) // value = "0x123454" // Retrieves what's after the matched suffix

value = **String_Substring**("0x12345433", "", "4", 2) // value = "0x1234" // Retrieves what's after the matched suffix. The matched amount moves from the back to the front.

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)

4.19 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, it returns a substring with array. [0] = empty and deals with empty strings by separators.

If separator is empty, it returns substring with array [0] = Input string and deals with empty strings by separators.

Note

```
value = String_Split("0x112345", "1", 0)    // value = {"0x","","2345"}
value = String_Split("0x112345", "1", 1)    // value = {"0x","2345"}
value = String_Split("", "1", 0)            // value = {""}     // length = 1
value = String_Split("", "1", 1)            // value = {}       // length = 0
value = String_Split("0x112345", "", 0)     // value = {"0x112345"}

s1 = "123, ""456,67"",89"
value = String_Split(s1, ",", 0)            // value = {"123", """456", "67""", "89"}   // length = 4
value = String_Split(s1, ",", 2)            // 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)

4.20 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

value = **String_Replace**("0x112345", "1", "2") // value = "0x222345"
value = **String_Replace**("0x112345", "", "2") // value = "0x112345"
value = **String_Replace**("0x112345", "1", "") // value = "0x2345"

4.21 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
)
```

Parameters

`string`	Input string

Return

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

Note

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

```
string s1 = "Hello    Hello    World    Hello    World"
string s2 = "HelloHelloWorldHelloWorld"
value = String_Trim(s1, "Hello")              // value = "    Hello    World    Hello    World"
value = String_Trim(s1, "World")              // value = "Hello    Hello    World    Hello    World"
value = String_Trim(s1, "", "Hello")          // value = "Hello    Hello    World    Hello    World"
value = String_Trim(s1, "", "World")          // value = "Hello    Hello    World    Hello    "
value = String_Trim(s1, "Hello", "World")     // value = "    Hello    World    Hello    "
value = String_Trim(s2, "Hello")              // value = "WorldHelloWorld"
value = String_Trim(s2, "World")              // value = "HelloHelloWorldHelloWorld"
value = String_Trim(s2, "", "Hello")          // value = "HelloHelloWorldHelloWorld"
value = String_Trim(s2, "", "World")          // value = "HelloHelloWorldHello"
value = String_Trim(s2, "Hello", "World")     // value = "WorldHello"
```

4.22 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

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

4.23 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

value = **String_ToUpper**("0x11Acz34") // value = "0X11ACZ34"

4.24 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[]  n1 = {100, 200, 30}
    byte[] n2 = {40, 50, 60}
    n3 = Array_Append(n1, n2)            // n3 = {100, 200, 30, 40, 50, 60}
    n1 = Array_Append(n1, 100)           // n1 = {100, 200, 30, 100}
    n1 = Array_Append(n1, n3)            // n1 = {100, 200, 30, 100, 100, 200, 30, 40, 50, 60}

?   float[] n1 = {1.1, 2.2, 3.3}
    float[] n2 = {0.4, 0.5}
    n3 = Array_Append(n1, n2)            // n3 = {1.1, 2.2, 3.3, 0.4, 0.5}
    n4 = Array_Append(n3, 5.678)         // n4 = {1.1, 2.2, 3.3, 0.4, 0.5, 5.678}

?   string[] n1 = {"123", "ABC", "456", "DEF"}
    string[] n2 = {"ABC", "123", "XYZ"}
    n3 = Array_Append(n1, n2)            // n3 = {"123", "ABC", "456", "DEF", "ABC", "123", "XYZ"}
    n4 = Array_Append(n2, "Hello World") // n4 = {"ABC", "123", "XYZ", "Hello World"}
```

4.25 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[] n1 = {100, 200, 30}
    int[] n2 = {40, 50, 60}
    n3 = Array_Insert(n1, 0, n2)      // n3 = {40, 50, 60, 100, 200, 30}    // Insert to the index 0
    n4 = Array_Insert(n1, 2, n2)      // n4 = {100, 200, 40, 50, 60, 30}    // Insert to the index 2
    n5 = Array_Insert(n1, -1, n2)     // n5 = {}    // The project will stop by error. Illegal index to start with

?   double[] n1 = {1.4, 2.6, 3.9}
    double[] n2 = {0.5, 0.7}
    n3 = Array_Insert(n1, 1, n2)      // n3 = {1.4, 0.5, 0.7, 2.6, 3.9}
    n4 = Array_Insert(n3, 4, 1.2345)// n4 = {1.4, 0.5, 0.7, 2.6, 1.2345, 3.9}
    n5 = Array_Insert(n3, 100, 9)     // 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.
```

4.26 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.

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

Return

?[]	The new array with elements removed after the index staring address.

Syntax2

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

Note

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

```
?   int[] n1 = {100, 200, 30, 40, 50, 60}
    n3 = Array_Remove(n1, -1)       // n3 = {}   // The project will stop by error. Illegal value to start with.
    n4 = Array_Remove(n1, 100)      // n4 = {}   // The project will stop by error. Illegal value to start with.
    n5 = Array_Remove(n1, 0)        // n5 = {200, 30, 40, 50, 60}   // Remove index 0
    n6 = Array_Remove(n1, 1, 2)     // n6 = {100, 40, 50, 60}       // Remove 2 elements from index 1
    n7 = Array_Remove(n1, 1, 100)      // n7 = {100}        // Remove 100 elements from index 1 (remove to the
                                       end of the array)
    n8 = Array_Remove(n1, Length(n1)-1)     // n8 = {100,200,30,40,50,60}      // Remove from the last of index
    n9 = Array_Remove(n1, Length(n1))    // n9 = {}   // The project will stop by error. Illegal value to start with.
```

4.27 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 arry1)-1)
`?[]`	Input array2 (Data type can be byte, int, float, double, bool, string)
`int`	The starting index of array2 (0 .. (length of arry2)-1)
`int`	The number of elements to be compared (0: return true, <0: return false)
	* 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[] n1 = {100, 200, 30}
    byte[] n2 = {100, 200, 30}
    Array_Equals(n1, n2)                    // true
    Array_Equals(n1, 0, n2, 0, 3)           // true
    Array_Equals(n1, 0, n2, 0, Length(n2))  // true

?   int[] n1 = {1000, 2000, 3000}
    int[] n2 = {1000, 2000, 3000, 4000}
    Array_Equals(n1, n2)                    // false
    Array_Equals(n1, 0, n2, 0, Length(n2))  // false    // compare 4 elements
    Array_Equals(n1, 0, n2, 0, 3)           // true
?   float[] n1 = {1.1, 2.2, 3.3}
```

```
    float[] n2 = {1.1, 2.2}
    Array_Equals(n1, n2)                      // false
    Array_Equals(n1, 0, n2, 0, Length(n2))    // true     // compare 2 elements
    Array_Equals(n1, 0, n2, 0, Length(n1))    // false
?   double[] n1 = {100, 200, 300, 3.3, 2.2, 1.1}
    double[] n2 = {100, 200, 400, 3.3, 2.2, 4.4}
    Array_Equals(n1, n2)                      // false
    Array_Equals(n1, 0, n2, 0, Length(n2))    // false
    Array_Equals(n1, 0, n2, 0, 2)             // true
    Array_Equals(n1, 3, n2, 3, 2)             // true
?   bool[] n1 = {true, false, true, true, true}
    bool[] n2 = {true, false, true, false, true}
    Array_Equals(n1, n2)                      // false
    Array_Equals(n1, 0, n2, 0, -1)            // false
    Array_Equals(n1, 0, n2, 0, 0)             // true     // compare 0 element
?   string[] n1 = {"123", "ABC", "456", "DEF"}
    string[] n2 = {"123", "ABC", "456", "DEF"}
    Array_Equals(n1, n2)                      // true
    Array_Equals(n1, -1, n2, 0, 4)            // false    // Invalid starting index
```

4.28 Array_IndexOf()

Search for the index number of the first occurrence within array elements.

Syntax 1

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

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.)
`int`	The initial index of the array to start searching

Return

`int`	`0..`(length of input array)`-1`	If the element is found , it returns the index number.
	`-1`	No element found

Syntax 2

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

Note

Same as syntax 1. It defaults 0 to the initial address of the parameter int as searching from the foremost.

```
?   byte[] n = {100, 200, 30, 100}
    value = Array_IndexOf(n, 200)       // 1
    value = Array_IndexOf(n, 2000)      // -1 // Since 2000 is not a byte type, it will be converted to int[], int to
                                        search.
    value = Array_IndexOf(n, 100, 1)    // 3

?   int[] n = {1000, 2000, 3000, 1000}
    value = Array_IndexOf(n, 200)       // -1
    value = Array_IndexOf(n, 1000)      // 0
    value = Array_IndexOf(n, 1000, 1)   // 3

?   float[] n = {1.1, 2.2, 3.3, 1.1}
    value = Array_IndexOf(n, 1.1)       // 0
    value = Array_IndexOf(n, 4.4)       // -1
    value = Array_IndexOf(n, 1.1, 1)    // 3

?   double[] n = {100, 200, 300, 3.3, 2.2, 1.1, 100}
    value = Array_IndexOf(n, 1.1)       // 5
    value = Array_IndexOf(n, 500)       // -1
    value = Array_IndexOf(n, 100, 1)    // 6

?   bool[] n = {true, false, true, true, true}
    value = Array_IndexOf(n, true)      // 0
    value = Array_IndexOf(n, false)          // 1
    value = Array_IndexOf(n, false, 2)       // -1
    value = Array_IndexOf(n, true, 2)        // 2
```

?

```
string[] n = {"123", "ABC", "456", "DEF", "123"}
value = Array_IndexOf(n, "456")          // 2
value = Array_IndexOf(n, "789")          // -1
value = Array_IndexOf(n, "123", 1)       // 4
value = Array_IndexOf(n, "AB")           // -1
```

4.29 Array_LastIndexOf()

Search for the index number of the last occurrence within array elements.

Syntax 1

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

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.)
`int`	The initial index of the array to start searching

Return

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

Syntax 2

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

Note

Same as syntax 1. It defaults 0 to the initial address of the parameter int as searching from the foremost.

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

?   byte[] n = {100, 200, 30, 100}
    value = Array_LastIndexOf(n, 200)          // 1
    value = Array_LastIndexOf(n, 2000)         // -1 // Since 2000 is not a byte type, it will be converted to int[], int
                                               to search.
    value = Array_LastIndexOf(n, 100, 1)       // 0

?   int[] n = {1000, 2000, 3000, 1000}
    value = Array_LastIndexOf(n, 200)          // -1
    value = Array_LastIndexOf(n, 1000)         // 3
    value = Array_LastIndexOf(n, 1000, 1)      // 0

?   float[] n = {1.1, 2.2, 3.3, 1.1}
    value = Array_LastIndexOf(n, 1.1)          // 3
    value = Array_LastIndexOf(n, 4.4)          // -1
    value = Array_LastIndexOf(n, 1.1, 1)       // 0

?   double[] n = {100, 200, 300, 3.3, 2.2, 1.1, 100}
    value = Array_LastIndexOf(n, 1.1)          // 5
    value = Array_LastIndexOf(n, 500)          // -1
    value = Array_LastIndexOf(n, 100, 1)       // 0

?   bool[] n = {true, false, true, true, true}
    value = Array_LastIndexOf(n, true)         // 4
```

```
        value = Array_LastIndexOf(n, false)          // 1
        value = Array_LastIndexOf(n, false, 2)       // 1
        value = Array_LastIndexOf(n, true, 2)        // 2

?       string[] n = {"123", "ABC", "456", "DEF", "123"}
        value = Array_LastIndexOf(n, "456")          // 2
        value = Array_LastIndexOf(n, "789")          // -1
        value = Array_LastIndexOf(n, "123", 1)       // 0
        value = Array_LastIndexOf(n, "AB")           // -1
```

4.30 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[] n = {100, 200, 30}
    n = Array_Reverse(n)        // n = {30, 200, 100}
?   int[] n = {1000, 2000, 3000}
    n = Array_Reverse(n)        // n = {3000, 2000, 1000}
?   float[] n = {1.1, 2.2, 3.3}
    n = Array_Reverse(n)        // n = {3.3, 2.2, 1.1}
?   double[] n = {100, 200, 300, 3.3, 2.2, 1.1}
    n = Array_Reverse(n)        // n = {1.1, 2.2, 3.3, 300, 200, 100}
?   bool[] n = {true, false, true, true, true}
    n = Array_Reverse(n)        // n = {true, true, true, false, true}
?   string[] n = {"123", "ABC", "456", "DEF"}
    n = Array_Reverse(n)        // 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[] n = {100, 200, 30}
    n = Array_Reverse(n,   2)  // n = {200, 100, 30}
                                          // 2 elements as a section, that is {100,200}{30}
    n = Array_Reverse(n,   4)  // n = {30, 200, 100}
                                          // 4 elements as a section, that is {100,200,30}
    n = Array_Reverse(n,   8)  // n = {30, 200, 100}
?   int[] n = {100, 200, 300, 400}
    n = Array_Reverse(n,   2)  // n = {200, 100, 400, 300}
```

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

4.31 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[] n =   {1000, 2000, 3000}
    n =   Array_Sort(n )          // n =   {1000, 2000, 3000}
?   double[] n =   {100, 200, 300, 3.3, 2.2, 1.1}
    n =   Array_Sort(n , 1)       // n =   {300, 200, 100, 3.3, 2.2, 1.1}
?   bool[] n =   {true, false, true, true, true}
    n =   Array_Sort(n , 1)       // n =   {true, true, true, true, false}
?   string[] n =   {"123", "ABC", "456", "DEF"}
    n =   Array_Sort(n )          // n =   {"123", "456", "ABC", "DEF"}
```

4.32 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[] n = {100, 200, 30}
    n1 = Array_SubElements(n1 , 0)      // n1 = {100, 200, 30}
    n1 = Array_SubElements(n1 , -1)     // n1 = {}
    n1 = Array_SubElements(n1 , 0, 3)   // n1 = {100, 200, 30}
    n1 = Array_SubElements(n1 , 1, 3)   // n1 = {200, 30}
    n1 = Array_SubElements(n1 , 2)      // n1 = {30}
    n1 = Array_SubElements(n1 , 3, 3)   // n1 = {}
?   int[] n = {1000, 2000, 3000}
    n1 = Array_SubElements(n1 , 0)      // n1 = {1000, 2000, 3000}
    n1 = Array_SubElements(n1 , -1)     // n1 = {}
    n1 = Array_SubElements(n1 , 1, 3)   // n1 = {2000, 3000}
    n1 = Array_SubElements(n1 , 2)      // n1 = {3000}
?   float[] n = {1.1, 2.2, 3.3}
    n1 = Array_SubElements(n1 , 0)      // n1 = {1.1, 2.2, 3.3}
    n1 = Array_SubElements(n1 , -1)     // n1 = {}
    n1 = Array_SubElements(n1 , 1, 3)   // n1 = {2.2, 3.3}
    n1 = Array_SubElements(n1 , 2)      // n1 = {3.3}
?   double[] n = {100, 200, 3.3, 2.2, 1.1}
    n1 = Array_SubElements(n1 , 0)      // n1 = {100, 200, 3.3, 2.2, 1.1}
    n1 =  Array_SubElements(n1 , -1)    // n1 =  {}
    n1 =  Array_SubElements(n1 , 1, 3)  // n1 =  {200, 3.3, 2.2}
    n1 =  Array_SubElements(n1 , 2)     // n1 =  {3.3, 2.2, 1.1}
?   bool[] n =  {true, false, true, true, true}
    n1 =  Array_SubElements(n1 , 0)     // n1 =  {true, false, true, true, true}
    n1 =  Array_SubElements(n1 , -1)    // n1 =  {}
    n1 =  Array_SubElements(n1 , 1, 3)  // n1 =  {false, true, true}
    n1 =  Array_SubElements(n1 , 2)     // n1 =  {true, true, true}
?   string[] n =  {"123", "ABC", "456", "DEF"}
    n1 =  Array_SubElements(n1 , 0)     // n1 =  {"123", "ABC", "456", "DEF"}
    n1 =  Array_SubElements(n1 , -1)    // n1 =  {}
    n1 =  Array_SubElements(n1 , 1, 3)  // n1 =  {"ABC", "456", "DEF"}
    n1 =  Array_SubElements(n1 , 2)     // n1 =  {"456", "DEF"}
```

4.33 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 i = 10000
value = ValueReverse(i, 0) // 10000=0x00002710  →  0x10270000   // value = 270991360
value = ValueReverse(i, 1) // 10000=0x2710       →  0x1027        // value = 4135
i = 100000                 // int32 value
value = ValueReverse(i, 0) // 100000=0x000186A0  →  0xA0860100 // value = -1601830656
value = ValueReverse(i, 1) // 100000=0x000186A0  →  0xA0860100 // value = -1601830656
value = ValueReverse(i, 2) // 100000=0x000086A0  →  0xA0860000 // 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 i = 40000

value = **ValueReverse**(i)   // 40000=0x471C4000 → 0x00401C47 // 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 i = 80000

value = **ValueReverse**(i)   // 80000=0x40F3880000000000 → 0x000000000088F340 // 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 i = "ABCDEF"

value = **ValueReverse**(i)   // 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[] i = {10000, 20000, 60000, 80000}

value = **ValueReverse**(i, 0) // value = {270991360, 541982720, 1625948160, -2143813376}

value = **ValueReverse**(i, 1) // value = {4135, 8270, 1625948160, -2143813376}

value = **ValueReverse**(i, 2) // 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[] i = {10000, 20000}

value = **ValueReverse**(i) // 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[] i = {10000, 20000}

value = **ValueReverse**(i) // value = {4.428251E-317,4.430275E-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[] i = {"ABCDEFG", "12345678"}

value = **ValueReverse**(i) // value = {"GFEDCBA", "87654321"}

4.34 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 n = 100
    value = GetBytes(n)          // value = {0x64}
    value = GetBytes(n, 0)       // value = {0x64}
    value = GetBytes(n, 1)       // value = {0x64}

?   byte[] n = {100, 200}        // Convert every element of the array to byte, 1 byte as a single unit.
    value = GetBytes(n)          // value = {0x64, 0xC8}
    value = GetBytes(n, 0)       // value = {0x64, 0xC8}
    value = GetBytes(n, 1)       // value = {0x64, 0xC8}

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

?   int[] n = {10000, 20000, 80000}
    // Convert every single element of the array to byte. For int32 data, works on 4 bytes sequentially.
    value = GetBytes(n)
    // value = {0x10, 0x27, 0x00, 0x00, 0x20, 0x4E, 0x00, 0x00, 0x80, 0x38, 0x01, 0x00}
    value = GetBytes(n, 0)
```

// value = {0x10, 0x27, 0x00, 0x00, 0x20, 0x4E, 0x00, 0x00, 0x80, 0x38, 0x01, 0x00}
value = **GetBytes**(n, 1)
// value = {0x00, 0x00, 0x27, 0x10, 0x00, 0x00, 0x4E, 0x20, 0x00, 0x01, 0x38, 0x80}

? float
value = **GetBytes**(123.456, 0) // value = {0x79, 0xE9, 0xF6, 0x42}
float n = -1.2345
value = **GetBytes**(n, 0) // value = {0x19, 0x04, 0x9E, 0xBF}
value = **GetBytes**(n, 1) // value = {0xBF, 0x9E, 0x04, 0x19}

? float[] n = {1.23, 4.56, -7.89}
// Convert every single element of the array to byte. For float data, works on 4 bytes sequentially.
value = **GetBytes**(n)
// value = {0xA4, 0x70, 0x9D, 0x3F, 0x85, 0xEB, 0x91, 0x40, 0xE1, 0x7A, 0xFC, 0xC0}
value = **GetBytes**(n, 0)
// value = {0xA4, 0x70, 0x9D, 0x3F, 0x85, 0xEB, 0x91, 0x40, 0xE1, 0x7A, 0xFC, 0xC0}
value = **GetBytes**(n, 1)
// value = {0x3F, 0x9D, 0x70, 0xA4, 0x40, 0x91, 0xEB, 0x85, 0xC0, 0xFC, 0x7A, 0xE1}

? double n = -1.2345
value = **GetBytes**(n, 0) // value = {0x8D, 0x97, 0x6E, 0x12, 0x83, 0xC0, 0xF3, 0xBF}
value = **GetBytes**(n, 1) // value = {0xBF, 0xF3, 0xC0, 0x83, 0x12, 0x6E, 0x97, 0x8D}

? double[] n = {1.23, -7.89}
// Convert every single element of the array to byte. For double data, works on 8 bytes sequentially.
value = **GetBytes**(n)
// value = {0xAE,0x47,0xE1,0x7A,0x14,0xAE,0xF3,0x3F,0x8F,0xC2,0xF5,0x28,0x5C,0x8F,0x1F,0xC0}
value = **GetBytes**(n, 0)
// value = {0xAE,0x47,0xE1,0x7A,0x14,0xAE,0xF3,0x3F,0x8F,0xC2,0xF5,0x28,0x5C,0x8F,0x1F,0xC0}
value = **GetBytes**(n, 1)
// value = {0x3F,0xF3,0xAE,0x14,0x7A,0xE1,0x47,0xAE,0xC0,0x1F,0x8F,0x5C,0x28,0xF5,0xC2,0x8F}

? bool flag = true //GetBytes converts true to 1, and false to 0.
value = **GetBytes**(flag) // value = {1}
value = **GetBytes**(flag, 0) // value = {1} // Because bool is 1 byte, Endian Parameters are not sufficient.
value = **GetBytes**(flag, 1) // value = {1}

? bool[] flag = {true, false, true, false, false, true, true}
value = **GetBytes**(flag) // value = {1, 0, 1, 0, 0, 1, 1}
value = **GetBytes**(flag, 0) // value = {1, 0, 1, 0, 0, 1, 1}
value = **GetBytes**(flag, 1) // value = {1, 0, 1, 0, 0, 1, 1}

? string n = "ABCDEFG" // string to encode in UTF8
value = **GetBytes**(n) // value = {0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47}
value = **GetBytes**(n, 0) // value = {0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47} // Endian Parameters not valid
value = **GetBytes**(n, 1) // value = {0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47} // Endian Parameters not valid

? string[] n = {"ABC", "DEF", "達明機器人" }
value = **GetBytes**(n)
// value = {0x41, 0x42, 0x43, 0x44, 0x45, 0x46,

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

value = **GetBytes**(n, 1)

// 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[] can maintain the original contents without separation bytes, 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

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

Syntax 4

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

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

Parameters

`int[]`	The input integer array (int[] type)
`int`	The input integer array follows Little Endian or Big Endian `0` Little Endian (Default) `1` Big Endian
`int`	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

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

4.35 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`	When the output string's notation is decimal, hexadecimal, or binary, the output string value is in decimal, hexadecimal, or binary.

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

?   byte[] n = {12, 34, 56}
    value = GetString(n)         // value = "{12,34,56}"
    value = GetString(n, 10)     // value = "{12,34,56}"
    value = GetString(n, 16)     // value = "{0x0C,0x22,0x38}"
    value = GetString(n, 2)      // value = "{0b00001100,0b00100010,0b00111000}"
    value = GetString(n, 16, 3)  // value = "{C,22,38}"
    value = GetString(n, 2, 2)   // value = "{0b1100,0b100010,0b111000}"

?   int n = 1234
    value = GetString(n)         // value = "1234"
    value = GetString(n, 10)     // value = "1234"
    value = GetString(n, 16)     // value = "0x000004D2"
    value = GetString(n, 2)      // value = "0b00000000000000000000010011010010"
    value = GetString(n, 16, 3)  // value = "4D2"
    value = GetString(n, 2, 2)   // value = "0b10011010010"

?   int[] n = {123, 345, -123, -456}
    value = GetString(n)         // value = "{123,345,-123,-456}"
    value = GetString(n, 10)     // value = "{123,345,-123,-456}"
    value = GetString(n, 16)     // value = "{0x0000007B,0x00000159,0xFFFFFF85,0xFFFFFE38}"
    value = GetString(n, 2)      // value = "{0b00000000000000000000000001111011,
                                              0b00000000000000000000000101011001,
                                              0b11111111111111111111111110000101,
                                              0b11111111111111111111111000111000}"
    value = GetString(n, 16, 3)  // value = "{7B,159,FFFFFF85,FFFFFE38}"
    value = GetString(n, 2, 2)   // value = "{0b1111011,
                                              0b101011001,
                                              0b11111111111111111111111110000101,
                                              0b11111111111111111111111000111000}"

?   float n = 12.34
    value = GetString(n)         // value = "12.34"
    value = GetString(n, 10)     // value = "12.34"
    value = GetString(n, 16)     // value = "0x414570A4"
```

```
        value = GetString(n, 2)       // value = "0b01000001010001010111000010100100"
        value = GetString(n, 16, 3) // value = "414570A4"
        value = GetString(n, 2, 2)   // value = "0b1000001010001010111000010100100"

?       float[] n = {123.4, 345.6, -123.4, -456.7}
        value = GetString(n)          // value = "{123.4,345.6,-123.4,-456.7}"
        value = GetString(n, 10)      // value = "{123.4,345.6,-123.4,-456.7}"
        value = GetString(n, 16)      // value = "{0x42F6CCCD,0x43ACCCCD,0xC2F6CCCD,0xC3E4599A}"
        value = GetString(n, 16, 3) // value = "{42F6CCCD,43ACCCCD,C2F6CCCD,C3E4599A}"

?       double n = 12.34
        value = GetString(n)          // value = "12.34"
        value = GetString(n, 10)      // value = "12.34"
        value = GetString(n, 16)      // value = "0x4028AE147AE147AE"
        value = GetString(n, 16, 3) // value = "4028AE147AE147AE"

?       double[] n = {123.45, 345.67, -123.48, -456.79}
        value = GetString(n)          // value = "{123.45,345.67,-123.48,-456.79}"
        value = GetString(n, 10)      // value = "{123.45,345.67,-123.48,-456.79}"
        value = GetString(n, 16)      // value = "{0x405EDCCCCCCCCCCD,0x40759AB851EB851F,
                                               0xC05EDEB851EB851F,0xC07C8CA3D70A3D71}"
        value = GetString(n, 16, 3) // value = "{405EDCCCCCCCCCCD,40759AB851EB851F,
                                               C05EDEB851EB851F,C07C8CA3D70A3D71}"

?       bool n = true
        value = GetString(n)          // value = "true"
        value = GetString(n, 16)      // value = "true"
        value = GetString(n, 2)       // value = "true"
        value = GetString(n, 16, 3) // value = "true"

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

?       string n = "1234567890"
        value = GetString(n)          // value = "1234567890"
        value = GetString(n, 16)      // value = "1234567890"
        value = GetString(n, 2)       // value = "1234567890"
        value = GetString(n, 16, 3) // value = "1234567890"

?       string[] n = {"123.45", "345.67", "-12""3.48", "-45A6.79"}
        value = GetString(n)          // value = "{123.45,345.67,-12""3.48,-45A6.79}"
        value = GetString(n, 1)       // value = "{"123.45","345.67","-12""3.48","-45A6.79"}"
        value = GetString(n, 2)       // value = "{123.45,345.67,-12"3.48,-45A6.79}"        // -12""3.48 displayed as -12"3.48
        value = GetString(n, 16, 3) // value = "{123.45,345.67,-12""3.48,-45A6.79}"
        value = GetString(n, 10, 3) // 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)
int	When the output string's notation is decimal, hexadecimal, or binary, the output string value is in decimal, hexadecimal, or binary.

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	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 n = 123
     value = GetString(n)                // value = "123"
     value = GetString(n, ";", 10)       // value = "123"
     value = GetString(n, "-", 16)       // value = "0x7B"
     value = GetString(n, "#", 2)        // value = "0b01111011"
     value = GetString(n, ",", 16, 3)    // value = "7B"
     value = GetString(n, ",", 2, 2)     // value = "0b1111011"
     * Separator is effective to array input only.

?    byte[] n = {12, 34, 56}
     value = GetString(n, "-")                  // value = "12-34-56"
     value = GetString(n, Ctrl("\r\n"), 10)  // value = "12\u0D0A34\u0D0A56"
     value = GetString(n, newline, 16)         // value = "0x0C\u0D0A0x22\u0D0A0x38"
     value = GetString(n, NewLine, 2)          // value = "0b00001100\u0D0A0b00100010\u0D0A0b00111000"
     value = GetString(n, "-", 16, 3)          // value = "C-22-38"
     value = GetString(n, "-", 2, 2)           // value = "0b1100-0b100010-0b111000"
     * \u0D0A is Newline control character, not string value.

?    string[] n = {"123.45", "345.67", "-12""3.48", "-45A6.79"}
     value = GetString(n, "-")           // value = "123.45-345.67-"-12""3.48"-"-45A6.79""
     value = GetString(n, "-", 1)        // value = ""123.45"-"345.67"-"-12""3.48"-"-45A6.79""
     value = GetString(n, "-", 2)        // 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 n = 123
    value = GetString(n)                        // value = "123"
    value = GetString(n, "[0]=", ";", 10)       // value = "123"
    value = GetString(n, "[0]=", "-", 16)       // value = "0x7B"
    value = GetString(n, "[0]=", "#", 2)        // value = "0b01111011"
    * Index and sepapator are only effective to array input.
?   byte[] n = {12, 34, 56}
    value = GetString(n, "[0]=", "-")                  // value = "[0]=12-[1]=34-[2]=56"
    value = GetString(n, "[0]=", Ctrl("\r\n"), 10)     // value = "[0]=12\u0D0A[1]=34\u0D0A[2]=56"
    value = GetString(n, "[0]=", newline, 16)          // value = "[0]=0x0C\u0D0A[1]=0x22\u0D0A[2]=0x38"
    value = GetString(n, "[0]=", "-", 16, 3)           // value = "[0]=C-[1]=22-[2]=38"
    value = GetString(n, "[0]=", "-", 2, 2)            // value = "[0]=0b1100-[1]=0b100010-[2]=0b111000"
```

* "[0]=" Support numeric format strings

* \u0D0A is Newline control character, not string value.

4.36 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 substtring to retrieve	
	`>=1`	Retrieve the nth matched substring
	`-1`	Retrieve the last matched substring
`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.

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 n = "$abcd$1234$ABCD$"
value = GetToken(n, "", "", 0)          // value = "$abcd$1234$ABCD$"
value = GetToken(n, "$", "$")           // value = "$abcd$"
value = GetToken(n, "$", "$", 0)        // value = ""
value = GetToken(n, "$", "$", 1)        // value = "$abcd$"
value = GetToken(n, "$", "$", 2)        // value = "$ABCD$"
value = GetToken(n, "$", "$", 3)        // value = ""
value = GetToken(n, "$", "$", -1, 1)    // value = "ABCD"
value = GetToken(n, "$", "$", 1, 1)     // value = "abcd"
value = GetToken(n, "$", "$", 2, 1)     // value = "ABCD"
value = GetToken(n, "$", "", 1)         // value = "$abcd"
value = GetToken(n, "$", "", 2)         // value = "$1234"
value = GetToken(n, "$", "", 3)         // value = "$ABCD"
value = GetToken(n, "$", "", 4)         // value = "$"
value = GetToken(n, "", "$", 1)         // value = "$"
value = GetToken(n, "", "$", 2)         // value = "abcd$"
value = GetToken(n, "", "$", 3)         // value = "1234$"
value = GetToken(n, "", "$", 4)         // value = "ABCD$"
string n = "$abcd$1234$ABCD$" + Ctrl("\r\n") + "56\r\n78$"
value = GetToken(n, "$", Ctrl("\r\n"), 1)    // value = "$abcd$1234$ABCD$\u0D0A"
value = GetToken(n, "$", newline, 2)         // value = ""
value = GetToken(n, "$", NewLine, 1, 1)      // value = "abcd$1234$ABCD$"     // Remove prefix and suffix
value = GetToken(n, Ctrl("\r\n"), "$", 1)    // value = "\u0D0A56\r\n78$"
value = GetToken(n, newline, "$", 2)         // value = ""
value = GetToken(n, NewLine, "$", 1, 1)      // value = "56\r\n78"
```

* \u0D0A 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

string	Input string	
byte[]	Prefix. The leading element of the substring, byte[] type	
byte[]	Suffix. The trailing element of the substring, byte[] type	
int	The number of the matched substtring to retrieve	
	>=1	Retrieve the nth matched substring
	-1	Retrieve the last matched substring
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.

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 totol matached 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 n = "$abcd$1234$ABCD$"
byte[] bb0 = {}, bb1 = {0x24}       // 0x24 is $
value = GetToken(n, bb0, bb0, 0)       // value = "$abcd$1234$ABCD$"
value = GetToken(n, bb1, bb1)          // value = "$abcd$"
value = GetToken(n, bb1, bb1, 0)       // value = ""
value = GetToken(n, bb1, bb1, 1)       // value = "$abcd$"
value = GetToken(n, bb1, bb1, 2)       // value = "$ABCD$"
value = GetToken(n, bb1, bb1, 3)       // value = ""
value = GetToken(n, bb1, bb1, 1, 1)    // value = "abcd"
value = GetToken(n, bb1, bb1, 2, 1)    // value = "ABCD"
value = GetToken(n, bb1, bb0, 1)       // value = "$abcd"
value = GetToken(n, bb1, bb0, 2)       // value = "$1234"
value = GetToken(n, bb1, bb0, 3)       // value = "$ABCD"
value = GetToken(n, bb1, bb0, 4)       // value = "$"
value = GetToken(n, bb0, bb1, 1)       // value = "$"
value = GetToken(n, bb0, bb1, 2)       // value = "abcd$"
value = GetToken(n, bb0, bb1, 3)       // value = "1234$"
value = GetToken(n, bb0, bb1, 4)       // value = "ABCD$"

string n = "$abcd$1234$ABCD$" + Ctrl("\r\n") + "56\r\n78$"
byte[] bb0 = {0x0D,0x0A}, bb1 = {0x24}       // 0x24 is $ // 0x0D,0x0A is \u0D0A
value = GetToken(n, bb1, bb0, 1)       // value = "$abcd$1234$ABCD$\u0D0A"
value = GetToken(n, bb1, bb0, 2)       // value = ""
value = GetToken(n, bb1, bb0, 1, 1)    // value = "abcd$1234$ABCD$"      // Removing the prefix and the suffix
value = GetToken(n, bb0, bb1, 1)       // value = "\u0D0A56\r\n78$"
value = GetToken(n, bb0, bb1, 2)       // value = ""
value = GetToken(n, bb0, bb1, 1, 1)    // value = "56\r\n78"
```

* \u0D0A is the Newline control character, not the string content.

Syntax 7

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

Parameters

`byte[]`	The input byte[]
`string`	Prefix. The leading element of the output byte[], byte[] type

string	Suffix. The trailing element of the output byte[], byte[] type
int	The number of the matched substring to retrieve
	>=1 Retrieve the nth matched substring
	-1 Retrieve the last matched substring
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.

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 s ="$abcd$1234$ABCD$"
byte[] n = GetBytes(s)
value = GetToken(n, "", "", 0)
            // value = {0x24,0x61,0x62,0x63,0x64,0x24,0x31,0x32,0x33,0x34,0x24,0x41,0x42,0x43,0x44,0x24}
value = GetToken(n, "$", "$")          // value = {0x24,0x61,0x62,0x63,0x64,0x24}
value = GetToken(n, "$", "$", 0)       // value = {}
value = GetToken(n, "$", "$", 1)       // value = {0x24,0x61,0x62,0x63,0x64,0x24}
value = GetToken(n, "$", "$", 2)       // value = {0x24,0x41,0x42,0x43,0x44,0x24}
value = GetToken(n, "$", "$", 1, 1)    // value = {0x61,0x62,0x63,0x64}
value = GetToken(n, "$", "$", 2, 1)    // value = {0x41,0x42,0x43,0x44}
value = GetToken(n, "$", "", 1)        // value = {0x24,0x61,0x62,0x63,0x64}
value = GetToken(n, "$", "", 2)        // value = {0x24,0x31,0x32,0x33,0x34}
value = GetToken(n, "$", "", 3)        // value = {0x24,0x41,0x42,0x43,0x44}
value = GetToken(n, "$", "", 4)          // value = {0x24}
value = GetToken(n, "", "$", 1)          // value = {0x24}
value = GetToken(n, "", "$", 2)          // value = {0x61,0x62,0x63,0x64,0x24}
value = GetToken(n, "", "$", 3)          // value = {0x31,0x32,0x33,0x34,0x24}
value = GetToken(n, "", "$", 4)          // value = {0x41,0x42,0x43,0x44,0x24}
string s ="$abcd$1234$ABCD$" + Ctrl("\r\n") + "56\r\n78$"
byte[] n = GetBytes(s)
value = GetToken(n, "$", Ctrl("\r\n"), 1)
    // value = {0x24,0x61,0x62,0x63,0x64,0x24,0x31,0x32,0x33,0x34,0x24,0x41,0x42,0x43,0x44,0x24,0x0D,0x0A}
value = GetToken(n, "$", Ctrl("\r\n"), 1, 1)
    // value = {0x61,0x62,0x63,0x64,0x24,0x31,0x32,0x33,0x34,0x24,0x41,0x42,0x43,0x44,0x24}
    // Removing prefix and suffix
value = GetToken(n, Ctrl("\r\n"), "$", 1)
    // value = {0x0D,0x0A,0x35,0x36,0x5C,0x72,0x5C,0x6E,0x37,0x38,0x24}
value = GetToken(n, Ctrl("\r\n"), "$", 1, 1)
    // value = {0x35,0x36,0x5C,0x72,0x5C,0x6E,0x37,0x38}
```

Syntax 10

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

Parameters

byte[]	The input byte[] array
byte[]	Prefix. The leading element of the output byte[]
byte[]	Suffix. The trailing element of the output byte[]
int	The number of the matched substring to retrieve
	>=1 Retrieve the nth matched substring
	-1 Retrieve the last matched substring
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.

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 s ="$abcd$1234$ABCD$"
byte[] n = GetBytes(s)
byte[] bb0 = {}, bb1 = {0x24}       // 0x24 is $
value = GetToken(n, bb0, bb0, 0)
                // value = {0x24,0x61,0x62,0x63,0x64,0x24,0x31,0x32,0x33,0x34,0x24,0x41,0x42,0x43,0x44,0x24}
value = GetToken(n, bb1, bb1)          // value = {0x24,0x61,0x62,0x63,0x64,0x24}
value = GetToken(n, bb1, bb1, 0)       // value = {}
value = GetToken(n, bb1, bb1, 1)       // value = {0x24,0x61,0x62,0x63,0x64,0x24}
value = GetToken(n, bb1, bb1, 2)       // value = {0x24,0x41,0x42,0x43,0x44,0x24}
value = GetToken(n, bb1, bb1, 1, 1)    // value = {0x61,0x62,0x63,0x64}
value = GetToken(n, bb1, bb1, 2, 1)    // value = {0x41,0x42,0x43,0x44}
value = GetToken(n, bb1, bb0, 1)       // value = {0x24,0x61,0x62,0x63,0x64}
value = GetToken(n, bb1, bb0, 2)       // value = {0x24,0x31,0x32,0x33,0x34}
value = GetToken(n, bb1, bb0, 3)       // value = {0x24,0x41,0x42,0x43,0x44}
value = GetToken(n, bb0, bb1, 1)       // value = {0x24}
value = GetToken(n, bb0, bb1, 2)       // value = {0x61,0x62,0x63,0x64,0x24}
value = GetToken(n, bb0, bb1, 3)       // value = {0x31,0x32,0x33,0x34,0x24}
string s ="$abcd$1234$ABCD$" + Ctrl("\r\n") + "56\r\n78$"
byte[] n = GetBytes(s)
byte[] bb0 = {0x0D,0x0A}, bb1 = {0x24}
value = GetToken(n, bb1, bb0, 1)
    // value = {0x24,0x61,0x62,0x63,0x64,0x24,0x31,0x32,0x33,0x34,0x24,0x41,0x42,0x43,0x44,0x24,0x0D,0x0A}
value = GetToken(n, bb1, bb0, 1, 1)
    // value = {0x61,0x62,0x63,0x64,0x24,0x31,0x32,0x33,0x34,0x24,0x41,0x42,0x43,0x44,0x24}
    // Remove prefix and suffix
value = GetToken(n, bb0, bb1, 1)
    // value = {0x0D,0x0A,0x35,0x36,0x5C,0x72,0x5C,0x6E,0x37,0x38,0x24}
value = GetToken(n, bb0, bb1, 1, 1)
    // value = {0x35,0x36,0x5C,0x72,0x5C,0x6E,0x37,0x38}
```

4.37 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 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.

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 n = "$abcd$1234$ABCD$"
value = GetAllTokens(n, "", "")           // value = {"$abcd$1234$ABCD$"}
value = GetAllTokens(n, "$", "$")         // value = {"$abcd$", "$ABCD$"}
value = GetAllTokens(n, "$", "$", 1)      // value = {"abcd", "ABCD"}
value = GetAllTokens(n, "$", "")          // value = {"$abcd", "$1234", "$ABCD", "$"}
value = GetAllTokens(n, "", "$", 1)       // value = {"", "abcd", "1234", "ABCD"}

string n = "#abcd$1234#ABCD$5678#"
value = GetAllTokens(n, "", "", 0)        // value = {"#abcd$1234#ABCD$5678#"}
value = GetAllTokens(n, "$", "", 0)       // value = {"$1234#ABCD","$5678#"}
value = GetAllTokens(n, "$", "", 1)       // value = {"1234#ABCD","5678#"}
value = GetAllTokens(n, "$", "", 2)       // value = {}
                                          // $ is not in the start of the input string. Returns an empty array.
value = GetAllTokens(n, "$", "", 3)       // value = {}
                                          // $ is not in the 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 of the input string. Returns an empty array.
value = GetAllTokens(n, "$", "$", 3)    // value = {}
                                        // $ is not in the 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"}
```

4.38 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 remains 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 errors in format setting, the default format will be applied as MM/dd/yyyy HH:mm:ss.

Note

```
value = GetNow("MM/dd/yyyy HH:mm:ss")          // value = 08/15/2017 13:40:30
value = GetNow("yyyy/MM/dd HH:mm:ss.ffff")     // value = 2017/08/15 13:40:30.1337
value = GetNow("yyyy-MM-dd hh:mm:ss tt")       // value = 2017-08-15 01:40:30 PM
```

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

```
value = GetNow()            // value = 08/15/2017 13:40:30
```

4.39 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

```
value = GetNowStamp()          // value = 2147483647
... others ...
value = GetNowStamp()          // 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

```
value = GetNowStamp(false)          // value = 2147483647
... others ...
value = GetNowStamp(false)          // value = -1        // Over flow
value = GetNowStamp(true)           // 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

```
value = GetNowStamp()          // value = 2147483546
... others ... (After 100ms)
diff = GetNowStamp(value)      // diff = 100
... others ... (After 200ms)
```

diff = **GetNowStamp**(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

value = **GetNowStamp**() // value = 2147483546
… others … (After 100ms)
diff = **GetNowStamp**(value) // diff = 100
… others … (After 200ms)
diff = **GetNowStamp**(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

value = **GetNowStamp**() // value = 41730494
… others … (After 60ms)
flag = **GetNowStamp**(value, 100) // diff = 60 // flag = false
… others … (After 60ms)
flag = **GetNowStamp**(value, 100) // diff = 120 // 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

```
value = GetNowStamp()                 // value = 41730494
… others … (After 60ms)
flag = GetNowStamp(value, 100)        // diff = 60          // flag = false
… others … (After 60ms)
flag = GetNowStamp(value, 100)        // diff = 120         // flag = true
```

4.40 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 World"      // 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.
```

4.41 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 n = 100
    value = Length(n)          // value = 1
    value = Length(100)        // value = 1
?   int n = 400
    value = Length(n)          // value = 4
    value = Length(400)        // value = 4
?   float n = 1.234
    value = Length(n)          // value = 4
    value = Length(1.234)      // value = 4
?   double n = 1.234
    value = Length(n)          // value = 8
    value = Length(1.234)      // value = 4
                               // float // Numbers would be stored as the smaller data type first.
?   bool n = true
    value = Length(n)          // value = 1
    value = Length(false)      // value = 1
?   string n = "A""BC"
    value = Length(n)          // value = 4
                               // The string is A"BC. Two double quotation marks represent " in string
    value = Length("")         // value = 0
    value = Length("123")      // value = 3
    value = Length(empty)      // value = 0
?   byte[] n = {100, 200, 30}
    value = Length(n)          // value = 3
?   int[] n = {}
    value = Length(n)          // value = 0
    n = {400, 500, 600}
    value = Length(n)          // value = 3
?   float[] n = {1.234}
    value = Length(n)          // value = 1
?   double[] n = {1.234, 200, -100, +300}
    value = Length(n)          // value = 4
?   bool[] n = {true, false, true, true, true, true, false}
```

```
        value = Length(n)               // value = 7
?       string[] n = {"A""BC", "123", "456", "ABC"}
        value = Length(n)               // value = 4
```

4.42 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

```
b = 0x0D0A
value = Ctrl(b)                // value = \r\n
value = Ctrl(0x0D0A)           // value = \r\n
value = Ctrl(0x0D000A09)       // value = \r\n\t   // 0x00 will not be transformed
value = Ctrl(0x0D300A09)       // value = \r0\n\t // 0x30 is transformed to 0
value = Ctrl(0x00)             // 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

```
b = "\r\n"
value = Ctrl(b)                // value = \r\n
value = Ctrl("\r\n")           // value = \r\n
value = Ctrl("\r\n\t")         // value = \r\n\t
value = Ctrl("\r0\n\t")        // value = \r0\n\t
value = Ctrl("\0")             // 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[] bb1 = {0xFF,0x55,0x31,0x32,0x33,0x00,0x35,0x36,0x0D,0x0A}
value = Ctrl(bb1)          // value = �U123 56\r\n
byte[] bb2 = {}
value = Ctrl(bb2)          // value = ""
```

4.43 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[] bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}
value = XOR8(bb1,0,Length(bb1))     // value = 0x6F
value = XOR8(bb1,0,-1)              // value = 0x6F
value = XOR8(bb1,1,-1)              // value = 0x7F
value = XOR8(bb1,-1,-1)             // value = 0
```

Syntax 2

```
byte XOR8(
    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[] bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}
value = XOR8(bb1,0,Length(bb1))          // value = 0x6F
value = XOR8(bb1,0)                      // value = 0x6F
value = XOR8(bb1)                        // value = 0x6F
```

```
bb1 = Byte_Concat(bb1, XOR(bb1))          // bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF, 0x6F}
```

4.44 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[] bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}
value = SUM8(bb1,0,Length(bb1))     // value = 0x6D
value = SUM8(bb1,0,-1)              // value = 0x6D
value = SUM8(bb1,1,-1)              // value = 0x5D
value = SUM8(bb1,-1,-1)             // 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[] bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}
value = SUM8(bb1,0,Length(bb1))          // value = 0x6D
value = SUM8(bb1,0)                      // value = 0x6D
value = SUM8(bb1)                        // value = 0x6D
bb1 = Byte_Concat(bb1, SUM8(bb1))        // bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF, 0x6D}
```

4.45 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[] bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}
value = **SUM16**(bb1,0,Length(bb1))   // value = {0x04, 0x6D}
value = **SUM16**(bb1,0,-1)   // value = {0x04, 0x6D}
value = **SUM16**(bb1,1,-1)   // value = {0x04, 0x5D}
value = **SUM16**(bb1,-1,-1)   // 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[] bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}
value = **SUM16**(bb1,0,Length(bb1))   // value = {0x04, 0x6D}
value = **SUM16**(bb1,0)   // value = {0x04, 0x6D}
value = **SUM16**(bb1)   // value = {0x04, 0x6D}
bb1 = **Byte_Concat**(bb1, **SUM16**(bb1))   // bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF, 0x04, 0x6D}

4.46 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[] bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}
value = SUM32(bb1,0,Length(bb1))    // value = {0x00, 0x00, 0x04, 0x6D}
value = SUM32(bb1,0,-1)             // value = {0x00, 0x00, 0x04, 0x6D}
value = SUM32(bb1,1,-1)             // value = {0x00, 0x00, 0x04, 0x5D}
value = SUM32(bb1,-1,-1)            // 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[] bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}
value = SUM32(bb1,0,Length(bb1))        // value = {0x00, 0x00, 0x04, 0x6D}
value = SUM32(bb1,0)                    // value = {0x00, 0x00, 0x04, 0x6D}
value = SUM32(bb1)                      // value = {0x00, 0x00, 0x04, 0x6D}
bb1 = Byte_Concat(bb1, SUM32(bb1))      // bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x04, 0x6D}
```

4.47 CRC16()

Utilize CRC 16 bits algorithm to computes the checksum

Syntax 1

```
byte[] CRC16(
    int,
    byte[],
    int,
    int
)
```

Parameters

int	CRC16 algorithm	
	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[] bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}
value = CRC16(0, bb1,0,Length(bb1))     // value = {0x2D, 0xD4}
value = CRC16(0, bb1,0,-1)              // value = {0x2D, 0xD4}
value = CRC16(0, bb1,1,-1)              // value = {0xEC, 0xC5}
value = CRC16(0, bb1,-1,-1)             // value = {0x00, 0x00}
value = CRC16(3, bb1,0,Length(bb1))     // value = {0x42, 0x12}
value = CRC16(4, bb1,0,Length(bb1))     // 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[] bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}
value = CRC16(0, bb1,0,Length(bb1))          // value = {0x2D, 0xD4}
value = CRC16(0, bb1,0)                      // value = {0x2D, 0xD4}
value = CRC16(0, bb1)                        // value = {0x2D, 0xD4}
bb1 = Byte_Concat(bb1, CRC16(0, bb1))        // 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[]))

4.48 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[] bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}

value = CRC32(bb1,0,Length(bb1))     // value = {0x43, 0xD5, 0xB9, 0xF8}
value = CRC32(bb1,0,-1)              // value = {0x43, 0xD5, 0xB9, 0xF8}
value = CRC32(bb1,1,-1)              // value = {0x08, 0xA5, 0x5B, 0xEB}
value = CRC32(bb1,-1,-1)             // 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[]
)
```

Note

Similar to Syntax1 with computing all the elements of the array

CRC32(byte[]) => **CRC32**(byte[], 0, Length(byte[]))

```
byte[] bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF}
value = CRC32(bb1,0,Length(bb1))          // value = {0x43, 0xD5, 0xB9, 0xF8}
value = CRC32(bb1,0)                      // value = {0x43, 0xD5, 0xB9, 0xF8}
value = CRC32(bb1)                        // value = {0x43, 0xD5, 0xB9, 0xF8}
```

```
bb1 = Byte_Concat(bb1, CRC32(bb1))
            // bb1 = {0x10, 0x20, 0x50, 0xF0, 0xFF, 0xFF, 0xFF, 0x43, 0xD5, 0xB9, 0xF8}
```

4.49 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"

value = ListenPacket("TMSCT", var_data1)       // $TMSCT,9,1,var_i++,*06\r\n
value = ListenPacket("", var_data2)            // $TMSCT,11,Hello World,*51\r\n      // Error for TMSCT
value = ListenPacket("", "2,Techman Robot")    // $TMSCT,15,2,Techman Robot,*57\r\n
value = ListenPacket("TMSTA", var_data2)       // $TMSTA,11,Hello World,*53\r\n      // Error for TMSTA
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
value = ListenPacket(var_data1)                // $TMSCT,15,1, var_counter++,*26\r\n
```

4.50 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 ip = "127.0.0.1"

byte b = 100
value = ListenSend(ip, 10, b)
    // send 0x64 to ipfilter "127.0.0.1"     // value = -2     // SubCmd must be between 90 and 99.
value = ListenSend(ip, 90, b)
    // send 0x64 to ipfilter "127.0.0.1"     // value = -1     // Supposedly Listen Server is not starting.
value = ListenSend(ip, 90, b)
    // send 0x64 to ipfilter "127.0.0.1"     // 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.
value = ListenSend(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)
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[] bb = {100, 200}
value = ListenSend(90, bb)
        // send 0x64 0xC8
        // $TMSTA,5,90,d�,*CF        // The value of {100, 200} is conversed to 0x64 0xC8
string[] ss = {"T", "M", "達明機器人" }
value = ListenSend(90, 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
```

4.51 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 i goes with "i". * If the variable is listed, the value of the variable will be used to send the matched object.	

Return

int	Sending times	
	> 0	Send success. The return value returns the sending times
	0	Send failed
	-1	TM Manager function is not enabled
	-9	Invalid Parameters

Note

```
string var_s = "ABC"
string var_s1 = " var_s"
string[] var_ss = {"ABC", " var_s", " var_s1"}
value = VarSync(1, 1000, " var_s")     // Send var_s variable object
value = VarSync(2, 2000, var_s)        // Send ABC variable object (Because the value of var_s is "ABC")
value = VarSync(3, 2000, var_ss)       // Send ABC, var_s, var_s1 variable object (From the value of ss string
                                       array)
value = VarSync(3, 2000, " var_ss")    // Send var_ss variable object
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, ?)

5. General Functions (Script)

5.1 Exit()

Stop the project running.

Syntax 1

```
bool Exit(
    bool,
    int
)
```

Parameter

bool	Whether to wait for the end of the motion command to stop the project	
	true	Wait (default)
	false	No wait
int	Ending code	
	> 0	Execute the closestop funciton.
	== 0	Evaluate if the project errs. (default)
	< 0	Execute the errorstop funciton.

Return

bool True Command accepted ; False Command rejected

Syntax 2

```
bool Exit(
    bool
)
```

Parameter

bool	Whether to wait for the end of the motion command to stop the project	
	true	Wait (default)
	false	No wait

Note

Same as syntax 1. It defaults 0 to the ending code.

Syntax 3

```
bool Exit(
)
```

Parameter

void No parameter

Return

bool True Command accepted ; False Command rejected

Note

Same as syntax 1. It defaults true to whether to wait for the end of the motion command to stop the project and 0 to the ending code.

Syntax 4

```
bool Exit(
    int
)
```

Parameter

int Ending code.

> 0	Execute the closestop funciton.
== 0	Same as syntax 1. Evaluates if the project errs.
< 0	Execute the errorstop funciton.

Return

`bool` `True` Command accepted ; `False` Command rejected

Note

Same as syntax 1. It defaults true to whether to wait for the end of the motion command to stop the project.

```
Exit()
// Wait for the end of the motion command, stop the project running, and assess if errors occurred.
// Run the closestop function next when there is no error.
// Run the errorstop function next when there is an error.
Exit(false)        // Wait not for the end of the motion command not, stop the project running shortly, and assess if
                   errors occurred.
Exit(false, 1)     // Wait not for the end of the motion command not, stop the project running shortly, and run the
                   closestop function.
Exit(1)     // Wait for the end of the motion command, stop the project running, and run the closestop function.
Exit(0)     // Wait for the end of the motion command, stop the project running, and assess if errors occurred.
Exit(-1)    // Wait for the end of the motion command, stop the project running, and run the errorstop function.
```

5.2 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.

Syntax 1

```
bool Pause(
)
```

Parameter

void	No parameter

Return

bool	True Command accepted ; False Command rejected

Note

Pause()

5.3 Resume()

Resume the project and the motion of the robot.

Syntax 1

```
bool Resume(
)
```

Parameter

`void`	No parameter

Return

`bool`	`True` Command accepted ; `False` Command rejected

Note

Resume()

5.4 WaitFor()

The loop wait condition stands or time out on waiting.

Syntax 1

```
bool WaitFor(
    bool,
    int
)
```

Parameter

bool	The loop wait condition. It can be true/false or a statement returns the bool value.	
int	wait time (millisecond)	
	< 0	wait indefinitely
	>= 0	time to wait

Return

bool	Return True for the loop wait condition stands and False for time out on waiting.

Note

```
int i = 0
bool flag = WaitFor(i++ > 100, 1000)
// The loop executes i++ and judges whether it is larger than 100. It exits the loop at once if the condition satisfies
(flag = true) or it is timeout after 1000ms (flag = false).
```

Syntax 2

```
bool WaitFor(
    int
)
```

Parameter

int	wait time (millisecond)	
	< 0	invalid
	>= 0	time to wait

Return

bool	Return True for the wait time stands and False for not. (Interrupted by the project stop)

Note

```
WaitFor(100)          // Timeout after 100ms of waiting.
```

5.5 Sleep()

Stop the thread in the time specified. Same as WaitFor(int).

Syntax 1

```
bool Sleep(
    int
)
```

Parameter

int	wait time (millisecond)	
	< 0	invalid
	>= 0	time to wait

Return

bool	Return True for the wait time stands and False for not. (Interrupted by the project stop)

Note

```
Sleep(100)          // Timeout after 100ms of waiting.
```

5.6 Display()

Display contents on the TMflow dashboard.

Syntax 1

```
bool Display(
    string,
    string,
    string,
    string
)
```

Parameter

`string`	Headline background color	
	`"Red"`	Set headline background color to red.
	`"Green"`	Set headline background color to green.
	`"Blue"`	Set headline background color to blue.
	`"Yellow"`	Set headline background color to yellow.
	`"Black"`	Set headline background color to black.
	`"White"`	Set headline background color to white.
	`"Gray"`	Set headline background color.to gray.
`string`	Headline text color	
	`"Red"`	Set headline text color to red.
	`"Green"`	Set headline text color to green.
	`"Blue"`	Set headline text color to blue.
	`"Yellow"`	Set headline text color to yellow.
	`"Black"`	Set headline text color to black.
	`"White"`	Set headline text color to white.
	`"Gray"`	Set headline text color to gray.
`string`	Headline text	
`string`	Text	

Return

`bool` Return `True` for set successfully and `False` for unsuccessfully.

Note

Display("Green", "Yellow", "Gripper Initial Finish", "Force = 30N") // Output the headline of Gripper Initial Finish with the text of Force = 30N to the TMflow dashboard and set headline text color to yellow and headline background color to green.

Syntax 2

```
bool Display(
    string,
    string
)
```

Note

Same as Syntax 1. It sets headline background color to white and headline text color to black by default.

Syntax 3

```
bool Display(
    string
)
```

Note

Same as Syntax 1. It sets headline background color to white and headline text color to black by default.

The headline text is an empty string.

6. Math Functions

6.1 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 i = 10
value = abs(i)    // 10
i = -10
value = abs(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 f = 10.1
value = abs(f)    // 10.1
f = -10.1
value = abs(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 d = 10.8
value = abs(d)    // 10.8
d = -10.8
value = abs(d)    // 10.8
```

6.2 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 b = 100
    value = pow(b, 2)     // 10000
    value = pow(b, -2)    // 0        // 0.0001, but int type
    value = pow(b, 0.1)   // 1        // 1.5848932, but int type
    value = pow(b, 2.1)   // 15848    // 15848.932, but int type
    value = pow(b, -2.1)  // 0        // 6.309574E-05, but int type
?   float b = -100
    value = pow(b, 2)     // 10000
    value = pow(b, -2)    // 0.0001
    value = pow(b, 0.2)   // Error // NaN
    value = pow(b, 2.2)   // Error // NaN
    value = pow(b, -2.2)  // Error // NaN
```

```
?    double b = 100
     value = pow(b, 2)        // 10000
     value = pow(b, -2)       // 0.0001
     value = pow(b, 0.31)     // 4.168694
     value = pow(b, 2.31)     // 41686.938
     value = pow(b, -2.31)    // 2.3988328E-05
```

6.3 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

```
value = sqrt(100)          // 10
value = sqrt(100.1234)     // 10.006168
value = sqrt(0.1234)       // 0.35128337
value = sqrt(-100)         // Error // NaN
value = sqrt(-100.1234)    // Error // NaN
value = sqrt(-0.1234)      // Error // NaN
```

6.4 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

```
value = ceil(100)            // 100
value = ceil(100.1234)       // 101
value = ceil(0.1234)         // 1
value = ceil(-100)           // -100
value = ceil(-100.1234)      // -100
value = ceil(-0.1234)        // 0
```

6.5 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
)
```

Parameter

`double`	input number in double

Return

`double`	Return a number in double rounded downward to its nearest integer

Note

```
value = floor(100)          // 100
value = floor(100.1234)     // 100
value = floor(0.1234)       // 0
value = floor(-100)         // -100
value = floor(-100.1234)    // -101
value = floor(-0.1234)      // -1
```

6.6 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)	
	0 .. 15	valid values
	< 0	value invalid, will use 0 by default
	> 15	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)

```
value = round(100)          // 100
value = round(100.456)      // 100
value = round(0.567)            // 1
value = round(-100)             // -100
value = round(-100.456)         // -100
value = round(-0.567)           // -1
value = round(100.345, 1)       // 100.3
value = round(100.345, 2)       // 100.35
value = round(-100.345, 1)      // -100.3
value = round(-100.345, 2)      // -100.35
value = round(-100.345, 16)     // -100
```

6.7 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 parameter

Return

`float`	Return a random number in float between 0 and 1.

Note

```
value = random()            // 0.9473762
value = random()            // 0.7764986
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

```
value = random(10)          // 8
value = random(10)          // 1
value = random(10)          // 5
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

```
value = random(5, 10)       // 8
value = random(5, 10)       // 8
value = random(5, 10)       // 6
value = random(5, -1)       // 5   // The upper bound is smaller than the lower bound. Returned the value of the
                            lower bound in integer.
```

6.8 sum()

Return the sum of the given numbers or the array of numbers.

Syntax 1

```
int sum(
    ?,
    ...
)
```

Parameter *(variable parameter amount)*

?	input value, can be in the type of byte, int, byte[], or int[] calculate the value of each parameter or each element in the array. It returns an error and stops if it comes with a non-numeric type.

Return

int	Return the sum, in the integer type, of the given numbers.

Syntax 2

```
double sum(
    ?,
    ...
)
```

Parameter *(variable parameter amount)*

?	input value, can be in the type of float, double, float[], or double[] calculate the value of each parameter or each element in the array. It returns an error and stops if it comes with a non-numeric type.

Return

double	Return the sum, in the double type, of the given numbers.

Note

```
int sum1 = sum(1,2,3,4,5)                                   // 15
int sum2 = sum(1,2,3,4,{5,6,7,8})                           // 36
int sum3 = sum(1,2,3,4,{5,6,7,8},1.2)                       // 37      // due to putting the int type
double sum4 = sum(1,2,3,4,{5,6,7,8},1.2)                    // 37.2
double sum5 = sum(1,2,3,4,{5,6,7,8},9.2)                    // 45.2
double sum6 = sum(1,2,3,4,{5,6,7,8},9.2,sum5,{1.2,3.4})     // 95
```

6.9 average()

Return the average of the given numbers or the array of numbers.

Syntax 1

```
double average(
    ?,
    ...
)
```

Parameter *(variable parameter amount)*

?	input value, can be in the type of byte, int, float, double, byte[], int[], float[], or double[] calculate the value of each parameter or each element in the array. It returns an error and stops if it comes with a non-numeric type.

Return

double	Return the average, in the double type, of the given numbers.

Note

```
double avg1 = average(1,2,3,4,5)                          // 3
double avg2 = average(1,2,3,4,{5,6,7,8},9.2)               // 5.02222222222222
double avg3 = average(1,2,3,4,sum({5,6,7,8}),9.2)          // 7.53333333333333
double avg4 = average(1,2,3,4,{5,6,7,8},9.2,{1.2,3.4})     // 4.52727272727273
```

6.10 stdevp()

Return the standard deviation $\sigma = \sqrt{\sum\left(X - \frac{\sum X}{N}\right)^2 / N}$ based on the entire population of the given numbers or the array of numbers.

Syntax 1

```
double stdevp(
    ?,
    ...
)
```

Parameter *(variable parameter amount)*

?	input value, can be in the type of byte, int, float, double, byte[], int[], float[], or double[] calculate the value of each parameter or each element in the array. It returns an error and stops if it comes with a non-numeric type.

Return

double	Return the standard deviation, in the double type, of the given numbers.

Note

```
double stdp1 = stdevp(1,2,3,4,5)                              // 1.4142135623731
double stdp2 = stdevp(1,2,3,4,{5,6,7,8},9.2)                  // 2.61694384000276
double stdp3 = stdevp(1,2,3,4,sum({5,6,7,8}),9.2)             // 8.66153694341958
double stdp4 = stdevp(1,2,3,4,{5,6,7,8},9.2,{1.2,3.4})        // 2.63165724111457
```

6.11 stdevs()

Return the standard deviation $s = \sqrt{\sum\left(X - \frac{\sum X}{N}\right)^2 / (N-1)}$ based on a sample of the given numbers or the array of numbers.

Syntax 1

```
double stdevs(
    ?,
    ...
)
```

Parameter *(variable parameter amount)*

?	input value, can be in the type of byte, int, float, double, byte[], int[], float[], or double[] calculate the value of each parameter or each element in the array. It returns an error and stops if it comes with a non-numeric type.

Return

double	Return the standard deviation, in the double type, of the given numbers.

Note

```
double stds1 = stdevs(1,2,3,4,5)                          // 1.58113883008419
double stds2 = stdevs(1,2,3,4,{5,6,7,8},9.2)              // 2.77568810287547
double stds3 = stdevs(1,2,3,4,sum({5,6,7,8}),9.2)         // 9.4882383331505
double stds4 = stdevs(1,2,3,4,{5,6,7,8},9.2,{1.2,3.4})    // 2.76010539983201
```

6.12 min()

Return the minimum of the given numbers or the array of numbers.

Syntax 1

```
int min(
    ?,
    ...
)
```

Parameter *(variable parameter amount)*

?	input value, can be in the type of byte, int, byte[], or int[] compare the value of each parameter or each element in the array. It returns an error and stops if it comes with a non-numeric type.

Return

int	Return the minimum, in the integer type, of the given numbers.

Syntax 2

```
double min(
    ?,
    ...
)
```

Parameter *(variable parameter amount)*

?	input value, can be in the type of float, double, float[], or double[] compare the value of each parameter or each element in the array. It returns an error and stops if it comes with a non-numeric type.

Return

double	Return the minimum, in the double type, of the given numbers.

Note

```
int min1 = min(1,2,3)                          // 1
int min2 = min(1,2,3,{-4,-5.3,6.2},1.2)        // -5       // due to putting the int type
double min3 = min(1,2,3,{-4,-5.3,6.2},1.2)     // -5.3
double min4 = min(1,2,3,{-0.2,-0.1,0.1},9.2)   // -0.2
```

6.13 max()

Return the maximum of the given numbers or the array of numbers.

Syntax 1

```
int max(
    ?,
    ...
)
```

Parameter *(variable parameter amount)*

?	input value, can be in the type of byte, int, byte[], or int[] compare the value of each parameter or each element in the array. It returns an error and stops if it comes with a non-numeric type.

Return

int	Return the maximum, in the integer type, of the given numbers.

Syntax 2

```
double max(
    ?,
    ...
)
```

Parameter *(variable parameter amount)*

?	input value, can be in the type of float, double, float[], or double[] compare the value of each parameter or each element in the array. It returns an error and stops if it comes with a non-numeric type.

Return

double	Return the maximum, in the double type, of the given numbers.

Note

```
int max1 = max(1,2,3)                         // 3
int max2 = max(1,2,3,{-4,-5.3,6.2},1.2)       // 6      // due to putting the int type
double max3 = max(1,2,3,{-4,-5.3,6.2},1.2)    // 6.2
double max4 = max(1,2,3,{-0.2,-0.1,0.1},9.2)  // 9.2
```

6.14 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

```
value = d2r(1)          // 0.017453292
```

6.15 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

```
value = r2d(1)          // 57.29578
```

6.16 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

```
value = sin(0)          // 0
value = sin(15)         // 0.25881904
value = sin(30)         // 0.5
value = sin(60)         // 0.8660254
value = sin(90)         // 1
```

6.17 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

```
value = cos(0)        // 1
value = cos(15)       // 0.9659258
value = cos(30)       // 0.8660254
value = cos(45)       // 0.70710677
value = cos(60)       // 0.5
```

6.18 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

```
value = tan(0)      // 0
value = tan(15)     // 0.2679492
value = tan(30)     // 0.57735026
value = tan(45)     // 1
value = tan(60)     // 1.7320508
```

6.19 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

```
value = asin(0)              // 0
value = asin(0.258819)       // 14.999998
value = asin(0.5)            // 30
value = asin(0.8660254)      // 60
value = asin(1)              // 90

value = asin(sin(15))        // 15
value = asin(sin(60))        // 60
```

6.20 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

```
value = acos(1)               // 0
value = acos(0.9659258)       // 15.000003
value = acos(0.8660254)       // 30.000002
value = acos(0.7071068)       // 44.999996
value = acos(0.5)             // 60

value = acos(cos(15))         // 15.000003
value = acos(cos(30))         // 30.000002
value = acos(cos(45))         // 45

value = acos(cos((double)15))    // 14.9999999999999996
value = acos(cos((double)30))    // 29.9999999999999993
```

6.21 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

```
value = atan(0)             // 0
value = atan(0.2679492)     // 15
value = atan(0.5773503)     // 30.000002
value = atan(1)             // 45
value = atan(1.732051)      // 60.000004

value = atan(tan(30))       // 30
value = atan(tan(60))       // 60
```

6.22 atan2()

Return the arctangent of the quotient of it 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

```
value = atan2(2, 1)        // 63.434948
value = atan2(1, 1)        // 45
value = atan2(-1, -1)      // -135
value = atan2(4, -3)       // 126.869896
```

6.23 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

```
value = log(16, 2)          // 4
value = log(16, 8)          // 1.3333334
value = log(16, 10)         // 1.20412
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

```
value = log(16, 2)          // 4
value = log(16)             // 2.7725887
value = log(2)              // 0.6931472
value = log(16)/log(2)      // 2.7725887/0.6931472 = 3.9999998557305
```

6.24 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

```
value = log(16, 10)       // 1.20412
value = log10(16)         // 1.20412
value = log(500, 10)      // 2.69897
value = log10(500)        // 2.69897
```

6.25 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[] vector1 = {3,4}
float[] vector2 = {3,4,5}
float[] vector3 = {3,4,5,6,8}
value = norm2(vector1)          // 5
value = norm2(vector2)          // 7.071068
value = norm2(vector3)          // 12.247449
```

6.26 dist()

Return the distance between the two coordinates.

Syntax 1

```
float dist(
    float[],
    float[]
)
```

Parameter

float[]	The first coordinate $\{X_{1(mm)}\quad Y_{1(mm)}\quad Z_{1(mm)}\quad RX_{1(°)}\quad RY_{1(°)}\quad RZ_{1(°)}\}$
float[]	The second coordinate $\{X_{2(mm)}\quad Y_{2(mm)}\quad Z_{2(mm)}\quad RX_{2(°)}\quad RY_{2(°)}\quad RZ_{2(°)}\}$

Return

float	The distance between the two coordinates

Note

```
float[] c1 = {100,200,100,30,50,20}
float[] c2 = {100,100,100,50,50,10}
value = dist(c1, c2)            // 100
```

6.27 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	$\{X_{1(mm)} \quad Y_{1(mm)} \quad Z_{1(mm)} \quad RX_{1(°)} \quad RY_{1(°)} \quad RZ_{1(°)}\}$
`float[]`	Second Point	$\{X_{2(mm)} \quad Y_{2(mm)} \quad Z_{2(mm)} \quad RX_{2(°)} \quad RY_{2(°)} \quad RZ_{2(°)}\}$
`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

$\{X_{trans} \quad Y_{trans} \quad Z_{trans} \quad RX_{trans} \quad RY_{trans} \quad RZ_{trans}\}$

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 = $\begin{bmatrix} R & P \\ 0 & 1 \end{bmatrix}$

$$R_n = \begin{bmatrix} cos(RZ_n)\,cos(RY_n) & -sin(RZ_n)cos(RX_n) + cos(RZ_n)sin(RY_n)sin(RX_n) & sin(RZ_n)sin(RX_n) + cos(RZ_n)sin(RY_n)cos(RX_n) \\ sin(RZ_n)cos(RY_n) & cos(RZ_n)cos(RX_n) + sin(RZ_n)sin(RY_n)sin(RX_n) & -cos(RZ_n)sin(RX_n) + sin(RZ_n)sin(RY_n)cos(RX_n) \\ -sin(RY_n) & cos(RY_n)sin(RX_n) & cos(RY_n)cos(RX_n) \end{bmatrix}$$

First Point = $\begin{bmatrix} R_1 & P_1 \\ 0 & 1 \end{bmatrix}$

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,-24.615868,-15.565178,-88.613686}
float[] var_trans_i = trans(var_P1, var_P2, true)
        // {176.10095,588.32776,-308.80237,3.7459252,23.13792,-92.46916}
```

6.28 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 $\{X_o \quad Y_o \quad Z_o \quad RX_o \quad RY_o \quad RZ_o\}$
`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 $\{\mathrm{X_{inv}} \quad \mathrm{Y_{inv}} \quad \mathrm{Z_{inv}} \quad \mathrm{RX_{inv}} \quad \mathrm{RY_{inv}} \quad \mathrm{RZ_{inv}}\}$ opposite to the input displacement and rotation angle $\{\mathrm{X_o} \quad \mathrm{Y_o} \quad \mathrm{Z_o} \quad \mathrm{RX_o} \quad \mathrm{RY_o} \quad \mathrm{RZ_o}\}$ 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,-300,12.483133,-18.590115,-60.283367}
float[] var_inv_i = inversetrans(var_P1, true)
    // {218.38095,142.13216,-268.52972,12.483133,-18.590115,-60.283367}
```

6.29 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 $\{X_i \quad Y_i \quad Z_i \quad RX_i \quad RY_i \quad RZ_i\}$	
`float[]`	the displacement and rotation angle $\{X_o \quad Y_o \quad Z_o \quad RX_o \quad RY_o \quad RZ_o\}$	
`bool`	The reference coordinate	
	`false`	Refer to the robot's base(default)
	`true`	Refer to the initial point

Return

`float[]`	the terminal point $\{X_t \quad Y_t \quad Z_t \quad RX_t \quad RY_t \quad RZ_t\}$ 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.

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)

$$P_t = P_i + P_{trans}$$
$$R_t = R_{trans} * R_i$$

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,-24.615868,-15.565178,-88.613686}
float[] var_trans_i = trans(var_P1, var_P2, true)
        // {176.10095,588.32776,-308.80237,3.7459252,23.13792,-92.46916}

float[] var_apply_RB = applytrans(var_P1, var_trans_RB)
        // {-400,200,50,-20,30,-44.999996}
float[] var_apply_i = applytrans(var_P1, var_trans_i, true)
        // {-400,200,50.000015,-20,30,-45}

float[] var_inv_RB = inversetrans(var_P1)
        // {-100,200,-300,12.483133,-18.590115,-60.283367}
float[] var_inv_i = inversetrans(var_P1, true)
        // {218.38095,142.13216,-268.52972,12.483133,-18.590115,-60.283367}

float[] var_apply_1 = applytrans(var_P1, var_inv_RB)
        // {0,0,0,-4.8045007E-07,1.4295254E-07,4.8365365E-07}
float[] var_apply_2 = applytrans(var_P1, var_inv_i, true)
        // {-3.845248E-06,6.1641267E-06,1.4917891E-06,-1.8922562E-07,-2.1415798E-07,6.352311E-07}
```

6.30 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 $\{X_{1(mm)} \quad Y_{1(mm)} \quad Z_{1(mm)} \quad RX_{1(°)} \quad RY_{1(°)} \quad RZ_{1(°)}\}$
`float[]`	Second Point $\{X_{2(mm)} \quad Y_{2(mm)} \quad Z_{2(mm)} \quad RX_{2(°)} \quad RY_{2(°)} \quad RZ_{2(°)}\}$
`float`	Ratio

Return

`float[]`	the linear interpolate point $\{X_i \quad Y_i \quad Z_i \quad RX_i \quad RY_i \quad RZ_i\}$ between initial point and endpoint according to the ratio. Return an empty array if unable to calculate.

Note

$$\{X_i \quad Y_i \quad Z_i \quad RX_i \quad RY_i \quad RZ_i\} \\ = (\{X_2 \quad Y_2 \quad Z_2 \quad RX_2 \quad RY_2 \quad RZ_2\} - \{X_1 \quad Y_1 \quad Z_1 \quad RX_1 \quad RY_1 \quad RZ_1\}) \times Ratio \\ + \{X_1 \quad Y_1 \quad Z_1 \quad RX_1 \quad RY_1 \quad RZ_1\}$$

```
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[] interp = interpoint(var_P1, var_P2, 0.5)
        // {-412.67075,-42.035904,369.254,172.91898,-20.690556,-69.33843}
```

6.31 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[]
)
```

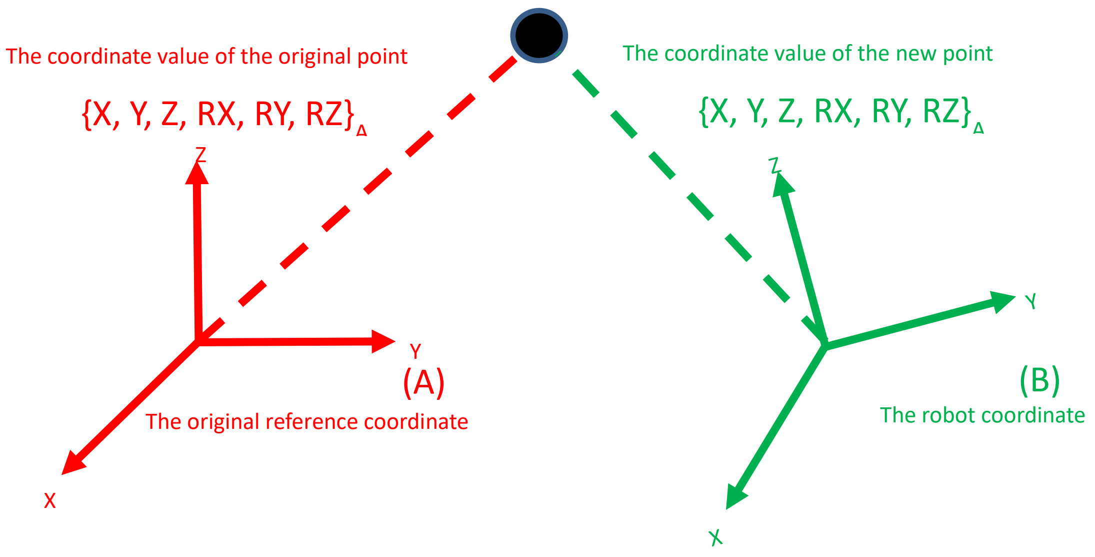
Parameter

`float[]`	The coordinate value of the original point $\{X_o \quad Y_o \quad Z_o \quad RX_o \quad RY_o \quad RZ_o\}_A$
`float[]`	The original reference coordinate system $\{X_{oa} \quad Y_{oa} \quad Z_{oa} \quad RX_{oa} \quad RY_{oa} \quad RZ_{oa}\}_A$
`float[]`	The new reference coordinate system $\{X_n \quad Y_n \quad Z_n \quad RX_n \quad RY_n \quad RZ_n\}_B$

Return

`float[]`	The coordinate value of the new point $\{X_{nb} \quad Y_{nb} \quad Z_{nb} \quad RX_{nb} \quad RY_{nb} \quad RZ_{nb}\}_B$ Return an empty array if unable to calculate.

Note


The coordinate value of the original point
{X, Y, Z, RX, RY, RZ}_A
Z
Y
X
(A)
The original reference coordinate
The coordinate value of the new point
{X, Y, Z, RX, RY, RZ}_A
Z
Y
X
(B)
The robot coordinate

```
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)
// f0 = {0.0020519744,1.9731047E-05,-0.0022721738,113.94231,14.9346,-123.19886}
// 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

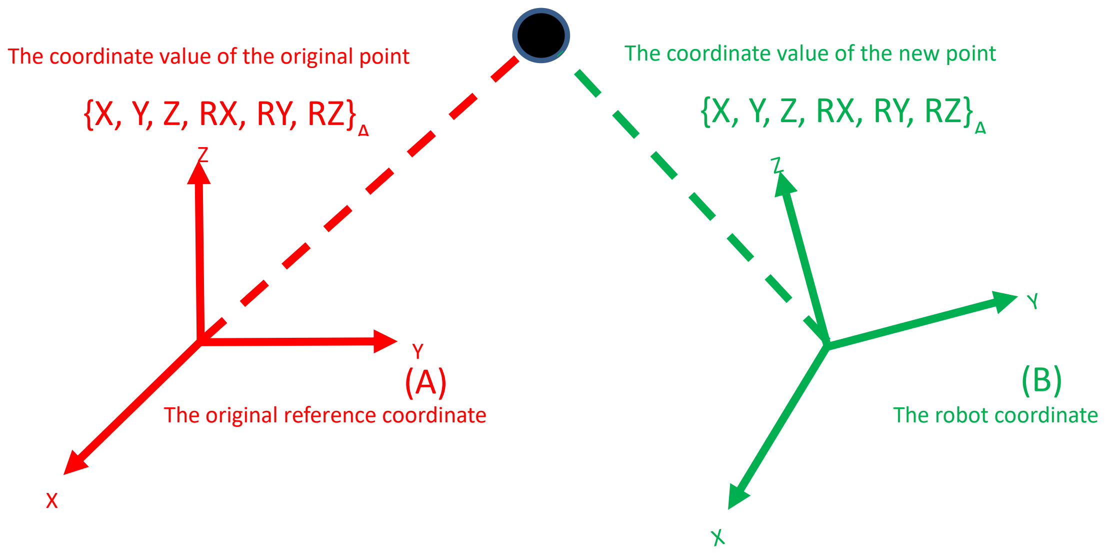
`float[]`	The coordinate value of the original point $\{X_o \quad Y_o \quad Z_o \quad RX_o \quad RY_o \quad RZ_o\}_A$
`float[]`	The original reference coordinate system $\{X_{oa} \quad Y_{oa} \quad Z_{oa} \quad RX_{oa} \quad RY_{oa} \quad RZ_{oa}\}_A$

Return

`float[]`	The coordinate value of the new point $\{X_{nr} \quad Y_{nr} \quad Z_{nr} \quad RX_{nr} \quad RY_{nr} \quad RZ_{nr}\}_R$ Return an empty array if unable to calculate.

Note

The usage is the same as Syntax1's except assuming the robot coordinate system $\{0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\}_R$ as the default new reference coordinate system.


The coordinate value of the original point
{X, Y, Z, RX, RY, RZ}A
Z
Y
X
(A)
The original reference coordinate
The coordinate value of the new point
{X, Y, Z, RX, RY, RZ}A
Z
Y
X
(B)
The robot coordinate

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,-105.84492}

6.32 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 $X_{1(mm)}$ $Y_{1(mm)}$ $Z_{1(mm)}$ $RX_{1(°)}$ $RY_{1(°)}$ $RZ_{1(°)}$
`float[]`	Any point on the X axis in the plane $X_{2(mm)}$ $Y_{2(mm)}$ $Z_{2(mm)}$ $RX_{2(°)}$ $RY_{2(°)}$ $RZ_{2(°)}$
`float[]`	Any point on the +X and +Y in the plane $X_{3(mm)}$ $Y_{3(mm)}$ $Z_{3(mm)}$ $RX_{3(°)}$ $RY_{3(°)}$ $RZ_{3(°)}$

* 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

$X_{p(mm)}$ $Y_{p(mm)}$ $Z_{p(mm)}$ $RX_{p(°)}$ $RY_{p(°)}$ $RZ_{p(°)}$

that coincided with the origin of the plane

$X_{1(mm)}$ $Y_{1(mm)}$ $Z_{1(mm)}$

and calculated with the corners of

$X_{2(mm)}$ $Y_{2(mm)}$ $Z_{2(mm)}$ $RX_{2(°)}$ $RY_{2(°)}$ $RZ_{2(°)}$

and

$X_{3(mm)}$ $Y_{3(mm)}$ $Z_{3(mm)}$ $RX_{3(°)}$ $RY_{3(°)}$ $RZ_{3(°)}$

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,-4.797323,416.2175,-168.6551,-2.780696,99.9553}
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

`float, float, float`	The origin coordinates in the coordinate plane to calculate $X_{1(mm)}$ $Y_{1(mm)}$ $Z_{1(mm)}$
`float, float, float`	The coordinates of any point on the X axis in the coordinate plane to calculate $X_{2(mm)}$ $Y_{2(mm)}$ $Z_{2(mm)}$

`float, float, float` 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.

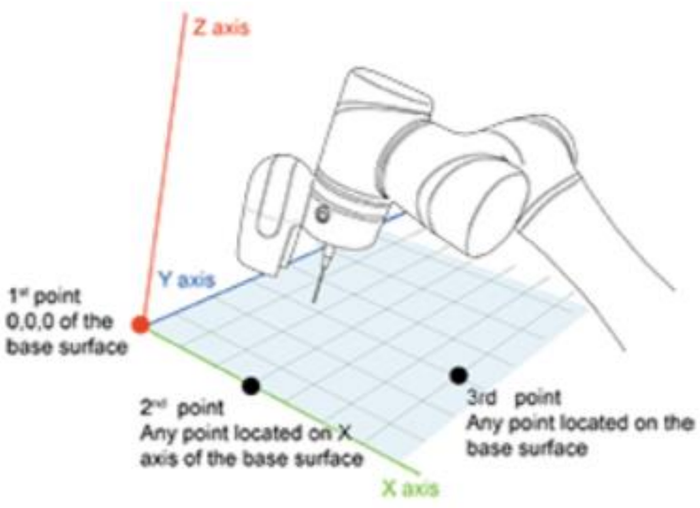
Return

`float[]` Definition parameters of the coordinate plane to calculate.

$X_{p(mm)}$ $Y_{p(mm)}$ $Z_{p(mm)}$ $RX_{p(°)}$ $RY_{p(°)}$ $RZ_{p(°)}$

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)
```


Z axis
Y axis
1st point
0,0,0 of the base surface
2nd point
Any point located on X axis of the base surface
3rd point
Any point located on the base surface
X axis

6.33 intercoord()

Convert two input planes into a new plane.

Syntax 1

```
float[] intercoord(
    float[],
    float[]
)
```

Parameters

`float[]`	The first input plane $X_{1(mm)}$ $Y_{1(mm)}$ $Z_{1(mm)}$ $RX_{1(°)}$ $RY_{1(°)}$ $RZ_{1(°)}$
`float[]`	The second input plane $X_{2(mm)}$ $Y_{2(mm)}$ $Z_{2(mm)}$ $RX_{2(°)}$ $RY_{2(°)}$ $RZ_{2(°)}$

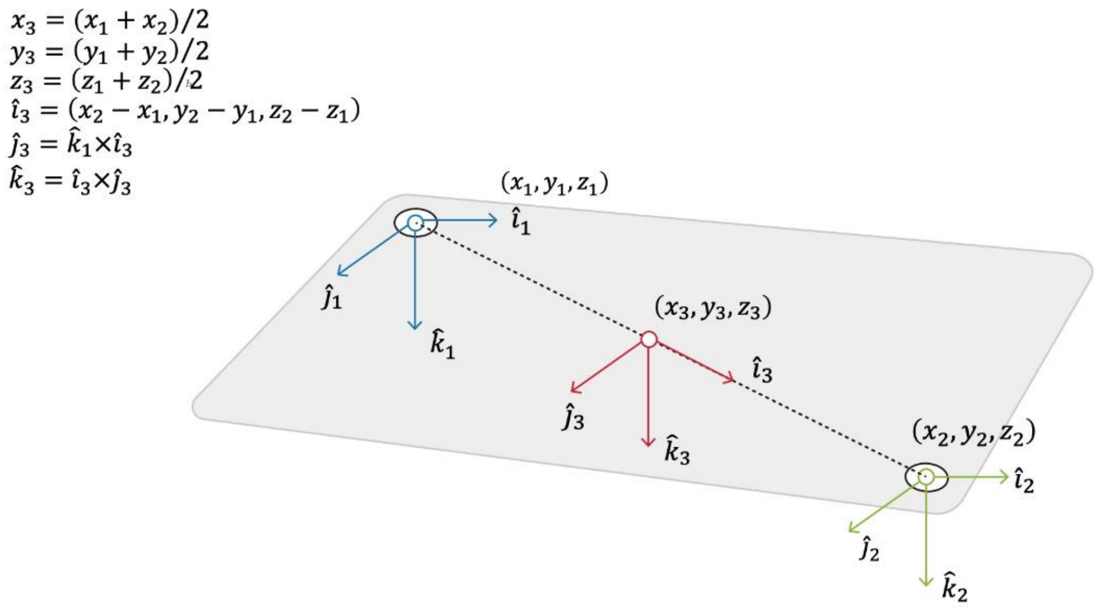
* The input points above are required to be described in the same coordinate system.

Return

`float[]`	The new plane converted by the two input planes. $X_{3(mm)}$ $Y_{3(mm)}$ $Z_{3(mm)}$ $RX_{3(°)}$ $RY_{3(°)}$ $RZ_{3(°)}$

Note

$$X_3 = (X_1 + X_2)/2$$
$$Y_3 = (Y_1 + Y_2)/2$$
$$Z_3 = (Z_1 + Z_2)/2$$
$$\widehat{\imath_3} = (X_2 - X_1, Y_2 - Y_1, Z_2 - Z_1,)$$
$$\widehat{\jmath_3} = (\widehat{k_1} \times \widehat{\imath_3})$$
$$\widehat{k_3} = (\widehat{\imath_1} \times \widehat{\jmath_3})$$



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)
  // {-5.460001,496.52502,141.245,176.06546,0.23468152,61.116673}

7. 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**, **XmlFiles**, and **TMcraft** only.

FileName.txt	The directory default to .\TextFiles (File path the same as .\TextFiles\FileName.txt)
.\TextFiles\FileName.txt	The file is in the local directory named **TextFiles**.
.\XmlFiles\FileName.xml	The file is in the local directory named **XmlFiles**.
.\XmlFiles\folder\file	The subdirector is in the local directory named **XmlFiles**.
..\folder	Unavailable.
.\TextFiles\..\..\folder	Unavailable.

   Void for absolute paths.

C:\file1	Void.
D:\folder\file2	Void.
\TextFiles\FileName.txt	Void.

2. External device path. Available to USB drives or SSD labelled **TMROBOT**.

\USB\TMROBOT	The root directory of the external USB drive.

3. Remote path. Available with the Network service in TMflow.

\\127.0.0.1\shared	SMB / CIFS
ftp://127.0.0.1	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 available 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 10MB (10485760 Bytes).
- The type of the array to read or write depends on the definition of the array.

7.1 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.
```

7.2 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!\u0D0A1Hello TM Robot!"

string var_s2 = **File_ReadText**(".\TextFiles\SampleFile1.txt") // "1Hello World!\u0D0A1Hello TM Robot!"

* \u0D0A 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.

7.3 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

`string`	File path
`int`	The number of the line to start to read
`int`	The amount of the lines to read

Return

`string[]`	Return the file content in the type of string separated by new line characters. If the number of the line to start to read <= 0, it returns an empty array. If the number of the line to start to read > the total number of lines, it returns an empty array. If the amount of the lines to read <= 0, it returns from the first line to the last line. If the amount of the lines to read > the total number of lines, 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
```

7.4 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 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 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 parameter

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. 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. If the array size is 0, it denotes read the end of the file already.

Note

```
.\TextFiles\SampleFile4.txt            .\TextFiles\SampleFile5.txt
1|  4Hello World!                      1|  5Hello World!
2|                                     2|  5Hello TM Robot!
3|  4Hello 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)                      // {}
```

7.5 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 parameter 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
```

7.6 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

`string`	File path	
`?`	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.	
`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 staring 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_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 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
```

? 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 = "ABCDEFG" // 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
```

7.7 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 staring 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_WriteLine("writeline.txt", var_s2)        // Overwrite "達明機器人\u0D0A " to the file
```

```
writeline.txt
1|  達明機器人
2|
```

```
var_flag = File_WriteLine("writeline.txt", var_s1)        // Overwrite "Hi TM Robot\u0D0A" 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\u0D0A" 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\u0D0A" to index 3 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
{100,200}
10000
{1.23,4.56,-7.89}
true

```

? string var_n11 = "ABCDEFG"

File_WriteLine("writeline2.txt", var_n11, 1)

```
writeline2.txt
{100,200}
10000
{1.23,4.56,-7.89}
true
ABCDEFG

```

? string[] var_n12 = {"ABC", "DEF", "達明機器人" }

File_WriteLine("writeline2.txt", var_n12, 1)

```
writeline2.txt
{100,200}
10000
{1.23,4.56,-7.89}
true
ABCDEFG
{ABC,DEF,達明機器人}

```

7.8 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 staring 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\u0D0A" to the file

```
writeline.txt
1|  達明機器人
2|
```

var_flag = **File_WriteLine**("writeline.txt", var_s1) // Overwrite "Hi TM Robot\u0D0A" 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\u0D0A" 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\u0D0A" 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
{100,200}
10000
{1.23,4.56,-7.89}

```

? bool var_n9 = true

File_WriteLine("writeline2.txt", var_n9, 1)

```
writeline2.txt
{100,200}
10000
{1.23,4.56,-7.89}
true

```

? string var_n11 = "ABCDEFG"

File_WriteLine("writeline2.txt", var_n11, 1)

```
writeline2.txt
{100,200}
10000
{1.23,4.56,-7.89}
true
ABCDEFG

```

? string[] var_n12 = {"ABC", "DEF", "達明機器人" }

File_WriteLine("writeline2.txt", var_n12, 1)

```
writeline2.txt
{100,200}
10000
{1.23,4.56,-7.89}
true
ABCDEFG
{ABC,DEF,達明機器人}

```

7.9 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.		
	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 staring 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 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 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
100
200
10000
1.23
4.56
-7.89

```

? string var_n11 = "ABCDEFG"

File_WriteLines("writelines2.txt", var_n11, 1)

```
writelines2.txt
100
200
10000
1.23
4.56
-7.89
ABCDEFG

```

? string[] var_n12 = {"ABC", "DEF", "達明機器人" }

File_WriteLines("writelines2.txt", var_n12, 1)

```
writelines2.txt
100
200
10000
1.23
4.56
-7.89
ABCDEFG
ABC
DEF
達明機器人

```

7.10 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.
```

7.11 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.
```

7.12 File_Delete()

Delete 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.
```

7.13 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.
```

7.14 File_CopyImage()

Copy the saved vision file.

Syntax 1

```
bool File_CopyImage(
    string,
    string,
    string,
    int
)
```

Parameters

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)

```
var_bool 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)
// false // 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.
```

7.15 File_GetImage()

After executing a task job in TMvision, the system keeps the Source image storage path and the Result image storage path in the image storage path buffer. Users can use this function to retrieve the file path stored in the buffer and copy it externally. This function takes priority to the oldest image storage path to get images (FIFO) and removes the path automatically later.

The maximum number of stored image paths in the buffer is 60. When a new image storage path to add to the buffer, if the capacity is insufficient, the oldest image storage path will be automatically removed from the butter, and the new image storage path will be automatically added to the buffer.

Syntax 1

```
string[] File_GetImage(
    int
)
```

Parameter

int	Waiting time out setting to retrieve the image storage path	
	< 0	Wait indefinitely until retrieved the image storage path. (default)
	= 0	Retrieve once.
	> 0	Once using waiting, the process stays at this command until retrieved the image storage path or waiting timeout and keeps on the following executions

Return

string[] If both paths are empty strings, it denotes the buffer does not come with any image storag path.

[0]	Source Image Path
[1]	Result Image Path

Syntax 2

```
string[] File_GetImage(
)
```

Note

Same as syntax 1. Set the waiting timeout to -1 as indefinitely waiting until retrieved the image storage path.

File_GetImage() => **File_GetImage**(-1)

```
string[] var_image = File_GetImage()        // Wait until retrieved the image storage path.
bool var_flag1 = File_CopyImage(var_image[0], "\USB\TMROBOT")
bool var_flag2 = File_CopyImage(var_image[1], "\USB\TMROBOT")
```

7.16 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	Failure	

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!
```

7.17 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 nth matched substring
	-1	Retrieve the last matched substring
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.

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
$Hello World!
$Hello TM Robot!
$Hi TM Robot!$
```

string var_n = "SampleFile7.txt"

string var_s = ""

var_s = **File_GetToken**(var_n, "", "", 0) // "$Hello World!\u0D0A$Hello TM Robot!\u0D0A$Hi TM Robot!$"

var_s = **File_GetToken**(var_n, "$", "$") // "$Hello World!\u0D0A$"

var_s = **File_GetToken**(var_n, "$", "$", 0) // ""

var_s = **File_GetToken**(var_n, "$", "$", 1) // "$Hello World!\u0D0A$"

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!\u0D0A"

var_s = **File_GetToken**(var_n, "$", "$", 2, 1) // "Hi TM Robot!"

var_s = **File_GetToken**(var_n, "$", "", 1) // "$Hello World!\u0D0A"

var_s = **File_GetToken**(var_n, "$", "", 2) // "$Hello TM Robot!\u0D0A"

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, "", "$", 2) // "Hello World!\u0D0A$"

var_s = **File_GetToken**(var_n, "", "$", 3) // "Hello TM Robot!\u0D0A$"

var_s = **File_GetToken**(var_n, "", "$", 4) // "Hi TM Robot!$"

var_s = **File_GetToken**(var_n, "$", Ctrl("\r\n"), 1) // "$Hello World!\u0D0A"

var_s = **File_GetToken**(var_n, "$", newline, 2) // "$Hello TM Robot!\u0D0A"

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"), "$", 1) // "\u0D0A$"

var_s = **File_GetToken**(var_n, newline, "$", 2) // "\u0D0A$"

var_s = **File_GetToken**(var_n, NewLine, "$", 1, 1) // ""

* \u0D0A denotes a new line character but not a string value.

```
.\TextFiles\SampleFile9.txt
#Hello World!
$Hello TM Robot!
$Hi TM Robot!$
```

string var_n = "SampleFile9.txt"

string var_s = ""

var_s = **File_GetToken**(var_n, "", "") // "#Hello World!\u0D0A$Hello TM Robot!\u0D0A$Hi TM Robot!$"

var_s = **File_GetToken**(var_n, "#", "") // "#Hello World!\u0D0A$Hello TM Robot!\u0D0A$Hi TM Robot!$"

```
var_s = File_GetToken(var_n, "", "$")                 // "#Hello World!\u0D0A$"
var_s = File_GetToken(var_n, "#", newline, 1, 0)      // "#Hello World!\u0D0A"
var_s = File_GetToken(var_n, "#", newline, 1, 1)      // "Hello World!"
var_s = File_GetToken(var_n, "#", newline, 1, 2)      // "#Hello World!\u0D0A"
var_s = File_GetToken(var_n, "#", newline, 1, 3)      // "Hello World!"
var_s = File_GetToken(var_n, "$", newline, 1, 0)      // "$Hello TM Robot!\u0D0A"
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!\u0D0A$Hello TM Robot!\u0D0A$Hi TM
                                                      Robot!$"
var_s = File_GetToken(var_n, "", "$", 1, 2)           // "#Hello World!\u0D0A$"
var_s = File_GetToken(var_n, "", "$", -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
byte[]	The prefix of the string to retrieve in the byte array
byte[]	The suffix of the string to retrieve in the byte array
int	The number of the matched substring to retrieve
>=1	Retrieve the nth matched substring
-1	Retrieve the last matched substring
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.

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|  Hi$ 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 \u0D0A

var_s = **File_GetToken**(var_n, bb0, bb0, 0) // "$Hello World\u0D0AHello$ TM Robot!\u0D0AHi$ TM Robot!$"

var_s = **File_GetToken**(var_n, bb1, bb1) // "$Hello World\u0D0AHello$"

var_s = **File_GetToken**(var_n, bb1, bb1, 0) // ""

var_s = **File_GetToken**(var_n, bb1, bb1, 1) // "$Hello World\u0D0AHello$"

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\u0D0AHello"

var_s = **File_GetToken**(var_n, bb1, bb1, 2, 1) // " TM Robot!"

var_s = **File_GetToken**(var_n, bb1, bb0, 1) // "$Hello World\u0D0AHello"

var_s = **File_GetToken**(var_n, bb1, bb0, 2) // "$ TM Robot!\u0D0AHi"

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\u0D0AHello$"

var_s = **File_GetToken**(var_n, bb0, bb1, 3) // " TM Robot!\u0D0AHi$"

var_s = **File_GetToken**(var_n, bb0, bb1, 4) // " TM Robot!$"

var_s = **File_GetToken**(var_n, bb1, bb2, 1) // "$Hello World\u0D0A"

var_s = **File_GetToken**(var_n, bb1, bb2, 2) // "$ TM Robot!\u0D0A"

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) // "\u0D0AHello$"

var_s = **File_GetToken**(var_n, bb2, bb1, 2) // "\u0D0AHi$"

```
var_s = File_GetToken(var_n, bb2, bb1, 1, 1)       // "Hello"
```

* \u0D0A denotes a new line character but not a string value.

7.18 File_GetAllTokens()

Read the file by the string pattern and retrieve all eligible substrings.

Syntax 1

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

Parameter

string		File path
string		The prefix of the string to retrieve
string		The suffix of the string to retrieve
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.

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 = {}

var_ss = **File_GetAllTokens**(var_n, "", "") // {"$Hello World!\u0D0A$Hello TM Robot!\u0D0A$Hi TM Robot!$"}

var_ss = **File_GetAllTokens**(var_n, "$", "$") // {"$Hello World!\u0D0A$", "$Hi TM Robot!$"}

var_ss = **File_GetAllTokens**(var_n, "$", "$", 1) // {"Hello World!\u0D0A", "Hi TM Robot!"}

var_ss = **File_GetAllTokens**(var_n, "$", "", 1) // {"Hello World!\u0D0A", "Hello TM Robot!\u0D0A", "Hi TM Robot!", ""}

```
.\TextFiles\SampleFile9.txt
#Hello World!
$Hello TM Robot!
$Hi TM Robot!$
```

```
string var_n = "SampleFile9.txt"
string[] var_ss = {}
var_ss = File_GetAllTokens(var_n, "", "")           // {"#Hello World!\u0D0A$Hello TM Robot!\u0D0A$Hi TM Robot!$"}
var_ss = File_GetAllTokens(var_n, "$", "$", 0)      // {"$Hello TM Robot!\u0D0A$"}
var_ss = File_GetAllTokens(var_n, "$", "$", 1)      // {"Hello TM Robot!\u0D0A"}
var_ss = File_GetAllTokens(var_n, "$", "$", 2)      // {}
var_ss = File_GetAllTokens(var_n, "$", "$", 3)      // {}
var_ss = File_GetAllTokens(var_n, "$", "", 0)       // {"$Hello TM Robot!\u0D0A", "$Hi TM Robot!", "$"}
var_ss = File_GetAllTokens(var_n, "$", "", 2)       // {}
var_ss = File_GetAllTokens(var_n, "#", "", 0)       // {"#Hello World!\u0D0A$Hello TM Robot!\u0D0A$Hi TM Robot!$"}
var_ss = File_GetAllTokens(var_n, "#", "", 1)       // {"Hello World!\u0D0A$Hello TM Robot!\u0D0A$Hi TM Robot!$"}
var_ss = File_GetAllTokens(var_n, "#", "", 2)       // {"#Hello World!\u0D0A$Hello TM Robot!\u0D0A$Hi TM Robot!$"}
var_ss = File_GetAllTokens(var_n, "#", "", 3)       // {"Hello World!\u0D0A$Hello TM Robot!\u0D0A$Hi TM Robot!$"}
```

8. Serial Port Functions

8.1 SerialPort Class

Use SerialPort class and declare variables to create a COM port device. The variable name will be the device name.

Construct 1

SerialPort *VariableName* = `string, int, string, int, float, int, bool, bool, bool`
SerialPort *VariableName* = `string, int, string, int, float, int`
SerialPort *VariableName* = `string, int, string, int, float`
SerialPort *VariableName* = `string, int`

Parameter

`string`	connection description		
`int`	bits per second, BaudRate		
`string`	parity check		"none", "odd", "even", "mark", "space" ("none" by default)
`int`	Data Bits	5, 6, 7, 8	(8 by default)
`float`	Stop Bits	1, 1.5, 2	(1 by default)
`int`	read/write timeout in millisecond	0 .. 10000	(10000 ms by default)
`bool`	DTR/DSR		true, false (false by default)
`bool`	RTS/CTS		true, false (false by default)
`bool`	XON/XOFF		true, false (false by default)

Note

```
SerialPort spd_c1 = "COM2",115200
  // construct a device, with Baudrate 115200
SerialPort spd_c2 = "COM2",115200,"none",8,1
  // construct a device with Baudrate 115200, Parity none, DataBits 8, StopBits 1
SerialPort spd_c3 = "COM2",115200,"none",8,1,10000
  // construct a device with Baudrate 115200, Parity none, DataBits 8, StopBits 1, Timeout 10000ms
```

- In a flow project, it will create a device from the serial port list and connect to it actively.
- In a script project, after creating a device by the syntax content, it will not connect to the device. It takes the open device function to connect.
- Whether using the device to read or write, it will ask for confirmation of connecting to the device.

8.2 com_open()

Open a Serial Port device.

Syntax 1

```
bool com_open(
    string
)
```

Parameter

`string`	Serial Port device name

Return

`bool`	`True`	Open successfully.
	`False`	Open unsuccessfully. (The project reports error.)

Note

SerialPort spd_dev = "COM2",115200

com_open("spd_dev")

8.3 com_close()

Close a Serial Port device.

Syntax 1

```
bool com_close(
    string
)
```

Parameter

`string`	Serial Port device name

Return

`bool`	`True`	Close successfully.
	`False`	Close unsuccessfully.

Note

```
SerialPort spd_dev = "COM2",115200
com_open("spd_dev")
com_close("spd_dev")
```

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 opened the Serial Port, it receives data from it continuously and puts them in the receiving buffer. Users can use the function com_read or associatd functions to get data from the buffer. When the project stops running, it closes the opened Serial Port and clears the receiving buffer.

The receiving buffer comes with a capacity limitation. If there is data coming to the buffer and the buffer is out of space, it removes the earliest data automatically for the latest data coming into the buffer.

8.4 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

Return

byte[]	Return all the data content. If the content is empty, it returns byte[0].

Note

ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
byte[] value = **com_read**("spd")
  // value 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 clears the received buffer.

Syntax 2

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

Parameters

string	The name of the device on the Serial Port	
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.

ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}

```
value = com_read("spd", 6)
  // value byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C}
  // ReceivedBuffer = {0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
value = com_read("spd", 100)
  // value byte[] = {}
  // Insufficient elements for no more than 100 elements in the received buffer and return byte[0].
  // ReceivedBuffer = {0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
value = com_read("spd", 0)
  // value byte[] = {0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}  // Read all elements
  // ReceivedBuffer = {}
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.
  // ReceivedBuffer = {0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38}      // Supposed it receives data after 50ms,
  //    value byte[] = {0x31,0x32,0x33,0x34}      // it reads 4 element and exits the function.
  // 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

`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.

```
ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
value = com_read("spd", "He", newline)              // prefix "He", suffix \u0D0A
   // value byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A}  // Hello,\u0D0A
   //vReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}    // retrieve the first match and reserve the rest
value = com_read("spd", "", newline, 1)             // prefix "", suffix \u0D0A. Remove both the prefix and the suffix.
   // value byte[] = {0x57,0x6F,0x72,0x6C,0x64}          // World
   // ReceivedBuffer = {}
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.
   //    value byte[] = {}
   // No matched terms to read. Read byte[0]. Wait for 100 ms.
   // ReceivedBuffer = {}
   // ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A}
   // value byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C}
   // ReceivedBuffer = {}

ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
value = com_read("spd", "lo", newline)       // prefix "lo", suffix \u0D0A
   // value byte[] = {0x6C,0x6F,0x2C,0x0D,0x0A}                        // lo,\u0D0A
   // The data before the first matched term, {0x48,0x65,0x6C}, will be removed.
   // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
   // retrieve the first match and reserve the rest
byte[] bb = {}
value = com_read("spd", bb, newline)                      // prefix byte[0], suffix \u0D0A
   //    value byte[] = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}              // World\u0D0A
   // ReceivedBuffer = {}
value = com_read("spd", bb, newline, 0, 100)      // prefix byte[0], suffix \u0D0A, 100ms
   //    value byte[] = {}
   // No matched terms to read. Read byte[0]. Wait for 100 ms.
```

```
    // ReceivedBuffer = {}
    // ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A}
    //    value 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,                   // $TM
                  0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}    // #Robot
value = com_read("spd", "#", newline, 2)          // prefix "#" suffix \u0D0A
    // value byte[] = {}                          // # not in the start. Return an empty array.
    // ReceivedBuffer = {0x24,0x48,0x65,0x6C,0x6C,0x6F,0x0D,0x0A,0x23,0x48,0x69,0x0D,0x0A,
                         0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}
value = com_read("spd", "$", 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
`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

```
ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
value = com_read("spd", newline)      // suffix \u0D0A
   //   value byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A}      // Hello,\u0D0A
   // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
   // retrieve the first match and reserve the rest
value = com_read("spd", newline)      // suffix \u0D0A
   //   value byte[] = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}            // World\u0D0A
   // ReceivedBuffer = {}

ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
value = com_read("spd", newline, 1)          // suffix \u0D0A
   //   value byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C}              // Hello,
   // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
   // retrieve the first match and reserve the rest
value = com_read("spd", newline, 1)          // suffix \u0D0A
   //   value byte[] = {0x57,0x6F,0x72,0x6C,0x64}                   // World
   // ReceivedBuffer = {}
value = com_read("spd", newline, 1, 100)  // suffix \u0D0A, 100ms
   //   value byte[] = {}                         // No matched terms to read. Read byte[0]. Wait for 100 ms.
   // ReceivedBuffer = {}
   // ReceivedBuffer = {0x31,0x32,0x33,0x34,0x35,0x36,0x0D,0x0A}
   //   value byte[] = {0x31,0x32,0x33,0x34,0x35,0x36}
   // ReceivedBuffer = {}
```

8.5 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

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	
`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

`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

`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

```
ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
string value = com_read_string("spd")
  //   value string = "Hello, World\u0D0A"
  // ReceivedBuffer = {}

ReceivedBuffer = {0x54,0x4D,0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA}
value = com_read_string("spd", 4)
  //   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}
value = com_read_string("spd", 5, 100)
  //   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.
  // ReceivedBuffer = {0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA}
  // ReceivedBuffer = {0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA, 0x0D, 0x0A}
  // value string = "機器人\u0D0A"
  // ReceivedBuffer = {}
```

```
ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
value = com_read_string("spd", "He", newline)  // prefix "He", suffix \u0D0A
   // value string = "Hello,\u0D0A"
   // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}     // retrieve the first match and reserve the rest
value = com_read_string("spd", "", newline, 1)
   // prefix "", suffix \u0D0A. Remove both the prefix and the suffix.
   //    value string = "World"
   // ReceivedBuffer = {}
value = com_read_string("spd", "", newline, 1, 100)
   //    value string = ""                  // No matched terms to read. Read an empty string. Wait for 100 ms.
   // ReceivedBuffer = {}
   // ReceivedBuffer = {0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA, 0x0D, 0x0A}
   //    value string = "機器人"

ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
value = com_read_string("spd", "lo", newline)   // prefix "lo", suffix \u0D0A
   //    value string = "lo,\u0D0A"
   // The data before the first matched term, "Hel", will be removed.
   // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
   // retrieve the first match and reserve the rest
value = com_read_string("spd", newline, 1)       // suffix \u0D0A
   //    value 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

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 = {}
ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64}
value = com_read_string("spd", newline)          // suffix \u0D0A
   //    value string = "Hello,\u0D0A"
   // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64}    // retrieve the first match and reserve the rest
value = com_read_string("spd", newline, 0)         // suffix \u0D0A
   //    value string = ""                          // No matched terms to read. Read an empty string.
   // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64}
value = com_read_string("spd", newline, 1, 100)        // suffix \u0D0A
   //    value string = ""
   // No matched terms to read. Read an empty string. Wait for 100 ms.
   // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64}
   // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A,0x31,0x32,0x33,0x0D,0x0A}
   //    value string = "World"
   // ReceivedBuffer = {0x31,0x32,0x33,0x0D,0x0A}    // retrieve the first match and reserve the rest
value = com_read_string("spd", newline, 2)         // suffix \u0D0A
   // value string = "123\u0D0A"
   // ReceivedBuffer = {}
```

8.6 com_write()

Write data to the Serial Port

Syntax 1

```
bool com_write(
    string,
    ?,
    int,
    int
)
```

Parameters

string	The name of the device on the Serial Port
?	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 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
?	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 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.

```
flag = com_write("spd", 100)            // write 0x64
flag = com_write("spd", 1000)           // write 0xE8 0x03 0x00 0x00 (int, Little Endian)
flag = com_write("spd", (float)1.234) // write 0xB6 0xF3 0x9D 0x3F (float, Little Endian)
flag = com_write("spd", (double)123.456)
    // write 0x77 0xBE 0x9F 0x1A 0x2F 0xDD 0x5E 0x40 (double, Little Endian)
flag = com_write("spd", "Hello, World"+newline)
    // write 0x48 0x65 0x6C 0x6C 0x6F 0x2C 0x20 0x57 0x6F 0x72 0x6C 0x64 0x0D 0x0A (string, UTF8)
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[] bb = {100, 200}
flag = com_write("spd", bb)             // write 0x64 0xC8
flag = com_write("spd", bb, 1, 1)       // write 0xC8
// Array. Retrieve 1 element from the index 1. [1]=200
flag = com_write("spd", bb, -1, 1)      // write 0x64
// Array. Retrieve 1 element from the index 0. [0]=100
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[] ss = {"TM", "", "達明機器人" }
flag = com_write("spd", ss)
    // write 0x54 0x4D 0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA
flag = com_write("spd", Byte_Concat(GetBytes(ss), GetBytes(newline)))
    // write 0x54 0x4D 0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA 0x0D 0x0A
flag = com_write("spd", 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
```

8.7 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

`string`	The name of the device on the Serial Port
`?`	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 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_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.

```
flag = com_writeline("spd", 100)              // write 0x64 0x0D 0x0A
flag = com_writeline("spd", 1000)        // write 0xE8 0x03 0x00 0x00 0x0D 0x0A (int, Little Endian)
flag = com_writeline("spd", (float)1.234)   // write 0xB6 0xF3 0x9D 0x3F 0x0D 0x0A (float, Little Endian)
flag = com_writeline("spd", (double)123.456)
        // write 0x77 0xBE 0x9F 0x1A 0x2F 0xDD 0x5E 0x40 0x0D 0x0A (double, Little Endian)
flag = com_write("spd", "Hello, World"+newline)
        // write 0x48 0x65 0x6C 0x6C 0x6F 0x2C 0x20 0x57 0x6F 0x72 0x6C 0x64 0x0D 0x0A (string, UTF8)
flag = com_writeline("spd", "Hello, World")
        // write 0x48 0x65 0x6C 0x6C 0x6F 0x2C 0x20 0x57 0x6F 0x72 0x6C 0x64 0x0D 0x0A (string, UTF8)
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[] bb = {100, 200}
flag = com_writeline("spd", bb)          // write 0x64 0xC8 0x0D 0x0A
flag = com_writeline("spd", bb, 1, 1)  // write 0xC8 0x0D 0x0A
                                          // Array. Retrieve 1 element from the index 1. [1]=200
flag = com_writeline("spd", bb, -1, 1) // write 0x64 0x0D 0x0A
                                          // Array. Retrieve 1 element from the index 0. [0]=100
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[] ss = {"TM", "", "達明機器人" }
flag = com_writeline("spd", ss)
    // write 0x54 0x4D 0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA 0x0D 0x0A
flag = com_write("spd", Byte_Concat(GetBytes(ss), GetBytes(newline)))
    // write 0x54 0x4D 0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA 0x0D 0x0A
flag = com_writeline("spd", ss)
    // write 0x54 0x4D 0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA 0x0D 0x0A
flag = com_writeline("spd", 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
```

9. Socket Functions

9.1 Socket Class

Use Socket class and declare variables to create a TCP/IP communication device. The variable name will be the device name.

Construct

```
Socket VariableName = string, int, int
Socket VariableName = string, int
```

Parameter

`string`	the IP address of the remote host
`int`	the connection port of the remote host
`int`	read/write timeout in millisecond 0 .. 10000 (10000ms by default)

Note

```
Socket ntd_d1 = "192.168.1.10", 12345
   // construct a device, with IP 192.168.1.10, Port 12345, Timeout 10000ms

Socket ntd_d2 = "192.168.1.11", 9999, 8000
   // construct a device, with IP 192.168.1.10, Port 9999, Timeout 8000ms
```

- In a flow project, it will create a device from the network device list and open it.
- In a script project, after creating a device by the syntax content, it will not connect to the device. It takes the open device function to connect.
- While reading or writing with the device, it confirms if it needs to connect to the device.

9.2 socket_open()

Open a TCP/IP device.

Syntax 1

```
bool socket_open(
    string
)
```

Parameter

`string`	TCP/IP device name

Return

`bool`	`True`	Open successfully.
	`False`	Open unsuccessfully.

Note

Socket ntd_dev = "192.168.1.10", 12345
socket_open("ntd_dev")

9.3 socket_close()

Close a TCP/IP device.

Syntax 1

```
bool socket_close(
    string
)
```

Parameter

`string`	TCP/IP device name

Return

`bool`	`True`	Open successfully.
	`False`	Open unsuccessfully.

Note

Socket ntd_dev = "192.168.1.10", 12345
socket_open("ntd_dev")
socket_close("ntd_dev")

When a flow project starts running as the Start node initiates, it launches the TCP/IP Socket Client to connect to the specified IP and port. However, as to the script project, users have to use the socket_open function to open and connect to the assigned device.

After connecting to the TCP/IP device, the system 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.

9.4 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

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	
`int`	Retrieve the fixed amount of byte (by the length of byte[])	
	<= 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

`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 2. 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 ㎳, 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 ㎳ 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	
`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.only. 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

`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 = {}
```

9.5 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

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	
`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

`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.only. 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

`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.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 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\u0D0A"
  // 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\u0D0A"
  // 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,\u0D0AWorld\u0D0A"
  // ReceivedBuffer = {}
ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
string value = socket_read_string("ntd_a", "He", newline)   // Prefix "He" Suffix \u0D0A
  // string = "Hello,\u0D0A"
  // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
```

```
var_value = socket_read_string("ntd_a", "", newline, 1)     // Prefix "" Suffix \u0D0A 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 = {}
```

9.6 socket_send()

Send the data value to a remote device.

Syntax 1

```
int socket_send(
    string,
    ?,
    int,
    int
)
```

Parameters

`string`	Network device name
`?`	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 number 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_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
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
```

9.7 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
?	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 number 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
```

10. 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	float[]	R/W	point coordinate{X,Y,Z,RX,RY,RZ}

Read values

```
float[] f = Point["P1"].Value           // In the item Point, the index is defined as the name of the point
                                        and the data type of string.
float f1 = Point["P1"].Value[0]         // The x value of "P1" can be obtained solely
```

Write values

```
Point["P1"].Value = {0, 0, 90, 0, 90, 0}    // Replace to the coordinate of "P1" with {0,0,90,0,90,0}
Point["P1"].Value[2] = 120                  // or replace the z value of "P1" with 12 solely
```

10.1 Point

Syntax

Base

`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 coordinate of the point	{X, Y, Z, RX, RY, RZ}, Size = 6
`Joint`	float[]	R/W	The joint angle	{J1, J2, J3, J4, J5, J6}, Size = 6
`Pose`	int[]	R/W	The pose of the robot	{Config1, Config2, Config3}, Size = 3
`Flange`	float[]	R	The coordinate of the flange's center	{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 coordinate of the teaching point	{X, Y, Z, RX, RY, RZ}, Size = 6
`TeachJoint`	float[]	R	The joint angle (the original angle at the teach point)	{J1, J2, J3, J4, J5, J6}, Size = 6
`TeachPose`	int[]	R	The original pose of the robot on the teaching point	{Config1, Config2, Config3}, Size = 3

*It recalcutes the the Joint and Flange when setting the Value, and it recalcutates the Value and Flange when setting the Joint. Therefore, it reports an error if it cannot calculate the set value.

Note

```
// Read values
float[] f = Point["P1"].Value              // Obtain the coordinate {X, Y, Z, RX, RY, RZ} of "P1"
float f1 = Point["P1"].Value[0]            // or retrieve the x value of "P1" solely
float f1 = Point["P1"].Value[6]            // Return error, exceeding the array's access range
string s =Point["P1"].BaseName             // 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)
```

10.2 Base

Syntax

Base

Base[string].attribute or
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 base value	{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 teach base value	{X, Y, Z, RX, RY, RZ}, Size = 6

Note

```
// Read values
float[] f = Base["RobotBase"].Value          // Obtain the base value {0,0,0,0,0,0} of the base "RobotBase"
float f1 = Base["base1"].Value[0]            // or retrieve the x value of "base1" solely
string s =Base["base1"].Type                 // s ="C"
s =Base[Point["P1"].BaseName].Type           // s ="R"     // Given the type of "P1" is "RobotBase"
float[] 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)
```

10.3 TCP

Syntax

TCP

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[] f = TCP["NOTOOL"].Value             // Obtain the value {0,0,0,0,0,0} of the TCP "NOTOOL"
float f1 = TCP["NOTOOL"].Value[0]           // or retrieve the x value of "NOTOOL" solely
float mass = TCP["T1"].Mass                 // mass = 2.0
float[] moi = TCP["T1"].MOI                 //    moi = {0,0,0}
float[] mcf = TCP["T1"].MCF                 // 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)
```

10.4 VPoint

Syntax

VPoint

`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[] f = VPoint["Job1"].Value          // Obtain the initial coordinate    {X, Y, Z, RX, RY, RZ} of VPoint "Job1"
float f1 = VPoint["Job1"].Value[0]        // or retrieve the x value of "Job1"
float f1 = VPoint["Job1"].Value[6]        // Return error, exceeding the array's access range
string s = VPoint["Job1"].BaseName        // s ="RobotBase"
// Write values
VPoint["Job1"].Value = {0, 0, 90, 0, 90, 0}          // Replace the initial coordinate of VPoint "Job1" with
                                                     {0,0,90,0,90,0}
VPoint["Job1"].Value[2] = 120                        // or replace the Z value of "Job1" with 120 solely
VPoint["Job1"].BaseName = "base1"                    // Read only, invalid operation
VPoint["Job1"].Value = {0, 0, 90, 0, 90}             // Return error, writing elements to the array do not match to
                                                     6 (writing 5 elements)
VPoint["Job1"].Value = {0, 0, 90, 0, 90, 0, 100}     // Return error, writing elements to the array do not match to
                                                     6 (writing 7 elements)
```

10.5 IO

Syntax

IO

IO[string].attribute

Item

IO	Input/Output

Index

string	The name of the control module	
	ControlBox	the control box
	EndModule	the end module
	ExtModuleN	the external module (N = 0 .. n)
	Safety	the safety module

Attribute

ControlBox / EndModule / ExtModuleN

Name	Type	Mode	Description	Format
DI	byte[]	R	Digital input	[0] = DI0 0: Low, 1: High [1] = DI1 [n] = Din
DO	byte[]	R/W	Digital output	[0] = DO0 0: Low, 1: High [1] = DO1 [n] = DOn
AI	float[]	R	Analog input	[0] = AI0 -10.24V .. +10.24V (Voltage) [1] = AI1 [n] = AIn
AO	float[]	R/W	Analog output	[0] = AO0 -10.00V .. + 10.00V (Voltage) [1] = AO1 [n] = AOn
InstantDI	byte[]	R	Digital input (Instant Command)	[0] = DI0 0: Low, 1: High [1] = DI1 [n] = Din
InstantDO	byte[]	R/W	Digital output (Instant Command)	[0] = DO0 0: Low, 1: High [1] = DO1 [n] = DOn
InstantAI	float[]	R	Analog input (Instant Command)	[0] = AI0 -10.24V .. +10.24V (Voltage) [1] = AI1 [n] = AIn
InstantAO	float[]	R/W	Analog output (Instant Command)	[0] = AO0 -10.00V .. + 10.00V (Voltage) [1] = AO1 [n] = AOn

* The sets of DI[n]/DO[n]/AI[n]/AO[n] vary from the 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] According to Safety Input Ports Assign SI[3] According to Safety Input Ports Assign

Name	Type	Mode	Description	Format
				.. SI[7] According to Safety Input Ports Assign
SO	byte[]	R	Safety function output	0: Low, 1: High SO[0] According to Safety Output Ports Assign SO[1] According to Safety Output Ports Assign SO[2] According to Safety Output Ports Assign SO[3] According to Safety Output Ports Assign .. SO[7] According to Safety Output Ports Assign

The differences between DI/DO/AI/AO and InstantDI/InstantDO/InstantAI/InstantAO

DI/DO/AI/AO is the queue command with reservations in the main flow of the project. If a DI/DO/AI/AO is after the robot motion function such as a point node with the mixture of trajectories, the DI/DO/AI/AO will be operated after the point node is finished. If using an InstantDI/InstantDO/InstantAI/InstantAO command, It will be operated while the point node is on the run and without waiting for the point node to finish.

In addition, if DI/DO/AI/AO turns to an instant command automatically in the thread page of a project as going without waiting for the point node to finish before running along, the result is the same as using InstantDI/InstantDO/InstantAI/InstantAO.

Note

```
// Read values
byte[] di = IO["ControlBox"].DI          // Obtain the digital input status of ControlBox
int dilen = Length(di)                    // Obtain the amount of digital PINs with the size of the array
byte di0 = IO["ControlBox"].DI[0]        // Obtain the status of ControlBox DI[0]
byte 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[] ai = IO["ControlBox"].AI         // Obtain the analog input status of ControlBox
float[] 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.
byte di7 = IO["ControlBox"].InstantDI[7]     // Obtain the status of ControlBox" DI[7] (instant execution)

// 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.
```

10.6 Robot

Syntax

Robot

Robot[int].attribute

Item

Robot

Index

int	The index of the robot fixed at 0

Attribute

Name	Type	Mode	Description	Format
CoordRobot	float[]	R	The TCP coordinate of the robot end point opposite to the RobotBase of the robot	{X, Y, Z, RX, RY, RZ}, Size = 6
CoordBase	float[]	R	The TCP coordinate of the robot end point opposite to the current base of the robot.	{X, Y, Z, RX, RY, RZ}, Size = 6
Joint	float[]	R	The current robot joint angle	{J1, J2, J3, J4, J5, J6}, Size = 6
BaseName	string	R	The name of the current base	"Base Name"
TCPName	string	R	The name of the current TCP	"TCP Name"
CameraLight	byte	R/W	The lighting of the robot's camera	0: Low (Off), 1: High (On)
InstantCameraLight	byte	R/W	The lighting of the robot's camera (Instant Command)	0: Low (Off), 1: High (On)
TCPForce3D	float	R	The current TCP force as the composite force of the robot base x, y, and z.	N
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[] rtool = Robot[0].CoordRobot      // Obtain the current TCP coordinate of the robot end point opposite
                                         to the RobotBase of the robot
float[] ftool = Robot[0].CoordBase       // Obtain the current TCP coordinate of the robot end point opposite
                                         to the current base of the robot.
float 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 of the robot solely.
f = Robot[0]. CoordBase[6]               // Return error, exceeding the array's access range
float[] joint = Robot[0].Joint           // Obtain the current robot joint angle
float j = Robot[0].Joint[0]              // or retrieve the current angle of the robot's 1st joint solely
string b = Robot[0].BaseName             // b = "RobotBase"
string t = Robot[0].TCPName              // t = "NOTOOL"
byte light = Robot[0].CameraLight        // light = 0 (OFF)
float tf3d = Robot[0].TCPForce3D         // tf3d = 1.234
float ts3d = Robot[0].TCPSpeed3D         // 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
```

10.7 FT

Syntax

FT

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 force value of the X axis	
Y	float	R	The force value of the y axis	
Z	float	R	The force 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 resultant force value	
T3D	float	R	The XYZ torque value	
Value	float[]	R	The XYZ resultant force value and torque value array.	{X, Y, Z, TX, TY, TZ}, Size = 6
ForceValue	float[]	R	The XYZ resultant force value array	{X, Y, Z}, Size = 3
TorqueValue	float[]	R	The XYZ torque value array	{TX, TY, TZ}, Size = 3
RefCoordX	float	R	The X-axis force value measured based on the reference coordinate system set in the node	
RefCoordY	float	R	The Y-axis force value measured based on the reference coordinate system set in the node	
RefCoordZ	float	R	The Z-axis force 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 resultant force value 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	

Name	Type	Mode	Description	Format
RefCoordForceValue	float[]	R	The XYZ resultant force 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

Attributes associated with RefCoord come with values when in Force Control.

Note

```
// Read values
float x = FT["fts1"].X                  // Obtain the current X-axis force value of F/T sensor "fts1"
float tx = FT["fts1"].TX                // Obtain the current X-axis torque value of F/T sensor "fts1"
float f3d = FT["fts1"].F3D              // Obtain the current XYZ resultant force value of F/T sensor "fts1"
float[] force = FT["fts1"].ForceValue   // Obtain the current XYZ resultant force value array of F/T sensor "fts1"
string 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
```

11. Robot Teach Class

11.1 TPoint Class

Use TPoint Class and declare variables to create the point names and the point values. The variable name will be the point name. *It takes calculations for the values of the coordinates and the angles to construct points. It returns an error if it fails to calculate tthe values of the coordinates and the angles with the parameter values.

Construct 1

TPoint *VariableName* = `float[]`

TPoint *VariableName* = `float, float, float, float, float, float`

Parameters

`float[]`	The Robot End TCP Coordinate: X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°) recorded by RobotBase and NOTOOL TCP.

Note

TPoint p1 = {0,-282.75,1094.9,90,0,0}

// the coordinate value 0,-282.75,1094.9,90,0,0 by RobotBase and NOTOOL

TPoint p2 = 517.5,-147.8,442.45,180,0,90

// the coordinate value 517.5,-147.8,442.45,180,0,90 by RobotBase and NOTOOL

Construct 2

TPoint *VariableName* = `string, float[]`

TPoint *VariableName* = `string, float, float, float, float, float, float`

Parameters

`string` Point Definitions. Default to "C".

`"C"`	Point coordinate. When the project operation ends, it does not write the values back to the record file. It uses the declared values with the next project operation.(Same as Construct 1)
`"D"`	Point coordinate. When the project operation ends, it writes the values back to the record file. It prioritizes using the record file with the next project operation. If the record file comes empty, it uses the declared values.
`"J"`	Joint angle. When the project operation ends, it does not the values back to the record file. It uses the declared values with the next project operation.
`"JD"`	Joint angle. When the project operation ends, it writes the values back to the record file. It prioritizes using the record file with the next project operation. If the record file comes empty, it uses the declared values.

`float[]` If expressed in coordinates, this is the six elements of the TCP coordinates at the robot end: X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°); if expressed in angles, the six elements of the robot joints: Joint 1(°), Joint 2(°), Joint 3(°), Joint 4(°), Joint 5(°), Joint 6(°) by the RobotBase base and NOTOOL TCP records.

Note

TPoint p3 = "D",{522.07,130.75,442.45,180,0,120}

TPoint p4 = "D",134.95,-147.8,1094.9,90,0,90 // Return error，unable to calculate the angle value by the coordinate value

TPoint p5 = "J",0,0,90,0,90,0 // the angle value 0,0,90,0,90,0 by RobotBase and NOTOOL

TPoint p6 = "JD",{30,0,90,0,90,0} // the angle value 30,0,90,0,90,0 by RobotBase and NOTOOL

Construct 3

TPoint *VariableName* = `string, float[], string, string`

TPoint *VariableName* = `string, float, float, float, float, float, float, string, string`

TPoint *VariableName* = `float[], string, string`
TPoint *VariableName* = `float, float, float, float, float, float, string, string`

Parameters

`string`	Point Definitions. Default to "C".
	`"C"` Point coordinate. When the project operation ends, it does not write the values back to the record file. It uses the declared values with the next project operation.
	`"D"` Point coordinate. When the project operation ends, it writes the values back to the record file. It prioritizes using the record file with the next project operation. If the record file comes empty, it uses the declared values.
	`"J"` Joint angle. When the project operation ends, it does not the values back to the record file. It uses the declared values with the next project operation.
	`"JD"` Joint angle. When the project operation ends, it writes the values back to the record file. It prioritizes using the record file with the next project operation. If the record file comes empty, it uses the declared values.
`float[]`	If expressed in coordinates, this is the six elements of the TCP coordinates at the robot end: X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°); if expressed in angles, the six elements of the robot joints: Joint 1(°), Joint 2(°), Joint 3(°), Joint 4(°), Joint 5(°), Joint 6(°). Record by the base name and the TCP name.
`string`	Base name. Use the current base name in the record if an empty string.
`string`	TCP name. Use the current TCP name in the record if an empty string.

Note

```
TBase base1 = 0,0,90,0,0,0
TTCP tcp1 = 0,0,10,0,0,0
TPoint p7 = "D",{0,-282.75,1094.9,90,0,0},"RobotBase","NOTOOL"
TPoint p8 = "J",0,0,90,0,90,0,"base1","NOTOOL"                  // by base1 and NOTOOL
TPoint p9 = "JD",{30,0,90,0,90,0},"RobotBase","tcp1"             // by RobotBase and tcp1
TPoint p0 = "C",{517.5,-147.8,342.45,180,0,90},"base1","tcp1"    // by base1 and tcp1
// The syntax below is in the non-define section.
ChangeBase("base1")
TPoint p00 = {517.5,-147.8,342.45,180,0,90},"","tcp1"           // by the current base1 and the assigned tcp1
```

Construct 4

TPoint *VariableName* = `string, float[], float[], string, string`
TPoint *VariableName* = `string, float[], float[]`
TPoint *VariableName* = `float[], float[], string, string`
TPoint *VariableName* = `float[], float[]`

Parameters

`string`	Point Definitions. Default to "C".
	`"C"` When the project operation ends, it does not write the values back to the record file. It uses the declared values with the next project operation.
	`"D"` When the project operation ends, it writes the values back to the record file. It prioritizes using the record file with the next project operation. If the record file comes empty, it uses the declared values.
`float[]`	Expressed in coordinates, this is the six elements of the TCP coordinates at the robot end: X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°).
`float[]`	Expressed in angles, the six elements of the robot joints: Joint 1(°), Joint 2(°), Joint 3(°), Joint 4(°), Joint 5(°), Joint 6(°). Record by the base name and the TCP name.
`string`	Base name. Use the current base name in the record if an empty string.
`string`	TCP name. Use the current TCP name in the record if an empty string.

Note

```
TBase base1 = 0,0,90,0,0,0
TTCP tcp1 = 0,0,10,0,0,0
TPoint tp1 = "C",{0,-282.75,1094.9,90,0,0},{0,0,0,0,0,0},"RobotBase","NOTOOL"
TPoint tp2 = "D",{517.5,-147.8,442.45,180,0,90},{0,0,90,0,90,0}          // by RobotBase and NOTOOL
TPoint tp3 = {0,-292.75,1004.9,90,0,0},{0,0,0,0,0,0},"base1","tcp1"      // by base1 and tcp1
TPoint tp4 = {134.95,-147.8,1094.9,90,0,90},{0,0,0,0,90,0}               // by RobotBase and NOTOOL
// The syntax below is in the non-define section.
ChangeBase("base1")
ChangeTCP("tcp1")
TPoint tp5 = {0,-292.75,1004.9,90,0,0},{0,0,0,0,0,0},"",""         // by the current base1 and the current tcp1
```

Member Attibutes

Name	Type	Mode	Description	Format
Value	float[]	R/W	Point coordinate	{X, Y, Z, RX, RY, RZ}, Size = 6
Joint	float[]	R/W	Joint angle	{J1, J2, J3, J4, J5, J6}, Size = 6
Pose	int[]	R/W	Robot pose	{Config1, Config2, Config3}, Size = 3
Flange	float[]	R	Flange center coordinate	{X, Y, Z, RX, RY, RZ}, Size = 6
BaseName	string	R	Base name	"Base Name"
TCPName	string	R	TCP name	"TCP Name"
TeachValue	float[]	R	Point coordinate (the original teach point coordinate)	{X, Y, Z, RX, RY, RZ}, Size = 6
TeachJoint	float[]	R	Joint angle (the original teach point angle)	{J1, J2, J3, J4, J5, J6}, Size = 6
TeachPose	int[]	R	Robot pose (the original teach point pose)	{Config1, Config2, Config3}, Size = 3

* It recalculates the Joint and the Flange while setting the Value, and it recalculates the Value and the Flange while setting the Joint. Therefore, it returns an error if failed to calculate.

* Only in the define variable section will the variable come with the original teach point value.

Note

```
TPoint P1 = {0,-282.75,1094.9,90,0,0}
                                // "P1"Coordinate value 0,-282.75,1094.9,90,0,0 by RobotBase and NOTOOL
// Read values
float[] f = P1.Value            // Get the point coordinate of the point "P1" {0,-282.75,1094.9,90,0,0}
float f1 = P1.Value[1]          // or get the Y value along of point "P1"
float f2 = P1.Value[6]          // Return error. Exceeding the array's access range
float[] f3 = P1.Joint           // Get the angle value of the point "P1" {0,0,0,0,0,0}
string s = P1.BaseName          // s = "RobotBase"
// Write values
P1.Value = {517.5,-147.8,442.45,180,0,90}  // Modify the point coordinate of the point "P1" to
                                           {517.5,-147.8,442.45,180,0,90}
P1.Value[2] = 450                          // or modify the z value along of point "P1" to 450
P1.Flange = {0,0,90,0,90,0}                // Read only, invalid operation
P1.Value = {0,0,90,0,90}    // Return error. The amount of the array elements to wirte does not matach to 6. (5)
P1.Pose = {1,2,4,0}         // Return error. The amount of the array elements to wirte does not matach to 3. (4)
```

11.2 TBase Class

Use TBase Class and declare variables to create the point names and the point values. The variable name will be the point name.

*The system base name comes with the attribute of the read-only mode and without the write mode.

"RobotBase"

Therefore, once the variable name is the system base name, it is for variable declaration only. The input value is void in the construct.

Construct 1

TBase *VariableName* = `float[]`

TBase *VariableName* = `float, float, float, float, float, float`

Parameters

`float[]`	Base Value: X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)

Note

TBase base1 = {0,0,90,0,0,0}   // Name: base1, Type: C, Value: 0,0,90,0,0,0

TBase base2 = 0,90,90,0,0,0   // Name: base2, Type: C, Value: 0,90,90,0,0,0

TBase RobotBase = 0,0,90,0,0,90

// Since RobotBase is the system base, the input values are void but variable declarations.

Construct 2

TBase *VariableName* = `string, float[]`

TBase *VariableName* = `string, float, float, float, float, float, float`

Parameters

`string` Base Types. Default to "C".

"C"	When the project operation ends, it does not write the values back to the record file. It uses the declared values with the next project operation.(Same as Construct 1)
"D"	When the project operation ends, it writes the values back to the record file. It prioritizes using the record file with the next project operation. If the record file comes empty, it uses the declared values.
"V"	Defined the same as type "D." This type goes with vision jobs mainly. The name must comply with the naming rule beginning with *vision_*.

* Only in the define variable section will the variable declaration come with the write back to the record file mechanism.

`float[]` Base Value: X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)

Note

TBase base5 = "C",{0,0,90,0,0,0}   // Name: base5, Type: C, Value: 0,0,90,0,0,0

TBase base6 = "D",0,90,90,0,0,0   // Name: base6, Type: D, Value: 0,90,90,0,0,0

TBase vision_base7 = "V",90,90,90,0,0,0   // Name: vision_base7, Type: V, Value: 90,90,90,0,0,0

TBase base8 = "V",90,90,90,0,0,0

// Return error. The name, base8 , does not comply the naming rule of the type V.

Member Attibutes

Name	**Type**	**Mode**	**Description**	**Format**
Value	float[]	R/W	Base value	{X, Y, Z, RX, RY, RZ}, Size = 6
Type	string	R	Base type	"R": Robot Base "C": Custom Base "D": Custom Base with write-back "V": Vision Base

Name	Type	Mode	Description	Format
TeachValue	float[]	R	Base value (the original teach point base value)	{X, Y, Z, RX, RY, RZ}, Size = 6

* Only in the define variable section will the variable come with the original teach base value.

Member Methods

Name	Description
GetValue()	Retrieve the base value
SetValue()	Set the base value

11.2.1 GetValue()

Retrieve the base value. (applicable to multiple bases created in the vision jobs)

Syntax 1

```
float[] GetValue(
    int
)
```

Parameters

int	Base Index. Users can assign from multiple bases created by vision one shot get all. The index is 0 to N and defaulted to 0.Base Index. Users can assign from multiple bases created by vision one shot get all. The index is 0 to N and defaulted to 0.

Return

float[]	Base Value: X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°) If the base index value is larger than the number of multiple bases, it returns an empty array.

11.2.2 SetValue()

Set the base value. (applicable to multiple bases created in the vision jobs)

Syntax 1

```
bool SetValue(
    int,
    float[]
)
```

Parameters

int	Base Index. Users can assign from multiple bases created by vision one shot get all. The index is 0 to N and defaulted to 0.
float[]	Base Value: X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)

Return

bool	Return `True` if successful. Return `False` if failed. If the base index value is larger than the number of multiple bases, it sets failed.

Note

```
TBase base1 = {0,0,90,0,0,0}              // Name: base1, Type: C, Value: 0,0,90,0,0,0
TBase RobotBase = 0,0,90,0,0,90
                   // Since RobotBase is the system base, the input values are void but variable declarations.
TBase base5 = "C",{0,0,90,0,0,0}          // Name: base5, Type: C, Value: 0,0,90,0,0,0
TBase base6 = "D",0,90,90,0,0,0           // Name: base6, Type: D, Value: 0,90,90,0,0,0
TBase vision_base8 = "V",90,90,90,0,0,0   // Name: vision_base8, Type: V, Value: 90,90,90,0,0,0
```

```
// Read values
float[] b1 = base1.Value                     // Base: "base1", Value: {0,0,90,0,0,0}
float[] b2 = RobotBase.Value                 // Base: "RobotBase", Value {0,0,0,0,0,0}
float b3 = base1.Value[2]                    // Z value of Base "base1": 90
string t1 = base5.Type                       // "C"
string t2 = base6.Type                       // "D"
string t3 = vision_base8.Type                // "V"
float[] b50 = base5.GetValue(0)              // {0,0,90,0,0,0}   Obtain the 1st base value in the base "base5".
float[] b51 = base5.GetValue(1)              // {} empty array  Obtain the 2nd base value in the base "base5".
float[] vb = vision_base8.GetValue(1)        // Obtain the 2nd base value in the base"vision_base8".
                                             // Suppose there is a vision job, it creates and updates the base value of
                                             vision_base8.
// Write values
RobotBase.Value = {0,0,90,0,90,0}            // Read only, invalid operation for "RobotBase" being the system base.
base1.Value = {0,90,0,0,90,0}                // Modify the base "base1" to {0,90,0,0,90,0}
base1.Value[4] = 120                         // or modify the the RY value of the base "base1" alone to 120.
base1.Value[6] = 120                         // Return error. Exceeding the array’s access range.
base1.Type = "C"                             // Read only, invalid operation
base1.Value = {0,0,90,0,90}
                                  // Return error. The amount of the array elements to wirte does not matach to 6. (5)
base1.Value = {0,0,90,0,90,0,100}
                                  // Return error. The amount of the array elements to wirte does not matach to 6. (7)
base6.Value = {30,90,90,0,0,0}               // Write back to the record file since "base6" is type D.
base5.SetValue(0, {90,90,90,0,0,0})          // True. Set the 1st coordinate value in the base "base5".
base5.SetValue(1, {0,0,0,90,90,90})          // False. Failed to set.
```

11.3 TTCP Class

Use TTCP Class and declare variables to create the point names and the point values. The variable name will be the point name.

*The system TCP name comes with the attribute of the read-only mode and without the write mode.

"NOTOOL"

"HandCamera"

Therefore, once the variable name is the system base name or is in the global TCP setting of the robot, it is for variable declaration only. The input value is void in the construct.

Construct 1

TTCP *VariableName* = `float[], float`

TTCP *VariableName* = `float[]`

TTCP *VariableName* = `float, float, float, float, float, float, float`

TTCP *VariableName* = `float, float, float, float, float, float`

Parameters

`float[]`	TCP value: X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)
`float`	Tool Mass (kg). Default to 0.

Note

TTCP tcp1 = {0,0,90,0,0,0} // Name: tcp1, Type: C, Value: 0,0,90,0,0,0, Mass: 0 kg

TTCP tcp2 = {0,0,90,0,0,0},2 // Name: tcp2, Type: C, Value: 0,0,90,0,0,0, Mass: 2 kg

TTCP tcp3 = 0,0,90,0,0,0 // Name: tcp3, Type: C, Value: 0,0,90,0,0,0, Mass: 0 kg

TTCP tcp4 = 0,0,90,0,0,0,2 // Name: tcp4, Type: C, Value: 0,0,90,0,0,0, Mass: 2 kg

TTCP NOTOOL = 0,0,90,0,0,90

// Since NOTOOL is the system tool name, the input values are void but variable declarations.

Construct 2

TTCP *VariableName* = `string, float[], float`

TTCP *VariableName* = `string, float[]`

TTCP *VariableName* = `string, float, float, float, float, float, float, float`

TTCP *VariableName* = `string, float, float, float, float, float, float`

Parameters

`string` Tool Types. Default to "C".

"C" When the project operation ends, it does not write the values back to the record file. It uses the declared values with the next project operation.(Same as Construct 1)

"D" When the project operation ends, it writes the values back to the record file. It prioritizes using the record file with the next project operation. If the record file comes empty, it uses the declared values.

"V" Defined the same as type "D." This type goes with vision jobs mainly. The name must comply with the naming rule beginning with *vision_*.

* Only in the define variable section will the variable declaration come with the write back to the record file mechanism.

`float[]`	TCP value: X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)
`float`	Tool Mass (kg). Default to 0.

Note

TTCP tcp5 = "C",{0,0,90,0,0,0} // Name: tcp5, Type: C, Value: 0,0,90,0,0,0, Mass: 0 kg

TTCP tcp6 = "D",{0,90,90,0,0,0},2 // Name: tcp6, Type: D, Value: 0,90,90,0,0,0, Mass: 2 kg

TTCP visionTCP_tcp7 = "V",90,90,90,0,0,0 // Name: visionTCP_tcp7, Type: V, Value: 90,90,90,0,0,0

TTCP tcp7v = "V",90,90,90,0,0,0

// Return error. The name, tcp7v, does not comply the naming rule of the type V.

TTCP tcp8 = "D",90,90,90,0,0,0,2 // Name: tcp8, Type: D, Value: 90,90,90,0,0,0, Mass: 2 kg

Construct 3

TTCP *VariableName* = `string, float[], float, float[], float[]`

TTCP *VariableName* = `float[], float, float[], float[]`

Parameters

`string` Tool Types. Default to "C".

"C" When the project operation ends, it does not write the values back to the record file. It uses the declared values with the next project operation.(Same as Construct 1)

"D" When the project operation ends, it writes the values back to the record file. It prioritizes using the record file with the next project operation. If the record file comes empty, it uses the declared values.

"V" Defined the same as type "D." This type goes with vision jobs mainly. The name must comply with the naming rule beginning with *vision_*.

* Only in the define variable section will the variable declaration come with the write back to the record file mechanism.

`float[]` TCP value: X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)

`float` Tool Mass (kg). Default to 0.

`float[]` Principal Moments of Inertia: Ixx(kg-mm^2), Iyy(kg-mm^2), Izz(kg-mm^2)

`float[]` Mass Center Frame with Principle Axes w.r.t Tool Frame: X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)

Note

TTCP tcp9 = {0,0,90,0,0,0},2,{2,0.5,0.5},{0,0,-80,0,0,0}

// Name: tcp9, Type: C, Value: 0,0,90,0,0,0, Mass: 2 kg, Inertia: 2,0.5,0.5, The Reference Coordinate: 0,0,-80,0,0,0

TTCP tcp0 = "D",{0,0,150,0,0,90},1,{1,0.5,0.5},{0,0,-80,0,0,0}

// Name: tcp0, Type: D, Value: 0,0,150,0,0,90, Mass: 1 kg, Inertia: 1,0.5,0.5, The Reference Coordinate: 0,0,-80,0,0,0

Member Attibutes

Name	Type	Mode	Description	Format
Value	float[]	R/W	TCP value	{X, Y, Z, RX, RY, RZ}, Size = 6
Mass	float	R/W	Mass	Mass in kg
MOI	float[]	R/W	Principal Moments of Inertia	{Ixx, Iyy, Izz}, Size = 3
MCF	float[]	R/W	Mass center frame with principle axes w.r.t tool frame	{X, Y, Z, RX, RY, RZ}, Size = 6
TeachValue	float[]	R	TCP value (the original TCP setting value)	{X, Y, Z, RX, RY, RZ}, Size = 6
TeachMass	float	R	Mass (the original TCP setting value)	Mass in kg
TeachMOI	float[]	R	Principal Moments of Inertia (the original TCP setting value)	{Ixx, Iyy, Izz}, Size = 3
TeachMCF	float[]	R	Mass center frame with principle axes w.r.t tool frame (the original TCP setting value)	{X, Y, Z, RX, RY, RZ}, Size = 6

* Only in the define variable section will the variable declaration come with the original TCP settng value.

Note

TTCP tcp1 = {0,0,90,0,0,0},3 // Name: tcp1, Type: C, Value: 0,0,90,0,0,0, Mass: 3 kg

TTCP NOTOOL = 0,0,90,0,0,90

// Since NOTOOL is the system tool name, the input values are void but variable declarations.

TTCP tcp9 = "D",{0,0,90,0,0,0},2,{2,0.5,0.5},{0,0,-80,0,0,0}

```
// Read values
float[] t0 = NOTOOL.Value            // Obtain the TCP value of the tool named NOTOOL: {0,0,0,0,0,0}
float[] t1 = tcp1.Value              // Obtain the TCP value of the tool named tcp1: {0,0,90,0,0,0}
float t2 = tcp1.Value[2]             // or obtain the Z value of the tool named tcp1 alone.
float mass = tcp1.Mass               // mass = 3
float[] mcf = tcp1.MCF               // mcf = {0,0,0,0,0,0}
float mass9 = tcp9.Mass              // mass9 = 2        // Obtain 2.1 kg if operate after tcp9.Mass = 2.1
float[] moi9 = tcp9.MOI              // moi9 = {2,0.5,0.5}
// Write values
NOTOOL.Value = {0, -10, 0, 0, 0, 0}  // Read only, invalid operation for NOTOOL being the system TCP.
tcp1.Value = {0, -10, 0, 0, 0, 0}    // Modify the TCP named tcp1 to {0,-10,0,0,0,0}
tcp1.Value[0] = 10                   // or modify the X value of tcp1 alone to 10
tcp1.Mass = 2.4                      // Modify the the mass of the TCP named tcp1 to 2.4 kg.
tcp9.Mass = 2.1                      // Modify the the mass of the TCP named tcp9 to 2.1 kg.
tcp1.MOI = {0, 0, 0, 1, 2}     // Return error. The amount of the array elements to wirte does not matach to 3. (5)
tcp1.MCF = {0, -20, 0, 0, 0, 0, 0}
                               // Return error. The amount of the array elements to wirte does not matach to 6. (7)
```

12. Robot Motion & Vision Job Function

For robot motion functions and vision job functions to be available if using flow projects, it requires the External Script to have the project flow enter the Listen node (external script control mode) or the script node. If using script project programming, they are available in the project programming directly.

Calling in the main thread is mandatory for robot motion functions and vision job functions moving to the initial position. Namely, the main flows in the flow projects or the main functions in the script projects. Calling motion functions is not allowed in other threads. All the motion processes will be queued temporarily and processed in sequence. Users can use the queue tag numbers to understand the current motion command process if necessary.

12.1 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

Same as syntax 1. It default to 0 not waiting for the tagging to continue processing.

12.2 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.

12.3 StopAndClearBuffer()

Stop the motion of the robot and clear existing commands of the robot in the buffer.

Syntax

```
bool StopAndClearBuffer(
)
```

Parameter

`void`	No parameter

Return

`bool`	`True` Command accepted ; `False` Command rejected

Note

StopAndClearBuffer()

12.4 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:

- `"C"` expressed in Cartesian coordinate
- `"J"` expressed in joint angles

#2: Speed format:

- `"P"` expressed as a percentage

#3: Blending format

- `"P"` expressed as a percentage

`float[]`	Motion target degree. If defined with Cartesian coordinate, it includes the Cartesian coordinate of tool center point: 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 (%)
`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) `"CPP"`, (2) `"JPP"`

```
float[] targetP1= {0,0,90,0,90,0}          // Declare a float array to store the target coordinate
PTP("JPP", targetP1,10,200,0,false)        // Move to 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[] targetP1 = {417.50,-122.30,343.90,180.00,0.00,90.00}
                // Declare a float array to store the target coordinate.
float[] pose = {0,2,4}                          // Declare a float array to store pose.
PTP("CPP", targetP1,50,200,0,false, pose)       // Move to 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: `"C"` expressed in Cartesian 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`	Motion target. If expressed in Cartesian coordinate, it includes the Cartesian coordinate of tool center point: X (mm), Y (mm), Z (mm), RX(°), RY(°), RZ(°). If expressed in joint angles, 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 (%)
`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) `"CPP"` and (2) `"JPP"`

```
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,-
                                                                           122.30,343.90,180.00,0.00,90.00,
                                                                           with PTP, speed = 10%, time to
                                                                           top speed = 200ms, pose = 024.
```

12.5 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 with Cartesian coordinate

#2: Speed format:

`"P"`	expressed by percentage
`"A"`	expressed with absolute speed and in synchronization with the project speed
`"D"`	expressed with absolute speed and not in synchronization with the project speed

#3: Blending format:

`"P"`	expressed by percentage
`"R"`	expressed by 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"`, (4) `"CAR"`, (5) `"CDP"`, (6) `"CDR"`

```
float[] Point1 = {417.50,-122.30,343.90,180.00,0.00,90.00}
        // Declare a float array to store the target coordinate
Line("CAR", Point1,100,200,50,false)        // Move to 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 with Cartesian coordinate
	#2: Speed format:
	`"P"` expressed by percentage
	`"A"` expressed with absolute speed and in synchronization with the project speed
	`"D"` expressed with absolute speed and not in synchronization with the project speed
	#3: Blending format:
	`"P"` expressed by percentage
	`"R"` expressed by 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"`, (4) `"CAR"`, (5) `"CDP"`, (6) `"CDR"`

Line("CAR", 417.50,-122.30,343.90,180.00,0.00,90.00,100,200,50,false)

// Move to 417.50,-122.30,343.90,180.00,0.00,90.00 with Line, velocity = 100mm/s, time to top speed = 200ms, blending radius = 50mm

12.6 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 with Cartesian coordinate
	#2: Speed format:
	`"P"` expressed by percentage
	`"A"` expressed with absolute speed and in synchronization with the project speed
	`"D"` expressed with absolute speed and not in synchronization with the project speed
	#3: Blending format:
	`"P"` expressed by 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"`, (2) `"CAP"`, (3) `"CDP"`

```
float[] PassP = {417.50,-122.30,343.90,180.00,0.00,90.00}
                        // Declare a float array to store the via point value
float[] 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", PassP, 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 with Cartesian coordinate

#2: Speed format:

- `"P"` expressed by percentage
- `"A"` expressed with absolute speed and in synchronization with the project speed
- `"D"` expressed with absolute speed and not in synchronization with the project speed

#3: Blending format:

- `"P"` expressed by 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"`, (2) `"CAP"`, (3) `"CDP"`

```
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
```

12.7 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 with joint angles
  - "C": expressed with Cartesian coordinate
- #2: Speed format:
  - "A" expressed with absolute speed and in synchronization with the project speed
  - "D" expressed with absolute speed and not in synchronization with the project speed
- #3: Blending format:
  - "P": expressed by 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) "CAP", (2) "CDP", (3) "JAP", (4) "JDP"

```
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:
  - "C": expressed with Cartesian coordinate

"J": expressed with joint angles

#2: Speed format:

"A" expressed with absolute speed and in synchronization with the project speed

"D" expressed with absolute speed and not in synchronization with the project speed

#3: Blending format:

"P": expressed by 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) "CAP", (2) "CDP", (3) "JAP", (4) "JDP"

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%

12.8 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 (%)
`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

Return

`bool` `True` Command accepted; `False` Command rejected (format error)

Note

Motion command parameter includes: (1) `"CPP"`, (2) `"TPP"` or (3) `"JPP"`

```
float[] relmove = {0,0,10,45,0,0}              // Declare a float array to store the relative motion target
Move_PTP("TPP", 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 (%)
`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

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

12.9 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 with w.r.t. current base "T": expressed with w.r.t. tool coordinate #2: Speed format: "P": expressed by percentage "A" expressed with absolute speed and in synchronization with the project speed "D" expressed with absolute speed and not in synchronization with the project speed #3: Blending format: "P": expressed by percentage "R": expressed by 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) "CDP", (6) "CDR", (7) "TPP", (8) "TPR", (9) "TAP", (10) "TAR", (11) "TDP", (12) "TDR"

```
float[] relmove = {0,0,10,45,0,0}                //Declare a float array to store the relative motion target
Move_Line("TAP", 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 with w.r.t. current base `"T"`: expressed with w.r.t. tool coordinate #2: Speed format: `"P"`: expressed by percentage `"A"` expressed with absolute speed and in synchronization with the project speed `"D"` expressed with absolute speed and not in synchronization with the project speed #3: Blending format: `"P"`: expressed by percentage `"R"`: expressed by 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) `"CDP"`, (6) `"CDR"`, (7) `"TPP"`, (8) `"TPR"`, (9) `"TAP"`, (10) `"TAR"`, (11) `"TDP"`, (12) `"TDR"`

```
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.
```

12.10 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 with w.r.t. current base `"T"`: expressed with w.r.t. tool coordinate `"J"`: expressed with joint angles #2: Speed format: `"A"` expressed with absolute speed and in synchronization with the project speed `"D"` expressed with absolute speed and not in synchronization with the project speed #3: Blending format: `"P"`: expressed by 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) `"CDP"`, (3) `"TAP"`, (4) `"TDP"`, (5) `"JAP"`, (6) `"JDP"`

```
float[] target = {0,0,10,45,0,0}          // Declare a float array to store the relative motion target
Move_PLine("CAP", 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 with w.r.t. current base `"T"`: expressed with w.r.t. tool coordinate `"J"`: expressed with joint angles #2: Speed format: `"A"` expressed with absolute speed and in synchronization with the project speed `"D"` expressed with absolute speed and not in synchronization with the project speed #3: Blending format: `"P"`: expressed by 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) `"CDP"`, (3) `"TAP"`, (4) `"TDP"`, (5) `"JAP"`, (6) `"JDP"`

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

12.11 ChangeBase()

Set the base to refer for consecutive motions.

Syntax 1

```
bool ChangeBase(
    string,
    int
)
```

Parameters

`string`	Base Name
`int`	Base Index. Users can assign from multiple bases created by vision one shot get all. The index is 0 to N and defaults to 0.

Return

`bool`	`True` Command accepted; `False` Command rejected

Syntax 2

```
bool ChangeBase(
    string
)
```

Note

Same as syntax 1. It defaults 0 to base index.

ChangeBase("RobotBase")

// Change the base to "RobotBase"

ChangeBase("vision_job1", 1)

// Change the base to the 2nd base value of "vision_job1". If the assigned base name or the base index does not exist, it returns an error.

ChangeBase("RobotBase", 10)

// Change the base to "RobotBase". Since it is the base name of the system, the base index is invalid.

Syntax 3

```
bool ChangeBase(
    float[]
)
```

Parameters

`float[]` Base parameters, combines X, Y, Z, RX, RY, RZ

Return

`True` Command accepted; `False` Command rejected

Note

float[] Base1 = {20,30,10,0,0,90} // Declare a float array to store the base value

ChangeBase(Base1) // Change the base to the assigned base.

Syntax 4

```
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

Note

```
ChangeBase(20,30,10,0,0,90)                              // Change the base value to {20,30,10,0,0,90}
```

12.12 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".
```

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[] Tool1 = {0,0,150,0,0,90}  // Declare a float array to store the TCP value
ChangeTCP(Tool1)                  // Change the TCP value to 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[] Tool1 = {0,0,150,0,0,90}    // Declare a float array to store the TCP value
ChangeTCP(Tool1,2)                  // Change the TCP value to 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[] Tool1 = {0,0,150,0,0,90}             // Declare a float array to store the TCP value
float[] 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(Tool1,2, COM1)                     // Change the TCP value to Tool1 with weight = 2kg and moment
                                             of inertia to 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, 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}

12.13 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
```

12.14 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 parameter. Use PVT mode with Joint Command by default.

Return

`bool`	`True` Command accepted; `False` Command rejected

Note

PVTEnter(1)

12.15 PVTExit()

Set PVT mode motion to exit

Syntax1

```
bool PVTExit(
)
```

Parameter

`void`	No parameter

Return

`bool`	`True` Command accepted; `False` Command rejected

Note

PVTExit()

12.16 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,-122.2,359.7,180,0,90,50,50,0,0,0,0,0.5)
PVTPoint(467.5,-72.2,359.7,180,0,90,-50,50,0,0,0,0,0.5)
PVTPoint(417.5,-72.2,359.7,180,0,90,0,0,0,0,0,0,0.5)
PVTPoint(417.5,-122.2,359.7,180,0,90,50,50,0,0,0,0,0.5)
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()
```

12.17 PVTPause()

Set PVT mode motion to pause

Syntax1

```
bool PVTPause(
)
```

Parameter

`void`	No parameter

Return

`bool`	`True` Command accepted; `False` Command rejected

Note

PVTPause()

12.18 PVTResume()

Set PVT mode motion to resume

Syntax1

```
bool PVTResume(
)
```

Parameter

`void`	No parameter

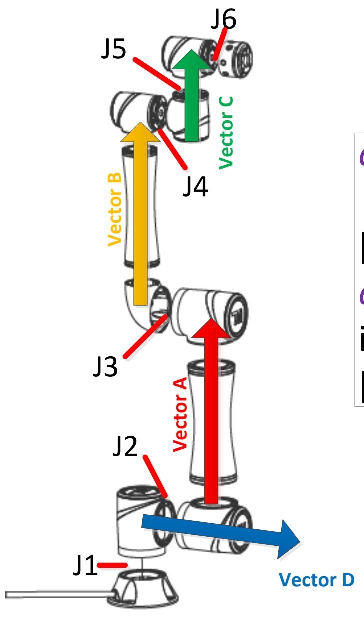
Return

`bool`	`True` Command accepted; `False` Command rejected

Note

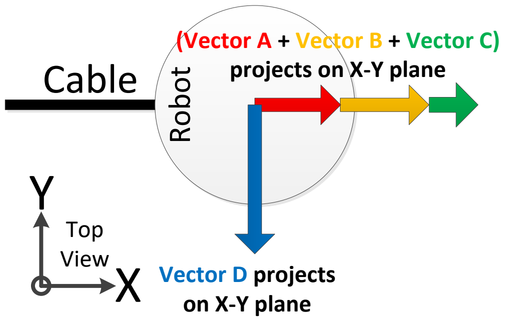
PVTResume()

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)

12.19 Vision_DoJob()

Execute the existing vision jobs in the project but not the ones with initial positions.
*Must create vision jobs in the project first without checking **Start at Initial Position**.
*It is required to codify the respective variable definitions in the define segment if using script projects.
* The respective variable definition values update after the vision job execution.

Syntax 1

```
bool Vision_DoJob(
    string
)
```

Parameter

string	Vision job name.

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

bool var_result = **Vision_DoJob**("job1")

12.20 Vision_DoJob_PTP()

Execute the existing vision jobs in the project, move to the initial position by the PTP motion, and queue with the motion commands.
*Must create vision jobs in the project first. Check **Start at Initial Position** for moving to the initial position and uncheck for not.
*It is required to codify the respective variable definitions in the define segment if using script projects.
* The respective variable definition values update after the vision job execution.

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`	Vision job failed. 1. The result is fail. 2. Fail to execute.

Syntax 2

```
bool Vision_DoJob_PTP(
    string,
    int,
    int
)
```

Note

Same as syntax 1. It defaults false to whether to use the smart pose of robot selection mode.
bool var_result = **Vision_DoJob_PTP**("job1", 100, 500)
var_result = **Vision_DoJob_PTP**("job1", 100, 500, true)

12.21 Vision_DoJob_Line()

Execute the existing vision jobs in the project, move to the initial position by the line motion, and queue with the motion commands.
*Must create vision jobs in the project first. Check **Start at Initial Position** for moving to the initial position and uncheck for not.
*It is required to codify the respective variable definitions in the define segment if using script projects.
* The respective variable definition values update after the vision job execution.

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

bool var_result = **Vision_DoJob_Line**("job1", 100)

Syntax 2

```
bool Vision_DoJob_Line(
    string,
    int,
    int
    bool
)
```

Parameters

`string`	Vision job name	
`int`	The speed setting, expressed in velocity (mm/s)	
`int`	The time interval to accelerate to top speed (ms)	
`bool`	Whether to link to the project speed.	
	`true`	Link to the project speed.
	`false`	Unlink to the project speed.

Return

`bool`	`True`	Vision job completed and the result is pass.	
	`False`	Vision job failed.	1. The result is fail.
			2. Fail to execute.

Syntax 3

```
bool Vision_DoJob_Line(
    string,
    int,
    int
)
```

Note

Same as systax 2. It defaults true to whether to link to the project speed.

bool var_result = **Vision_DoJob_Line**("job1", 100, 500)
var_result = **Vision_DoJob_Line**("job1", 100, 500, false)

13. Vision Functions

13.1 Vision_IsJobAvailable()

Check if the vision job in the current project present and valid

Syntax 1

```
bool Vision_IsJobAvailable(
    string
)
```

Parameter

string	Vision job name

Return

bool	True	the vision present and valid
	False	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")

13.2 Vision_GetOutputArraySize()

Retrieve the array size of the 2D output variables after the vision job execution.

Syntax 1

```
int Vision_GetOutputArraySize(
    string
)
```

Parameter

`string`	The variable name as the output by the vision job with the data type of string array.

Return

`int`	the find object index. (the row count of the 2D array)

Syntax 2

```
int Vision_GetOutputArraySize(
    string,
    int
)
```

Parameter

`string`	The variable name as the output by the vision job with the data type of string array.
`int`	the find object index. (the row index of the 2D array)

Return

`int`	The index of the assigned row. (row index). Retrieve the output result array size of the assigned object to locate.

13.3 Vision_GetOutputArrayValue()

Retrieve the array values of the 2D output variables after the vision job execution.

Syntax 1

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

Parameter

`string`	The variable name as the output by the vision job with the data type of string array.
`int`	the find object index. (the row index of the 2D array)

Return

`string[]`	The output result array value of the assigned row index (row index).

Syntax 2

```
string Vision_GetOutputArrayValue(
    string,
    int,
    int
)
```

Parameter

`string`	The variable name as the output by the vision job with the data type of string array.
`int`	the find object index. (the row index of the 2D array)
`int`	The output result array value of the assigned column index

Return

`string`	The output result value of the assigned object and the location.

Note

After vision job execution, if the result is output as a 2D array of data, it requires this function to retrieve the relevant information. It denotes retrieving the result data of the vision job and the module by substituting the parameter for the variable name. For example, "job1_Count_Blob_1_DetectObjectX_TM" is an output variable of the vision job, job1, and output by Counting (Blobs) module 1. If the output value is a 2D array, each row in the array stands for a find object result, and each column denotes numerous outcomes.

For example, the result array is { {"0", "1", "2"}, {"3", "4"}, {"5"} }

Retrieve the find object count

int var_count = **Vision_GetOutputArraySize**("job1_Count_Blob_1_DetectObjectX_TM")

// Retrieve the row count (i.e. the find object count) of the variable *job1_Count_Blob_1_DetectObjectX_TM*

// var_count = 3

int var_length = **Vision_GetOutputArraySize**("job1_Count_Blob_1_DetectObjectX_TM", 1)

// Retrieve the output result array size at row 1 (the 2nd object) of the variable *job1_Count_Blob_1_DetectObjectX_TM*

// var_length = 2

Retrieve the result value of the find object

string[] var_ss = **Vision_GetOutputArrayValue**("job1_Count_Blob_1_DetectObjectX_TM", 0)

// Retrieve the output result at row 0 (the 1st object) of the variable *job1_Count_Blob_1_DetectObjectX_TM*

// var_ss = {"0", "1", "2"}

string var_s = **Vision_GetOutputArrayValue**("job1_Count_Blob_1_DetectObjectX_TM", 2, 0)

```
    // Retrieve the output result at row 2 (the 3rd object) and column 0 (item 1) of the variable
    job1_Count_Blob_1_DetectObjectX_TM
    // var_s = "5"
string var_se = Vision_GetOutputArrayValue("job1_Count_Blob_1_DetectObjectX_TM", 3, 0)
    // Return Error. Exceeded the access range (find object count: 3, index: 0 .. 2)
```

13.4 Vision_GetTriggerJobOutputCount()

Retrieve the output variable count in the buffer after the vision IO Trigger Job execution.

Syntax 1

```
int Vision_GetTriggerJobOutputCount(
    string
)
```

Parameter

`string`	The variable name as output by the IO Trigger Job with the data type of string array.

Return

`int`	The output variable count of the designated variable in the buffer.

13.5 Vision_GetTriggerJobOutputValue()

Retrieve the output variable values after the vision IO Trigger Job execution.. (Retrieve and remove from the buffer.)

Syntax 1

```
string[] Vision_GetTriggerJobOutputValue(
    string
)
```

Parameter

`string`	The variable name as output by the IO Trigger Job with the data type of string array.

Return

`string[]`	The output values of the designated variables in the buffer.

Note

After the vision IO Trigger Job execution (Enabling the IO trigger mode is a must during vision editing.), the result updates to the variable after each IO trigger. Since the output of the IO Trigger Job is multiple outputs, to prevent the output data from being overwritten because of out-of-process, it stores the output data in the buffer one by one (first in, first out), and users can use the function calls to retrieve the count or the values in the buffer. For example, job1_Count_Blob_1_DetectObjectX_TM is an output variable of the vision job job1 to be output by the Counting (Blobs) module 1. After the vision execution, it updates and stores the result in the buffer at each IO trigger, so users can call the functions to retrieve the count or the values of the variable job1_Count_Blob_1_DetectObjectX_TM in the buffer.

The buffer comes with a capacity limit. When results are about to enter, if the buffer capacity is insufficient, it removes the oldest data automatically and adds the latest to the buffer.

Suppose the current content in the buffer goes by { {"0", "1", "2"}, {"3", "4"}, {"5"} }

Retrieve the output variable count of the vision trigger job in the buffer.

```
int var_count = Vision_GetTriggerJobOutputCount ("job1_Count_Blob_1_DetectObjectX_TM")
  // var_count = 3
```

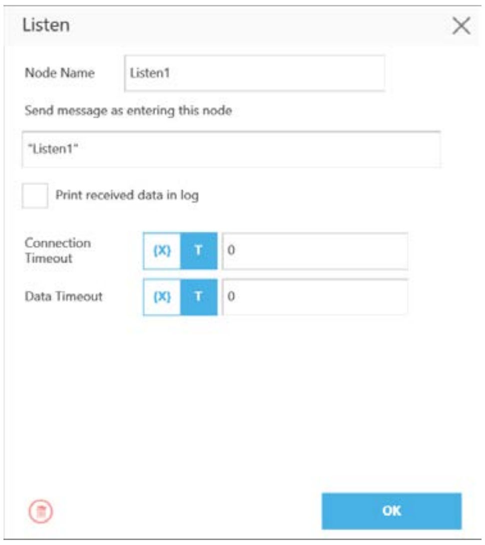
Retrieve the output variable values in the buffer.

```
string[] var_s = Vision_GetTriggerJobOutputValue ("job1_Count_Blob_1_DetectObjectX_TM")
  // var_s = {"0", "1", "2"}
  // After retrieving, the content in the buffer turns to { {"3","4"}, {"5"} }.
var_count = Vision_GetTriggerJobOutputCount ("job1_Count_Blob_1_DetectObjectX_TM")
  // var_count = 2
var_s = Vision_GetTriggerJobOutputValue ("job1_Count_Blob_1_DetectObjectX_TM")
  // var_s = {"3","4"}
  // After retrieving, the content in the buffer turns to { {"5"} }.
var_s = Vision_GetTriggerJobOutputValue ("job1_Count_Blob_1_DetectObjectX_TM")
  // var_s = {"5"}
  // After retrieving, the content in the buffer turns to { }.
var_s = Vision_GetTriggerJobOutputValue ("job1_Count_Blob_1_DetectObjectX_TM")
  // var_s = {}
var_count = Vision_GetTriggerJobOutputCount ("job1_Count_Blob_1_DetectObjectX_TM")
  // var_count = 0
```

14. External Script

14.1 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.



1. **Send Message:** When entering this node, it will initiate a message
2. **Print Log:** Enable Communication Log (shown on the right)
3. **Connection Timeout:** When entering this node, if more than the time (milliseconds) is not connected, it will be overtime.
   If <= 0, no timeout
4. **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 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.

14.2 ScriptExit()

Exit external script control mode.

Syntax 1

```
bool ScriptExit(
)
```

Parameters

`void`	No parameter

Return

`bool`	`True` Command accepted; `False` Command rejected (format error)

Note

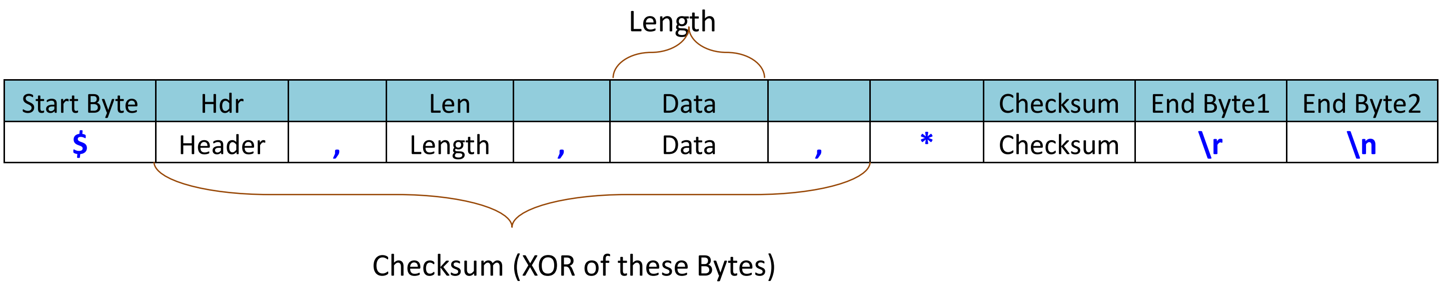
Exit the external script control mode and wait for the command to finish, and then quit the listen node and move on with the pass route.

* Execute via TMSCT communication packets
* Functions after ScriptExit() will not be executed such as

```
<     $TMSCT,78,2,ChangeBase("RobotBase")\r\n
      ChangeTCP("NOTOOL")\r\n
      ScriptExit()\r\n              // Exit the external script control mode.
      ChangeLoad(10.1),*6C\r\n      // ChangeLoad will not be executed.
```

* After exiting the script mode, it is required to wait for all the commands and the functions to complete executions until quitting the listen node and moving on with the pass route. At the time being of waiting for quitting the listen node, it is not in the external script control mode, so no more external commands will be accepted and CPEER error packets will be replied.

14.3 Communication Protocol



Start Byte	Hdr		Len		Data			Checksum	End Byte1	End Byte2
$	Header	**,**	Length	**,**	Data	**,**	*****	Checksum	**\r**	**\n**

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

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:

$\boxed{\text{TMSCT,100,Data,}}$ → `$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 ^ 0x4D ^ 0x53 ^ 0x43 ^ 0x54 ^ 0x2C ^ 0x35 ^ 0x2C ^ 0x31 ^ 0x30 ^ 0x2C ^ 0x4F ^ 0x4B ^ 0x2C = 0x6D

CS = 6D (0x36 0x44)

14.4 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)

1. 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

2. 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.

14.5 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	`true`	Indicates Tag Number complete
	`false`	Indicates Tag Number incomplete
	`none`	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
```

14.6 CPERR

Start Byte	Hdr		Len		Data			Checksum	End Byte1	End Byte2
$	*CPERR*	**,**	Length	**,**	Data	**,**	*****	Checksum	**\r**	**\n**

Error Code
Code (00 .. FF)

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
```

14.7 Priority Commands

Due to the serial execution nature of the TMscript 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 TMscript 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 TMscript 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.
```

```
4.       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 external 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 external 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.
    <    $TMSCT,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.
    <   $TMSCT,23,3,StopAndClearBuffer(1),*55\r\n
                                       // When receiving the command StopAndClearBuffer(1), 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,4,3,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.
```
    >   $TMSCT,4,2,OK,*5F\r\n          // Respond 2 OK
    >   $TMSTA,10,01,02,true,*67\r\n  // Tag 2 finished
```

```
5.  <   $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,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.
    <   $TMSCT,23,3,StopAndClearBuffer(2),*56\r\n
                                       // When receiving the command StopAndClearBuffer(2), 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,4,3,OK,*5E\r\n      // Respond 3 OK
```
It clears and exits the current Script 1 in execution, and clears all the Script programs in the Received Script buffer. Therefore, it clears Script 2 without the response to Script 2.

15. Modbus Functions

15.1 ModbusTCP Class

Use Modbus TCP class and declare variables to create a Modbus TCP device. The variable name will be the device name.

Construct 1

```
ModbusTCP VariableName = string, int, int
ModbusTCP VariableName = string, int
ModbusTCP VariableName = string
```

Parameters

`string`	remote host IP address	
`int`	remote host connection port	(502 by default)
`int`	read/write timeout in millisecond	0 .. 10000 (10000ms by default)

Note

```
ModbusTCP mtcp1 = "192.168.1.10"
  // construct a device, with IP 192.168.1.10
ModbusTCP mtcp2 = "192.168.1.10", 502
  // construct a device, with IP 192.168.1.10, Port 502
ModbusTCP mtcp3 = "192.168.1.10", 502, 8000
  // construct a device, with IP 192.168.1.10, Port 502, Timeout 8000ms
```

* After construction, either in flow projects or script projects, the device will not connect actively until proceeding to read or write.

Member Methods

Name	Description
Preset()	Configure the preset ModbusTCP parameters.
IODDPreset()	Read IODD file and configure the preset ModbusTCP parameters.

15.1.1 Preset()

Configure the preset ModbusTCP parameters.

Syntax 1

```
bool Preset(
    string,
    byte,
    string,
    int,
    string,
    int
)
```

Parameters

`string`	preset device name	
`byte`	Slave ID	
`string`	signal type	"DO", "DI", "RO", "RI"
`int`	starting address	
`string`	type	"bool", "byte", "int16", "int32", "float", "double", "string"
`int`	suffix parameter, when type is	

"int32"	0 Little Endian (CD AB)	1 Big Endian (AB CD) (default)
"float"	0 Little Endian (CD AB)	1 Big Endian (AB CD) (default)
"double"	0 Little Endian (CD AB)	1 Big Endian (AB CD) (default)
"string"	address count (0 by default)	

This is invalid for other types.

Return

`bool` preset successfully `True`, preset unsuccessfully `False`

Syntax 2

```
bool Preset(
    string,
    byte,
    string,
    int,
    string
)
```

Note

Same as syntax 1. Fill default to suffix parameter by default.

Note

```
ModbusTCP mbus1 = "127.0.0.1"
    // construct a device, with IP 127.0.0.1, Port 502, Timeout 8000ms

mbus1.Preset("light", 1, "DO", 7206, "bool")       // set preset device name to "light"
mbus1.Preset("9000", 1, "RO", 9000, "string")      // set preset device name to "9000"
    // Return error, naming must be in alphanumeric combination.

mbus1.Preset("preset_9000", 1, "RO", 9000, "int", 0)
    // set preset device name to "preset_9000" // Little Endian
mbus1.Preset("preset_9000", 1, "RO", 9000, "int", 1)
    // set preset device name to "preset_9000" // Big Endian
mbus1.Preset("preset_9000", 1, "RO", 9000, "int")
    // set preset device name to "preset_9000" // Big Endian
mbus1.Preset("preset_9000", 1, "RO", 9000, "string", 5)
    // set preset device name to "preset_9000" // string type
    // Once the name, preset_9000, exists, it overwrites the content to the configuration of the same name by the
    processing sequence.
bool flag = modbus_read("mbus1", "light")          // flag = false     // suppose camera light is off
modbus_write("mbus1", "light", true)               // write true       // camera light is on
flag = modbus_read("mbus1", "light")               // flag = true

modbus_write("mbus1 ", "preset_9000", Ctrl("\0\0\0\0\0\0\0\0\0\0"))
  // clears preset_9000 in string type and occupies five addresses (5 RO = 10 bytes)

modbus_write("mbus1", "preset_9000", 1234)         // write "1234"    // because preset_9000 is in string type
int var_i = modbus_read("mbus1", "preset_9000")    // var_i = 1234    // it tries to convert "1234" to integer.
  // because preset_9000 is in string type and occupies five addresses, it then reads 10 bytes and converts by strings. (ends when
encountered 0x00.)

modbus_write("mbus1", "preset_9000", "HelloWorld")     // write "HelloWorld"
```

```
string var_s = modbus_read("mbus1", "preset_9000")        // var_s = "HelloWorld"
var_i = modbus_read("mbus1", "preset_9000")
// Return error, unable to convert HelloWorld to integer.

modbus_write("mbus1", "preset_9000", 1234)                // write "1234"
  // contiune the last piece of data and write "HelloWorld" // therefore, the current data in the address 9000 is "1234oWorld"
var_i = modbus_read("mbus1", "preset_9000")
// Return error, unable to convert 1234oWorld to integer.
```

15.1.2 IODDPreset()

Read IODD file and configure the preset ModbusTCP parameters.

Syntax 1

```
bool IODDPreset(
    string,
    byte,
    int,
    int
)
```

Parameters

string	IODD file name (read the IODD file stored in the directory .\XmlFiles\IODD at local host)
byte	Slave ID
int	starting address In
int	starting address Out

Return

bool	preset successfully True, preset unsuccessfully False

Note

```
ModbusTCP mbus1 = "192.168.1.10"
// construct a device, with IP 192.168.1.10, Port 502, Timeout 10000ms

mbus1.IODDPreset("OMRON-E2EQ-X3B4-IL2-20170301-IODD1.1.xml", 1, 100, 200)
  // load the file, .\XmlFiles\IODD\OMRON-E2EQ-X3B4-IL2-20170301-IODD1.1.xml, and add the preset
  configuration parameters.
  // define preset names in the same way as flow projects rules.
```

15.2 ModbusRTU Class

Use Modbus TCP class and declare variables to create a Modbus TCP device. The variable name will be the device name.

Construct

ModbusRTU *VariableName* = `string, int, string, int, float, int, bool, bool, bool`
ModbusRTU *VariableName* = `string, int, string, int, float, int`
ModbusRTU *VariableName* = `string, int, string, int, float`
ModbusRTU *VariableName* = `string, int`

Parameters

`string`	connection description		
`int`	bits per second, BaudRate		
`string`	parity check		"none", "odd", "even", "mark", "space" ("none" by default)
`int`	Data Bits	5, 6, 7, 8	(8 by default)
`float`	Stop Bits	1, 1.5, 2	(1 by default)
`int`	read/write timeout in millisecond		0 .. 10000 (10000 ms by default)
`bool`	DTR/DSR		true, false (false by default)
`bool`	RTS/CTS		true, false (false by default)
`bool`	XON/XOFF		true, false (false by default)

Note

```
ModbusRTU mrtu1 = "COM2",115200
  // construct a device with Baudrate 115200
ModbusRTU mrtu2 = "COM2",115200,"none",8,1
  // construct a device with Baudrate 115200, Parity none, DataBits 8, StopBits 1
ModbusRTU mrtu3 = "COM2",115200,"none",8,1,10000
  // construct a device with Baudrate 115200, Parity none, DataBits 8, StopBits 1, Timeout 10000ms
```

* After construction, either in flow projects or script projects, the device will not connect actively until proceeding to read or write.

Member Methods

Name	Description
Preset()	Configure the preset ModbusTCP parameters.
IODDPreset()	Read IODD file and configure the preset ModbusTCP parameters.

15.2.1 Preset()

Configure the preset ModbusTCP parameters.

Syntax 1

```
bool Preset(
    string,
    byte,
    string,
    int,
    string,
    int
)
```

Parameters

`string`	preset device name		
`byte`	Slave ID		
`string`	signal type	"DO", "DI", "RO", "RI"	
`int`	starting address		
`string`	type	"bool", "byte", "int16", "int32", "float", "double", "string"	
`int`	suffix parameter, when type is		
	"int32"	0 Little Endian (CD AB)	1 Big Endian (AB CD) (default)
	"float"	0 Little Endian (CD AB)	1 Big Endian (AB CD) (default)
	"double"	0 Little Endian (CD AB)	1 Big Endian (AB CD) (default)
	"string"	address count (0 by default)	
	This is invalid for other types.		

Return

`bool` preset successfully `True`，preset unsuccessfully `False`

Syntax 2

```
bool Preset(
    string,
    byte,
    string,
    int,
    string
)
```

Note

Same as syntax 1. Fill default to suffix parameter by default.

Note

```
ModbusRTU mbus1 = "COM2",115200
   // construct a device with Baudrate 115200, Parity none, DataBits 8, StopBits 1, Timeout 10000ms

mbus1.Preset("light", 1, "DO", 7206, "bool")       // set preset device name to "light"
mbus1.Preset("9000", 1, "RO", 9000, "string")     // set preset device name to "9000"
        // Return error, naming must be in alphanumeric combination.

mbus1.Preset("preset_9000", 1, "RO", 9000, "int", 0)
        // set preset device name to "preset_9000" // Little Endian
mbus1.Preset("preset_9000", 1, "RO", 9000, "int", 1)
        // set preset device name to "preset_9000" // Big Endian
mbus1.Preset("preset_9000", 1, "RO", 9000, "int")
        // set preset device name to "preset_9000" // Big Endian
mbus1.Preset("preset_9000", 1, "RO", 9000, "string", 5)
        // set preset device name to "preset_9000" // string type
        // Once the name, preset_9000, exists, it overwrites the content to the configuration of the same name by the
        processing sequence.

bool flag = modbus_read("mbus1", "light")          // flag = false     // suppose camera light is off
modbus_write("mbus1", "light", true)               // write true       // camera light is on
flag = modbus_read("mbus1", "light")               // flag = true

modbus_write("mbus1 ", "preset_9000", Ctrl("\0\0\0\0\0\0\0\0\0\0"))
```

// clears preset_9000 in string type and occupies five addresses (5 RO = 10 bytes)

```
modbus_write("mbus1", "preset_9000", 1234)          // write "1234"    // because preset_9000 is in string type
int var_i = modbus_read("mbus1", "preset_9000")     // var_i = 1234    // it tries to convert "1234" to integer.
  // because preset_9000 is in string type and occupies five addresses, it then reads 10 bytes and converts by strings. (ends when
encountered 0x00.)

modbus_write("mbus1", "preset_9000", "HelloWorld")        // write "HelloWorld"
string var_s = modbus_read("mbus1", "preset_9000")        // var_s = "HelloWorld"
var_i = modbus_read("mbus1", "preset_9000")
// Return error, unable to convert HelloWorld to integer.

modbus_write("mbus1", "preset_9000", 1234)                // write "1234"
  // contiune the last piece of data and write "HelloWorld" // therefore, the current data in the address 9000 is "1234oWorld"
var_i = modbus_read("mbus1", "preset_9000")
// Return error, unable to convert 1234oWorld to integer.
```

15.2.2 IODDPreset()

Read IODD file and configure the preset ModbusTCP parameters.

Syntax 1

```
bool IODDPreset(
    string,
    byte,
    int,
    int
)
```

Parameters

string	IODD file name (read the IODD file stored in the directory .\XmlFiles\IODD at local host)
byte	Slave ID
int	starting address In
int	starting address Out

Return

bool	preset successfully True, preset unsuccessfully False

Note

```
ModbusRTU mbus1 = "COM2",115200
  // construct a device with Baudrate 115200, Parity none, DataBits 8, StopBits 1, Timeout 10000ms

mbus1.IODDPreset("OMRON-E2EQ-X3B4-IL2-20170301-IODD1.1.xml", 1, 100, 200)
  // load the file, .\XmlFiles\IODD\OMRON-E2EQ-X3B4-IL2-20170301-IODD1.1.xml, and add the preset
  configuration parameters.
  // define preset names in the same way as flow projects rules.
```

15.3 modbus_open()

Open the connection to the Modbus TCP/RTU device.

Syntax 1

```
bool modbus_open(
    string
)
```

Parameters

`string`	TCP/RTU device name

Return

`bool`	`True`	open successfully
	`False`	open unsuccessfully

Note

```
ModbusTCP mbus1 = "127.0.0.1"      // construct a device, with IP 127.0.0.1, Port 502, Timeout 10000ms
modbus_open("mbus1")               // connect to the device with IP: 127.0.0.1 and port: 502.
```

15.4 modbus_close()

Close the connection from the Modbus TCP/RTU device.

Syntax 1

```
bool modbus_close(
    string
)
```

Parameters

`string`	TCP/RTU device name

Return

`bool`	`True`	close successfully
	`False`	close unsuccessfully

Note

```
ModbusTCP mbus1 = "127.0.0.1"       // construct a device, with IP 127.0.0.1, Port 502, Timeout 10000ms
modbus_open("mbus1")                // connect to the device with IP: 127.0.0.1 and port: 502.
modbus_close("mbus1")               // close the connection.
```

15.5 modbus_read()

Modbus TCP/RTU read function

Syntax 1 (TCP/RTU)

```
? modbus_read(
    string,
    string
)
```

Parameters

`string`	TCP/RTU device name
`string`	The predefined parameters belong to TCP/RTU device

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 follows 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)

```
value = modbus_read("TCP_1", "preset_800")     // value = 1       // DO 1 address = 1 bit
value = modbus_read("TCP_1", "preset_7202")    // value = true    // DI 1 address = 1 bit
```

value = **modbus_read**("TCP_1", "preset_9000") // value = ab1234cd // RO 4 address = 8 bytes size
value = **modbus_read**("TCP_1", "preset_7001") // 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		
`byte`	Slave ID		
`string`	Read type		
	`DO`	Digital Output	(FC 01 Read Coil Status)
	`DI`	Digital Input	(FC 02 Read Input Status)
	`RO`	Register Output	(FC 03 Read Holding Registers)
	`RI`	Register Input	(FC 04 Read Input Registers)
`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

value = **modbus_read**("TCP_1", 0, "DO", 800, 4)
// value = {0,0,0,0} // DO 4 address = 4 bit to byte array
value = **modbus_read**("TCP_1", 0, "DI", 7202, 3)
// value = {1,0,0} // DI 3 address = 3 bit to byte array
value = **modbus_read**("TCP_1", 0, "RO", 9000, 6)
// value = {0x54,0x65,0x63,0x68,0x6D,0x61,0x6E,0xE9,0x81,0x94,0xE6,0x98}
// RO 6 address = 12 bytes size
value = **modbus_read**("TCP_1", 0, "RI", 7001, 12)
// 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
value = **modbus_read**("TCP_1", 0, "RI", 7301, 6)
// value = {0x07,0xE2,0x00,0x05,0x00,0x12,0x00,0x0F,0x00,0x0A,0x00,0x39}

```
// RI 6 address = 12 bytes size
```

15.6 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
`byte`	Slave ID
`string`	Read type

`DO`	Digital Output	(FC 01 Read Coil Status)
`DI`	Digital Input	(FC 02 Read Input Status)
`RO`	Register Output	(FC 03 Read Holding Registers)
`RI`	Register Input	(FC 04 Read Input Registers)

`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

```
value = modbus_read_int16("TCP_1", 0, "DO", 800, 4)
        // byte[] = {0,0,0,0}        to int16[]    value = {0,0}        // byte[0][1] , byte[2][3]
value = modbus_read_int16("TCP_1", 0, "DI", 7202, 3)
        // byte[] = {1,0,0}          to int16[]    value = {256,0}
        // byte[0][1] , byte[2][3] // Fill up to [3] automatically
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[]         value = {21605,25448,28001,28393,-32364,-6504}
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}
        // to int16[]         value = {10544,-24756,-15492,-26214,17502,-4915,17076,0,-32768,0,0,0}
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[]         value = {2018,5,18,15,49,35}
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[]         value = {2018,5,18,15,49,35}
```

15.7 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		
byte	Slave ID		
string	Read type		
	DO	Digital Output	(FC 01 Read Coil Status)
	DI	Digital Input	(FC 02 Read Input Status)
	RO	Register Output	(FC 03 Read Holding Registers)
	RI	Register Input	(FC 04 Read Input Registers)
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

```
value = modbus_read_int32("TCP_1", 0, "DO", 800, 4)
        // byte[] = {0,0,0,0}        to int32[]     value = {0}  // byte[0][1][2][3]
value = modbus_read_int32("TCP_1", 0, "DI", 7202, 3)
        // byte[] = {1,0,0}          to int32[]     value = {16777216}       // byte[0][1][2][3] // Fill up to [3] automatically.
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[]         value = {1415930728,1835101929,-2120948072}
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}
        // to int32[]         value = {691052364,-1015244390,1147071693,1119092736,-2147483648,0}
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[]         value = {132251653,1179663,3211299}
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}
```

15.8 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		
`byte`	Slave ID		
`string`	Read type		
	`DO`	Digital Output	(FC 01 Read Coil Status)
	`DI`	Digital Input	(FC 02 Read Input Status)
	`RO`	Register Output	(FC 03 Read Holding Registers)
	`RI`	Register Input	(FC 04 Read Input Registers)
`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

value = modbus_read_float("TCP_1", 0, "DO", 800, 4)
        // byte[] = {0,0,0,0}      to float[]     value = {0}  // byte[0][1][2][3]
value = modbus_read_float("TCP_1", 0, "DI", 7202, 3)
        // byte[] = {1,0,0}        to float[]     value = {2.350989E-38} // byte[0][1][2][3]
        // Fill up to [3] automatically.
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[]        value = {3.940861E+12,4.360513E+27,-5.46975E-38}
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}
        // to float[]        value = {3.921802E-14,-252.6,891.7,90,0,0}
value = modbus_read_float("TCP_1", 0, "RI", 7001, 12, 0)   // byte[2][3][0][1]
        // to float[]        value = {0x9F4C2930,0x999AC37C,0xECCD445E,0x000042B4,0x00008000,0x00000000}
                                   = {-4.323275E-20,-1.600218E-23,-1.985221E+27,2.392857E-41,4.591775E-41,0}
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[]        value = {3.400471E-34,1.65306E-39,5.326512E-39}
```

15.9 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		
byte	Slave ID		
string	Read type		
	DO	Digital Output	(FC 01 Read Coil Status)
	DI	Digital Input	(FC 02 Read Input Status)
	RO	Register Output	(FC 03 Read Holding Registers)
	RI	Register Input	(FC 04 Read Input Registers)
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

```
value = modbus_read_double("TCP_1", 0, "DO", 800, 4)
        // byte[] = {0,0,0,0}        to double[]          value = {0}  // byte[0][1][2][3][4][5][6][7]
value = modbus_read_double("TCP_1", 0, "DI", 7202, 3)
        // byte[] = {1,0,0}          to double[]          value = {7.2911220195564E-304}    // byte[0][1][2][3][4][5][6][7]
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[]       value = {3.65481260356117E+98,-4.87647898854073E-301}
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[]       value = {2.76472410615396E-110,2.2818627604613E+21,0}
value = modbus_read_double("TCP_1", 0, "RI", 7001, 12, 0)        // byte[6][7][4][5][2][3][0][1]
        // to double[]       value = {0x999AC37C9F4C2930,0x000042B4ECCD445E,0x0000000000008000}
                                  = {-2.4604103205376E-185,3.62371629877526E-310,1.6189543082926E-319}
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[]       value = {1.06475148078395E-270,1.52982527955113E-308}
```

15.10 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		
byte	Slave ID		
string	Read type		
	DO	Digital Output	(FC 01 Read Coil Status)
	DI	Digital Input	(FC 02 Read Input Status)
	RO	Register Output	(FC 03 Read Holding Registers)
	RI	Register Input	(FC 04 Read Input Registers)
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 達明机器手臂"

// 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}

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�

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�

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 達明机器手臂 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

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

15.11 modbus_write()

Modbus TCP/RTU write function

Syntax 1 (TCP/RTU)

```
bool modbus_write(
    string,
    string,
    ?,
    int
)
```

Parameters

`string`	TCP/RTU Device Name
`string`	TCP/RTU The predefined parameters belong to TCP/RTU device
`?`	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)

`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

When this parameter is skipped (As shown in Syntax2), the predefined address length will be applied.

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 2 (TCP/RTU)

```
bool modbus_write(
    string,
    string,
    ?,
)
```

Note

Similar to Syntax1 with predefined address length to write. If the predefined address length <= 0, it will write all the data.

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

preset_800	DO	800	bool	
preset_9000	RO	9000	string	4

```
modbus_write("TCP_1", "preset_800", 1)          // write 1 (true)
modbus_write("TCP_1", "preset_800", 0)          // write 0 (false)
bool flag = true
modbus_write("TCP_1", "preset_800", flag)       // write 1 (true)
modbus_write("TCP_1", "preset_800", false)      // write 0 (false)

string ss = "ABCDEFGHIJKLMNOPQRST"              // 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", 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		
byte	Slave ID		
string	Write type		
	DO	Digital Output	(FC 05/15 Write Single/Multiple Coil(s))
	RO	Register Output	(FC 06/16 Write Single/Multiple Register(s))
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, bb) => **modbus_write**("TCP_1", 0, "RO", 9000, 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[] bb = {10, 20, 30}

modbus_write("TCP_1", 0, "DO", 800, bb) // write 1,1,1

```
                                                // Zero value, write 0. Non-zero value, write 1.
modbus_write("TCP_1", 0, "DO", 800, bb, 2)      // write 1,1
                                                // Address number = 2, only write 2 addresses.
modbus_write("TCP_1", 0, "DO", 800, true)       // write 1
int i = 10000
modbus_write("TCP_1", 0, "RO", 9000, i)         // write 0x00,0x00,0x27,0x10
                                                // with int32 BigEndian (AB CD) default
bb = GetBytes(i, 0, 1)                          // bb = {0x10,0x27}
                                                // transfer to int16 LittleEndian (CD AB)
modbus_write("TCP_1", 0, "RO", 9000, bb)        // write 0x10,0x27
string[] n = {"ABC", "12", "34"}
modbus_write("TCP_1", 0, "RO", 9000, n, 2)      // write ABC1
                                                // Only 2 addresses available, the exceeding values cannot
                                                be applied.
modbus_write("TCP_1", 0, "RO", 9000, n, 5)      // write ABC12340
                                                // The data needs 4 addresses (0xAB 0xC1 0x23 0x40)
```

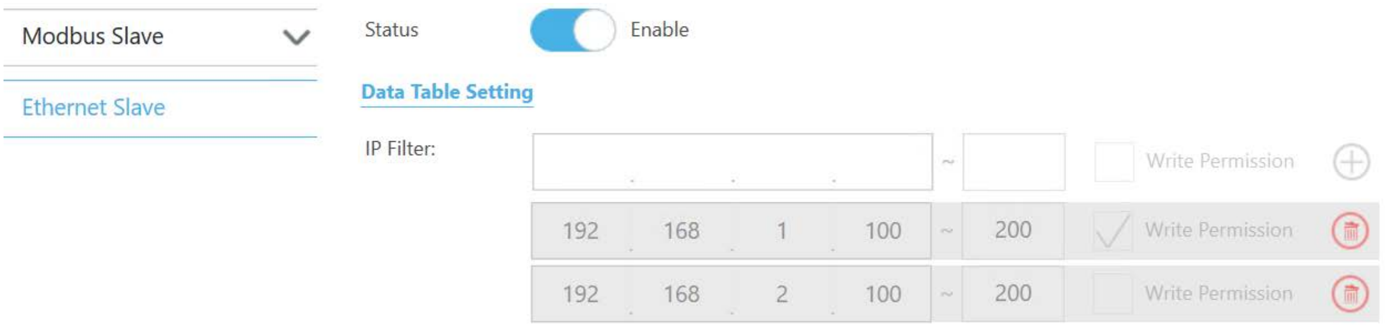
16. TM Ethernet Slave

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 window.

IP	TMflow → System → Network → IP Address
Port	5891

16.1 GUI Setting



IP Filter						Write Permission
192	168	1	100	~	200	✓
192	168	2	100	~	200	

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 1 192.168.1.100 ~ 200 denotes IP 192.168.1.100, 192.168.1.101, … , and 192.168.1.200 are available for connections.

Group 2 192.168.2.100 ~ 200 denotes IP 192.168.2.100, 192.168.2.101, … , and 192.168.2.200 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 1, 192.168.1.100 ~ 200, has permission to write, so clients connecting within this group range sending data to Ethernet Slave makes Ethernet Slave write data. Group 2, 192.168.2.100 ~ 200, does not have permission to write. When sending data to Ethernet Slave, it does not write data and will respond with the error code of write permission.

16.2 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

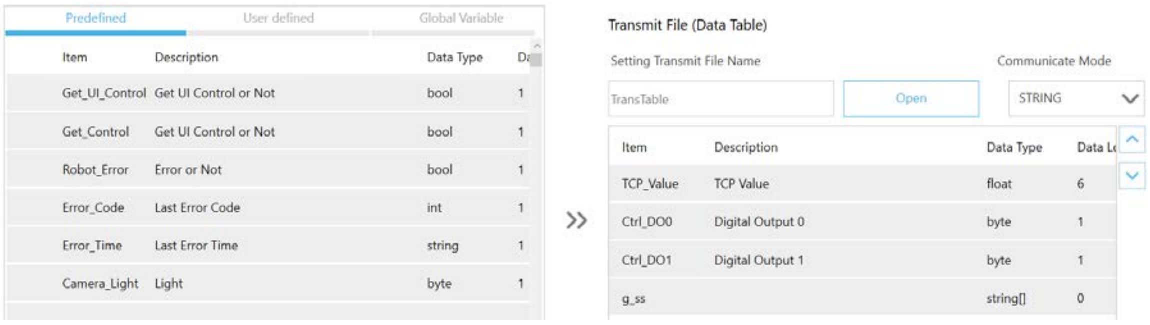
`string`	Item name

Return

`?`	Return value by set data type

Note

Suppose taking TCP_Value float[], Ctrl_DO0 byte, Ctrl_DO1 byte, and g_ss string[] as the communication data table.



Item	Description	Data Type	Da
Get_UI_Control	Get UI Control or Not	bool	1
Get_Control	Get UI Control or Not	bool	1
Robot_Error	Error or Not	bool	1
Error_Code	Last Error Code	int	1
Error_Time	Last Error Time	string	1
Camera_Light	Light	byte	1

Transmit File (Data Table)

Setting Transmit File Name: TransTable — Open

Communicate Mode: STRING

Item	Description	Data Type	Data Le
TCP_Value	TCP Value	float	6
Ctrl_DO0	Digital Output 0	byte	1
Ctrl_DO1	Digital Output 1	byte	1
g_ss		string[]	0

```
float[] fva0= svr_read("TCP_Value")          // {1, 1, 1, 0.1, 0.2, 0.1}
byte b0 = svr_read("Ctrl_DO0")               // 0
byte b1 = svr_read("Ctrl_DO1")               // 1
string[]ss = svr_read("g_ss")                // {"Hi","TM","Robot"}
ss = g_ss                                    // make the variable to take, g_ss, a variable name directly
byte st = svr_read("Robot_Link")             // 0 (Robot disconnected) 1 (Robot connected)

float[] fva1 = svr_read("TCP_Value")         // Report error. Suppose Ethernet Slave is not launched.
float[] fva2 = svr_read("TCP_Value1")        // Report error. Item name TCP_Value1 does not exist.

float[] fva3 = svr_read("g_ff")              // Report error.. Item name g_ff does not exist in the communication
                                             table.
// If g_ff is assumed to exist among the global variables, it cannot be accessed because not it does not load all global
variables.
float[] fva4 = svr_read("Coord_Base_Flange")    // {0.01,-252.6,891.7,90,0,0}
// Although added as the communication data, it's accessible for item name Coord_Base_Flange is in the system
definitions.
```

16.3 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

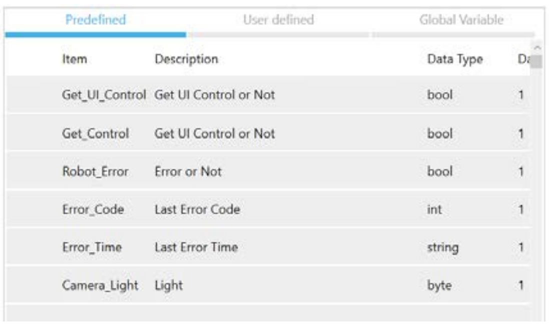
string	Item name
?	Item value

Return

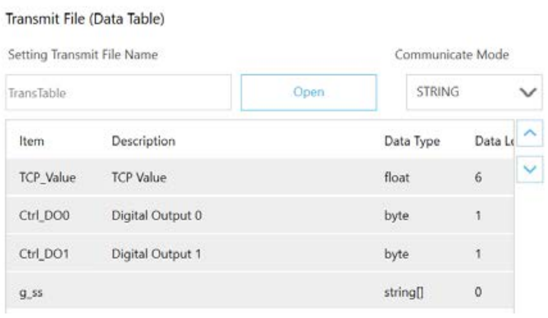
bool	True	Write successfully	
	False	Write failed	Possible causes 1. Item name does not exist. 2. Unable to write the read-only item name. 3. Item value to write mismatched with item data type.

Note

Suppose taking TCP_Value float[], Ctrl_DO0 byte, Ctrl_DO1 byte, and g_ss string[] in the communication data table.



Item	Description	Data Type	Da
Get_UI_Control	Get UI Control or Not	bool	1
Get_Control	Get UI Control or Not	bool	1
Robot_Error	Error or Not	bool	1
Error_Code	Last Error Code	int	1
Error_Time	Last Error Time	string	1
Camera_Light	Light	byte	1



Item	Description	Data Type	Data Le
TCP_Value	TCP Value	float	6
Ctrl_DO0	Digital Output 0	byte	1
Ctrl_DO1	Digital Output 1	byte	1
g_ss		string[]	0

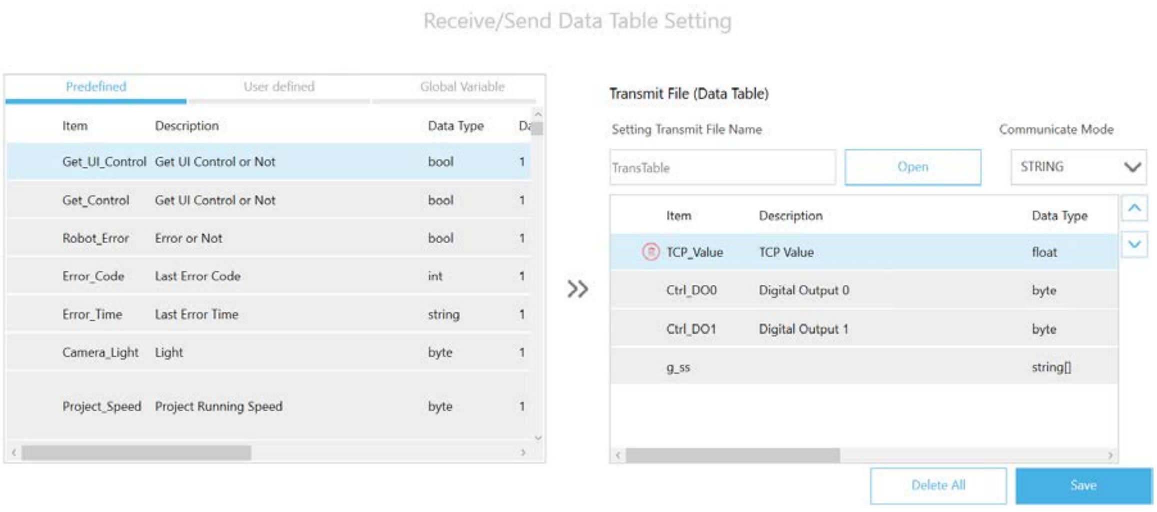
```
float[] tvalue = {1,2,3,0.1,0.2,0.3}
bool flag = false
flag = svr_write("TCP_Value", tvalue)       // flag = false    read-only, invalid process (not an error)
flag = svr_write("Ctrl_DO0", 1)             // flag = true , Ctrl_DO0 = 1
flag = svr_write("Ctrl_DO1", 0)             // flag = true , Ctrl_DO1 = 0

flag = svr_write("TCP_Value", tvalue)       // Error. Suppose Ethernet Slave is not launched.
flag = svr_write("TCP_Value1", tvalue)      // Error. Item name TCP_Value1 does not exist.
flag = svr_write("Ctrl_DO0", "True")        // Error. Item name Ctrl_DO0 writes value as string (the data type is
                                            set to byte)
```

16.4 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 communication 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 Setting

Predefined | User defined | Global Variable

Item	Description	Data Type	Da
Get_UI_Control	Get UI Control or Not	bool	1
Get_Control	Get UI Control or Not	bool	1
Robot_Error	Error or Not	bool	1
Error_Code	Last Error Code	int	1
Error_Time	Last Error Time	string	1
Camera_Light	Light	byte	1
Project_Speed	Project Running Speed	byte	1

Transmit File (Data Table)

Setting Transmit File Name: TransTable

Communicate Mode: STRING

Item	Description	Data Type
TCP_Value	TCP Value	float
Ctrl_DO0	Digital Output 0	byte
Ctrl_DO1	Digital Output 1	byte
g_ss		string[]

The configuration interface is a left-to-right mechanism. Users can add items at the left to the communication data table at the right and adjust the arrangement order of each item in the communication data table at the right. In the content to send, there will always be an item, ***Robot_Link***, predefined in Ethernet Slave as a type of byte with the attribute of read-only to denote whether to connect to the robot.

1. ***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.

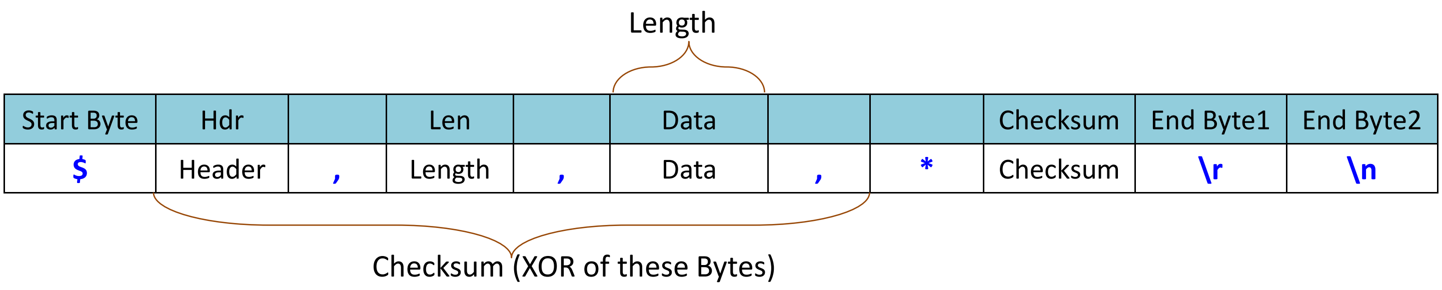
2. ***User defined***

   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.

3. ***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.

16.5 Communication Protocol



Start Byte	Hdr		Len		Data			Checksum	End Byte1	End Byte2
$	Header	**,**	Length	**,**	Data	**,**	*****	Checksum	**\r**	**\n**

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)

16.6 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 User defined area and the Global Variable 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        // transaction ID S0, string format, set Ctrl_DO0=1
>    $TMSVR,10,S0,0,00,OK,*18\r\n
                          // server responds transaction ID S0, mode 0, error code 00, correct writing

<    $TMSVR,16,S1,99,Ctrl_DO0=1,*46\r\n       // transaction ID S1, mode 99
>    $TMSVR,18,S1,0,01,NotSupport,*0E\r\n
                          // server responds transaction ID S1, mode 0, error code 01, mode not support

<    $TMSVR,15,S2,2,Ctrl_DO0=1,*74\r\n        // transaction ID S2, string format, set Ctrl_DO0=1
>    $TMSVR,23,S2,0,02,WritePermission,*6A\r\n
     // server responds transaction 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        // transaction ID S3, JSON format, set Ctrl_DO0=1
>    $TMSVR,19,S3,0,03,InvalidData,*74\r\n
     // server responds transaction ID S3, mode 0, error code 03, JSON format, data format (JSON) mismatched with
the content data format (STRING)

<    $TMSVR,16,S4,2,Ctrl_DO32=1,*40\r\n       // transaction ID S4, string format, set Ctrl_DO32=1
>    $TMSVR,26,S4,0,04,NotExist;Ctrl_DO32,*58\r\n
     // server responds transaction ID S4, mode 0, error ode 04, item Ctrl_DO32 does not exist.
```

< $*TMSVR*,17,S5,*2*,Robot_Link=1,*07\r\n // transaction ID S5, string format, set Robot_Link=1

> $*TMSVR*,27,S5,*0*,05,ReadOnly;Robot_Link,*1E\r\n
// server responds transaction 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 // transaction ID S6, string format, set adata={1,2,3,4}

> $*TMSVR*,23,S6,*0*,06,ModeError;adata,*2D\r\n
// server responds transaction ID S6, mode 0, error code 06, M/A mode mismatched while writing (suppose it Manual Mode while writing).

< $*TMSVR*,18,S7,*2*,adata={1,2,3},*47\r\n // transaction ID S7, string format, set adata={1,2,3}

> $*TMSVR*,24,S7,*0*,07,ValueError;adata,*42\r\n
// server responds transaction 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.

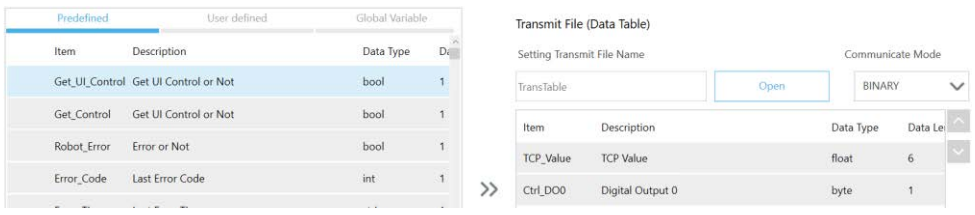
Data

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

Suppose taking Check TCP_Value float[] and Ctrl_DO0 byte as the communication data and transmitting in binary mode.



Item	Description	Data Type	Da
Get_UI_Control	Get UI Control or Not	bool	1
Get_Control	Get UI Control or Not	bool	1
Robot_Error	Error or Not	bool	1
Error_Code	Last Error Code	int	1

Transmit File (Data Table)

Setting Transmit File Name: TransTable

Communicate Mode: BINARY

Item	Description	Data Type	Data Le
TCP_Value	TCP Value	float	6
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.

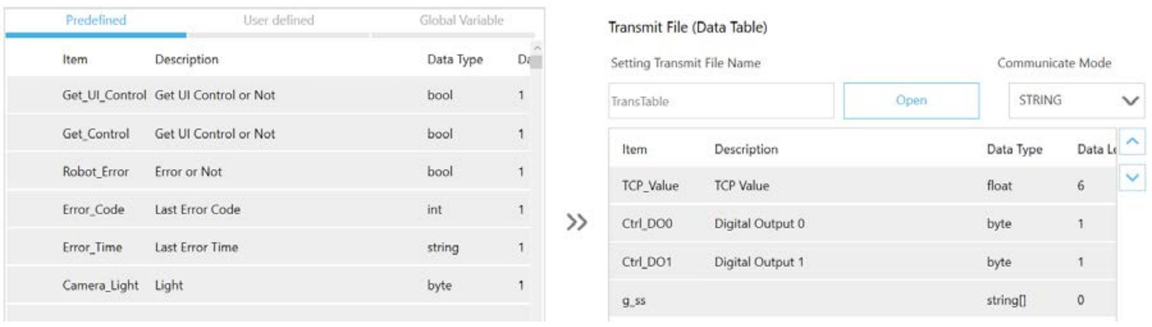
Data

ID		Mode		Content
Transaction ID	,	2	,	Item and Value

Item	**=**	***Value***	**\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.

Suppose taking Check TCP_Value float[] and Ctrl_DO0 byte, Ctrl_DO1 byte, g_ss string[] as the communication data and transmitting in string mode.



Item	Description	Data Type	Da
Get_UI_Control	Get UI Control or Not	bool	1
Get_Control	Get UI Control or Not	bool	1
Robot_Error	Error or Not	bool	1
Error_Code	Last Error Code	int	1
Error_Time	Last Error Time	string	1
Camera_Light	Light	byte	1

Transmit File (Data Table)

Setting Transmit File Name: TransTable

Communicate Mode: STRING

Item	Description	Data Type	Data Le
TCP_Value	TCP Value	float	6
Ctrl_DO0	Digital Output 0	byte	1
Ctrl_DO1	Digital Output 1	byte	1
g_ss		string[]	0

```
>   $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

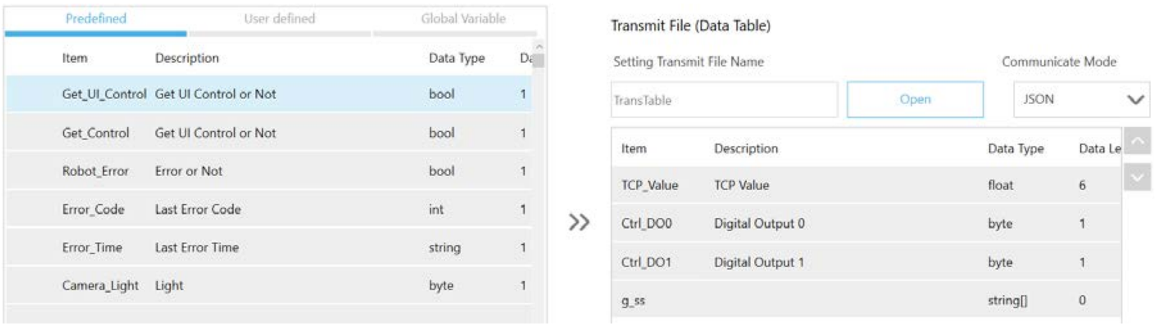
ID		Mode		Content
Transaction ID	,	3	,	Item and Value

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.

Suppose taking TCP_Value float[] and Ctrl_DO0 byte, Ctrl_DO1 byte, g_ss string[] as the communication data and transmitting in JSON mode.


Predefined | User defined | Global Variable

Item	Description	Data Type	Da
Get_UI_Control	Get UI Control or Not	bool	1
Get_Control	Get UI Control or Not	bool	1
Robot_Error	Error or Not	bool	1
Error_Code	Last Error Code	int	1
Error_Time	Last Error Time	string	1
Camera_Light	Light	byte	1

Transmit File (Data Table)

Setting Transmit File Name: TransTable Open

Communicate Mode: JSON

Item	Description	Data Type	Data Le
TCP_Value	TCP Value	float	6
Ctrl_DO0	Digital Output 0	byte	1
Ctrl_DO1	Digital Output 1	byte	1
g_ss		string[]	0

```
>   $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)

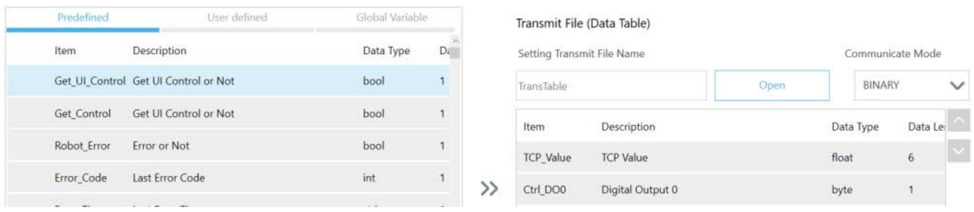
ID		Mode		Content
Transaction ID	,	*11*	,	Item

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

Suppose taking TCP_Value float[] and Ctrl_DO0 byte as the communication data and transmitting in binary mode.


Predefined | User defined | Global Variable

Item	Description	Data Type	Da
Get_UI_Control	Get UI Control or Not	bool	1
Get_Control	Get UI Control or Not	bool	1
Robot_Error	Error or Not	bool	1
Error_Code	Last Error Code	int	1

Transmit File (Data Table)

Setting Transmit File Name: TransTable | Open

Communicate Mode: BINARY

Item	Description	Data Type	Data Le
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 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)

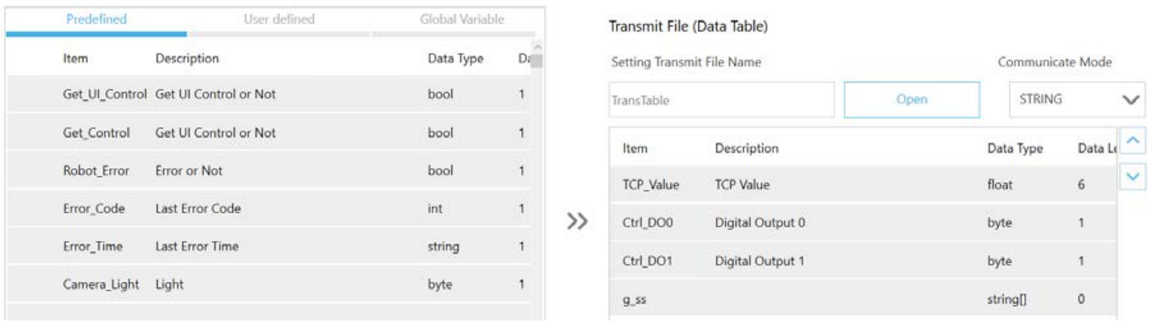
ID		Mode		Content
Transaction ID	*,*	*12*	*,*	Item and Value

Item	***\r\n***	***…***

No Value required.

Item	Item name
\r\n	The newline characters. Required only as a delimiter if the next item comes.

Suppose taking TCP_Value float[] and Ctrl_DO0 byte, Ctrl_DO1 byte, g_ss string[] as the communication data and transmitting in STRING mode.



Item	Description	Data Type	Da
Get_UI_Control	Get UI Control or Not	bool	1
Get_Control	Get UI Control or Not	bool	1
Robot_Error	Error or Not	bool	1
Error_Code	Last Error Code	int	1
Error_Time	Last Error Time	string	1
Camera_Light	Light	byte	1

Transmit File (Data Table)

Setting Transmit File Name: TransTable   Open

Communicate Mode: STRING

Item	Description	Data Type	Data L
TCP_Value	TCP Value	float	6
Ctrl_DO0	Digital Output 0	byte	1
Ctrl_DO1	Digital Output 1	byte	1
g_ss		string[]	0

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)

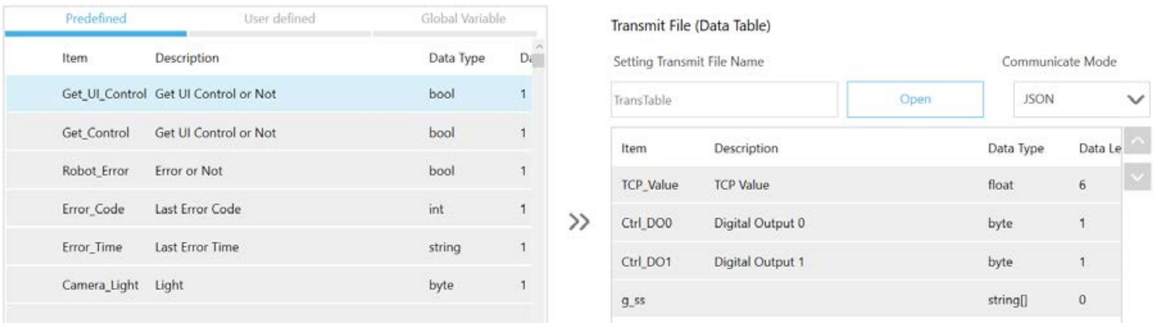
ID		Mode		Content
Transaction ID	*,*	***13***	*,*	Item and Value

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

Suppose taking TCP_Value float[] and Ctrl_DO0 byte, Ctrl_DO1 byte, g_ss string[] as the communication data and transmitting in JSON mode.



Item	Description	Data Type	Da
Get_UI_Control	Get UI Control or Not	bool	1
Get_Control	Get UI Control or Not	bool	1
Robot_Error	Error or Not	bool	1
Error_Code	Last Error Code	int	1
Error_Time	Last Error Time	string	1
Camera_Light	Light	byte	1

Item	Description	Data Type	Data Le
TCP_Value	TCP Value	float	6
Ctrl_DO0	Digital Output 0	byte	1
Ctrl_DO1	Digital Output 1	byte	1
g_ss		string[]	0

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

17. 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 Definition Section	Read/Write	Read

17.1 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_ba = **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_b = **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 default.
* 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,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        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,
        0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,
        0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,
        0x00000000,0x00000000,0x00000000} (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}
var_ia = profinet_read_input("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,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
        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[] = {0x00000001,0x00000002,0x00000003,0x00000004,0x00000005,0x00000006,
        0x00000007,0x00000008,0x00000009,0x0000000A,0x0000000B,0x0000000C,
        0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,
        0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,
        0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000} (Little Endian)
    // int[] = {1,2,3,4,5,6,7,8,9,10,11,12,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
float[] var_fa = profinet_read_input("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 = profinet_read_input("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,
        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[] = {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[] = {2.1,2.2,2.3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
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[] = {1.1,1.2,1.3,2.1,2.2,2.3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
```

17.2 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[] 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[] 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[] 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[] 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[] 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
)
```

Parameters

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 = profinet_read_input_int(164)
    // byte[] = {0xE4,0x07,0x00,0x00} (Little Endian)        to int
    // int = 0x000007E4 (Little Endian)
    // int = 2020
var_i = profinet_read_input_int(164)
    // byte[] = {0x00,0x00,0x07,0xE4} (Big Endian)           to int
    // int = 0x000007E4 (Big Endian)
    // int = 2020
```

17.3 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

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_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_f = **profinet_read_input_float**(284)

// byte[] = {0x00,0x00,0x80,0x3F} (Little Endian) to float

// float = 0x3F800000 (Little Endian)

// float = 1.0

var_f = **profinet_read_input_float**(284)

// byte[] = {0x3F,0x80,0x00,0x00} (Big Endian) to float

// float = {0x3F800000} (Big Endian)

// float = {1.0}

17.4 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_s = **profinet_read_input_string**(148,16)

    // byte[] = {0x54,0x4D,0x35,0x2D,0x37,0x30,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
    // string = "TM5-700"

var_s = **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"

var_s = **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"

var_s = **profinet_read_input_string**(148,10)

    // byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6,0x98,0x8E}
    // string = "abcd 達明"

var_s = **profinet_read_input_string**(148,8)

    // byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6}
    // string = "abcd 達�"

17.5 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 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_b = profinet_read_input_bit(148,0)
    // 0x30      get bit: "0"
    // 0
var_b = 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`	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 = 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}
```

17.6 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_ba = **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_b = **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}

var_int 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,
                0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)      to int[]
        // int[] = {0x00000004,0x00000005,0x00000006,0x00000007,0x00000008,0x00000009,
                0x0000000A,0x0000000B,0x0000000C,0x00000000,0x00000000,0x00000000,
                0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,
                0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,
                0x00000000,0x00000000,0x00000000} (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}
    var_ia = profinet_read_output("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,
                0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                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[] = {0x00000001,0x00000002,0x00000003,0x00000004,0x00000005,0x00000006,
                0x00000007,0x00000008,0x00000009,0x0000000A,0x0000000B,0x0000000C,
                0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,
                0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,
                0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000} (Little Endian)
        // int[] = {1,2,3,4,5,6,7,8,9,10,11,12,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}

    float var_f = profinet_read_output("Current_TCP_Mass")
        // byte[] = {0x40,0x40,0x00,0x00} (Big Endian)          to float
        // float = 0x40400000 (Big Endian)
        // float = 3.0

    var_fa = profinet_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 = profinet_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 = profinet_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 = profinet_read_output ("RobotModel",0,16)
        // byte[] = {0x54,0x4D,0x35,0x2D,0x37,0x30,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
        // string = "TM5-700"
    var_s = profinet_read_output ("RobotModel",4,3)
        // byte[] = {0x37,0x30,0x30}
        // string = "700"
```

```
var_s = profinet_read_output ("RobotModel")
        // byte[] = {0x54,0x4D,0x35,0x2D,0x37,0x30,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
        // string = "TM5-700"
```

17.7 profinet_read_output_int()

Read the output table content and convert the data to the 32-bit integer.

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_ia = 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}
var_ia = 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}
var_ia = 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}

var_ia = 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}

var_ia = 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}

    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}
```

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_i = profinet_read_output_int(556)
        // byte[] = {0xE4,0x07,0x00,0x00} (Little Endian)        to int
        // int = 0x000007E4 (Little Endian)
        // int = 2020
    var_i = profinet_read_output_int(556)
        // byte[] = {0x00,0x00,0x07,0xE4} (Big Endian)           to int
        // int = 0x000007E4 (Big Endian)
        // int = 2020
```

17.8 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_fa = 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}
var_fa = 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}
var_fa = 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}

var_fa = 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}

var_fa = 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}

    var_fa = 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_f = profinet_read_output_float(676)
    // byte[] = {0x00,0x00,0x80,0x3F} (Little Endian) to float
    // float = 0x3F800000 (Little Endian)
    // float = 1.0
var_f = profinet_read_output_float(676)
    // byte[] = {0x3F,0x80,0x00,0x00} (Big Endian) to float
    // float = 0x3F800000 (Big Endian)
    // float = 1.0
```

17.9 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_s = **profinet_read_output_string**(540,16)

// byte[] = {0x54,0x4D,0x35,0x2D,0x37,0x30,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}

// string = "TM5-700"

var_s = **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"

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,0x31,0x32,0x33,0x34,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}

// string = "abcd 達明機器人 1234"

var_s = **profinet_read_output_string**(540,10)

// byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6,0x98,0x8E}

// string = "abcd 達明"

var_s = **profinet_read_output_string**(540,8)

// byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6}

// string = "abcd 達�"

17.10 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_b = profinet_read_output_bit(540,0)
    // 0x30      get bit: "0"
    // 0
var_b = 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}
```

17.11 profinet_write_output()

Write data to the output table.

Syntax 1

```
bool profinet_write_output (
    int,
    ?,
    int
)
```

Parameters

`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,0x31,0x32,0x33,0x34,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
        // 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,
                    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
        // 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,
                    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
        // 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}
```

17.12 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 nth 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_b = profinet_read_output(540)
    // 0xF0
profinet_write_output_bit(540,1,1)
var_b = profinet_read_output_bit(540,1)
    // 0xF2      get bit: "1"
    // 1
profinet_write_output_bit(540,7,0)
var_b = 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	he nth 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 = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1}
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[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1}
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[] = {0,0,0,1,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0}
```

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

`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 = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1}
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}
```

18. 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 Definition Section	Read/Write	Read

18.1 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} (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} (Big Endian)
    // float[] = {2.1,2.2,2.3,0,0,0,0,0,0,0,0,0}

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} (Big Endian)      to float[]
    // float[] = {0x3F8CCCCD,0x3F99999A,0x3FA66666,0x40066666,0x400CCCCD,0x40133333,
          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}
```

18.2 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
```

18.3 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

18.4 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�"
```

18.5 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 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_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}
```

18.6 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(
    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"
```

18.7 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}
```

Syntax 3

```
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
```

18.8 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
```

18.9 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�"
```

18.10 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}
```

18.11 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

`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 eip_write_output(
    int,
    ?,
    int,
    int
)
```

Parameter

`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
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}
```

18.12 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`	he nth 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

`string`	Item name
`int`	The nth 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("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

`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
`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 4

```
bool eip_write_output_bit(
    int,
    int,
    byte[],
    int
)
```

Parameter

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

Parameter

`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 = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1}
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 = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1}
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[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1}
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[] = {0,0,0,1,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0}
```

19. Force Control Functions

19.1 FTSensor Class

Use FTSensor class and declare variables to create a Force Torque Sensor device. The variable name will be the device name.

Construct 1

```
FTSensor VariableName = string, string, float[], float[], float
FTSensor VariableName = string, float[], float[], float
FTSensor VariableName = string, string
FTSensor VariableName = string
```

Parameters

`string`	Supported models from the sensor suppliers
	`"ATI_Axia80"`
	`"OnRobot_HEX-E"`
	`"OnRobot_HEX-H"`
	`"ROBOTIQ_FT300"`

`string`	connection description required when using serial port for communication
`float[]`	Position setting: X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)
`float[]`	TCP value: X(mm), Y(mm), Z(mm)
`float`	Tool Mass: kg

Note

```
FTSensor fts_1 = "ATI_Axia80","COM2"      // Construct the device ATI_Axia80. The connection parameters are void for
                                           using EtherCAT communication.
FTSensor fts_2 = "ATI_Axia80"             // Construct the device ATI_Axia80
FTSensor fts_3 = "ROBOTIQ_FT300","COM2"
      // Construct the device ROBOTIQ_FT300. Require to assign the connection port for using serial port communication.
FTSensor fts_4 = "ROBOTIQ_FT300","COM2",{0,0,0,0,0,0},{0.12,0.24,0.36},1
      // Construct the device ROBOTIQ_FT300. Require to assign the connection port for using serial port communication and to
      configure the settings of the sensor position and tool mass center.
```

* If not filling position setting, TCP value, or tool mass, it uses the parameters in the force sensor setting in the project.

* After construction, either in flow projects or script projects, the device will not connect actively until proceeding to read or write.

Member Attributes

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 torque value	
Value	float[]	R	The XYZ force strength value and torque value array.	{X, Y, Z, TX, TY, TZ}, Size = 6
ForceValue	float[]	R	The XYZ force strength value array	{X, Y, Z}, Size = 3
TorqueValue	float[]	R	The XYZ 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

Attributes associated with RefCoord come with values when in Force Control motions.

Member Methods

Name	Description
Open()	Connect to the sensor device.
Close()	Disconnect from the sensor device.

19.1.1 Open()

Open the connection to the Force Torque Sensor.

Syntax 1

```
bool Open(
)
```

Parameters

void	No parameter

Return

bool	True	open successfully
	False	open unsuccessfully (the project returns an error)

* After opening the device, it proceeds to the connection to communicate. It may take a while to get the values due to different sensor models.

19.1.2 Close()

Close the connection from the Force Torque Sensor.

Syntax 1

```
bool Close(
)
```

Parameters

void	No parameter

Return

bool	True	close successfully
	False	close unsuccessfully

Note

```
FTSensor fts_1 = "ROBOTIQ_FT300","COM2"
fts_1.Open()     // Connect to the device.
fts_1.Close()    // Disconnect from the device.
```

19.2 Force Class

Use Force class and declare variables for users to set the robot target force and torque parameters to reach a variety of associated force control motions.

Construct 1

Force *VariableName* = `string`

Parameters

`string`	sensor name

Note

```
FTSensor fts1 = "ATI_Axia80"     // Construct the device ATI_Axia80.
Force fc1 = "fts1"               // Declare the force control variable and tether fts1 as the sensor name.
```

Member Methods

Name	Defaults	Description
Reset()		Reset all the force control motion parameters to the default except the sensor name.
Frame()	1	Set the reference coordinate of the force control motion.
StopDuration()	200	The duration of compliance stop. Set the length of time to switch between force control and position control.
Distance()	-	The moving distance of the force control motion (Available only in the SetPoint F/T operation mode.)
FTSet()	0,false,5,2 1,false,5,2 2,false,5,2 3,false,0.5,2 4,false,0.5,2 5,false,0.5,2	The force and the torque of the force control motion. The positive and negative values denote the force control direction. Users can adjust the PID control parameters.
Trajectory()	-	The trajectory path of the force control motion. (Trajectory F/T operation mode.)
Timeout()	-	Stop condition of timeout (Available only in the single F/T operation mode.)
AllowPosTol()	-	Stop condition of tolerant position error
DInput()	-	Stop condition of digital input
AInput()	-	Stop condition of analog input
FTReached()	-	The stop condition of monitoring the values of force, torque, or resultant force on the sensor.
Condition()	-	Stop condition of the conditional expression
Start()	true,false	Start the force control motion.
Stop()		Stop the force control motion.

19.2.1 Reset()

Reset all the force control motion parameters to the default except the sensor name.

Syntax 1

```
void Reset(
```

)

Parameters

`void`	No parameter

Return

`void`	No return

19.2.2 Frame()

Set the reference coordinate of the force control motion.

Syntax 1

```
void Frame(
    int
)
```

Parameters

`int` The base associated with the force control motion

0	The robot base
1	Tool: the base is coupled with the orientation of the tool coordinate. (default)
2	The current base
3	Trajectory: the base changes with the path.

Return

`void`	No return

Syntax 2

```
void Frame(
    string
)
```

Parameters

`string`	Point name. Use the TCP coordinate of the point as the base associate with the force control motion.

Return

`void`	No return

Syntax 3

```
void Frame(
    float[],
    string
)
```

Parameters

`float[]`	The TCP coordinate of the robot end point: X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)
`string`	Base name. Use the current base name if empty string.

Return

`void`	No return

Syntax 4

```
void Frame(
    float[]
)
```

Note

Same as syntax 3. Fill RobotBase in Base name.

Syntax 5

```
void Frame(
    float, float, float, float, float, float,
    string
)
```

Note

Same as syntax 3. Replace the type float[] with the parameters in float .

Syntax 6

```
void Frame(
    float, float, float, float, float, float
)
```

Note

Same as syntax 4. Replace the type float[] with the parameters in float .

19.2.3 StopDuration()

Set the duration of compliance stop. Set the length of time to switch between force control and position control.

Syntax 1

```
void StopDuration(
    int
)
```

Parameters

`int`	The duration of compliance stop in milliseconds

Return

`void`	No return

19.2.4 Distance()

Set the moving distance of the force control motion. (Available only in the SetPoint F/T operation mode.)

Syntax 1

```
void Distance(
    int
)
```

Parameters

`int` Distance mm

< 0	No limit to the moving distance.
>= 0	Limited to the moving distance. (Distance calculated from the starting point where the force control motion begins.)

Return

`void`	No return

19.2.5 FTSet()

Set the force and the torque of the force control motion. The positive and negative values denote the force control direction. Users can adjust the PID control parameters.

Syntax 1

```
void FTSet(
    int or string,
    bool,
    float,
    int
)
```

Parameters

`int or string`
Axis force or axis torque

0	or	"FX"	FX (N)	3	or	"TX"	TX (Nm)
1	or	"FY"	FY (N)	4	or	"TY"	TY (Nm)
2	or	"FZ"	FZ (N)	5	or	"TZ"	TZ (Nm)

`bool`
Whether to enable the control of the assigned axis force or axis torque

false	Disable
true	Enable

`float`
The control value of force or torque. The positive and negative values denote the force control direction.

`int`
PID control parameter

(weak)				(strong)
0	1	2	3	4

Return

`void` No return

Syntax 2

```
void FTSet(
    int or string,
    bool,
    float
)
```

Note

Same as syntax 1. Fill 2 to PID control parameter.

Syntax 3

```
void FTSet(
    int or string,
    bool
)
```

Note

Same as syntax 1 parameter definition. Use to set whether to enable the control of the assigned axis force or axis torque

Syntax 4

```
void FTSet(
    int or string,
    float
)
```

Note

Same as syntax 1 parameter definition. Use to set the control value of force or torque.

Syntax 5

```
void FTSet(
    int or string,
    int
)
```

Note

Same as syntax 1 parameter definition. Use to set PID control parameter.

Syntax 6

```
void FTSet(
    int or string,
    float[]
)
```

Parameters

`int or string`

Axis force or axis torque

0	or	"FX"	FX (N)	3	or	"TX"	TX (Nm)
1	or	"FY"	FY (N)	4	or	"TY"	TY (Nm)
2	or	"FZ"	FZ (N)	5	or	"TZ"	TZ (Nm)

`float[]` PID control parameter {Kp, Ki, Kd}

19.2.6 Trajectory()

.

Set the trajectory path of the force control motion. (Trajectory F/T operation mode.)

Syntax 1

```
void Trajectory(
    string
)
```

Parameters

`string` Subflow name. Use in flow projects only. Return errors if use in script projects.

Return

`void` No return

Syntax 2

```
void Trajectory(
    ?
)
```

Parameters

`?` Trajectory path. Be a statement or a customized function

Return

`void` No return

Syntax 3

```
void Trajectory(
)
```

Parameters

`void`

No parameter for cancelling trajectory F/T operation mode. (Change to the SetPoint F/T operation mode.)

Return

`void`	No return

19.2.7 Timeout()

Set the stop condition of timeout. (Available only in the SetPoint F/T operation mode.)

Syntax 1

```
void Timeout(
    int
)
```

Parameters

`int`	Timeout in milliseconds	
	< 0	Disable
	>= 0	Timeout duration

Return

`void`	No return

Syntax 2

```
void Timeout(
)
```

Parameters

`void`	No parameter for cancelling the stop condition.

Return

`void`	No return

19.2.8 AllowPosTol()

Set the stop condition of the tolerant position error.

Syntax 1

```
void AllowPosTol(
    int
)
```

Parameters

`int`	Error distance in mm	
	< 0	Disable
	>= 0	Distance of the tolerant error.

Return

`void`	No return

Syntax 2

```
void AllowPosTol(
)
```

Parameters

`void`	No parameter for cancelling the stop condition.

Return

`void`	No return

19.2.9 DInput()

Set the stop condition of the digital input.

Syntax 1

```
void DInput(
    string,
    int,
    int or string
)
```

Parameters

string	Control module name	
	ControlBox	The control box
	EndModule	The end module
	ExtModuleN	The external module (N = 0 .. n)
int	Input channel 0 .. n	
int or string		
	Set the stop condition to Low/High.	
	0 or "L"	Low
	1 or "H"	High

Return

void	No return

Syntax 2

```
void DInput(
)
```

Parameters

void	No parameter for cancelling the stop condition.

Return

void	No return

19.2.10 AInput()

Set the stop condition of the analog input.

Syntax 1

```
void AInput(
    string,
    int,
    int or string,
    float
)
```

Parameters

string	Control module name	
	ControlBox	The control box
	EndModule	The end module
	ExtModuleN	The external module (N = 0 .. n)
int	Input channel 0 .. n	
int or string		
	Set the condition to judge	
	0 or ">"	Greater than
	1 or ">="	Greater than or equal to

2	or	"=="	
			Equal to (Recommend not to use since it is not easy to hold the equal condition with analog input.)
3	or	"<="	Less than or equal to
4	or	"<"	Less than

`float` Condition value

Return

`void` No return

Syntax 2

```
void AInput(
)
```

Parameters

`void` No parameter for cancelling the stop condition.

Return

`void` No return

19.2.11 FTReached()

Set the stop condition of monitoring the values of force, torque, or resultant force on the sensor.

Syntax 1

```
void FTReached(
    int or string,
    bool,
    float
)
```

Parameters

`int` The values of force, torque, or resultant force.

0	or	"FX"	FX (N)	3	or	"TX"	TX (Nm)
1	or	"FY"	FY (N)	4	or	"TY"	TY (Nm)
2	or	"FZ"	FZ (N)	5	or	"TZ"	TZ (Nm)
6	or	"F3D"	F3D (N)	7	or	"T3D"	T3D (Nm)

`bool` Whether to enable monitoring the assigned the values of force, torque, or resultant force.

false	Disable
true	Enable

`float` The monitoring value

Return

`void` No return

Syntax 2

```
void FTReached(
    int or string,
    bool
)
```

Note

Same as syntax 1. Use to whether to enable monitoring the assigned the values of force, torque, or resultant force.

Syntax 3

```
void FTReached(
    bool
)
```
Parameters

`bool`	Whether to enable the absolute values monitoring.	
	false	Disable
	true	Enable

Syntax 4

```
void FTReached(
)
```
Parameters

`void`	No parameter for cancelling the stop condition.

Return

`void`	No return

19.2.12 Condition()

Set the stop condition of the conditional expression.

Syntax 1

```
void Condition(
    bool or ?
)
```
Parameters

`bool or ?`	Condition. Be true/false or a statement returning bool values.

Return

`void`	No return

Syntax 2

```
void Condition(
)
```
Parameters

`void`	No parameter for cancelling the stop condition.

Return

`void`	No return

19.2.13 Start()

Start the force control motion.

Syntax 1

```
int Start(
    bool,
    bool
)
```
Parameters

`bool`	Zero out force sensor before execution.

	`true`	Enable (default)
	`false`	Disable
`bool`	Enable tool gravity compensation	
	`true`	Enable
	`false`	Disable (default)

Return

`int`	Return the result value after the force control motion stops.	
	0	Not Working
	1	Working
	2	Timeout
	3	Distance reached
	4	IO triggered
	5	Resisted
	6	Error
	14	Over Speed
	201	Digital IO triggered
	202	Analog IO triggered
	203	Variable
	204	Force is satisfied
	205	Allowable Position Tolerances
	206	Motion Finish

Syntax 2

```
int Start(
    bool
)
```

Note

Same as syntax 1. Enabling the tool gravity compensation parameter will retrieve the parameter values of the associated device name in the force sensor setting of the project. Fill false if not retrieving the associated device name.

Syntax 3

```
int Start(
)
```

Note

Same as syntax 1. Fill true to zero out force sensor before execution. Enabling the tool gravity compensation parameter will retrieve the parameter values of the associated device name in the force sensor setting of the project. Fill false if not retrieving the associated device name.

19.2.14 Stop()

Stop the force control motion.

Syntax 1

```
int Stop(
)
```

Parameters

`void`	No parameter

Return

`int`	Return the result value after the force control motion stops.

Setting Parameters

```
FTSensor fts1 = "ATI_Axia80"           // Construct the device ATI_Axia80.
Force fc1 = "fts1"                     // Declare the force control variable and tether fts1 as the sensor name.

(1)
    fc1.Frame(3)                           // Set the force control coordinate as trajectory.
    fc1.Frame({517.5,-147.8,442.45,180,0,90}) // Set the force control coordinate as point (will overwrite the
                                           previous setting)
    fc1.Frame("P1")                        // Set the force control coordinate as point (will overwrite the
                                           previous setting)
    fc1.Reset()                            // Reset all parameters (reserve tethering "fts1" as sensor name)
(2)
    fc1.Frame(3)
    fc1.Frame({517.5,-147.8,442.45,180,0,90})
    fc1.Frame("P1")
    fc1.FTSet(0, true, 5, 0)               // Set axis FX with force control 5N with PID 0.
    fc1.FTSet(1, true, 6, 1)               // Set axis FY with force control 6N with PID 1.
    fc1.FTSet(2, true, 7, 3)               // Set axis FZ with force control 7N with PID 3.
    fc1.FTSet("FZ", true, 8)               // Set axis FZ with force control 8N with PID 2. (will overwrite the
                                           previous FZ setting)
    fc1.Reset()                            // Reset all parameters (reserve tethering "fts1" as sensor name)
(3)
    fc1.Frame(1)                           // Set the force control coordinate as tool.
    fc1.StopDuration(300)                  // Set the duration of compliance stop. to 300ms
    // Force and torque control
    fc1.Distance(1000)                     // Set moving distance to 1000mm
    fc1.FTSet("FZ", true, 5)               // Set axis FZ with force control 5N
    // Stop conditions
    fc1.Timeout(10000)                     // Set timeout to 10000ms
    fc1.AllowPosTol(100)                   // Set the tolerant error to 100mm
    fc1.DInput("ControlBox", 0, "H")       // When ControlBox DI0 is High
    fc1.AInput("ControlBox", 0, ">=", 3.3) // When ControlBox AI0 is greater than or equal to 3.3V
    fc1.FTReached(2, true, 1)              // Axis FZ satisfies 1N
    fc1.FTReached("FZ", true, 2)           // Axis FZ satisfies 2N (will overwrite the previous FZ setting)
    fc1.FTReached(true)                    // Enable the absolute value monitoring
    int count = 0
    fc1.Condition(count > 100)             // conditional expression
    fc1.Reset()                            // Reset all parameters (reserve tethering "fts1" as sensor name)
```

SetPoint F/T Operation Mode

SetPoint mode applies mainly to the robot touching the target with force control.

```
FTSensor fts1 = "ATI_Axia80"
Force fc1 = "fts1"
(1)
    PTP("JPP",{0,0,90,0,90,0},50,200,0,false)
    fc1.Distance(100)                          // Set moving distance to 100mm
    fc1.FTSet("FZ", true, 5)                   // Set axis FZ with force control 5N
    fc1.Start()                                // Start the force control motion. Zero out force sensor before
                                               execution.
    //    Set axis FZ with force control 5N and the moving distance of 100mm, and the robot will move along with the Z-
          direction. The robot stops moving once it is 100mm away from the starting point. However, the motion is still
          under the force control, namely in the Start() function.

(2)
    PTP("JPP",{0,0,90,0,90,0},50,200,0,false)
    fc1.FTSet("FZ", true, 5)                   // Set axis FZ with force control 5N
    fc1.AllowPosTol(80)                        // The stop condition of the tolerant position error.
    int re = fc1.Start()                       // Start the force control motion. Zero out force sensor before
                                               execution.
    //    Set axis FZ with force control 5N, and the robot will move along with the Z-direction. Under the concurrent
          monitoring stop conditions, by satisfying the stop condition, the robot stops moving once it is 80mm away
          from the starting point, exits the Start() function, and returns the value 205.

(3)
    PTP("JPP",{0,0,90,0,90,0},50,200,0,false)
    fc1.Frame(1)                               // Set the force control coordinate as工具
    fc1.StopDuration(300)                      // Set the duration of compliance stop.to 300ms
    // Force and torque control
    fc1.Distance(200)                          // Set moving distance to 200mm
    fc1.FTSet("FZ", true, 5)                   // Set axis FZ with force control 5N
    // Stop conditions
    fc1.Timeout(10000)                         // Set timeout to 10000ms
    fc1.AllowPosTol(1000)                      // Set the tolerant error to 1000mm
    fc1.DInput("ControlBox", 0, "H")           // When ControlBox DI0 is High
    fc1.AInput("ControlBox", 0, ">=", 3.3)     // When ControlBox AI0 >= 3.3V
    int count = 0
    fc1.Condition(count > 100)                 // conditional expression
    int re = fc1.Start()                       // Start the force control motion. Zero out force sensor before
                                               execution.
    //    Set axis FZ with force control 5N and the moving distance of 200mm, and the robot will move along with the Z-
          direction. Under the concurrent monitoring stop conditions, the robot stops moving once it is more than
          200mm from the starting point. However, the motion is still under force control, and the stop condition
          monitoring continues. Not until any stop conditions are satisfied will the force control motion be stopped, the
          Start() function exit, and the result values after stopping be returned.
```

Trajectory F/T Operation Mode

Trajectory mode applies mainly to the robot touching the target with robot motion control and force control.

```
FTSensor fts1 = "ATI_Axia80"
Force fc1 = "fts1"
(1)
        fc1.FTSet("FZ", true, 2)                // Set axis FZ with force control 2N
        fc1.Trajectory("FTSubflow0")            // Set FTSubflow0 as the trajectory path. (Use in flow projects only.)
        int re = fc1.Start()                    // Start the force control motion. Zero out force sensor before execution.
        //      Suppose FTSubflow0 comes with a few point nodes to finish.
        //      Set axis FZ with force control 2N and the trajectory path as the subflow FTSubflow0. The robot follows the
                nodes in the subflow to control and with force control concurrently. When the subflow ends, it stops the force
                control motion, exits Start() function, and returns the value 206.
(2)
        fc1.FTSet("FZ", true, 2)                // Set axis FZ with force control 2N
        fc1.Frame(3)                            // Set the force control coordinate as trajectory
        fc1.Trajectory("FTSubflow0")            // Set FTSubflow0 as the trajectory path. (Use in flow projects only.)
        int re = fc1.Start()                    // Start the force control motion. Zero out force sensor before execution.
        //      Suppose FTSubflow0 comes with a few point nodes to finish.
        //      Set axis FZ with force control 2N and the trajectory path as the subflow FTSubflow0. The robot follows the
                nodes in the subflow to control and with force control concurrently. When the subflow ends, it stops the force
                control motion, exits Start() function, and returns the value 206.

(3)
        fc1.FTSet("FZ", true, 2)                // Set axis FZ with force control 2N
        fc1.Frame(1)                            // Set the force control coordinate as tool
        fc1.Trajectory("FTSubflow1")            // Set FTSubflow1 as the trajectory path. (Use in flow projects only.)
        // Stop conditions
        fc1.Timeout(10000)                      // Timeout is invalid in the trajectory mod.
        fc1.AllowPosTol(1000)                   // Set the tolerant error to 1000mm
        fc1.DInput("ControlBox", 0, "H")        // When ControlBox DI0 is High
        fc1.FTReached("FZ", true, 2)            // Axis FZ satisfies 2N
        fc1.FTReached(true)                     // Enable the absolute value monitoring
        int re = fc1.Start()                    // Start the force control motion. Zero out force sensor before execution.
        //      Suppose FTSubflow1 comes with a few point nodes to loop the loop.
        //      Set axis FZ with force control 2N and the trajectory path as the subflow FTSubflow1. The robot follows the
                nodes in the subflow to control and with force control concurrently. Since the subflow is looping the loop, the
                robot motion, the force control, and the stop condition monitoring keep on. Not until any stop conditions are
                satisfied will the force control motion be stopped, the Start() function exit, and the result values after stopping
                be returned.

(4)
        define
        {
                FTSensor fts1 = "ATI_Axia80"
                Force fc1 = "fts1"
                int count = 0
        }
        main
        {
```

```
        fc1.FTSet("FZ", true, 1, 0)             // Set axis FZ with force control 8N with PID 0
        fc1.Frame(1)                            // Set the force control coordinate as tool
        fc1.Trajectory(FTMotion())              //Set FTMotion() customized function as the trajectory path.
        // Stop conditions
        fc1.Timeout(10000)                      // Timeout is invalid in the trajectory mode.
        fc1.AllowPosTol(1000)                   // Set the tolerant error to 1000mm
        fc1.DInput("ControlBox", 0, "H")        // When ControlBox DI0 is High
        fc1.FTReached("FZ", true, 2)            // Axis FZ satisfies 2N
        fc1.FTReached(true)                     // Enable the absolute value monitoring
        fc1.Condition(count++ > 200000)         // conditional expression
        int re = fc1.Start()

                                                // Start the force control motion. Zero out force sensor before
                                                execution.
        Display(re)
}
void FTMotion()
{
        while (true)
        {
                PTP("JPP",{15,0,90,0,90,0},50,200,100,false)
                PTP("JPP",{15,0,75,0,90,0},50,200,100,false)
                PTP("JPP",{-15,0,75,0,90,0},50,200,100,false)
                PTP("JPP",{-15,0,90,0,90,0},50,200,100,false)
                Sleep(10)
        }
}
//      Set axis FZ with force control 1N, PID 0, and the trajectory path as FTMotion() customized function. The robot
        follows the contents of FTMotion() to control and with force control concurrently. Since this function is
        looping the loop, the robot motion, the force control, and the stop condition monitoring keep on. Not until
        any stop conditions are satisfied will the force control motion be stopped, the Start() function exit, and the
        result values after stopping be returned.
```

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.
438B Alexandra Road, #08-01/02 Alexandra
Technopark, Singapore 119968
Tel: (65) 6835-3011 Fax: (65) 6835-3011

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) 21-6023-0333 Fax: (86) 21-5037-2388

Authorized Distributor:

©OMRON Corporation 2023 All Rights Reserved.
In the interest of product improvement,
specifications are subject to change without notice.

Cat. No. I664-E-02 0723 (0121)

19888-820B