欧姆龙
协作机器人：TMscript
语言手册

I664-CN5-02

协议条款及条件

质保责任限制条款

质保

- 排他性质保

  欧姆龙的排他性质保条款保证，产品自欧姆龙销售之日起12个月内（或欧姆龙书面确认的其他期限）不存在材料和工艺方面的缺陷。欧姆龙对所有其他明示或暗示的质保概不负责。

- 限制

  欧姆龙对产品的非侵权性、适销性或特定用途的适用性不做任何明示或暗示的保证或陈述。买方承认其已自行确定本产品将适当地满足其预期用途的要求。

  此外，欧姆龙对基于产品侵权或其他任何知识产权侵权的任何索赔或费用不做任何保证，亦不承担任何责任。

- 买方补救措施

  欧姆龙在本协议项下的义务就是，由欧姆龙自行选择：（i）更换（采用最初交付的形式，且由买方负责拆卸或更换产品的人工费用）不符合规定的产品；（ii）维修不符合规定的产品；（iii）向买方支付与不符合规定产品购买价格相等的金额或将此金额存入买方账户；前提是在任何情况下，欧姆龙对产品的质保、维修、赔偿或任何其他索赔或费用概不负责，除非欧姆龙经过分析确认产品得到了妥善处理、储存、安装和维护，且未受到污染、滥用、误用或不当修改。买方退货的任何产品必须在发货前获得欧姆龙的授权。欧姆龙公司对产品与任何电气或电子元件、电路、系统组件或任何其他材料、物质或环境的组合使用而产生的适用性、不适用性或结果概不负责。以口头或书面形式提供的任何意见、建议或信息都不应被解释为对上述质保的修改或补充。

请访问：http://www.omron.com/global/或联系您的欧姆龙销售代表获取相关出版资料。

责任限制：其他

欧姆龙公司对以任何方式与产品有关的特殊、非直接、附带或间接损害、利润或生产损失、商业损失概不负责，无论该索赔是否基于合同、质保、疏忽或严格责任。

此外，在任何情况下，欧姆龙公司的责任均不超过所主张责任所依据产品的个别价格。

应用注意事项

适用性

欧姆龙公司对买方应用或使用本产品时是否遵守适用于本产品组合的任何标准、规范或法规概不负责。应买方要求，欧姆龙将提供适用的第三方认证文件，文件确定了适用于产品的额定值和使用限制。该信息本身并不足以完全确定产品与最终产品、机器、系统或其他应用或用途组合的适用性。买方应全权负责确定特定产品与买方的应用、产品或系统的适合性。买方在任何情况下都应对产品的应用承担责任。

在未确保整个系统的设计旨在解决相关风险，以及在未确保欧姆龙产品经过适当评级和安装，可在整个设备或系统中用于预期用途的情况下，切勿将本产品用于会严重危及生命或造成巨大财产损失的应用中。

可编程产品

欧姆龙公司对用户的可编程产品的编程或由此产生的任何后果概不负责。

免责声明

性能数据

欧姆龙公司网站、目录以及其他材料中提供的数据应作为用户确定适宜性的指南，但并不构成质保。性能数据可能是欧姆龙测试条件的结果，用户必须将其与实际的应用要求相关联。实际性能以欧姆龙的质保和责任限制条款的规定为准。

- 规格变更

产品规格和配件可能会由于改进及其他原因而随时更改。每当发布的额定值或功能发生变化时，或当发生重大的结构更改时，我们通常都会更改部件编号。然而，产品的某些规格如有更改，恕不另行通知。如有疑问，可指定特殊的部件编号来修复或建立针对您应用的关键规格。请随时咨询您的欧姆龙销售代表，以确认所购买产品的实际规格。

错误和疏忽

欧姆龙公司提供的信息已被核实，并被认为是准确的；但欧姆龙公司对文书、印刷或校对方面的错误或疏忽概不负责。

网络安全威胁责任声明

为维护系统的安全性和可靠性，应执行稳健的网络安全防御计划，涵盖以下的部分或全部方面：

防病毒保护

- 请在连接控制系统的电脑上安装最新版本的企业级杀毒软件并及时维护。
- 如需在控制系统或设备上使用 USB 存储器等外部存储设备，应事先进行病毒扫描。

防止非法访问

- 导入物理控制，确保只有授权人员才能访问控制系统及设备。
- 通过将控制系统及设备的网络连接限制在最低程度，防止未获信任的设备访问。
- 通过部署防火墙，阻止未使用的通信端口及限制系统间的通信。将控制系统及设备的网络与 IT 网络隔离。
- 在控制系统及设备的远程访问中导入多重要素认证。
- 采用复杂密码并频繁更换。

数据输入输出保护

- 定期备份并更新数据，以防数据丢失。
- 请确认备份、范围检查等妥当性，以防对控制系统和设备的输入输出数据受到意外修改。
- 定期验证数据保护范围以适应变更。
- 通过安排测试恢复检验备份的有效性，确保能够从事故中顺利恢复。
- 进行安全设计如紧急停机、应急运行等，以应对数据遭到篡改及异常情况。

补充建议

- 经由外部网络环境连接 SCADA、HMI 等未经授权的终端或未经授权的服务器，可能会面临恶意伪装、数据篡改等网络安全问题。
- 请客户自行采取充分有效的安全防护措施以防范网络攻击，例如限制终端访问，使用配备安全功能的终端，对面板设置区域实施上锁管理等。
- 构建网络基础设施时，可能会因电缆断线、未经授权的网络设备的影响，导致通信故障的发生。
- 请采取充分有效的安全防护措施，例如通过对面板设置区域实施上锁管理等方法，限制无权限人员对网络设备的物理访问。
- 使用配备 SD 存储卡功能的设备时，可能存在第三方通过拔出或非法卸载移动存储介质等方式非法获取、篡改、替换移动存储介质内的文件及数据的安全风险。

修订历史表

版本	日期	说明
1.00	2021 年 1 月	初版
2.00	2023 年 6 月	新增 2.0 功能

1. 概述

TMscript 是适用于流程项目和脚本项目的 Techman 机器人编程语言。其主要应用范围如下表所示。

- 应用范围

应用	流程项目			脚本项目
	设置节点（及其他节点）	Listen 节点（外部命令）	脚本节点	
表达式	✓	✓	✓	✓
define/main/closestop/errorstop				✓
自定义函数				✓
注释		✓	✓	✓
项目内的全局变量	✓	✓	✓	✓
函数内的局部变量		✓	✓	✓
多行输入		✓	✓	✓
条件语句		✓	✓	✓
循环语句		✓	✓	✓
分支语句		✓	✓	✓
线程函数				✓
通用函数	✓	✓	✓	✓
通用函数（脚本）		✓	✓	✓
数学函数	✓	✓	✓	✓
文件函数	✓	✓	✓	✓
串行端口类		✓	✓	✓
串行通讯函数	✓	✓	✓	✓
Socket 类				
Socket 函数	✓	✓	✓	✓
参数化对象	✓	✓	✓	✓
机器人示教类		✓	✓	✓
机器人运动及视觉任务函数		✓	✓	✓
视觉函数	✓	✓	✓	✓
外部命令		✓		
Modbus TCP/RTU 类		✓	✓	✓
Modbus 函数	✓	✓	✓	✓
TM Ethernet Slave	✓	✓	✓	✓
Profinet 函数	✓	✓	✓	✓
EtherNet/IP 函数	✓	✓	✓	✓
FTSensor 类		✓	✓	✓
Force 类		✓	✓	✓

2. 表达式

2.1 类型

可在变量管理器中声明多种数据类型的变量。

byte	8 位整数	无符号	0 至 255	3 位有效数字
int	32 位整数	有符号	-2147483648 至 2147483647	10 位有效数字
float	32 位浮点数	有符号	-3.4028235E+38 至 3.4028235E+38	7 位有效数字
double	64 位浮点数	有符号	-1.7976931348623157E+308 至 1.7976931348623157E+308	15 位有效数字
bool	布尔值		true 或 false	
string	字符串			

对于函数，整数类型可进一步分为 int16 和 int32 两种。默认类型为 int32。

int16	16 位整数	有符号	-32768 至 32767	5 位有效数字
int32	32 位整数	有符号	-2147483648 至 2147483647	10 位有效数字

2.2 变量和常量

1. 变量

根据变量命名规则，变量名只能包含数字、下划线和大小写英文字符。

数字　　0123456789
字符　　a-z、A-Z、_

示例

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

不带双引号的字符串将被视为变量。

```
s = s1 + " and " + s2      // s = "DEF and 123"
                           //s、s1、s2 为变量，" and "为字符串。
```

除变量外，该命名规则也适用于常量、数字、字符串和布尔值，但需要用双引号括起来的字符串常量除外。

在 TMflow 中生成的变量会附带基于来源的前缀。如需使用变量写入或读取，用户必须输入包括前缀在内的完整变量名，如 var_s1 或 g_s2。附加前缀的规则请参见变量设置页面中的相关说明。

2. 数字

- 支持十进制整数、十进制浮点数、二进制数、十六进制整数和科学记数法。

十进制整数	123
	-123
	+456
十进制浮点数	34.567
	-8.9
二进制数	0b0000111
	0B1110000
十六进制整数	0x123abc
	0X00456DEF
科学记数法	3.4e5
	2.3E-4

- 二进制和十六进制表示法不支持浮点数。
- 数字表示法不区分大小写。

  例如：

  0b0011　等同于 0B0011
  0xabcD　等同于 0XABCD、0xABCd、0Xabcd 等
  3.4e5　　等同于 3.4E5

- 使用数字作为常量时，系统将自动决定数字的数据类型。规则是从小到大决定位类型，例如：

```
100          //数据类型：byte          //100 在 byte 数据类型的取值范围内。
1000         //数据类型：int
1.11         //数据类型：float         //1.11 在 float 数据类型的取值范围内。
```

可通过变量声明或转换指定数据类型，例如：

```
byte b = 100   //变量 b = 100（数据类型：byte）
int i = 100    //变量 i = 100（数据类型：int）
(int)100       //常量 100（数据类型：int）
(float)100     //常量 100（数据类型：float）
```

对于函数调用，系统将决定参数的数据类型并选择相应语法。若无相应语法，系统将根据从小到大决定位类型的规则决定兼容语法，例如：

```
GetBytes(100, 0, 0)  //{0x64,0x00,0x00,0x00}//100 和 0 的数据类型为 byte，但语法为 GetBytes(int, int, int)。此时，系统会将 100 和 0 转换为 int 数据类型，继续执行调用语法。
GetBytes(100)        //{0x64}//100 的数据类型为 byte，而 GetBytes(?)语法支持任意数据类型。此时，系统会将 100 视为 byte 数据类型的常量，继续执行调用语法。
```

- byte 类型只能表示 0 至 255 的 8 位无符号数字。因此，若通过赋值或计算赋给 byte 类型负值，byte 类型仍将只保存 8 位无符号值。

  例如：

```
byte b = -100          //错误     //-100 超出了 byte 类型的取值范围。
byte b = 0 - 100       // b = 156// 0-100=-100（0xFFFFFF9C）。保存 8 位，即 0x9C。该值等于 156。
b = 0 - 1              // b = 255// 0-1=-1（0xFFFFFFFF），保存 8 位，即 0xFF，该值等于 255。
b = 255 + 1            // b = 0    // 255+1=256（0x100），保存 8 位，即 0x00，该值等于 0。
```

- int 类型只能表示-2147483648 至 2147483647 的 32 位有符号数字。若计算结果超出此取值范围，int 类型仍将只保存 32 位有符号整数值。

  例如：

```
int i = -2147483648 - 1  // i = 2147483647   // -2147483648 - 1 = -2147483649（0xFFFFFFFF 7FFFFFFF），保存 32 位，即 0x7FFFFFFF。该值等于 2147483647。
i = 2147483647 + 1       // i = -2147483648   // 2147483647 + 1 = 2147483648（0x80000000），按有符号整数值的方法保存 32 位。其值等于-2147483648
```

3. 字符串

输入字符串常量时，应用成对双引号括起字符串，以免混淆变量和字符串。

例如：

```
"Hello World! "
"Hello TM""5"  （若字符串中包含"，应使用两个（""）而不是一个（"）。）
```

- 不支持双引号间的控制字符。

  例如：

```
"Hello World!\r\n"   （输出将为字符串 Hello World!\r\n）
```

- 无双引号时将按以下规则编译
  1. 数字将被视为数字
  2. 若存在相应的变量，数字和字符的组合将被视为变量。
  3. 若不存在相应变量，编译时上述组合将被视为字符串，同时发出警告消息。
- 字符串和变量的组合
  1. 在双引号间，变量无法作为变量被组合

     例如：

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

  2. 标准语法：使用双引号括起字符串，并使用加号（+）连接变量和数字

     示例：

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

  3. 兼容语法（不推荐）：可使用单引号括起变量，但将发出警告消息。

     例如：

```
单引号    "Hi, 's' Robot"         // s1 = "Hi, TM5 Robot"
          "Hi, 'x' Robot"         // s1 = "Hi, 'x' Robot"        //由于变量 x 不存在，'x'被视为字符串
```

  4. 若使用单引号，则无法使用数组索引获取元素值。应使用带双引号的标准格式。

     例如：

```
string[] ss = {"Techman", "Robot"}
"Hi, 's' 'ss[0]' Robot"                // s1 = "Hi, TM5 'ss[0]' Robot"     //'ss[0]'无效
```

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

5. 无法使用"'表示单引号，若用户需要输入'（变量名）'，应使用带双引号的标准格式。
   例如：

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

   //若需要获得 s1 = "Hi, 's' Robot"，请使用以下语法。

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

- 对于控制字符（如换行），请使用 Ctrl()命令。
  例如：

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

  Hi,

  TM5 Robot

- 保留字符与变量类似，不需要双引号。（但不支持单引号）
  1. empty 空字符串，等同于""
  2. newline 或 NewLine 换行，等同于 Ctrl("\r\n")或 Ctrl(0x0D0A)

4. 布尔值

true 或 false 的逻辑值。

表示 true 值	true True
表示 false 值	false False

布尔值区分大小写。误用大写字母（如 TRue）将被视为变量或字符串。

2.3 数组

- 数组是一组数据类型相同的数据。数组的初始值通过{}赋予，每个元素均将保留其数据类型的特征。
  例如：

```
int[] i = {0,1,2,3}                    //元素为数字数据类型
string[] s = {"ABC", "DEF", "GHI"}     //元素为字符串数据类型
bool[] bb = {true, false, true}        //元素为布尔值数据类型
```

- 可利用索引获取特定元素的值，索引从 0 开始
  例如：

索引 0 1 2 3 4 5 6 7

数组  共有 8 个元素

A[0] A[1] A[2] A[3] A[4] A[5] A[6] A[7]

有效索引值为[0] ~ [7]. 若访问的索引值超出此范围（如 A[8]），则会出现错误。

- 仅支持一维数组。索引编号上限为 2048。
- 数组的大小可能因函数返回值或赋值而改变。元素编号上限为 2048。这一特性使数组可满足网络节点中的各种函数和应用的需求。
  例如：

```
string[] ss = {empty, empty, empty}              //该字符串数组的初始大小为 3 个元素
ss = String_Split("A_B_C_D_F_G_H", "_")          //拆分字符串后，该字符串数组包含 7 个元素
len = Length(ss)                                 // len = 7
ss = String_Split("A,B", ",")                    //拆分字符串后，该字符串数组包含 2 个元素
len = Length(ss)                                 // len = 2
```

2.4 运算符

- 运算符表如下所示。
- 计算优先遵循运算符优先级，其次遵循结合律。
  例如：
  从左向右结合
  A = A * B / C % D ➔ (A = (((A * B) / C) % D))
  1 2 3 4
  从右向左结合
  A -= B += 10 && ! !D ➔ (A -= (B += (10 && (! (!D)))))
  1 2 3 4 5
- 计算将按运算数的类型进行。
  1. 两值均为整数类型时，计算将按整数类型进行，例如：
     int var_a = 10
     int var_b = 3
     float var_c = var_a / var_b
     根据运算符优先级，先计算/，再计算=。
     var_a / var_b = 10 / 3 = 3 （var_a 和 var_b 均为整数）
     var_c = 3 （浮点数 var_c 被赋值整数 3）
  2. 两值之一为浮点数类型时，计算将按浮点数类型进行，例如：
     int var_a = 10
     float var_b = 3
     float var_c = var_a / var_b
     var_a / var_b = 10 / 3 = 3.333333 （var_b 为浮点数）
     var_c = 3.333333 （var_c 被赋值浮点数 3.333333）
     var_c = var_a / 3
     var_a / 3 = 10 / 3 = 3 （var_a 和 3 均为整数）
     var_c = 3
     var_c = var_a / 3.0
     var_a / 3.0 = 10 / 3.0 = 3.333333 （3.0 为浮点数）
     var_c = 3.333333

优先级 从高到低	运算符	名称	示例	要求	结合
17	++	后缀增量	i++	整型变量	从左向右
	--	后缀减量	i--		
	()	函数调用	int x = f()		
	[]	分配存储空间	array[4] = 2	数组变量	
16	++	前缀增量	++i	整型变量	从右向左
	--	前缀减量	--i		
	+	一元加	int i = +1	数值变量、常数	
	-	一元减	int i = -1		
	!	逻辑非（NOT）	if (!done) …	布尔值变量、常数	

优先级 从高到低	运算符	名称	示例	要求	结合
	~	按位非	flag1 = ~flag2	整型变量、常数	
14	*	乘	int i = 2 * 4	数值变量、常数	从左向右
	/	除	float f = 10.0 / 3.0		
	%	取模（取整数余数）	int rem = 4 % 3		
13	+	加	int i = 2 + 3	数值变量、常数	
	-	减	int i = 5 - 1		
12	<<	按位左移	int flags = 33 << 1	整型变量、常数	
	>>	按位右移	int flags = 33 >> 1		
11	<	小于	if (i < 42)…	数值变量、常数	
	<=	小于等于	if (i <= 42)…		
	>	大于	if (i > 42)…		
	>=	大于等于	if (i >= 42)…		
10	==	等于	if (i == 42)…		
	!=	不等于	if (I != 42)…		
9	&	按位 AND	flag1 = flag2 & 42	整型变量、常数	
8	^	按位 XOR	flag1 = flag2 ^ 42		
7	|	按位 OR	flag1 = flag2 | 42		
6	&&	逻辑 AND	if (condition A && condition B)		
5	||	逻辑 OR	if (condition A || condition B)		
4	c ? t : f	三元条件	int i = a > b ? a : b		从右向左
3	=	基本赋值	int a = b	左侧：数值变量 右侧：数值变量、常数	
	+=	加赋值	a += 3		
	-=	减赋值	b -= 4		
	*=	乘赋值	a *= 5		
	/=	除赋值	a /= 2		
	%=	取模赋值	a %= 3		
	<<=	按位左移赋值	flags <<= 2	左侧：整型变量 右侧：整型变量、常数	
	>>=	按位右移赋值	flags >>= 2		
	&=	按位 AND 赋值	flags &= new_flags		
	^=	按位 XOR 赋值	flags ^= new_flags		
	|=	按位 OR 赋值	flags |= new_flags		

2.5 数据类型转换

- 数据类型之间可以相互转换，以用于变量/常量或数组中。
- 只能在形式相同的容器间转换，如变量/常量转换或数组转换。无法将变量转换为数组或将数组转换为变量。

原类型	转换后类型	示例	结果
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 等同于 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（非 0 等同于 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（非 0 等同于 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	错误
	int	int i = (int)False	错误
	float	float f = (float)true	错误
	double	double d = (double)false	错误
	string	string s = (string)True	s = "true"（小写）
string	byte	byte b1 = (byte)"1.23" byte b2 = (byte)"XYZ"	1 错误（无法将"XYZ"转换为值）
	int	int i = (int)"1.23"	1
	float	float f1 = (float)"1.23" float f2 = (float)"A1"	1.23 错误（无法将"A1"转换为值）
	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（字符串"true"转换为 true） flag2= false（字符串"false"转换为 false） flag3 = true（非空字符串转换为 true） flag4 = false（空字符串转换为 false）

- 数组的转换方法与上表一致。转换将针对数组中的每个元素进行。

```
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}
```

- 若转换时出现以下情况，则将返回错误消息。
  - 无法将布尔值（true/false）或非数值字符串（"XYZ"）等正确地转换为数值。

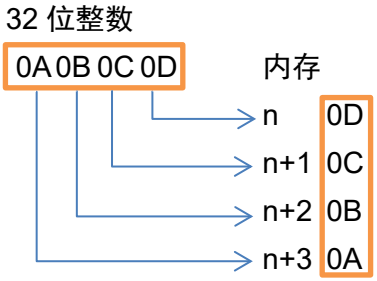
    ```
    int value = (int)true        // 错误
    int value = (int)"XYZ"       // 错误
    ```
  - 试图将 NaN 或 Infinity 等无效浮点数转换为 float 或 double 类型。

    ```
    string   dvalue = "1.79769e+308"
    float f = (float) dvalue     // 错误 1.79769e+308 为有效的 double 类型数字，但无法转换为 float 类型。
    ```

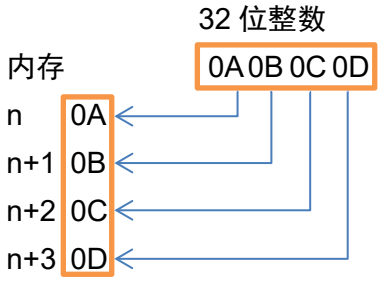
2.6 字节顺序和转换

内存或通信网络中的应用中的数据包含多个字节，由于最小单位是字节，所以要按顺序表达式对多字节进行排序。这种顺序就是字节顺序。

- 小端序： 将多个字节中的低位置于较小的地址，高位置于较大的地址。



- 大端序： 将多个字节中的高位置于较小的地址，低位置于较大的地址。



排序顺序不同可能导致转换结果不同，因此应将这一因素纳入考虑。例如，整数类型为 32 位，亦即占用 4 字节。若 int 值为 300，则其 16 位表达式为 0x0000012C。若内存存储或通信网络使用小端序，则顺序为

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

若采用小端序，则将获取 0x0000012C；若采用大端序，则将获取 0x2C010000。这两个值完全不同，可能导致结果不同。

TMscript 提供了多种机器人编程函数。在这些函数中，int、float 和 double 等数值类型的转换同样遵循一般字节顺序规则，此规则通常适用于文件函数、通信相关函数（如 SerialPort、Socket、Modbus）或值到字节转换相关函数。除非另有规定，函数默认使用小端序。对于 string 类型，将采用兼容 ASCII 字符的 UTF-8 编码格式，无需区分字节顺序即可固定字节顺序。对于 bool 类型，布尔值 true 将被转换为 1、false 将被转换为 0。详情请参见下表。

类型	位/字节	转换方法	示例	结果
byte	8 位/1 字节	小端序	byte b = 100	0x64
int	32 位/4 字节	小端序	int i = 300	0x2C 0x01 0x00 0x00
float	32 位/4 字节	IEEE-754、小端序	float f = 300	0x00 0x00 0x96 0x43
double	64 位/8 字节	IEEE-754、小端序	double d = 300	0x00 0x00 0x00 0x00 0x00 0xC0 0x72 0x40
bool	bool 值	True:1, False:0	bool bf = true	0x01
string	string 值	UTF-8	"TM Robot"	0x54 0x4D 0x20 0x52 0x6F 0x62 0x6F 0x74

值转换方法是通信网络应用的重要基础。应用两侧应采用相同的转换方法，以便正确解析数据内容。例如：

- **socket_send**("ntd_a", 9000)
  根据函数定义，将发送 0x28,0x23,0x00,0x00（int、小端序）至设备 ntd_a。
  若接收设备使用不同的方法解析，则将得到不同的值。例如：

int、大端序	0x28 0x23 0x00 0x00 = 673382400
float、小端序	0x00 0x00 0x23 0x28 = 1.2612E-41
string、UTF-8	0x28 0x23 0x00 0x00 = "(#"
int、小端序	0x00 0x00 0x23 0x28 = 9000

- **socket_send**("ntd_a", (float)9000)
  根据函数定义，将发送 0x00,0xA0,0x0C,0x46（int、小端序）至设备 ntd_a。
  若接收设备使用不同的方法解析，则将得到不同的值。例如：

int、大端序	0x00 0xA0 0x0C 0x46 = 10488902	
int、小端序	0x46 0x0C 0xA0 0x00 = 1175232512	
float、大端序	0x00 0xA0 0x0C 0x46 = 1.4698082E-38	
string、UTF-8	0x00 0xA0 0x0C 0x46 = ""	//遇到 0x00 时字符串结束。
float、小端序	0x46 0x0C 0xA0 0x00 = 9000	

2.7 警告

若出现下列情况，将提示警告消息。

- 未用双引号括起字符串常量。

```
string s = Hello            //警告 Hello
```

- 字符串常量内部存在单引号。

```
string s0 = "World"
string s1 = "Hello ‘s0’"    //警告 ‘s0’
```

- 将浮点数值赋给整型常量，导致部分有效数字丢失，如：

```
int i = 0
float f = 1.234
i = f               //警告 i = 1
```

- 将值赋给有效数字较少的变量，导致部分有效数字丢失，如：

```
byte b = 100
int i = 1000
float f = 1.234
double d = 2.345
b = i               //警告 b = 232  //     byte 只能容纳 0 至 255 的值
f = d               //警告 f = 2.345
```

- 将字符串值赋给数值变量时，将进行字符串至数字的转换。若可执行转换，则将提示警告消息，否则项目会因错误停止，如：

```
int i = "1234"              //警告 i = 1234
int j = "0x89AB"            //警告 j = 35243
int k = "0b1010"            //警告 k = 10
string s1 = 123             //警告 s1 = "123" //数字转字符串
string s2 = "123"
int x = s2                  //警告 x = 123 //字符串转数字
//下列代码可编译，但编译时会出现警告，执行时则会因错误停止。
S2 = "XYZ"
x = s2                      //警告 //因错误停止执行    // s = "XYZ"无法被转换为数字
s2 = ""
x = s2                      //警告 //因错误停止执行    // s = ""无法被转换为数字
```

- 在函数中使用字符串参数作为数值参数时，如：

```
Ctrl(0x0A0B0C0D0E)    //警告 //0x0A0B0C0D0E 不是 int 类型（超出 32 位）
                      //由于存在其他语法 Ctrl(string)，该参数将被应用于 Ctrl(string)
```

虽然带警告消息的项目仍可执行，但强烈建议您更正警告消息中的所有错误，从而消除不可预知的问题，防止项目因错误停止。

- 如何通过修复消除错误消息

1. 使用双引号括起字符串常量

```
string s = "Hello"
```

2. 使用 + 连接字符串常量和字符串变量

```
string s0 = "World"
string s1 = "Hello " + s0
```

3. 进行数值转换时明确类型

```
float   f = 1.234
int i = (int) f             //使用(int)转换类型，处理后 i = 1//浮点数被转换为整数。
```

3. 脚本项目编程

3.1 define

该函数定义了同一项目内的全局变量。项目运行时将优先处理项目定义部分的所有变量。

语法

```
define
```

例如：

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

3.2 main

该函数是项目运行时调用的第一个函数，也是项目的初始函数。

语法

```
main
```

例如：

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

项目开始时运行 main 函数，其结果为：

Hello Techman Robot

Hi TM Robot　　//仪表板仅显示最后显示的内容，因此仪表板上的内容为 Hi TM Robot。

3.3 closestop

无任何错误而进入停止状态的项目将转至该函数。直至该函数结束，项目才会停止。

语法

```
closestop
```

例如：

```
closestop
{
    IO["ControlBox"].DO = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}      //将控制柜 DO 设为 Low。
}
```

项目运行后，按下停止按钮将使项目运行 closestop 函数。

重要提示

该函数无法与运动命令同时执行，最多执行 12 秒，随后强制关闭。

若运行 closestop 函数时出现错误（包括系统中出现的其他错误），将不会运行 errorstop 函数，项目直接结束运行。

例如：

```
main
{
    Exit()      //令项目停止。假设没有出现错误，则将继续运行 closestop 函数。
}
closestop
{
    int zero = 0
    int k = 100 / zero        //closestop 函数中出现被零除错误。项目直接结束运行。
    Display("closestop")   //由于出现错误，本行不会运行。
}
```

注

项目进入停止状态时，页面将移除运行状态，但仍会继续运行函数。直至该函数结束，项目才会停止运行。项目停止运行前，按下执行按钮运行项目将返回错误。

3.4 errorstop

因任何错误进入停止状态的项目将转至该函数。直至该函数结束，项目才会停止。

语法

```
errorstop
```

例如：

```
main
{
    int zero = 0
    int k = 100 / zero          //出现被零除错误。
}
errorstop
{
    IO["ControlBox"].DO = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}      //将控制柜 DO 设为 Low。
}
```

项目运行后，项目将生成被零除错误并转至 errorstop 函数。

重要提示

该函数无法与运动命令同时执行，最多执行 12 秒，随后强制关闭。

若运行 errorstop 函数时出现错误（包括系统中出现的其他错误），将不会运行 errorstop 函数，项目直接结束运行。

例如：

```
main
{
    int k = 100 / (byte)"0.1"   //出现被零除错误。
}
errorstop
{
    int k = 100 / (byte)"0.1"   //errorstop 函数中出现被零除错误。项目直接结束运行。
    Display("errorstop ")       //由于出现错误，本行不会运行。
}
```

注

项目进入停止状态时，页面将移除运行状态，但仍会继续运行函数。直至该函数结束，项目才会停止运行。项目停止运行前，按下执行按钮运行项目将返回错误。

3.5 自定义函数

除内置的 define、main、closetop 和 errorstop 函数外，脚本项目还为用户提供了自定义函数。下方的自定义函数 Show()没有输入参数，也没有返回值（void）。

例如：

```
main
{
    Show()    //调用自定义函数 Show()。
}
void Show()
{
    Display("Hello TM AI Cobot")
}
```

自定义函数支持参数的输入和输出。下例中有两个自定义函数。Add()可将用户输入的两个值相加，并返回相加的结果；Sub()可将用户输入的两个值相减，并返回相减的结果。需要注意，只要返回值不为void，就必须返回类型与定义相同的值。例如Add()采用float类型，则返回的变量也必须为float类型。

例如：

```
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
}
```

重要提示

若函数定义了返回类型，则必须返回相同的类型。若定义为 void，则无需使用 return。

3.6 注释

将代码的一部分用作注释。编译器会忽略被注释的代码。用户可注释单行或多行代码，这些代码不会执行。

- **单行注释**

用户可在代码开头使用//进行注释，完成单行注释，如下所示。编译器将忽略 a = a + 1。

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

- **多行注释**

用户可将/*和*/分别用作注释的开头和末尾进行注释。如下所示，编译器将忽略/*和*/之间的 int a = 0 和 a = a + 1。

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

3.7 变量

用户可通过声明变量进一步实现计算、读写和传递参数等应用。变量分为全局变量和局部变量。

重要提示

变量区分大小写，例如 TMrobot 和 TMROBOT 是两个不同的变量。

- **全局变量**

在定义部分内声明的变量为项目内的全局变量。可在同一项目的函数中读写此类变量。如下所示，若在定义部分内声明了变量 b，主函数运行时会先调用 Test1()，并将该变量设为 20。随后显示变量值时，就会得到最后为 b 设置的值，即 20。

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

重要提示

项目内的全局变量不同于机器人全局变量管理器中的全局变量。项目内的全局变量可跨项目内的函数生效。而机器人全局变量管理器中的全局变量不仅可跨项目内的函数，还可跨不同项目内的函数生效。

重要提示

用户只能在 TMflow 操作界面的全局变量管理器中创建全局变量，不能使用 TMscript 语法创建。

- **局部变量**

在定义部分外（如在 main、closestop、errorstop 和自定义函数中）声明的变量为局部变量，只能在函数内使用。

```
int sum((int s)
{
    int a = 0
}
//定义了两个变量int s和int a，sum()结束时，这两个局部变量将消失。
```

同一函数定义部分内的变量不能重名，也不能与定义部分和机器人全局变量管理器中的全局变量重名。

```
define
{
    int b = 0
    string g_s1 = ""      //假设在全局变量管理器中定义了 g_s1，则此时将发生变量重名。
}
void sum(int s)
{
    int a = 0
    int b = 1             //与定义部分内的 int b 重名。
    float a = 0           //与 int a 重名。
}
```

不同函数部分内的局部变量相互独立。例如，Test1()和 Test2()中都声明了变量 a。由于在不同函数内，这两个变量相互独立，只在各自的函数部分内有效。

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

局部变量的作用域可自上而下覆盖。用户也可在条件语句或循环语句中声明仅在该条件语句或循环语句内存在的变量。变量不存在于作用域外。因此，变量一旦离开作用域，就会释放变量数据。而 SerialPort、Socket 和 Modbus 等类声明的变量此时会关闭设备。

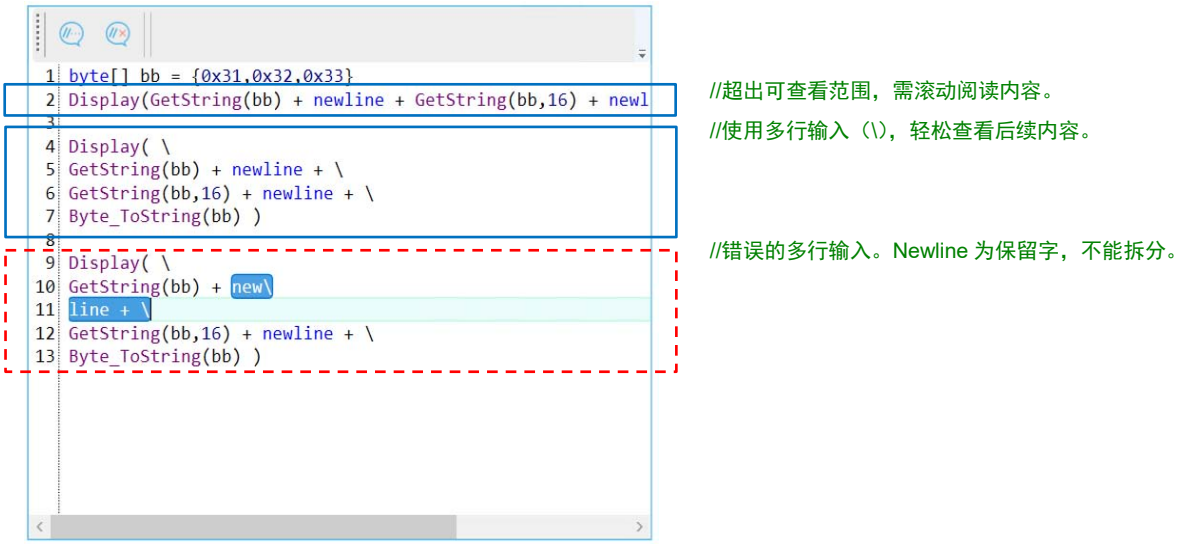
```
void Test1()
{
        int a = 10                              变量 a 的有效作用域
        if (a > 5)
        {                                       变量 b 的有效作用域
            int b = 20
            Display("A Value: " + a)
        }
        Display("B Value: " + b)    //警告。b将被视为字符串而非变量。
}
void Test2()
{
        if (true)
        {                                               变量 bbb 的有效作用域
            Socket bbb = "127.0.0.1",12345
            socket_open("bbb")
            Sleep(5000)
        } //等待 5 秒后，变量 bbb 将因离开 if 条件语句作用域被释放，连接关闭。
        Sleep(10000)
}
```

在流程项目中，若用户希望创建项目内的全局变量，则只能在变量管理器或设备管理器中创建。用户无法在流程项目中使用脚本创建全局变量。此外，由于每个进程节点（包括脚本节点）都是一个独立函数，在节点内声明的变量均为局部变量。退出节点后，节点内的变量将不复存在，变量数据也将被释放。

3.8 多行输入

待输入表达式内容过多不利于维护或调试。用户可在行尾添加\以实现连续内容的多行输入。多行输入内容将被视为同一行内的表达式，直到没有\为止，且最后以换行符结束。

使用多行输入时，仍需保持单词输入正确，不能随意使用\将单词分为连续的多行。



3.9 条件语句

项目运行期间，可能需要考虑针对任务失败、函数和通信错误等各种情况采用不同方法。此时，用户可使用条件语句，根据结果值采用不同路径。目前可以使用 if 和 switch 这两种条件语句。

3.9.1 if

if 语句可对括号内的条件表达式进行判断。若条件满足，则执行语句 1。

```
if (conditional expression)
{
    statement 1          //条件满足。
}
```

反之亦然。若条件满足，则转至语句 1。否则将转至语句 2。

```
if (conditional expression)
{
    statement 1          //条件满足。
}
else
{
    statement 2          //条件不满足。
}
```

用户还可使用 if……else if……else 语句检查更多可能情况，依次判断条件 1、2、3 和最后的其他情况条件。

```
if (条件表达式 1)
{
    statement 1          //条件 1 满足。
}
else if (conditional expression 2)
{
    statement 2          //条件 2 满足。
}
else if (conditional expression 3)
{
    statement 3          //条件 3 满足。
}
else
{
    statement 4          //条件 1、2、3 均不满足。
}
```

举例如下。若 Score 为 100，将显示“Full Score”。若 Score 在 60 至 99 之间，将显示“Excellent”，其他情况下将显示“Failed”。用户可使用 if 语句编写条件。

```
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
```

注：
if (Score==100)中的 == 表示比较，int Score = 65 中的 = 表示赋值。

条件表达式的评估标准如下。对评估而言布尔值具有诸多优点，推荐采用。

- 布尔值　　true 或 True 表示条件满足，false 或 False 表示条件不满足。
- 数值　　非零表示条件满足，零表示条件不满足。
- 字符串值　true 或 True 表示条件满足，其他字符串值表示条件不满足。

3.9.2 switch

在条件判断方面，switch 语句与 if 语句类似。对于类似的条件，switch 语句采用另一种编写方式。

```
switch (变量或表达式)
{
    case condition 1 satisfies
        statement 1          //条件 1 满足。
        break
    case condition 2 satisfies
        statement 2          //条件 2 满足。
        break
    default
        statement 3          //条件 1、2 均不满足。
        break
}
```

如下例所示，其可根据变量的内容值匹配多种结果。

```
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
```

此外，switch 语句也支持表达式。在下例中，若 Score 为 100，将显示“Full Score”。若 Score 在 60 至 99 之间，将显示“Excellent”，其他情况下将显示“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
```

其与 if 语句的区别在于

- switch 语句只获取一次值并多次比较结果值，而 if 语句每次比较都会获取一次。因此，switch 对 DI 状态的判断更加准确。
- 基于以上原因，switch 语句不支持不同性质的条件，而 if 语句支持。例如：

```
void Test1()
{
	int Payload = 4
	int Length = 130
	if (Payload > 4)
	{
		Display("Green", "Yellow", "Payload", "")
	}
	else if (Length > 70)
	{
		Display("Green", "Yellow", "Length", "")
	}
	//无法使用switch语句编写此处的if……else if语句。
}
```

3.10 循环语句

项目运行期间，可能需要反复计算某些值或反复检查条件是否满足。在这些情况下，需要使用循环语句反复处理语句中的代码，直至条件满足。目前可以使用 for、while 和 do-while 这三种循环语句。

3.10.1 for

for 循环语法包括四个部分：初始化部分、循环条件部分、语句和迭代运算部分。

```
for (初始化部分;循环条件部分;迭代运算部分)
{
    statement
}
```

执行顺序如下所示。

1. 初始化部分：转至 for 语法时，初始化部分会执行一次。通常用于声明变量。（变量为 for 语法内的局部变量。）
2. 循环条件部分：根据条件决定是否继续执行 for 循环。条件满足（true）或不存在时，将继续执行 for 循环，直至条件不满足（false）时退出 for 循环。
3. 执行语句：待执行的语句。
4. 迭代运算部分：执行语句后，迭代运算部分执行一次，然后返回循环条件部分判断条件。

下例为 for 循环的基本应用。本例将对从 0 到输入的 K 值的所有整数求和。

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

用户可以并用多个 for 循环。下例通过 for 循环显示了乘法表。

```
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)
}
```

用户可在 for 循环中按需选用四个部分。

```
void Test1()
{
    int i = 0
    for (i = 3; i < 4; )
        i++
    // i = 3
    // i < 4      // i++        // i = 4
    // i < 4      //false，退出 for
    for ( ; i < 5; i++)
    {
    }
    //由于未复位 i 值，将继续使用 i =4。
    // i < 5      //            // i++, i = 5
    // i < 5      //false，退出 for

    for ( ; ; )                //循环条件部分不存在，因此将继续执行。
    {
        // …
    }
}
```

3.10.2 while

while 循环仅含一个布尔值条件表达式。条件满足（true）则执行语句，直至条件不满足（false）时退出 while 循环。因此，while 循环的执行次数可为零或更多，首次执行时就有可能因条件不满足而退出。

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

下例对等差数列 0、1、2、3、…、N 求和，直至总和为 500500，并获取 N 值。

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

3.10.3 do while

do while 循环的语法与 while 循环的类似。while 循环先检查条件是否满足，再执行代码块中的语句；而 do while 循环先执行命令，再检查条件是否满足。因此，do while 循环至少会执行语句一次。

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

如下例所示，若使用 do while 循环，将显示 Hello TM Robot。

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

与之相反，若使用 while 循环，则不会显示任何结果。

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

3.11 分支语句

3.11.1 break

适用于在不满足循环条件时退出 for、while 或 do while 的最后一个循环语句并退出循环。如下例所示，一旦 i 大于 10，就会退出并结束 for 循环。

```
int sum(int k)
{
    int result = 0
    for (int i = 0; i < k; i++)
    {
        if (i > 10)
        {
            break                    直接退出最近的 for i 循环。
        }
        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                直接退出最近的 for j 循环。
        }
        result += newline
    }
    Display("Green", "Yellow", "multiplication table", result)
}
```

3.11.2 continue

适用于 for、while 和 do while 循环语句，但与 break 语句不同的是，continue 会结束最近的当前循环并开始下一次循环，而非退出循环。如下例所示，只要遇到偶数，当前循环就会直接结束，开始下一次循环。因此只会记录奇数。

```
void Test1()
{
    string result = ""
    for (int i = 0; i < 10; i++)        结束当前循环，返回至下一次循环。
    {
        if (i % 2 == 0)
        {
            continue
        }
        result += i + ", "
    }
    Display("Odd value: " + result)
```

```
}
//奇数值： 1, 3, 5, 7, 9
```

3.11.3 return

return 语句会使函数内的语句结束执行，并将值返回函数调用者。例如：

```
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    //一旦 i == 3，就会直接返回 result，不会继续执行
                                     语句。
        result += i
    }
    return result
}
```

3.12 线程

在项目编程过程中，当涉及异步并行操作时，用户需要使用线程。用户可新建线程用于操作，以使项目符合多任务处理概念。但是，用户也应注意执行时线程间不存在优先级，当存在多个线程时，每个线程都将独立运行。

项目运行期间首先调用的 main 函数也是作为线程运作的。该线程与其他线程的区别在于，该线程调用的 main 函数和在 main 函数内调用的其他自定义函数拥有调用机器人运动函数的权限。其他新建线程则没有调用机器人运动函数的权限。这是为了保障机器人运动进程的一致性，由于每个线程独立处理，若允许其他线程使用机器人运动函数，则很容易导致机器人运动进程混乱或中断。

不同于其他线程，主线程拥有调用机器人运动函数的权限。其他新建线程则无法调用机器人运动函数。这是为了保障机器人运动进程的一致性。由于每个线程独立运行，若其他线程也可调用机器人运动函数，则很容易导致机器人运动中断或混乱。

除机器人运动函数调用权限外，用户还可设置线程是否连续执行，亦即项目暂停时线程是否会受到影响。若将线程设为连续执行，则即使项目暂停该线程也会继续运行。此设置适用于通信应用。

若使用的线程在循环（使用循环语句）内运作，则有必要额外添加睡眠函数以释放线程占用的 CPU 利用率。围绕线程工作时，建议额外添加睡眠函数，由于 CPU 利用率过高可能导致执行效率低下，因此必须注意。

3.12.1 ThreadRun()

新建线程并使用新线程执行语句。前一个线程将继续执行。

语法 1

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

参数

?	语句或自定义函数	
bool	线程是否继续执行而不暂停	
	true	继续执行而不暂停
	false	不继续执行而暂停（默认值）

返回值

int	返回新建线程的ID。	
	> 0	创建成功
	<= 0	创建失败

*新建线程的 ID 由系统决定，非固定值。执行中的线程编号不会重复，但线程停止后编号可能重复出现。

语法 2

```
int ThreadRun(
    ?
)
```

注

与语法 1 相同。线程是否继续执行参数被默认设为 false。

3.12.2 ThreadID()

获取当前线程的 ID。

语法 1

```
int ThreadID(
)
```

参数

void	无参数

返回值

int	返回线程的ID。

注

```
main
{
    int tid = ThreadRun(ThreadTest1(), false)
    Sleep(1000)
    ThreadRun(ThreadTest2(tid), true)
}
void ThreadTest1()
{
    Display("Hello ThreadTest1() " + ThreadID())
}
void ThreadTest2(int k)
{
    Display("Hello ThreadTest2() " + ThreadID() + " " + k)
}
// Hello ThreadTest1() 18
// Hello ThreadTest2() 61 18
```

新建一个线程并继续执行 ThreadTest1()。
之前的线程继续执行。
新建一个线程并继续执行 ThreadTest2()。

下面介绍新建线程以运行语句的执行顺序。

```
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
```

- 执行顺序为

1. 在主线程中获取线程 ID 并将该值赋给局部变量 tid。
2. 新建线程并执行 ThreadTest3(tid, ThreadID())语句。
3. 由于 tid 是局部变量，可以导入 tid 值。
4. 由于 ThreadID()是函数，而调用函数时会使用新线程，获得的线程 ID 将不同于 tid。
5. 然后调用 ThreadTest3()函数，将其和获得的线程 ID 分别导入函数中的 k1 和 k2
6. 调用 ThreadTest3()函数中的 ThreadID()函数获取线程 ID。由于与 ThreadTest3()函数处于相同线程内，k2 将拥有相同的 ID，而 k1 将拥有不同的 ID。

3.12.3 ThreadState()

获取特定线程 ID 的状态。

语法 1

```
int ThreadState(
    int
)
```

参数

int	线程ID	

返回值

int	返回专用线程的状态	
	-1	该线程不存在。
	0	该线程正在执行。
	1	该线程正在请求停止。
	2	该线程已停止。

语法 2

```
int ThreadState(
)
```

注

与语法 1 相同。线程 ID 被默认设为函数调用的当前线程 ID。

3.12.4 ThreadExit()

请求特定线程 ID 停止执行。

语法 1

```
int ThreadExit(
    int
)
```

参数

int	线程ID	

返回值

int	返回请求特定线程停止的结果	
	-1	该线程不存在。
	0	请求的线程已停止执行。

语法 2

```
int ThreadExit(
)
```

注

与语法 1 相同。线程 ID 被默认设为函数调用的当前线程 ID。

```
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)                //使用循环语句等待线程t4停止。
    //由于ThreadTest4中不存在循环，线程将在执行结束后停止。
    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)
    //使用WaitFor等待线程t5停止。
    title = "t5 " + st + " " + flag  // t5 0 false
    //由于ThreadTest5中存在循环，线程不会停止。
    //因此，WaitFor将于1000 ms超时后退出。所以flag = false。
    Sleep(1000)
    if (flag == false)
    {
        ThreadExit(t5)               //请求线程t5退出。
    }
    flag = WaitFor((st = ThreadState(t5)) == 2, 1000)
    //使用WaitFor等待线程t5停止。
    //由于系统请求线程t5停止，WaitFor条件保持不变。
    Display("t5 " + st + " " + flag, "Hello main() " + ThreadID()) // t5 2 true
}
void ThreadTest5()
{
    while (true)                     //循环语句
    {
        Display(title , "Hello ThreadTest5() " + ThreadID())
        Sleep(100)
        //使用循环语句时，建议额外使用睡眠函数释放线程占用的 CPU 利用率。
    }
}
```

4. 通用函数

4.1 Byte_ToInt16()

将特定字节数组的前 2 字节转换为整数，并以 int 类型返回。

语法 1

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

参数

byte[]	字节数组
int	字节数组遵循小端序还是大端序
	0 小端序（默认值）
	1 大端序
int	转换为有符号int16（有符号数字）还是无符号int16（无符号数字）
	0 有符号int16（默认值）
	1 无符号int16

返回值

int	由从索引[0]开始的2字节转换的有符号或无符号int16。 由于只需要2字节，对应的字节数组索引将是[0][1]。若数据不够长，转换前会将其填充至2字节。

注

```
byte[] bb1 = {0x90, 0x01, 0x05}
byte[] bb2 = {0x01} //由于 bb2[]不足 2 字节， 转换前将被填充至 2 字节。

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
```

语法 2

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

注

与语法 1 类似，返回值为有符号 int16

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

语法 3

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

注

与语法 1 类似，输入为小端序，返回值为有符号 int16

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

4.2 Byte_ToInt32()

将字节数组的前 4 字节转换为整数，并以 int 类型返回。

语法 1

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

参数

byte[]	输入的字节数组
int	输入的字节数组遵循小端序还是大端序
	0 小端序（默认值）
	1 大端序

返回值

int	由从索引[0]开始的4字节转换的无符号int32。 由于只需要4字节，对应的字节数组索引将是[0][1][2][3]。若数据不够长，转换前会将其填充至4字节。

注

```
byte[] bb1 = {0x01, 0x02, 0x03, 0x4F, 1}
byte[] bb2 = {0x01, 0x02, 0x03}
                                    //由于 bb2[]不足 4 字节，转换前将被填充至 4 字节。
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
```

语法 2

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

注

与语法 1 类似，输入为小端序

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

4.3 Byte_ToFloat()

将字节数组的前 4 字节转换为浮点数，并以浮点类型返回。

语法 1

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

参数

byte[]	输入的字节数组
int	输入的字节数组遵循小端序还是大端序
	0 小端序（默认值）
	1 大端序

返回值

float	由从索引[0]开始的4字节转换的浮点数。 由于只需要4字节，对应的字节数组索引将是[0][1][2][3]。若数据不够长，转换前会将其填充至4字节。

注

```
byte[] bb1 = {0x01, 0x02, 0x03, 0x4F, 1}
byte[] bb2 = {0x01, 0x02, 0x03}     //由于 bb2[]不足 4 字节，转换前将被填充至 4 字节。
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
```

语法 2

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

注

与语法 1 类似，输入为小端序

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

4.4 Byte_ToDouble()

将字节数组的前 8 字节转换为浮点数，并以 double 类型返回。

语法 1

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

参数

byte[]	输入的字节数组
int	输入的字节数组遵循小端序还是大端序
	0　小端序（默认值）
	1　大端序

返回值

double	由从索引[0]开始的8字节转换的浮点数。 由于只需要8字节，对应的字节数组索引将是[0][1][2][3][4][5][6][7]。若数据不够长，转换前会将其填充至8字节。

注

```
byte[] bb1 = {0x01, 0x02, 0x03, 0x4F, 1}      //由于 bb1[]不足 8 字节，转换前将被填充至 8 字节。
byte[] bb2 = {0x01, 0x02, 0x03}     //由于 bb1[]不足 8 字节，转换前将被填充至 8 字节。
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
```

语法 2

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

注

与语法 1 类似，输入为小端序

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

4.5 Byte_ToInt16Array()

以 2 字节为单位将字节数组转换为整数，并以 int[]类型返回。

语法 1

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

参数

`byte[]` 输入的字节数组

`int` 输入的字节数组遵循小端序还是大端序

- `0` 小端序（默认值）
- `1` 大端序

`int` 转换为有符号int16（有符号数字）还是无符号int16（无符号数字）

- `0` 有符号int16（默认值）
- `1` 无符号int16

返回值

`int[]` 由字节数组从索引[0]开始以2字节为单位转换成的整型数组

注

```
byte[] bb1 = {0x90, 0x01, 0x02, 0x03, 0x04}       //剩余部分不足 2 字节，转换前将被填充至 2 字节。
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}
```

语法 2

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

注

与语法 1 类似，返回值为有符号 int16

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

语法 3

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

注

与语法 1 类似，输入为小端序，返回值为有符号 int16

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

4.6 Byte_ToInt32Array()

以 4 字节为单位将字节数组转换为整数，并以 int[]类型返回

语法 1

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

参数

byte[]	输入的字节数组
int	输入的字节数组遵循小端序还是大端序
0	小端序（默认值）
1	大端序

返回值

int[]	由字节数组从索引[0]开始以4字节为单位转换成的整型数组

注

```
byte[] bb1 = {0x01, 0x02, 0x03, 0x04, 0x05}        //剩余部分不足 4 字节，转换前将被填充至 4 字节。
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}
```

语法 2

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

注

与语法 1 类似，输入为小端序。

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

4.7 Byte_ToFloatArray()

以 4 字节为单位将字节数组转换为浮点数，并以 float[]类型返回。

语法 1

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

参数

byte[]	输入的字节数组
int	输入的字节数组遵循小端序还是大端序
0	小端序（默认值）
1	大端序

返回值

float[]	由字节数组从索引[0]开始以4字节为单位转换成的浮点型数组

注

```
byte[] bb1 = {0x01, 0x02, 0x03, 0x04, 0x05}
                //剩余部分不足 4 字节，转换前将被填充至 4 字节。
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}
```

语法 2

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

注

与语法 1 类似，输入为小端序

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

4.8 Byte_ToDoubleArray()

以 8 字节为单位将字节数组转换为双精度浮点数，并以 double[]类型返回。

语法 1

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

参数

byte[]	输入的字节数组
int	输入的字节数组遵循小端序还是大端序
0	小端序（默认值）
1	大端序

返回值

double[]	由字节数组从索引[0]开始以8字节为单位转换成的浮点型数组

注

```
byte[] bb1 = {0x01, 0x02, 0x03, 0x04, 0x05}        //剩余部分不足 8 字节，转换前将被填充至 8 字节。
byte[] bb2 = {1, 2, 3, 4}          //剩余部分不足 8 字节，转换前将被填充至 8 字节。

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}
```

语法 2

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

注

与语法 1 类似，输入为小端序

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

4.9 Byte_ToString()

将字节数组转换为字符串

语法 1

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

参数

`byte[]` 输入的字节数组

`int` 应用于输入的字节数组的字符编码规则

- `0` UTF8（默认值）（止于0x00）
- `1` 十六进制编码的二进制
- `2` ASCII（止于0x00）

返回值

`string` 由字节数组转换的字符串。转换从索引[0]开始。

注

```
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 代表控制字符 SOH，而非字符串值。

语法 2

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

注

与语法 1 类似，应用 UTF8 字符编码规则

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

4.10 Byte_Concat()

连接两字节数组，或连接一个数组与一字节值。

语法 1

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

参数

`byte[]` 输入的字节数组

`byte` 将连接至字节数组后的字节值

返回值

`byte[]` 由输入的字节数组和字节值组成的字节数组

注

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

value = Byte_Concat(bb1, 12) // value = {0x31, 0x32, 0x33, 0x00, 0x4F, 0x01, 0x0C}
```

语法 2

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

参数

`byte[]` 输入的字节数组1

`byte[]` 输入的字节数组2，将连接至数组1末尾后

返回值

`byte[]` 连接输入的数组而组成的字节数组。

注

```
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}
```

语法 3

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

参数

`byte[]` 输入的字节数组1

`byte[]` 输入的字节数组2，将连接至数组1末尾后

`int` 将连接的数组2中的元素数量

0..数组2的长度	有效数量
<0	无效。将改为应用数组2的长度。
>数组2的长度	无效。将改为应用数组2的长度。

返回值

byte[] 连接输入的数组而组成的字节数组。

注

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} //仅连接数组2中的2个元素
value = Byte_Concat(bb1,   bb2, -1)       // value = {0x31, 0x32, 0x33, 0x00, 0x4F, 0x01, 0x01, 0x02, 0x03} //-1为无效值
value = Byte_Concat(bb1, bb2, 10)         // value = {0x31, 0x32, 0x33, 0x00, 0x4F, 0x01, 0x01, 0x02, 0x03} //10已超出数组大小

                                          //可使用 Length()获取数组大小
value = Byte_Concat(bb1, bb2, Length(bb2))
                                          // value = {0x31, 0x32, 0x33, 0x00, 0x4F, 0x01, 0x01, 0x02, 0x03}
```

语法 4

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

参数

byte[]	输入的字节数组1	
int	数组1的起始索引	
	0..（数组1的长度）-1	有效
	<0	起始索引将为0
	>=（数组1的长度）	起始索引将为数组2的长度（对于超出数组2的长度的索引，将取得空值）
int	将连接的数组1中的元素数量	
	0.. （数组1的长度）	有效
	<0	无效，将改为应用数组1的长度
	>（数组1的长度）	无效，将改为应用数组1的长度
	若起始索引和特定元素数量之和超出数组1的长度，多余的索引将被弃用。	
byte[]	输入的字节数组2，将连接至数组1末尾后	
int	数组2的起始索引	
	0..（数组2的长度）-1	有效
	<0	起始索引将为0
	>=（数组2的长度）	起始索引将为数组2的长度（对于超出数组2的长度的索引，将取得空值）
int	将连接的数组2中的元素数量	
	0..（数组2的长度）	有效
	<0	无效。将改为应用数组2的长度。
	>（数组2的长度）	无效。将改为应用数组2的长度。
	若起始索引和特定元素数量之和超出数组2的长度，多余的索引将被弃用。	

返回值

`byte[]` 连接输入的数组而组成的字节数组。

注

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}

语法 5

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

参数 *（参数数量不固定）*

`byte` 输入的字节值
`byte[]` 输入的字节数组
依次连接各参数的内容。不是字节或字节数组的参数将被忽略，并继续连接下一个参数。

返回值

`byte[]` 以字节为单位连接参数并返回一个新的字节数组。

注

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}
//参数"AB"为字符串型。忽略。
value = **Byte_Concat**(bb1, "AB", bb2, 1000) // value = {0x31,0x32,0x00,0x4F,0x01,0x01,0x02,0x03}
//参数"AB"为字符串型。忽略。
//参数1000为整型。忽略。

4.11 String_ToInteger()

将字符串转换为整数（int 类型）

语法 1

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

参数

string	输入的字符串。
int	输入的字符串的表示法为十进制、十六进制还是二进制
10	十进制或自动检测格式（默认值）
16	十六进制
2	二进制
字符串的表示法	
123	十进制
0x7F	十六进制
0b101	二进制

返回值

int	由输入的字符串转换成的整数值。若表示法无效，将返回0。

注

```
value = String_ToInteger("1234", 10)        // value = 1234
value = String_ToInteger("1234", 16)        // value = 4660
value = String_ToInteger("1234", 2)         // value = 0          //无效的二进制格式
value = String_ToInteger("1100", 2)         // value = 12
value = String_ToInteger("0x1234", 10)      // value = 4660       //自动检测为十六进制格式
value = String_ToInteger("0x1234", 16)      // value = 4660
value = String_ToInteger("0x1234", 2)       // value = 0          //无效的二进制格式
value = String_ToInteger("0b1100", 10)      // value = 12         //自动检测为二进制格式
value = String_ToInteger("0b1100", 16)      // value = 725248     //有效的十六进制数
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          //无效的十六进制格式
value = String_ToInteger("-0b1100", 2)      // value = 0          //无效的二进制格式
```

语法 2

```
int String_ToInteger(
    string
)
```

注

与语法 1 类似，采用十进制或自动检测格式

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

语法 3

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

参数

string[]	输入的字符串数组
int	输入的字符串数组中的元素的表示法为十进制、十六进制还是二进制
	10　十进制或自动检测格式（默认值）
	16　十六进制
	2　二进制
	字符串的表示法
	123　十进制
	0x7F　十六进制
	0b101　二进制
	*一个数组中的所有元素的表示法必须相同

返回值

int[]	由输入的字符串数组转换成的整型数组。若表示法无效，将返回0。

注

```
ss = {"12", "ab", "cc", "dd", "10"}
value = String_ToInteger(ss)        // value = {12, 0, 0, 0, 10} //"ab"、"cc"、"dd"为无效的十进制数
value = String_ToInteger(ss, 2)     // value = {0, 0, 0, 0, 2}  //"12"、"ab"、"cc"、"dd"为无效的二进制数
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"为无效的十进制数
```

4.12 String_ToFloat()

将字符串转换为浮点数（浮点型）

语法 1

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

参数

string	输入的字符串
int	输入的字符串的表示法为十进制、十六进制还是二进制格式
10	十进制或自动检测格式（默认值）
16	十六进制
2	二进制
字符串的表示法	
123	十进制
0x7F	十六进制
0b101	二进制

返回值

float　由输入的字符串转换成的浮点数。若表示法无效，将返回 0。

注

```
value = String_ToFloat("12.34", 10)          // value = 12.34
value = String_ToFloat("12.34", 16)          // value = 0          //无效的十六进制格式
value = String_ToFloat("12.34", 2)           // value = 0          //无效的二进制格式
value = String_ToFloat("11.00", 2)           // value = 0          //无效的二进制格式
value = String_ToFloat("0x1234", 10)         // value = 6.53E-42   //自动检测为十六进制格式
value = String_ToFloat("0x1234", 16)         // value = 6.53E-42
value = String_ToFloat("0x1234", 2)          // value = 0          //无效的二进制格式
value = String_ToFloat("0b1100", 10)         // value = 1.7E-44    //自动检测为二进制格式
value = String_ToFloat("0b1100", 16)         // value = 1.016289E-39 //有效的十六进制格式
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          //无效的十六进制格式
value = String_ToFloat("-0b1100", 2)         // value = 0          //无效格式
```

语法 2

```
float String_ToFloat(
    string
)
```

注

与语法 1 类似，采用十进制或自动检测格式

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

语法 3

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

参数

`string[]`	输入的字符串数组
`int`	输入的字符串数组中的元素的表示法为十进制、十六进制还是二进制
	`10` 十进制或自动检测格式（默认值）
	`16` 十六进制
	`2` 二进制
	字符串的表示法
	123 十进制
	0x7F 十六进制
	0b101 二进制
	*一个数组中的所有元素的表示法必须相同

返回值

`float[]`	由输入的字符串数组转换成的浮点型数组。若表示法无效，将返回 0。

注

```
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()

将字符串转换为浮点数（double 类型）

语法 1

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

参数

`string` 输入的字符串

`int` 输入的字符串的表示法为十进制、十六进制还是二进制格式

- `10` 十进制或自动检测格式（默认值）
- `16` 十六进制
- `2` 二进制

字符串的表示法

- 123 十进制
- 0x7F 十六进制
- 0b101 二进制

返回值

`double` 由输入的字符串转换成的浮点数。若表示法无效，将返回 0。

注

```
value = String_ToDouble("12.34", 10)      // value = 12.34
value = String_ToDouble("12.34", 16)      // value = 0          //无效的十六进制格式
value = String_ToDouble("12.34", 2)       // value = 0          //无效的二进制格式
value = String_ToDouble("11.00", 2)       // value = 0          //无效的二进制格式
value = String_ToDouble("0x1234", 10)     // value = 2.3023E-320  //自动检测为十六进制格式
value = String_ToDouble("0x1234", 16)     // value = 2.3023E-320
value = String_ToDouble("0x1234", 2)      // value = 0          //无效的二进制格式
value = String_ToDouble("0b1100", 10)     // value = 6E-323     //自动检测为二进制格式
value = String_ToDouble("0b1100", 16)     // value = 3.5832E-318  //有效的十六进制格式
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          //无效的十六进制格式
value = String_ToDouble("-0b1100", 2)     // value = 0          //无效的二进制格式
```

语法 2

```
double String_ToDouble(
    string
)
```

注

与语法 1 类似，采用十进制或自动检测格式

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

语法 3

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

参数

string[]	输入的字符串数组
int	输入的字符串数组中的元素的表示法为十进制、十六进制还是二进制
	10　十进制或自动检测格式（默认值）
	16　十六进制
	2　二进制
	字符串的表示法
	123　十进制
	0x7F　十六进制
	0b101　二进制
	*一个数组中的所有元素的表示法必须相同

返回值

double[]	由输入的字符串数组转换成的浮点型数组。若表示法无效，将返回 0。

注

```
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()

将字符串转换为字节数组

语法 1

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

参数

`string` 输入的字符串

`int` 应用于输入的字符串的字符编码规则

- `0` UTF8（默认值）
- `1` 十六进制编码的二进制 //止于无效的十六进制值
- `2` ASCII

返回值

`byte[]` 由输入的字符串转换成的字节数组

注

```
value = String_ToByte("12345", 0)          // value = {0x31, 0x32, 0x33, 0x34, 0x35}
value = String_ToByte("12345", 1)          // value = {0x12, 0x34, 0x50}     //以0填充不足部分
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}       //由于x为无效的十六进制值，只转换了0
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为无效的十六进制值
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为无效的十六进制值
```

语法 2

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

注

与语法 1 类似，采用 UTF8 格式

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

4.15 String_IndexOf()

从左向右搜索特定字符串首次出现的地址。

语法 1

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

参数

string	输入的字符串
string	要搜索的特定字符串
int	初始地址，将从此处开始搜索

返回值

int	0..（字符串的长度）-1	若找到特定字符串，将返回索引编号
	-1	未找到
	0	特定字符串为""或空字符串

语法 2

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

注

与语法 1 相同。参数 int 的初始地址默认为 0，即从最左端开始搜索。

```
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   //从索引编号1开始搜索
index = String_IndexOf("012314", "1", 2)    // 4   //从索引编号2开始搜索
index = String_IndexOf("012314", "1", 10)   // -1
```

语法 3

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

参数

string	输入的字符串数组
string	要搜索的特定字符串
int	数组初始索引，将从此处开始搜索

返回值

int	0..数组大小 -1	若找到字符串，将返回字符串数组的相应索引。
	-1	未找到

*将从初始索引开始从左向右搜索数组索引。

*若数组元素可用，将使用String_IndexOf(string, string)搜索。将返回数组元素的索引编号，但不返回字符串的索引编号。

语法 4

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

注

与语法 3 相同。参数 int 的初始索引默认为 0，即从最前端的数组元素开始搜索。

```
string[] ss = {"012314", "ABCDEF", "123TM"}
int index = String_IndexOf(ss, "1")      // 0
index = String_IndexOf(ss, "")           // 0   //由于使用String_IndexOf搜索字符串，可以搜索""。
index = String_IndexOf(ss, "d")          // -1
index = String_IndexOf(ss, "1", 1)       // 2
index = String_IndexOf(ss, "1", 10)      // -1
```

4.16 String_LastIndexOf()

从右向左搜索特定字符串最后一次出现的地址。

语法 1

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

参数

string	输入的字符串
string	要搜索的特定字符串
int	初始地址，将从此处开始搜索

返回值

int	0..（字符串的长度）-1	若找到特定字符串，将返回索引编号
	-1	未找到
	（字符串的长度）	特定字符串为""或空字符串

语法 2

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

注

与语法 1 相同。参数 int 的初始地址默认为 0，即从最右端开始搜索。

```
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   //从索引编号1开始搜索
index = String_LastIndexOf("012314", "1", 2)    // 1   //从索引编号2开始搜索
index = String_LastIndexOf("012314", "1", 10)   // 4
index = String_LastIndexOf("012314", "4", 10)   // 5
```

语法 3

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

参数

string	输入的字符串数组
string	要搜索的特定字符串
int	数组初始索引，将从此处开始搜索

返回值

int	0..数组大小 -1	若找到字符串，将返回字符串数组的相应索引。
	-1	未找到

*将从初始索引开始从右向左搜索数组索引。
*若数组元素可用，将使用String_IndexOf(string, string)搜索。将返回数组元素的索引编号，但不返回字符串的索引编号。

语法 4

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

注

与语法 3 相同。参数 int 的初始索引默认为 0，即从最后端的数组元素开始搜索整个数组。

```
string[] ss = {"012314", "ABCDEF", "123TM"}
int index = String_LastIndexOf(ss, "1")      // 2
index = String_LastIndexOf(ss, "")           // 2   //由于使用String_IndexOf搜索字符串，可以搜索""。
index = String_LastIndexOf(ss, "d")          // -1
index = String_LastIndexOf(ss, "1", 1)       // 0
index = String_LastIndexOf(ss, "1", 10)      // 2
```

4.17 String_DiffIndexOf()

从起始地址开始比较，查找两个字符串首次出现差异的地址。

语法 1

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

参数

string	输入的字符串 1
string	输入的字符串2
int	初始地址，将从此处开始比较

返回值

int	0..（字符串的长度）-1	若发现差异，将返回差异的索引编号。
	-1	未发现差异。即两个字符串一致。
	-2	初始地址同时超出两个字符串的长度。

语法 2

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

注

与语法 1 相同。参数 int 的初始地址默认为 0，即从最左端开始搜索。

```
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   //从索引5开始比较f和F。
index = String_DiffIndexOf(s1, s3, 7)               // 7   //从索引7开始比较'\0'和2。
index = String_DiffIndexOf(s1, "abcdef", 7)         // -1  //两个字符串一致。
index = String_DiffIndexOf(s1, "ABCDEF", 7)         // -2  //从索引7开始比较。同时超出两个字符串的长度。
```

4.18 String_Substring()

从输入的字符串中获取子字符串

语法 1

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

参数

`string` 输入的字符串
`int` 子字符串的起始字符位置（0~（输入的字符串的长度）-1）
`int` 子字符串的长度

返回值

`string` 子字符串
若起始字符位置<0，将返回空字符串
若起始字符位置>=输入的字符串的长度，将返回空字符串
若子字符串的长度<0，子字符串将以输入的字符串的最后一个字符结束
若起始字符位置和子字符串的长度之和超出输入的字符串的长度，子字符串将以输入的字符串的最后一个字符结束

注

```
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"
```

语法 2

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

注

与语法 1 类似，子字符串将以输入的字符串的最后一个字符结束
String_Substring(str, 2) => **String_Substring**(str, 2, 最大长度)

语法 3

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

参数

`string` 输入的字符串
`string` 要搜索的目标字符串，若能找到，子字符串将始于目标字符串位置
`int` 子字符串的长度

返回值

`string` 子字符串

若目标字符串为空字符串，子字符串将始于索引0
若未找到目标字符串，将返回空字符串
若子字符串的长度<0，子字符串将以输入的字符串的最后一个字符结束
若起始字符位置和子字符串的长度之和超出输入的字符串的长度，子字符串将以输入的字符串的最后一个字符结束

注

该语法与**String_Substring**(str, **String_IndexOf**(str1), int)相同

```
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"
```

语法 4

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

注

与语法 3 类似，子字符串将以输入的字符串的最后一个字符结束

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

语法 5

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

参数

string	输入的字符串
string	前缀。子字符串的开头元素
string	后缀。子字符串的结尾元素
int	出现次数

返回值

string 子字符串

若前缀和后缀均为空字符串，将返回输入的字符串

若出现次数<=0，将返回空字符串

注

```
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" //获取匹配的前缀之后的内容
value = String_Substring("0x12345122", "1", "", 2)          // value = "122"  //获取匹配的前缀之后的内容。匹配次数从前向后计算。
```

```
value = String_Substring("0x12345122", "1", "", 4)      // value = ""
value = String_Substring("0x12345433", "", "4", 1)      // value = "0x123454" //获取匹配的后缀之后的内容。

value = String_Substring("0x12345433", "", "4", 2)      // value = "0x1234"    //获取匹配的后缀之后的内容。匹配次数从后向前计算。
```

语法 6

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

注

与语法 5 类似，返回首次出现的子字符串

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

4.19 String_Split()

使用特定分隔符分割字符串。

语法 1

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

参数

string	输入的字符串
string	分隔符（字符串）
int	格式
0	分割，然后保留空字符串
1	分割，然后剔除空字符串
2	分割时跳过双引号内的元素，然后保留空字符串
3	分割时跳过双引号内的元素，然后剔除空字符串

返回值

string[]	分割后的子字符串 若输入的字符串为空字符串，则返回的子字符串数组中的[0] = 空字符串，然后将按分隔符处理空字符串。 若分隔符为空字符串，则返回的子字符串数组中的[0] = 输入的字符串，然后将按分隔符处理空字符串。

注

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

语法 2

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

注

与语法 1 类似，分割，然后保留空字符串

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

4.20 String_Replace()

将输入的字符串中出现的所有特定字符串替换为另一个特定字符串，然后作为新字符串返回

语法 1

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

参数

`string` 输入的字符串
`string` 旧值，待替换字符串
`string` 新值，用于替换出现的所有旧值的字符串

返回值

`string` 以新值替换输入值中的旧值后得到的字符串。若旧值为空字符串，将返回输入的字符串

注

```
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()

从输入的字符串中删除开头和结尾出现的所有特定字符或空白字符，然后作为新字符串返回

语法 1

```
string String_Trim(
    string
)
```

参数

`string` 输入的字符串

返回值

`string` 删除开头和结尾出现的所有空白字符后得到的字符串

注

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

空白字符

\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

语法 2

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

参数

`string` 输入的字符串
`string` 要删除的开头出现的特定字符

返回值

`string` 删除开头出现的所有特定字符后得到的字符串

语法 3

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

参数

`string` 输入的字符串
`string` 要删除的开头出现的特定字符
`string` 要删除的结尾出现的特定字符

返回值

`string` 删除开头和结尾出现的所有特定字符后得到的字符串

注

```
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()

将字符串中的所有英文字母统一为小写

语法 1

```
string String_ToLower(
    string
)
```

参数

`string` 输入的字符串

返回值

`string` 将所有英文字母统一为小写后得到的字符串。非英文字母的字符保持不变。

注

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

4.23 String_ToUpper()

将字符串中的所有英文字母统一为大写

语法 1

```
string String_ToUpper(
    string
)
```

参数

`string` 输入的字符串

返回值

`string` 将所有英文字母统一为大写后得到的字符串。非英文字母的字符保持不变。

注

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

4.24 Array_Append()

将新数据作为元素添加至数组末尾。

语法 1

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

参数

?[]	参数 1，将被添加元素的数组。支持的类型：byte、int、float、double、bool 和 string。
? or ?[]	参数 2，要添加的数据或数组。其类型必须与将被添加元素的数组的类型相同。 *两个参数的类型必须相同。

返回值

?[]	将参数2元素添加至参数1后得到的新数组。

注

```
?   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()

将数据作为元素插入数组。

语法 1

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

参数

?[]	参数 1，将被插入元素的数组。支持的类型：byte、int、float、double、bool 和 string。	
int	参数 1 的索引起始地址。	
	0 数组1的长度 - 1	合法值
	>= 数组1的长度	合法值，值将被置于参数1末尾。
	< 0	非法值，项目将因错误停止。
? or ?[]	参数 2，要插入的数据或数组。其类型必须与将被插入元素的数组的类型相同。*两个参数的类型必须相同。	

返回值

?[]	将参数2元素插入参数1中的索引起始地址后得到的新数组。

注

```
?   int[] n1 = {100, 200, 30}
    int[] n2 = {40, 50, 60}
    n3 = Array_Insert(n1, 0, n2)      // n3 = {40, 50, 60, 100, 200, 30}      //插入至索引0
    n4 = Array_Insert(n1, 2, n2)      // n4 = {100, 200, 40, 50, 60, 30}      //插入至索引2
    n5 = Array_Insert(n1, -1, n2)     // n5 = {}   //  项目将因错误停止。起始索引非法

?   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} //超出索引范围。值将被置于数组末尾。
```

4.26 Array_Remove()

从数组中删除作为元素的数据。

语法 1

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

参数

?[]	参数 1，将被删除元素的数组。支持的类型：byte、int、float、double、bool 和 string。	
int	参数 1 的索引起始地址。将从此处开始删除。	
	0 ~ 参数1的长度 - 1	合法值
	>= 参数1的长度	非法值，项目将因错误停止。
	< 0	非法值，项目将因错误停止。
int	要删除的元素数量	
	> 0	将从索引起始地址开始删除该数量的元素，元素不足则删除至数组末尾。
	< 0	数量将为0，不会删除任何元素。

返回值

?[]	删除索引起始地址后的元素后得到的新数组。

语法 2

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

注

与语法 1 相同。要删除的元素数量默认为 1。

```
?   int[] n1 = {100, 200, 30, 40, 50, 60}
    n3 = Array_Remove(n1, -1)        // n3 = {}   //  项目将因错误停止。起始值非法。
    n4 = Array_Remove(n1, 100)       // n4 = {}   //  项目将因错误停止。起始值非法。
    n5 = Array_Remove(n1, 0)         // n5 = {200, 30, 40, 50, 60}      //删除索引0
    n6 = Array_Remove(n1, 1, 2)      // n6 = {100, 40, 50, 60}          //从索引1开始删除2个元素
    n7 = Array_Remove(n1, 1, 100)    // n7 = {100}          //从索引1开始删除100个元素（删除至数组末尾）
    n8 = Array_Remove(n1, Length(n1)-1)     // n8 = {100,200,30,40,50,60}    //从最后一个索引开始删除
    n9 = Array_Remove(n1, Length(n1))       // n9 = {}   //  项目将因错误停止。起始值非法。
```

4.27 Array_Equals()

判断两个特定数组是否相同。

语法 1

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

参数

?[]	输入的数组 1（数据类型可为 byte、int、float、double、bool、string）
?[]	输入的数组 2（数据类型可为 byte、int、float、double、bool、string） *数组 1 和数组 2 的数据类型必须相同。

返回值

bool	两个数组是否相同？
true	两个数组相同
false	两个数组不相同

语法 2

```
bool Array_Equals(
    ?[],vv
    int,
    ?[],
    int,
    int
)
```

参数

?[]	输入的数组 1（数据类型可为 byte、int、float、double、bool、string）
int	数组 1 的起始索引（0~（数组 1 的长度）-1）
?[]	输入的数组 2（数据类型可为 byte、int、float、double、bool、string）
int	数组 2 的起始索引（0~（数组 2 的长度）-1）
int	要比较的元素数量（0：返回 true，<0：返回 false） *数组 1 和数组 2 的数据类型必须相同。

返回值

bool	两个数组中的特定元素是否相同？
true	相同
false	不相同（或参数无效）

注

```
?   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    //比较4个元素
    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     //比较2个元素
    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     //比较0个元素
?   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    //起始索引无效
```

4.28 Array_IndexOf()

搜索数组中出现的首个特定元素的索引编号。

语法 1

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

参数

?[]	输入的数组（数据类型可为 byte、int、float、double、bool、string）
?	要搜索的目标元素（数据类型必须与输入的数组?[]相同，但不得为数组。）
int	数组初始索引，将从此处开始搜索

返回值

int	0..（输入的数组的长度）-1	若找到特定元素，将返回索引编号。
	-1	未找到元素

语法 2

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

注

与语法 1 相同。参数 int 的初始地址默认为 0，即从最前端开始搜索。

```
? byte[] n = {100, 200, 30, 100}
  value = Array_IndexOf(n, 200)          // 1
  value = Array_IndexOf(n, 2000)         // -1 //由于2000不是字节类型，将转换为int[]和int进行搜索。
  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()

搜索数组中出现的最后一个特定元素的索引编号。

语法 1

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

参数

?[]	输入的数组（数据类型可为 byte、int、float、double、bool、string）
?	要搜索的目标元素（数据类型必须与输入的数组?[]相同，但不得为数组。）
int	数组初始索引，将从此处开始搜索

返回值

int	0..（输入的数组的长度）-1	若找到特定元素，将返回索引值
	-1	未找到元素

语法 2

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

注

与语法 1 相同。参数 int 的初始地址默认为 0，即从最前端开始搜索。

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

? byte[] n = {100, 200, 30, 100}
    value = Array_LastIndexOf(n, 200)        // 1
    value = Array_LastIndexOf(n, 2000)       // -1 //由于2000不是字节类型，将转换为int[]和int进行搜索。
    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()

反转数组中元素的顺序

语法 1

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

参数

?[]	输入的数组（数据类型可为 byte、int、float、double、bool、string）

返回值

?[]	反转后的数组

注

```
?   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"}
```

语法 2

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

参数

?[]	输入的数组（数据类型可为 byte、int、float、double、bool、string）
int	要将几个元素视为一个部分进行反转
2	2 个元素为一个部分
4	4 个元素为一个部分
8	8 个元素为一个部分
*同一部分中元素的顺序将被反转，但各部分的顺序将保持不变	

返回值

?[]	反转后的数组

注

```
?   byte[] n = {100, 200, 30}
    n = Array_Reverse(n,   2)     // n = {200, 100, 30}
                                  //2个元素为一个部分，即{100,200}{30}
    n = Array_Reverse(n,   4)     // n = {30, 200, 100}
                                  //4个元素为一个部分，即{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个元素为一个部分，即{100,200}{300,400}
```

```
    n = Array_Reverse(n,  4)   // n = {400, 300, 200, 100}
                                   //4个元素为一个部分，即{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个元素为一个部分，即{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个元素为一个部分，即{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()

对数组中的元素进行排序

语法 1

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

参数

?[]	输入的数组（数据类型可为 byte、int、float、double、bool、string）
int	排序方向
	0 升序（默认值）
	1 降序

返回值

?[]	排序后的数组

语法 2

```
?[] Array_Sort(
    ?[]
)
```

注

与语法 1 类似，排序方向为升序

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()

从输入的数组中获取子元素

语法 1

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

参数

?[]	输入的数组（数据类型可 fbyte、int、float、double、bool、string）
int	子元素的起始索引。（0~（数组的长度）-1）
int	子元素中的元素数量

返回值

?[]	取自输入的数组的子元素 若起始索引<0，子元素将为空数组 若起始索引>=输入的数组的长度，子元素将为空数组 若子元素中的元素数量<0，子元素将始于起始索引并结束于输入的数组的最后一个元素 若起始索引与元素数量之和超出输入的数组的长度，子元素将始于起始索引并结束于输入的数组的最后一个元素

语法 2

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

注

与语法 1 类似，但子元素将始于起始索引并结束于输入的数组的最后一个元素

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()

反转输入的数据中的字节单元（int 为 2 字节或 4 字节、float 为 4 字节、double 为 8 字节）的顺序；或反转字符串中字符的顺序。

语法 1

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

参数

int	输入的值
int	输入的值为 int32 还是 int16 格式 0 int32（默认值） 1 int16。若数据不符合int16格式，则将改为应用int32格式。 2 int16。强制应用int16格式。若输入int32数据，可能丢失部分字节

返回值

int	反转输入的值中的字节单元顺序后得到的值。将以4字节为单位反转int32数据。将以2字节为单位反转int16数据。

注

```
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
```

语法 2

```
int ValueReverse(
    int
)
```

参数

int	输入的值

注

与语法 1 类似，采用 int32 输入格式

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

语法 3

```
float ValueReverse(
    float
)
```

参数

float	输入的值

返回值

float	反转输入的值中的字节单元顺序后得到的值。将以4字节为单位反转float数据。

注

```
float i = 40000
value = ValueReverse(i)      // 40000=0x471C4000 → 0x00401C47 // value = 5.887616E-39
```

语法 4

```
double ValueReverse(
    double
)
```

参数

`double` 输入的值

返回值

`double` 反转输入的值中的字节单元顺序后得到的值。将以8字节为单位反转double数据。

注

```
double i = 80000
value = ValueReverse(i)      // 80000=0x40F3880000000000 → 0x000000000088F340 // value = 4.43432217445369E-317
```

语法 5

```
string ValueReverse(
    string
)
```

参数

`string` 输入的字符串

返回值

`string` 反转输入的字符串中的字符顺序后得到的值。

注

```
string i = "ABCDEF"
value = ValueReverse(i)      // value = "FEDCBA"
```

语法6

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

参数

`int[]` 输入的数组值

`int` 输入的值为 int32 还是 int16 格式

- `0` int32（默认值）
- `1` int16。若数据不符合int16格式，则将改为应用int32格式。
- `2` int16。强制应用int16格式。若输入int32数据，可能丢失部分字节

返回值

`int[]` 反转输入的数组中的每个元素内的字节单元顺序后得到的数组。

注

```
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}
```

语法 7

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

参数

`int[]` 输入的数组值

注

与语法 6 类似，输入的整数采用 int32 格式

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

语法 8

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

参数

`float[]` 输入的数组值

返回值

`float[]` 反转输入的数组中的每个元素内的字节单元顺序后得到的数组。

注

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

语法 9

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

参数

`double[]` 输入的数组值

返回值

`double[]` 反转输入的数组中的每个元素内的字节单元顺序后得到的数组。

注

```
double[] i = {10000, 20000}
value = ValueReverse(i)      // value = {4.428251E-317,4.430275E-317}
```

语法 10

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

参数

`string[]` 输入的字符串数组

返回值

`string[]` 反转输入的字符串数组中的每个元素内的字符串后得到的字符串数组。

注

```
string[] i = {"ABCDEFG", "12345678"}
value = ValueReverse(i)      // value = {"GFEDCBA", "87654321"}
```

4.34 GetBytes()

将任意数据类型转换为字节数组。

语法 1

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

参数

?	输入的数据。数据类型可为 int、float、double、bool、string 或 array。
int	输入的整数或浮点型数据遵循小端序还是大端序 0 小端序（默认值） 1 大端序 是否使用0x00 0x00分隔输入的字符串数组数据的各元素 0 不使用0x00 0x00分隔（默认值） 1 使用0x00 0x00分隔

返回值

byte[]	由输入的数据转换成的字节数组

语法 2

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

注

与语法 1 相同，小端序/大端序参数默认为 0，即基于小端序返回字节数组

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}   //将数组中的每个元素转换为字节，1 字节作为 1 个单元。
  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}
  //将数组中的每个元素转换为字节。各 int32 数据依次转换为 4 字节。
  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}
//将数组中的每个元素转换为字节。各 float 数据依次转换为 4 字节。
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}
//将数组中的每个元素转换为字节。各 double 数据依次转换为 8 字节。
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 会将 true 转换为 1、将 false 转换为 0。
value = GetBytes(flag)            // value = {1}
value = GetBytes(flag, 0)         // value = {1}    //由于bool为1字节，字节顺序参数不适用。
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"   //要转换为 UTF8 编码的字符串
value = GetBytes(n)      // value = {0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47}
value = GetBytes(n, 0)   // value = {0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47} //字节顺序参数无效
value = GetBytes(n, 1)   // value = {0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47} //字节顺序参数无效

? 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}

*将string[]转换为byte[]时，若不使用分隔字节，则可保留原有内容，但将无法有效地将byte[]转换回string[]。

*若在数组元素之间插入分隔字节（2个连续的0x00），则可有效地将string[]转换回string[]，但若字符串的值包含0x00 0x00，将可能出现转换错误。

语法 3

将整数（int 类型）转换为字节数组。

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

参数

int	输入的整数（int 类型）
int	输入的值遵循小端序还是大端序
	0 小端序（默认值）
	1 大端序
int	输入的整数值的数据类型为 int32 还是 int16
	0 int32（默认值）
	1 int16。若数据不符合int16格式，则将改为应用int32格式。
	2 int16。强制应用int16格式。若输入int32数据，可能丢失部分字节。

返回值

byte[] 由输入的整数转换成的字节数组。将以4字节为单位转换int32数据。将以2字节为单位转换int16数据。

注

```
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}   //字节丢失
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}   //字节丢失
```

语法 4

将整型数组（int[]类型）转换为字节数组

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

参数

`int[]` 输入的整型数组（int[]类型）

`int` 输入的整型数组遵循小端序还是大端序

- `0` 小端序（默认值）
- `1` 大端序

`int` 输入的整型数组的数据类型为 int32 还是 int16

- `0` int32（默认值）
- `1` int16. 若数据不符合int16格式，则将改为应用int32格式
- `2` int16. 强制应用int16格式。若输入int32数据，可能丢失部分字节。

返回值

`byte[]` 由输入的整型数组转换成的字节数组。每个元素独立转换，组成数组。将以4字节为单位转换int32数据。将以2字节为单位转换int16数据。

注

```
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}   //字节丢失
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}   //字节丢失
```

4.35 GetString()

将任意数据类型转换为字符串

语法 1

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

参数

`?` 输入的数据。数据类型可为 int、float、double、bool、string 或 array。

`int` 输出的字符串的表示法为十进制、十六进制或二进制时，输出的字符串值为十进制、十六进制还是二进制。

10	十进制
16	十六进制
2	二进制

字符串的表示法

123	十进制
0x7F	十六进制
0b101	二进制

输入的值为字符串数组时，输出的字符串值是否为标准字符串格式。

0 或 10	自动检测。若字符串中的值包含双引号或逗号，则转换为标准字符串格式。
1	强制转换为标准字符串格式
其他	不转换

`int` 输出的字符串的格式（仅适用于十六进制或二进制数字）

0	填充数位。添加前缀0x或0b，例如0x0C或0b00001100
1	填充数位。不加前缀0x或0b，例如0C或00001100
2	不填充数位。添加前缀0x或0b，例如0xC或0b1100
3	不填充数位。不加前缀0x或0b，例如C或1100

返回值

`string` 由输入的数据转换成的字符串。若无法转换输入的数据，将返回空字符串。若输入的数据为数组，将分别转换每个元素，并以"{ , , }"格式返回

语法 2

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

注

与语法 1 类似，填充数位并添加前缀 0x 或 0b。

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

语法 3

```
string GetString(
    ?
)
```

注

与语法 1 相同。输出的字符串的表示法参数默认为 10，输出的字符串的格式参数默认为 0。

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

GetString(?) => **GetString**(?, 0, 0) //假设?为字符串数组

```
? 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 = "{0b00000000000000000000001111011,
                                            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 = "0b10000010100010101110000101001000"
```

```
?   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显示为-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}"
                                          //默认使用自动检测
```

语法 4

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

参数

?	输入的数据。数据类型可为 int、float、double、bool、string 或 array。
string	用于输出的字符串的分隔符（仅对数组输入有效）
int	输出的字符串的表示法为十进制、十六进制或二进制时，输出的字符串值为十进制、十六进制还是二进制。
	10　十进制
	16　十六进制
	2　二进制
	字符串的表示法
	123　十进制
	0x7F　十六进制
	0b101　二进制
	输入的值为字符串数组时，输出的字符串值是否为标准字符串格式。
	0 或 10　自动检测。若字符串中的值包含双引号或分隔符，则转换为标准字符串格式。
	1　强制转换为标准字符串格式
	其他　不转换
int	输出的字符串的格式（仅适用于十六进制或二进制数字）
	0　填充数位。添加前缀0x或0b，例如0x0C或0b00001100
	1　填充数位。不加前缀0x或0b，例如0C或00001100
	2　不填充数位。添加前缀0x或0b，例如0xC或0b1100
	3　不填充数位。不加前缀0x或0b，例如C或1100

返回值

string	由输入的数据转换成的字符串。若无法转换输入的数据，将返回空字符串。 若输入的数据为数组，将分别转换每个元素，并以带特定的分隔符的字符串形式返回

语法 5

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

注

与语法 4 相同，填充数位并添加前缀 0x 或 0b

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

语法 6

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

注

与语法 4 相同。输出的字符串的表示法参数默认为 10，输出的字符串的格式参数默认为 0。

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

GetString(?) => **GetString**(?, 0, 0) //假设?为字符串数组

```
?   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"
    *分隔符仅对数组输入有效。

?   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 为控制字符“换行”，而非字符串值。

?   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"
                                      //难以区分分隔符和负号。
```

语法 7

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

参数

?	输入的数据。数据类型可为 int、float、double、bool、string 或 array。
string	输出的字符串的索引，用于数组输入。（仅当?为数组类型数据时有效） *支持数值格式字符串
string	用于输出的字符串的分隔符（仅对数组输入有效）
int	输出的字符串的表示法为十进制、十六进制还是二进制（仅适用于十六进制或二进制数字）

```
            10      十进制
            16      十六进制
            2       二进制
            字符串的表示法
            123     十进制
            0x7F    十六进制
```

0b101　二进制
输入的值为字符串数组时，输出的字符串值是否为标准字符串格式。
0 或 10 自动检测。若字符串中的值包含双引号或分隔符，则转换为标准字符串格式。
1　强制转换为标准字符串格式
其他　不转换

int　输出的字符串的格式（仅适用于十六进制或二进制数字）
0　填充数位。添加前缀0x或0b，例如0x0C或0b00001100
1　填充数位。不加前缀0x或0b，例如0C或00001100
2　不填充数位。添加前缀0x或0b，例如0xC或0b1100
3　不填充数位。不加前缀0x或0b，例如C或1100

返回值

string 转换值后返回的字符串。若无法转换，将返回空字符串。
若输入为数组类型，数组中的元素将转换为附带元素索引值格式字符串前缀、用分隔符隔开的字符串返回。
不会有大括号。

语法 8

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

注

与语法 7 类似，填充数位并添加前缀。
GetString(?, str, str, 16)　=>　**GetString**(?, str, str, 16, 0)

语法 9

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

注

与语法 7 类似，输出为十进制，填充数位并添加前缀。
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"
    *索引和分隔符仅对数组输入生效。
?   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]="支持数值格式字符串
    *\u0D0A 为控制字符“换行”，而非字符串值。
```

4.36 GetToken()

从输入的字符串中获取子字符串，或从输入的 byte[]数组中获取子数组

语法 1

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

参数

string	输入的字符串	
string	前缀。子字符串的开头元素	
string	后缀。子字符串的结尾元素	
int	要获取第几个匹配的子字符串	
	>=1	获取第n个匹配的子字符串
	-1	获取最后一个匹配的子字符串
int	删除选项	
	0	第一个匹配项无需位于输入的字符串的开头，且不删除前缀和后缀。（默认值）
	1	第一个匹配项无需位于输入的字符串的开头，且将删除前缀和后缀。
	2	第一个匹配项必须位于输入的字符串的开头，且不删除前缀和后缀。
	3	第一个匹配项必须位于输入的字符串的开头，且将删除前缀和后缀。

返回值

string 由输入的字符串的一部分组成的字符串
若前缀和后缀均为空字符串，将返回输入的字符串
若要获取的匹配的子字符串的编号<=0或大于匹配的子字符串的总数，将返回空字符串
若删除选项为2或3，则获取的第一个匹配项必须位于输入的字符串的开头，否则将返回空字符串。

语法 2

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

注

与语法 1 类似，保留前缀和后缀。
GetToken(str,str,str,1) => **GetToken**(str,str,str,1,0)

语法 3

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

注

与语法 1 类似，返回第一个匹配项并保留前缀和后缀。

```
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$"  //删除前缀和后缀
value = GetToken(n, Ctrl("\r\n"), "$", 1)       // value = "\u0D0A56\r\n78$"
value = GetToken(n, newline, "$", 2)            // value = ""
value = GetToken(n, NewLine, "$", 1, 1)         // value = "56\r\n78"
*\u0D0A 为控制字符“换行”，而非字符串值。

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, "", "$", 4)             // value = ""
value = GetToken(n, "$", "$", 1, 2)         // value = ""
                                            //与$匹配的字符串不位于输入的字符串的开头。
```

```
value = GetToken(n, "$", "$", -1, 2)      // value = ""
                                          //与$匹配的字符串不位于输入的字符串的开头。
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"
```

语法 4

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

参数

string	输入的字符串	
byte[]	前缀。子字符串的开头元素，字节数组类型	
byte[]	后缀。子字符串的结尾元素，字节数组类型	
int	要获取第几个匹配的子字符串	
	>=1	获取第n个匹配的子字符串
	-1	获取最后一个匹配的子字符串
int	删除选项	
	0	第一个匹配项无需位于输入的字符串的开头，且不删除前缀和后缀。（默认值）
	1	第一个匹配项无需位于输入的字符串的开头，且将删除前缀和后缀。
	2	第一个匹配项必须位于输入的字符串的开头，且不删除前缀和后缀。
	3	第一个匹配项必须位于输入的字符串的开头，且将删除前缀和后缀。

返回值

string 由输入的字符串的一部分组成的字符串

若前缀和后缀均为空字符串，将返回输入的字符串

若要获取的匹配的子字符串的编号<=0或大于匹配的子字符串的总数，将返回空字符串

若删除选项为2或3，则获取的第一个匹配项必须位于输入的字符串的开头，否则将返回空字符串。

语法 5

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

注

与语法 4 类似，保留前缀和后缀

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

语法 6

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

注

与语法 4 类似，返回第一个匹配项并保留前缀和后缀

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

```
string n = "$abcd$1234$ABCD$"
byte[] bb0 = {}, bb1 = {0x24}              //0x24 为$
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 为$ //0x0D,0x0A 为 \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$"      //删除前缀和后缀
value = GetToken(n, bb0, bb1, 1)           // value = "\u0D0A56\r\n78$"
value = GetToken(n, bb0, bb1, 2)           // value = ""
value = GetToken(n, bb0, bb1, 1, 1)        // value = "56\r\n78"
```

*\u0D0A 为控制字符“换行”，而非字符串的内容。

语法 7

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

参数

byte[] 输入的字节数组

string 前缀。输出的字节数组的开头元素，字节数组类型

string 后缀。输出的字节数组的结尾元素，字节数组类型

int	要获取第几个匹配的子字符串
>=1	获取第n个匹配的子字符串
-1	获取最后一个匹配的子字符串
int	删除选项
0	第一个匹配项无需位于输入的字符串的开头，且不删除前缀和后缀。（默认值）
1	第一个匹配项无需位于输入的字符串的开头，且将删除前缀和后缀。
2	第一个匹配项必须位于输入的字符串的开头，且不删除前缀和后缀。
3	第一个匹配项必须位于输入的字符串的开头，且将删除前缀和后缀。

返回值

byte[] 由输入的字节数组的一部分组成的byte[]
若前缀和后缀均为空字符串，将返回输入的数组
若要获取的匹配的子字符串的编号<=0或大于匹配的子字符串的总数，将返回空数组
若删除选项为2或3，则获取的第一个匹配项必须位于输入的字符串的开头，否则将返回空字符串。

语法 8

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

注

与语法 7 类似，保留前缀和后缀
GetToken(byte[],str,str,1) => **GetToken**(byte[],str,str,1,0)

语法 9

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

注

与语法 7 类似，返回第一个匹配项并保留前缀和后缀。
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}
    //删除前缀和后缀
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}
```

语法10

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

参数

byte[]	输入的byte[]
byte[]	前缀。输出的byte[]的开头元素
byte[]	后缀。输出的byte[]的结尾元素
int	要获取第几个匹配的子字符串
>=1	获取第n个匹配的子字符串
-1	获取最后一个匹配的子字符串
int	删除选项
0	第一个匹配项无需位于输入的字符串的开头，且不删除前缀和后缀。（默认值）
1	第一个匹配项无需位于输入的字符串的开头，且将删除前缀和后缀。
2	第一个匹配项必须位于输入的字符串的开头，且不删除前缀和后缀。
3	第一个匹配项必须位于输入的字符串的开头，且将删除前缀和后缀。

返回值

byte[] 由输入的字节数组的一部分组成的byte[]

若前缀和后缀均为空字符串，将返回输入的数组

若要获取的匹配的子字符串的编号<=0或大于匹配的子字符串的总数，将返回空数组

若删除选项为2或3，则获取的第一个匹配项必须位于输入的字符串的开头，否则将返回空字符串。

语法 11

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

注

与语法 10 类似，保留前缀和后缀

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

语法 12

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

注

与语法 10 类似，返回第一个匹配项并保留前缀和后缀。

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

```
string s ="$abcd$1234$ABCD$"
byte[] n = GetBytes(s)
byte[] bb0 = {}, bb1 = {0x24}  //0x24 为$
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}
    //删除前缀和后缀
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()

从输入的字符串中获取所有满足给定条件的子字符串

语法 1

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

参数

string	输入的字符串	
string	前缀。子字符串的开头元素	
string	后缀。子字符串的结尾元素	
int	删除选项	
	0	第一个匹配项无需位于输入的字符串的开头，且不删除前缀和后缀。（默认值）
	1	第一个匹配项无需位于输入的字符串的开头，且将删除前缀和后缀。
	2	第一个匹配项必须位于输入的字符串的开头，且不删除前缀和后缀。
	3	第一个匹配项必须位于输入的字符串的开头，且将删除前缀和后缀。

返回值

string[] 由从输入的字符串中获取的所有子字符串组成的字符串数组
若前缀和后缀均为空字符串，将返回输入的数组
若删除选项为2或3，则获取的第一个匹配项必须位于输入的字符串的开头，否则将返回空字符串。

语法 2

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

注

与语法 1 类似，保留前缀和后缀

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, "$", "", 2)      // value = {}
                                         //$不位于输入的字符串的开头。返回空数组。
value = GetAllTokens(n, "$", "", 3)      // value = {}
                                         //$不位于输入的字符串的开头。返回空数组。
```

```
value = GetAllTokens(n, "$", "$", 0)    // value = {"$1234#ABCD$"}
value = GetAllTokens(n, "$", "$", 1)    // value = {"1234#ABCD"}
value = GetAllTokens(n, "$", "$", 2)    // value = {}
                                        //$不位于输入的字符串的开头。返回空数组。
value = GetAllTokens(n, "$", "$", 3)    // value = {}
                                        //$不位于输入的字符串的开头。返回空数组。
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()

获取当前系统时间

语法 1

```
string GetNow(
    string
)
```

参数

`string` 日期和时间格式字符串，定义日期和时间值的文本表示方法。各说明符的定义如下所示。未包含的字符串将保持不变。

`d`	月中日期，从1到31。
`dd`	月中日期，从01到31。
`ddd`	周几，缩写。
`dddd`	周几，全称。
`f`	日期和时间值中的十分之一秒。
`ff`	日期和时间值中的百分之一秒。
`fff`	日期和时间值中的毫秒。
`ffff`	日期和时间值中的万分之一秒。
`h`	小时，使用12小时制，从1到12。
`hh`	小时，使用12小时制，从01到12。
`H`	小时，使用24小时制，从0到23。
`HH`	小时，使用24小时制，从00到23。
`m`	分钟，从0到59。
`mm`	分钟，从00到59。
`M`	月份，从1到12。
`MM`	月份，从01到12。
`MMM`	月份，缩写。
`MMMM`	月份，全称。
`s`	秒钟，从0到59。
`ss`	秒钟，从00到59。
`t`	AM/PM标识符的首字母。
`tt`	AM/PM标识符。
`y`	年份，从0到99。
`yy`	年份，从00到99。
`yyyy`	年份，四位数字。
`/`	日期分隔符。

返回值

`string` 当前的日期和时间。格式设置有误时将采用默认格式MM/dd/yyyy HH:mm:ss。

注

```
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
```

语法 2

```
string GetNow(
)
```

参数

`void` 无定义格式。将采用默认格式“MM/dd/yyyy HH:mm:ss”

返回值

`string` 当前的日期和时间。

注

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

4.39 GetNowStamp()

获取总运行时间或总运行时间差值

语法 1

```
int GetNowStamp(
)
```

参数

```
    void    无参数
```

返回值

```
    int     当前项目的总运行时间，单位为ms。上限为2147483647 ms
            < 0     溢出，总运行时间无效
```

注

```
    value = GetNowStamp()          // value = 2147483647
    ...其他...
    value = GetNowStamp()          // value = -1      //溢出
```

语法 2

```
double GetNowStamp(
    bool
)
```

参数

```
    bool    是否使用double格式记录项目的总运行时间？
            true    使用double类型，上限为9223372036854775807 ms
            false   使用int32类型，上限为2147483647 ms
```

返回值

```
    double  当前项目的总运行时间
            < 0     溢出。总运行时间无效。
```

注

```
    value = GetNowStamp(false)          // value = 2147483647
    ...其他...
    value = GetNowStamp(false)          // value = -1      //溢出
    value = GetNowStamp(true)           // value = 3147483647
```

语法 3

```
int GetNowStamp(
    int
)
```

参数

```
    int     之前记录的运行时间，单位为ms
```

返回值

```
    int     当前运行时间和输入的运行时间之间的差值，单位为ms。
            运行时间差值 = 当前运行时间 - 输入的运行时间
            < 0     运行时间差值无效，可能是因为输入的运行时间大于当前运行时间或溢出。
```

注

```
    value = GetNowStamp()          // value = 2147483546
    ...其他...（经过100 ms后）
    diff = GetNowStamp(value)      // diff = 100
    ...其他...（经过200 ms后）
    diff = GetNowStamp(value)      // diff = -1       //值超出2147483647
```

语法 4

```
double GetNowStamp(
    double
)
```

参数

`double` 之前记录的运行时间，单位为ms

返回值

`double` 当前运行时间和输入的运行时间之间的差值，单位为ms。
运行时间差值 = 当前运行时间 - 输入的运行时间
`< 0` 运行时间差值无效，可能是因为输入的运行时间大于当前运行时间或溢出。

注

```
value = GetNowStamp()          // value = 2147483546
...其他...（经过100 ms后）
diff = GetNowStamp(value)      // diff = 100
...其他...（经过200 ms后）
diff = GetNowStamp(value)      // diff = 200
```

语法 5

```
bool GetNowStamp(
    int,
    int
)
```

参数

`int` 之前记录的运行时间，单位为ms
`int` 预期运行时间差值

返回值

`bool` 当前运行时间和输入的运行时间之间的差值是否大于预期运行时间差值。
`true` （当前运行时间 - 输入的运行时间）>=预期运行时间
或时间差值小于零，即溢出
`false` （当前运行时间 - 输入的运行时间）<=预期运行时间

注

```
value = GetNowStamp()                  // value = 41730494
...其他...（经过60 ms后）
flag = GetNowStamp(value, 100)         // diff = 60     // flag = false
...其他...（经过60 ms后）
flag = GetNowStamp(value, 100)         // diff = 120    // flag = true
```

语法 6

```
bool GetNowStamp(
    double,
    double
)
```

参数

`double` 之前记录的运行时间，单位为ms
`double` 预期运行时间差值

返回值

`bool` 当前运行时间和输入的运行时间之间的差值是否大于预期运行时间差值。
`true` （当前运行时间 - 输入的运行时间）>=预期运行时间
或时间差值小于零，即溢出
`false` （当前运行时间 - 输入的运行时间）<=预期运行时间

注

```
value = GetNowStamp()                  // value = 41730494
…其他…（经过60 ms后）
flag = GetNowStamp(value, 100)         // diff = 60       // flag = false
…其他…（经过60 ms后）
flag = GetNowStamp(value, 100)         // diff = 120      // flag = true
```

4.40 GetVarValue()

获取变量的值。用户可使用字符串组合成变量名，然后获取组合成的变量的值。

语法 1

```
? GetVarValue(
    string
)
```

参数

`string` 变量名

返回值

`?` 返回变量的值。返回值的类型取决于变量的定义。
若变量不存在，将返回错误。

注

```
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"
                                         //获取var_s1的值
var_re = GetVarValue(var_t + "1")        // var_re = "Hello World"    // var_b + "1" = " var_s1"
                                         //获取var_s1的值
var_re = GetVarValue(var_t + "2")        // var_re = "Hi TM Robot"    // var_b + "2" = " var_s2"
                                         //获取var_s2的值
var_re = GetVarValue(var_t)              //错误       // var_t = " var_s"
                                         //获取var_s的值，但该变量不存在。
```

4.41 Length()

获取输入的数据的字节数量、字符串的长度或数组的长度（数组中的元素数量）

语法 1

```
int Length(
    ?
)
```

参数

? 输入的数据。数据类型可为整数、浮点数、布尔值、字符串或数组。

返回值

int 数据的长度
若输入了整数、浮点数或布尔值，将返回其字节数量。
若输入了字符串，将返回字符串的长度。
若输入了数组，将返回数组中的元素数量

注

```
? 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 //优先使用较小的数据类型存储数字。
? bool n = true
  value = Length(n)              // value = 1
  value = Length(false)          // value = 1
? string n = "A""BC"
  value = Length(n)              // value = 4
                                 //字符串为A"BC。在字符串中，连续两个双引号代表"
  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()

将整数或字符串更改为控制字符

语法 1

```
string Ctrl(
    int
)
```

参数

`int` 遵循大端序格式输入的整数。最多可转换4个字符。不会转换0x00。

返回值

`string` 由输入的整数转换成的字符串（含控制字符）

注

```
b = 0x0D0A
value = Ctrl(b)                 // value = \r\n
value = Ctrl(0x0D0A)            // value = \r\n
value = Ctrl(0x0D000A09)        // value = \r\n\t   //不会转换0x00
value = Ctrl(0x0D300A09)        // value = \r0\n\t //0x30被转换为0
value = Ctrl(0x00)              // value = ""       //空字符串不等同于 NULL。NULL 的代码为 Ctrl("\0")
```

语法 2

```
string Ctrl(
    string
)
```

参数

`string` 输入的字符串。转换遵循以下规则。未列出的字符串将保持不变。

`\0`	0x00	空值
`\a`	0x07	响铃
`\b`	0x08	退格
`\t`	0x09	水平制表符
`\r`	0x0D	回车
`\v`	0x0B	垂直制表符
`\f`	0x0C	换页
`\n`	0x0A	换行

返回值

`string` 由输入的整数转换成的字符串（含控制字符）

注

```
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
```

语法 3

```
string Ctrl(
    byte[]
)
```

参数

`byte[]` 输入的字节数组。转换将从索引[0]开始，直至数组末尾。（0x00也会被转换）

返回值

`string` 由输入的整数转换成的字符串（含控制字符）

注

```
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()

利用 XOR8 位算法计算校验码

语法 1

```
byte XOR8(
    byte[],
    int,
    int
)
```

参数

byte[]	输入的字节数组	
int	起始索引	
	0..(array size-1)	有效
	<0	无效。返回初始值0
	>=array size	无效。返回初始值0
int	要计算的元素数量。 若要计算的元素数量<0，则计算将结束于数组中的最后一个元素 若起始索引和要计算的元素数量之和超出数组大小，则计算将结束于数组中的最后一个元素。	

返回值

byte	校验码。

注

```
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
```

语法 2

```
byte XOR8(
    byte[],
    int
)
```

注

与语法 1 类似，计算至数组中的最后一个元素

XOR8(byte[], int) => **XOR8**(byte[], int, Length(byte[]))

语法 3

```
byte XOR8(
    byte[]
)
```

注

与语法 1 类似，计算数组中的所有元素

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()

利用 SUM8 位算法计算校验码

语法 1

```
byte SUM8(
    byte[],
    int,
    int
)
```

参数

byte[]	输入的字节数组
int	起始索引 0..array size-1 有效 <0 无效。返回初始值0 >=array size 无效。返回初始值0
int	要计算的元素数量。 若要计算的元素数量<0，则计算将结束于数组中的最后一个元素 若起始索引和要计算的元素数量之和超出数组大小，则计算将结束于数组中的最后一个元素。

返回值

byte	校验码。

注

```
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
```

语法 2

```
byte SUM8(
    byte[],
    int
)
```

注

与语法 1 类似，计算至数组中的最后一个元素

SUM8(byte[], int) => **SUM8**(byte[], int, Length(byte[]))

语法 3

```
byte SUM8(
    byte[]
)
```

注

与语法 1 类似，计算数组中的所有元素

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()

利用 SUM16 位算法计算校验码

语法 1

```
byte[] SUM16(
    byte[],
    int,
    int
)
```

参数

byte[]	输入的字节数组
int	起始索引 0..array size-1 有效 <0 无效。返回初始值0 >=array size 无效。返回初始值0
int	要计算的元素数量。 若要计算的元素数量<0，则计算将结束于数组中的最后一个元素 若起始索引和要计算的元素数量之和超出数组大小，则计算将结束于数组中的最后一个元素。

返回值

byte[] 校验码。长度为16位2字节（校验码遵循大端序）

注

```
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}
```

语法 2

```
byte[] SUM16(
    byte[],
    int
)
```

注

与语法 1 类似，计算至数组中的最后一个元素

SUM16(byte[], int) => **SUM16**(byte[], int, Length(byte[]))

语法 3

```
byte[] SUM16(
    byte[]
)
```

注

与语法 1 类似，计算数组中的所有元素

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()

利用 SUM32 位算法计算校验码

语法 1

```
byte[] SUM32(
    byte[],
    int,
    int
)
```

参数

byte[]	输入的字节数组
int	起始索引 0..array size-1 有效 <0 无效。返回初始值0 >=array size 无效。返回初始值0
int	要计算的元素数量。 若要计算的元素数量<0，则计算将结束于数组中的最后一个元素 若起始索引和要计算的元素数量之和超出数组大小，则计算将结束于数组中的最后一个元素。

返回值

byte[]	校验码。长度为32位4字节（校验码遵循大端序）

注

```
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}
```

语法 2

```
byte[] SUM32(
    byte[],
    int
)
```

注

与语法 1 类似，计算至数组中的最后一个元素

SUM32(byte[], int) => **SUM32**(byte[], int, Length(byte[]))

语法 3

```
byte[] SUM32(
    byte[]
)
```

注

与语法 1 类似，计算数组中的所有元素

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()

利用 CRC16 位算法计算校验码

语法 1

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

参数

int	CRC16算法		
	0 CRC16	//初始值为0x0000	//多项式为0xA001
	1 CRC16 (Modbus)	//初始值为0xFFFF	//多项式为0xA001
	2 CRC16 (Sick)	//初始值为0x0000	//多项式为0x8005
	3 CRC16-CCITT (0x1D0F)	//初始值为0x1D0F	//多项式为0x1021
	4 CRC16-CCITT (0xFFFF)	//初始值为0xFFFF	//多项式为0x1021
	5 CRC16-CCITT (XModem)	//初始值为0x0000	//多项式为0x1021
	6 CRC16-CCITT (Kermit)	//初始值为0x0000	//多项式为0x8408
	7 CRC16 Schunk Gripper	//初始值为0xFFFF	//多项式为0x1021
byte[]	输入的字节数组		
int	起始索引		
	0..array size-1 有效		
	<0 无效。返回初始值		
	>=array size 无效。返回初始值		
int	要计算的元素数量。		
	若要计算的元素数量<0，则计算将结束于数组中的最后一个元素		
	若起始索引和要计算的元素数量之和超出数组大小，则计算将结束于数组中的最后一个元素。		

返回值

byte[] 校验码。长度为16位2字节（校验码遵循大端序）

注

```
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}
```

语法 2

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

注

与语法 1 类似，计算至数组中的最后一个元素

CRC16(int, byte[], int) => **CRC16**(int, byte[], int, Length(byte[]))

语法 3

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

注

与语法 1 类似，计算数组中的所有元素

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}
```

语法 4

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

注

与语法 1 类似，CRC16 算法为 0（CRC16）

CRC16(byte[], int, int) => **CRC16**(0, byte[], int, int)

语法 5

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

注

与语法 1 类似，CRC16 算法为 0（CRC16）且计算至数组中的最后一个元素

CRC16(byte[], int) => **CRC16**(0, byte[], int, Length(byte[]))

语法 6

```
byte[] CRC16(
    byte[]
)
```

注

与语法 1 类似，CRC16 算法为 0（CRC16）且计算数组中的所有元素

CRC16(byte[]) => **CRC16**(0, byte[], 0, Length(byte[]))

4.48 CRC32()

利用 CRC32 位算法计算校验码

语法 1

```
byte[] CRC32(
    byte[],
    int,
    int
)
```

参数

byte[]	输入的字节数组
int	起始索引 0..array size-1 有效 <0 无效。返回初始值0 >=array size 无效。返回初始值0
int	要计算的元素数量。 若要计算的元素数量<0，则计算将结束于数组中的最后一个元素 若起始索引和要计算的元素数量之和超出数组大小，则计算将结束于数组中的最后一个元素。

返回值

byte[] 校验码。校验码长度为32位4字节（校验码遵循大端序）

注

```
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}
```

语法 2

```
byte[] CRC32(
    byte[],
    int
)
```

注

与语法 1 类似，计算至数组中的最后一个元素

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

语法 3

```
byte[] CRC32(
    byte[]
)
```

注

与语法 1 类似，计算数组中的所有元素

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()

将字符串的内容封包为用于 Listen 节点（外部命令控制模式）的兼容协议

语法 1

```
string ListenPacket(
    string,
    string
)
```

参数

`string` 用户定义的标头。若为空字符串，将应用默认字符串“TMSCT”

`string` Listen节点通信格式中的数据部分

返回值

`string` 已封包的数据（包括标头、数据长度和校验码）

注

```
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      //TMSCT错误
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      //TMSTA错误
value = ListenPacket("TMSTA", "00")           // $TMSTA,2,00,*41\r\n
```

语法 2

```
string ListenPacket(
    string
)
```

参数

`string` Listen节点通信格式中的数据部分（标头为TMSCT）

返回值

`string` 已封包的数据（包括标头、数据长度和校验码）

注

```
string var_data1 = "1,var_counter++"          // ScriptID, ScriptLanguage
value = ListenPacket(var_data1)               // $TMSCT,15,1, var_counter++,*26\r\n
```

4.50 ListenSend()

向当前连接至监听服务器的客户端设备发送 Listen 节点通信协议 TMSTA。

语法 1

```
int ListenSend(
    string,
    int,
    ?
)
```

参数

string	用于筛选目标 IP，例如 127.0.0.1 代表将发送至所有从 127.0.0.1 连接的客户端设备。
int	用于发送自定义数据消息的TMSTA SubCmd编号，仅限90~99
?	要发送的值。支持的类型：byte、int、float、double、bool 和 string。 将以小端序转换数值、以UTF8编码转换字符串值。

返回值

int	返回的结果	
	0	发送成功
	-1	错误。监听服务器未启动。
	-2	错误。SubCmd必须在90至99之间。

语法 2

```
int ListenSend(
    int,
    ?
)
```

参数

int	用于发送自定义数据消息的TMSTA SubCmd编号，仅限90~99
?	要发送的值。支持的类型：byte、int、float、double、bool 和 string。 将以小端序转换数值、以UTF8编码转换字符串值。

返回值

int	返回的结果	
	0	发送成功
	-1	错误。监听服务器未启动。
	-2	错误。SubCmd必须在90至99之间。

注

若不筛选目标 IP，将向所有已连接的客户端设备发送数据消息。

注

```
string ip = "127.0.0.1"

byte b = 100
value = ListenSend(ip, 10, b)
    //发送 0x64 至 IP 筛选器"127.0.0.1"          // value = -2     //SubCmd 必须在 90 至 99 之间。
value = ListenSend(ip, 90, b)
    //发送 0x64 至 IP 筛选器"127.0.0.1"          // value = -1     //假设监听服务器未启动。
value = ListenSend(ip, 90, b)
    //发送 0x64 至 IP 筛选器"127.0.0.1"          // value = 0      //发送成功
    //筛选 IP 127.0.0.1 并发送至通过该 IP 连接至监听服务器的设备。
    // $TMSTA,4,90,d,*06                        //100 被转换为 0x64。
```

```
value = ListenSend(ip, 90, 123456)
    //发送0x40 0xE2 0x01 0x00至IP筛选器"127.0.0.1"
    // $TMSTA,7,90,@□ ,*C2
    //123456 被转换为 0x40 0xE2 0x01 0x00（int，小端序）
value = ListenSend(90, "123.456")
    //发送0x31 0x32 0x33 0x2E 0x34 0x35 0x36
    //若不筛选目标 IP，将向所有已连接的客户端设备发送数据消息。
    // $TMSTA,10,90,123.456,*7E
    //"123.456"被转换为0x31 0x32 0x33 0x2E 0x34 0x35 0x36（string，UTF8）。
byte[] bb = {100, 200}
value = ListenSend(90, bb)
    //发送0x64 0xC8
    // $TMSTA,5,90,d□,*CF    //{100, 200}被转换为0x64 0xC8
string[] ss = {"T", "M", "達明機器人" }
value = ListenSend(90, ss)
    //发送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()

将变量对象发送至 TMmanager（机器人管理系统）
*执行该函数时，在成功发送对象或达到重试次数上限之前，流程不会继续。

语法 1

```
int VarSync(
    int,
    int,
    ?
)
```

参数

int	重试次数上限	
	<= 0	出现错误时持续重试。
int	两次重试间的时长（单位为毫秒）	
	< 0	时长无效。将应用默认值1000 ms
?	字符串或字符串数组。要发送的变量的名称。	
	可列举多个项目。不会发送不确定的变量，只发送其他确定的变量。	
	*项目为变量名，而非等于变量，如 i 和"i"不同。	
	*若列举了变量，则将发送与列举的变量的值一致的对象。	

返回值

int	发送次数	
	> 0	发送成功。将返回发送次数作为返回值
	0	发送失败
	-1	未启用TM Manager功能
	-9	参数无效

注

```
string var_s = "ABC"
string var_s1 = " var_s"
string[] var_ss = {"ABC", " var_s", " var_s1"}
value = VarSync(1, 1000, " var_s")      //发送变量对象var_s
value = VarSync(2, 2000, var_s)         //发送变量对象ABC（因为var_s的值是"ABC"）
value = VarSync(3, 2000, var_ss)        //发送变量对象ABC、var_s、var_s1（根据字符串数组ss的值）
value = VarSync(3, 2000, " var_ss")     //发送变量对象var_ss
value = VarSync(4, 2000, " var_ss", " var_s1", "ABC")       //发送变量对象var_ss、var_s1、ABC
```

语法 2

```
int VarSync(
    int,
    ?
)
```

注

与语法 1 相同，两次重试间的时间默认为 1000 ms。
VarSync(int, ?) => **VarSync**(int, 1000, ?)

语法 3

```
int VarSync(
    ?
)
```

注

与语法 1 相同，两次重试间的时间默认为 1000 ms 且重试次数无限

VarSync(?)　=>　**VarSync**(0, 1000, ?)

5. 通用函数（脚本）

5.1 Exit()

令项目停止运行。

语法 1

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

参数

bool	是否等待运动命令结束再令项目停止	
	true	等待（默认值）
	false	不等待
int	结束代码	
	> 0	执行closestop函数。
	== 0	评估项目是否出错。（默认值）
	< 0	执行errorstop函数。

返回值

bool	True 已接受命令；False 已拒绝命令

语法 2

```
bool Exit(
    bool
)
```

参数

bool	是否等待运动命令结束再令项目停止	
	true	等待（默认值）
	false	不等待

注

与语法 1 相同。结束代码默认为 0。

语法 3

```
bool Exit(
)
```

参数

void	无参数

返回值

bool	True 已接受命令；False 已拒绝命令

注

与语法 1 相同。是否等待运动命令结束再令项目停止参数默认为 true，结束代码默认为 0。

语法 4

```
bool Exit(
    int
)
```

参数

int	结束代码。

> 0	执行closestop函数。
== 0	与语法1相同。评估项目是否出错。
< 0	执行errorstop函数。

返回值

bool	True 已接受命令；False 已拒绝命令

注

与语法 1 相同。是否等待运动命令结束再令项目停止参数默认为 true。

```
Exit()
//等待运动命令结束，令项目停止运行，然后评估是否发生错误。
//若无错误，随后运行 closestop 函数。
//若有错误，随后运行 errorstop 函数。
Exit(false)     //不等待运动命令结束，立刻令项目停止运行，然后评估是否发生错误。
Exit(false, 1)  //不等待运动命令结束，立刻令项目停止运行，然后运行 closestop 函数。
Exit(1)   //等待运动命令结束，令项目停止运行，然后运行 closestop 函数。
Exit(0)   //等待运动命令结束，令项目停止运行，然后评估是否发生错误。
Exit(-1)  //等待运动命令结束，令项目停止运行，然后运行 errorstop 函数。
```

5.2 Pause()

令项目和机器人运动暂停，但无法暂停的线程和外部命令除外。可通过使用 Resume()或按下机器人操纵杆上的执行按钮继续。

语法 1

```
bool Pause(
)
```

参数

`void` 无参数

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令

注

Pause()

5.3 Resume()

使项目和机器人运动继续。

语法 1

```
bool Resume(
)
```

参数

`void` 无参数

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令

注

Resume()

5.4 WaitFor()

循环等待条件成立或等待超时。

语法 1

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

参数

bool	循环等待条件。可为 true/false 或返回 bool 值的语句。	
int	等待时间（单位为毫秒）	
	< 0	无限期等待
	>= 0	等待时间

返回值

bool	若循环等待条件成立则返回True，若等待超时则返回False。

注

```
int i = 0
bool flag = WaitFor(i++ > 100, 1000)
//执行 i++然后判断 i 是否大于 100，如此循环。一旦条件满足（flag = true）或于 1000 ms 后超时（flag = false），立即退出循环。
```

语法 2

```
bool WaitFor(
    int
)
```

参数

int	等待时间（单位为毫秒）	
	< 0	无效
	>= 0	等待时间

返回值

bool	若在等待时间内则返回True，否则返回False。（因项目停止而中断）

注

```
WaitFor(100)        //等待 100 ms 后超时。
```

5.5 Sleep()

令线程停止特定时间。与 WaitFor(int)相同。

语法 1

```
bool Sleep(
    int
)
```

参数

int	等待时间（单位为毫秒）	
	< 0	无效
	>= 0	等待时间

返回值

bool	若在等待时间内则返回True，否则返回False。（因项目停止而中断）

注

```
Sleep(100)          //等待 100 ms 后超时。
```

5.6 Display()

在 TMflow 仪表板上显示内容。

语法 1

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

参数

string	标题背景颜色	
	"Red"	将标题背景颜色设为红色。
	"Green"	将标题背景颜色设为绿色。
	"Blue"	将标题背景颜色设为蓝色。
	"Yellow"	将标题背景颜色设为黄色。
	"Black"	将标题背景颜色设为黑色。
	"White"	将标题背景颜色设为白色。
	"Gray"	将标题背景颜色设为灰色。
string	标题文本颜色	
	"Red"	将标题文本颜色设为红色。
	"Green"	将标题文本颜色设为绿色。
	"Blue"	将标题文本颜色设为蓝色。
	"Yellow"	将标题文本颜色设为黄色。
	"Black"	将标题文本颜色设为黑色。
	"White"	将标题文本颜色设为白色。
	"Gray"	将标题文本颜色设为灰色。
string	标题文本	
string	文本	

返回值

bool	若设置成功则返回True，不成功则返回False。

注

```
Display("Green", "Yellow", "Gripper Initial Finish", "Force = 30N")    //向 TMflow 仪表板输出标题 Gripper Initial Finish 和文本 Force = 30N，并将标题文本颜色设为黄色、将标题背景颜色设为绿色。
```

语法 2

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

注

与语法 1 相同。默认将标题背景颜色设为白色、将标题文本颜色设为黑色。

语法 3

```
bool Display(
    string
)
```

注

与语法 1 相同。默认将标题背景颜色设为白色、将标题文本颜色设为黑色。标题文本为空字符串。

6. 数学函数

6.1 abs()

返回特定数字的绝对值

语法 1

```
int abs(
    int
)
```

参数

`int` 以整数形式输入的数字

返回值

`int` 以整数形式返回的输入的数字的绝对值

注

```
int i = 10
value = abs(i)      // 10
i = -10
value = abs(i)      // 10
```

语法 2

```
float abs(
    float
)
```

参数

`float` 以浮点数形式输入的数字

返回值

`float` 以浮点数形式返回的输入的数字的绝对值

注

```
float f = 10.1
value = abs(f)      // 10.1
f = -10.1
value = abs(f)      // 10.1
```

语法 3

```
double abs(
    double
)
```

参数

`double` 以双精度浮点数形式输入的数字

返回值

`double` 以双精度浮点数形式返回的输入的数字的绝对值

注

```
double d = 10.8
value = abs(d)      // 10.8
d = -10.8
value = abs(d)      // 10.8
```

6.2 pow()

返回特定底数和指数的幂运算结果

语法 1

```
int pow(
    int,
    double
)
```

参数

int	以整数形式输入的底数
double	指数

返回值

int	以整数形式返回的幂运算结果

语法 2

```
float pow(
    float,
    double
)
```

参数

float	以浮点数形式输入的底数
double	指数

返回值

float	以浮点数形式返回的幂运算结果

语法 3

```
double pow(
    double,
    double
)
```

参数

double	以双精度浮点数形式输入的底数
double	指数

返回值

double	以双精度浮点数形式返回的幂运算结果

注

```
?   int b = 100
    value = pow(b, 2)       // 10000
    value = pow(b, -2)      // 0       //0.0001，但转换为 int 类型
    value = pow(b, 0.1)     // 1       //1.5848932，但转换为 int 类型
    value = pow(b, 2.1)     // 15848   // 15848.932，但转换为 int 类型
    value = pow(b, -2.1)    // 0       // 6.309574E-05，但转换为 int 类型
?   float b = -100
    value = pow(b, 2)       // 10000
    value = pow(b, -2)      // 0.0001
    value = pow(b, 0.2)     //错误// NaN
    value = pow(b, 2.2)     //错误// NaN
    value = pow(b, -2.2)    //错误// 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()

返回指定数字的平方根

语法 1

```
float sqrt(
    float
)
```

参数

`float` 以浮点数形式输入的数字

返回值

`float` 以浮点数形式返回的平方根

语法 2

```
double sqrt(
    double
)
```

参数

`double` 以双精度浮点数形式输入的数字

返回值

`double` 以双精度浮点数形式返回的平方根

注

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

6.4 ceil()

返回对数字向上取整的结果。

语法 1

```
float ceil(
    float
)
```

参数

`float` 以浮点数形式输入的数字

返回值

`float` 以浮点数形式返回的对数字向上取整的结果

语法 2

```
double ceil(
    double
)
```

参数

`double` 以双精度浮点数形式输入的数字

返回值

`double` 以双精度浮点数形式返回的对数字向上取整的结果

注

```
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()

返回对数字向下取整的结果。

语法 1

```
float floor(
    float
)
```

参数

`float` 以浮点数形式输入的数字

返回值

`float` 以浮点数形式返回的对数字向下取整的结果

语法 2

```
double floor(
    double
)
```

参数

`double` 以双精度浮点数形式输入的数字

返回值

`double` 以双精度浮点数形式返回的对数字向下取整的结果

注

```
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()

返回对数字四舍五入的结果。

语法 1

```
float round(
    float,
    int
)
```

参数

float	以浮点数形式输入的数字	
int	返回值小数点后有几位（默认为0，即对数字四舍五入取整）	
	0 .. 15	有效值
	< 0	无效值，将使用默认值0
	> 15	无效值，将使用默认值0

返回值

float	以浮点数形式返回的对数字四舍五入的结果。

语法 2

```
float round(
    float
)
```

注

与语法 1 相同。小数点后默认保留 0 位。

round(float) => **round**(float, 0)

语法 3

```
double round(
    double,
    int
)
```

参数

double	以双精度浮点数形式输入的数字	
int	返回值小数点后有几位（默认为0，即对数字四舍五入取整）	
	0 .. 15	有效值
	< 0	无效值，将使用默认值0
	> 15	无效值，将使用默认值0

返回值

double	以双精度浮点数形式返回的对数字四舍五入的结果。

语法 4

```
double round(
    double
)
```

注

与语法 3 相同。小数点后默认保留 0 位。

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()

返回在 0 至 1 之间的随机浮点数，或在下限至上限之间的随机整数。

语法 1

```
float random(
)
```

参数

void	无参数

返回值

float	返回一个在0至1之间的随机浮点数。

注

```
value = random()        // 0.9473762
value = random()        // 0.7764986
value = random()        // 0.9911129
```

语法 2

```
int random(
    int
)
```

参数

int	随机数的上限

返回值

int	返回一个在0至上限之间的随机整数

注

```
value = random(10)          // 8
value = random(10)          // 1
value = random(10)          // 5
value = random(-10)         // 0   //上限值必须大于0。
```

语法 3

```
int random(
    int,
    int
)
```

参数

int	随机数的下限
int	随机数的上限必须大于下限，否则将返回整数下限值。

返回值

int	返回一个在下限至上限之间的随机整数。

注

```
value = random(5, 10)       // 8
value = random(5, 10)       // 8
value = random(5, 10)       // 6
value = random(5, -1)       // 5   //上限小于下限。以整数形式返回的下限值。
```

6.8 sum()

返回给定数字或数字数组的总和。

语法 1

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

参数 *（参数数量不固定）*

?	输入的值，类型可为 byte、int、byte[]或 int[] 计算每个参数或数组中的每个元素的值。若遇到非数值类型，则返回错误并停止计算。

返回值

int	以整数类型返回给定数字的总和。

语法 2

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

参数 *（参数数量不固定）*

?	输入的值，类型可为 float、double、float[]或 double[] 计算每个参数或数组中的每个元素的值。若遇到非数值类型，则返回错误并停止计算。

返回值

double	以double类型返回给定数字的总和。

注

```
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      //由于需转换为 int 类型
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()

返回给定数字或数字数组的平均值。

语法 1

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

参数 *（参数数量不固定）*

?	输入的值，类型可为 byte、int、float、double、byte[]、int[]、float[]或 double[] 计算每个参数或数组中的每个元素的值。若遇到非数值类型，则返回错误并停止计算。

返回值

double	以double类型返回给定数字的平均值。

注

```
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()

返回给定数字或数字数组的总体标准差$\sigma = \sqrt{\sum\left(X - \frac{\sum X}{N}\right)^2 / N}$。

语法 1

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

参数 *（参数数量不固定）*

?	输入的值，类型可为 byte、int、float、double、byte[]、int[]、float[]或 double[] 计算每个参数或数组中的每个元素的值。若遇到非数值类型，则返回错误并停止计算。

返回值

double	以double类型返回给定数字的标准差。

注

```
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()

返回给定数字或数字数组的样本标准差$s = \sqrt{\sum\left(X - \frac{\sum X}{N}\right)^2/(N-1)}$。

语法 1

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

参数 *（参数数量不固定）*

?	输入的值，类型可为 byte、int、float、double、byte[]、int[]、float[]或 double[] 计算每个参数或数组中的每个元素的值。若遇到非数值类型，则返回错误并停止计算。

返回值

double	以double类型返回给定数字的标准差。

注

```
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()

返回给定数字或数字数组中的最小值。

语法 1

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

参数 *（参数数量不固定）*

?	输入的值，类型可为 byte、int、byte[]或 int[] 比较每个参数或数组中的每个元素的值。若遇到非数值类型，则返回错误并停止比较。

返回值

int	以整数类型返回给定数字中的最小值。

语法 2

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

参数 *（参数数量不固定）*

?	输入的值，类型可为 float、double、float[]或 double[] 比较每个参数或数组中的每个元素的值。若遇到非数值类型，则返回错误并停止比较。

返回值

double	以double类型返回给定数字中的最小值。

注

```
int min1 = min(1,2,3)                                  // 1
int min2 = min(1,2,3,{-4,-5.3,6.2},1.2)                // -5      //由于需转换为 int 类型
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()

返回给定数字或数字数组中的最大值。

语法 1

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

参数 *（参数数量不固定）*

?	输入的值，类型可为 byte、int、byte[]或 int[] 比较每个参数或数组中的每个元素的值。若遇到非数值类型，则返回错误并停止比较。

返回值

int	以整数类型返回给定数字中的最大值。

语法 2

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

参数 *（参数数量不固定）*

?	输入的值，类型可为 float、double、float[]或 double[] 比较每个参数或数组中的每个元素的值。若遇到非数值类型，则返回错误并停止比较。

返回值

double	以double类型返回给定数字中的最大值。

注

```
int max1 = max(1,2,3)                          // 3
int max2 = max(1,2,3,{-4,-5.3,6.2},1.2)        // 6      //由于需转换为 int 类型
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()

将角度值转换为弧度值

语法 1

```
float d2r(
    float
)
```

参数

`float` 以浮点数形式输入的角度值

返回值

`float` 以浮点数形式返回的弧度值

语法 2

```
double d2r(
    double
)
```

参数

`double` 以双精度浮点数形式输入的角度值

返回值

`double` 以双精度浮点数形式返回的弧度值

注

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

6.15 r2d()

将弧度值转换为角度值

语法 1

```
float r2d(
    float
)
```

参数

`float` 以浮点数形式输入的弧度值

返回值

`float` 以浮点数形式返回的角度值

语法 2

```
double r2d(
    double
)
```

参数

`double` 以双精度浮点数形式输入的弧度值

返回值

`double` 以双精度浮点数形式返回的角度值

注

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

6.16 sin()

返回输入的角度值的正弦

语法 1

```
float sin(
    float
)
```

参数

`float` 以浮点数形式输入的角度值

返回值

`float` 以浮点数形式返回的输入的角度值的正弦

语法 2

```
double sin(
    double
)
```

参数

`double` 以双精度浮点数形式输入的角度值

返回值

`double` 以双精度浮点数形式返回的输入的角度值的正弦

注

```
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()

返回输入的角度值的余弦

语法 1

```
float cos(
    float
)
```

参数

`float` 以浮点数形式输入的角度值

返回值

`float` 以浮点数形式返回的输入的角度值的余弦

语法 2

```
double cos(
    double
)
```

参数

`double` 以双精度浮点数形式输入的角度值

返回值

`double` 以双精度浮点数形式返回的输入的角度值的余弦

注

```
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()

返回输入的角度值的正切

语法 1

```
float tan(
    float
)
```

参数

`float` 以浮点数形式输入的角度值

返回值

`float` 以浮点数形式返回的输入的角度值的正切

语法 2

```
double tan(
    double
)
```

参数

`double` 以双精度浮点数形式输入的角度值

返回值

`double` 以双精度浮点数形式返回的输入的角度值的正切

注

```
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()

以角度值形式返回输入的值的反正弦

语法 1

```
float asin(
    float
)
```

参数

`float` 以浮点数形式输入的正弦值，必须在-1至1之间

返回值

`float` 以浮点数角度值形式返回的输入的值的反正弦

语法 2

```
double asin(
    double
)
```

参数

`double` 以双精度浮点数形式输入的正弦值，必须在-1至1之间

返回值

`double` 以双精度浮点数角度值形式返回的输入的值的反正弦

注

```
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()

以角度值形式返回输入的值的反余弦

语法 1

```
float acos(
    float
)
```

参数

`float` 以浮点数形式输入的余弦值，必须在-1至1之间

返回值

`float` 以浮点数形式返回的角度值

语法 2

```
double acos(
    double
)
```

参数

`double` 以双精度浮点数形式输入的余弦值，必须在-1至1之间

返回值

`double` 以双精度浮点数形式返回的角度值

注

```
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))   // 14.999999999999996
value = acos(cos((double)30))   // 29.999999999999993
```

6.21 atan()

以角度值形式返回输入的值的反正切

语法 1

```
float atan(
    float
)
```

参数

`float` 以浮点数形式输入的正切值

返回值

`float` 以浮点数形式返回的角度值

语法 2

```
double atan(
    double
)
```

参数

`double` 以双精度浮点数形式输入的正切值

返回值

`double` 以双精度浮点数形式返回的角度值

注

```
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()

返回引数的商的反正切

语法 1

```
float atan2(
    float,
    float
)
```

参数

`float` 输入的代表Y坐标的浮点数
`float` 输入的代表X坐标的浮点数

返回值

`float` 以浮点数形式返回的角度值

语法 2

```
double atan2(
    double,
    double
)
```

参数

`double` 输入的代表Y坐标的双精度浮点数
`double` 输入的代表X坐标的双精度浮点数

返回值

`double` 以双精度浮点数形式返回的角度值

注

```
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()

返回输入的值的自然对数

语法 1

```
float log(
    float,
    double
)
```

参数

`float` 以浮点数形式输入的值

`double` 对数的底数

返回值

`float` 以浮点数形式返回的输入的值以底数参数为底的对数

语法 2

```
double log(
    double,
    double
)
```

参数

`double` 以双精度浮点数形式输入的值

`double` 对数的底数

返回值

`double` 以双精度浮点数形式返回的输入的值以底数参数为底的对数

注

```
value = log(16, 2)      // 4
value = log(16, 8)      // 1.3333334
value = log(16, 10)     // 1.20412
value = log(16, 16)     // 1
```

语法 3

```
float log(
    float
)
```

参数

`float` 以浮点数形式输入的值

返回值

`float` 以浮点数形式返回的输入的值以e为底的自然对数

语法 4

```
double log(
    double
)
```

参数

`double` 以双精度浮点数形式输入的值

返回值

`double` 以双精度浮点数形式返回的输入的值以e为底的自然对数

注

```
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()

返回输入的值以 10 为底的对数

语法 1

```
float log10(
    float
)
```

参数

`float` 以浮点数形式输入的值

返回值

`float` 以浮点数形式返回的输入的值以10为底的对数

语法 2

```
double log10(
    double
)
```

参数

`double` 以双精度浮点数形式输入的值

返回值

`double` 以双精度浮点数形式返回的输入的值以10为底的对数

注

```
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()

返回特定向量的 2 范数。

语法 1

```
float norm2(
    float[]
)
```

参数

float[]	将求取该向量的2范数（又称欧几里得范数、向量长度）。

返回值

float	特定向量的2范数（又称欧几里得范数、向量长度）

注

$$\|v\| = \sqrt{\sum_{i=1}^{i=N} |v_i|^2}$$

```
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()

返回两个坐标间的距离。

语法 1

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

参数

float[]	第一个坐标	$\{X_{1(mm)}\ Y_{1(mm)}\ Z_{1(mm)}\ RX_{1(°)}\ RY_{1(°)}\ RZ_{1(°)}\}$
float[]	第二个坐标	$\{X_{2(mm)}\ Y_{2(mm)}\ Z_{2(mm)}\ RX_{2(°)}\ RY_{2(°)}\ RZ_{2(°)}\}$

返回值

float	两个坐标间的距离

注

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

6.27 trans()

返回从一个特定点到另一个特定点的位移和旋转角度。

语法 1

```
float[] trans(
    float[],
    float[],
    bool
)
```

参数

float[]	第一个点	$\{X_{1(mm)} \quad Y_{1(mm)} \quad Z_{1(mm)} \quad RX_{1(°)} \quad RY_{1(°)} \quad RZ_{1(°)}\}$
float[]	第二个点	$\{X_{2(mm)} \quad Y_{2(mm)} \quad Z_{2(mm)} \quad RX_{2(°)} \quad RY_{2(°)} \quad RZ_{2(°)}\}$
bool	参考坐标	
	false	参考机器人基准（默认值）
	true	参考第一个点

返回值

float[]	从第一个点到第二个点的位移和旋转角度。 $\{X_{trans} \quad Y_{trans} \quad Z_{trans} \quad RX_{trans} \quad RY_{trans} \quad RZ_{trans}\}$ 若无法计算，将返回空数组。

语法 2

```
float[] trans(
    float[],
    float[]
)
```

注

与语法 1 相同。默认将参考坐标设为 false，即参考机器人基准。

原始变换矩阵 = $\begin{bmatrix} R & P \\ 0 & 1 \end{bmatrix}$

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

第一个点 = $\begin{bmatrix} R_1 & P_1 \\ 0 & 1 \end{bmatrix}$

第二个点 = $\begin{bmatrix} R_2 & P_2 \\ 0 & 1 \end{bmatrix}$

若 *参考坐标*参数为`false`（参考坐标为机器人基准）

$$P_{trans} = P_2 - P_1$$
$$R_{trans} = R_2 * {R_1}^{-1}$$

若 *参考坐标*参数为 `true`（参考坐标为第一个点）

$$P_{trans} = {R_1}^{-1} * (P_2 - P_1)$$
$$R_{trans} = {R_1}^{-1} * R_2$$

```
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()

返回与输入的位移和旋转角度 {x, y, z, rx, ry, rz}相反的位移和旋转角度 {x, y, z, rx, ry, rz}。

语法 1

```
float[] inversetrans(
    float[],
    bool
)
```

参数

float[]	输入的位移和旋转角度 $\{X_o \quad Y_o \quad Z_o \quad RX_o \quad RY_o \quad RZ_o\}$
bool	参考坐标
false	参考机器人基准（默认值）
true	参考输入的位移和旋转角度

返回值

float[]	与输入的位移和旋转角度相反的 $\{X_{inv} \quad Y_{inv} \quad Z_{inv} \quad RX_{inv} \quad RY_{inv} \quad RZ_{inv}\}$ 位移和旋转角度 $\{X_o \quad Y_o \quad Z_o \quad RX_o \quad RY_o \quad RZ_o\}$ 若无法计算，将返回空数组。

语法 2

```
float[] inversetrans(
    float[]
)
```

注

与语法 1 相同。默认将参考坐标设为 false，即参考机器人基准。

原始变换矩阵 = $\begin{bmatrix} R & P \\ 0 & 1 \end{bmatrix}$

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

初始点 = $\begin{bmatrix} R_i & P_i \\ 0 & 1 \end{bmatrix}$

若 *参考坐标*参数为false（参考坐标为机器人基准）

$P_{inv} = -P_i$

$R_{inv} = {R_i}^{-1}$

若 *参考坐标*参数为 true（参考坐标为输入的 {x, y, z, rx, ry, rz}，等于变换矩阵的逆矩阵）

$P_{inv} = {R_i}^{-1} * (-P_i)$

$R_{inv} = {R_i}^{-1}$

```
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()

将位移和旋转角度应用于特定点，返回计算出的终点。

语法 1

```
float[] applytrans(
    float[],
    float[],
    bool
)
```

参数

float[]	初始点	{X_i Y_i Z_i RX_i RY_i RZ_i}
float[]	位移和旋转角度	{X_o Y_o Z_o RX_o RY_o RZ_o}
bool	参考坐标	
	false	参考机器人基准（默认值）
	true	参考初始点

返回值

float[]	将位移和旋转角度应用于初始点后计算出的终点 {X_t Y_t Z_t RX_t RY_t RZ_t} 若无法计算，将返回空数组。

语法 2

```
float[] applytrans(
    float[],
    float[]
)
```

注

与语法 1 相同。默认将参考坐标设为 false，即参考机器人基准。

原始变换矩阵 $= \begin{bmatrix} R & P \\ 0 & 1 \end{bmatrix}$

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

初始点 $= \begin{bmatrix} R_i & P_i \\ 0 & 1 \end{bmatrix}$

若 *参考坐标*参数为false（参考坐标为机器人基准）

$P_t = P_i + P_{trans}$

$R_t = R_{trans} * R_i$

若 *参考坐标*参数为 true（参考坐标为初始点）

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

$R_t = R_i * R_{trans}$

```
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()

根据比例返回两特定点之间的插值点

语法 1

```
float[] interpoint(
    float[],
    float[],
    float
)
```

参数

float[]	第一个点 $\{X_{1(mm)} \quad Y_{1(mm)} \quad Z_{1(mm)} \quad RX_{1(°)} \quad RY_{1(°)} \quad RZ_{1(°)}\}$
float[]	第二个点 $\{X_{2(mm)} \quad Y_{2(mm)} \quad Z_{2(mm)} \quad RX_{2(°)} \quad RY_{2(°)} \quad RZ_{2(°)}\}$
float	比例

返回值

float[]	根据比例得出的初始点和终点间的线性插值点 $\{X_i \quad Y_i \quad Z_i \quad RX_i \quad RY_i \quad RZ_i\}$。若无法计算，将返回空数组。

注

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

```
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()

通过坐标系转换将原坐标值转换成以新坐标系表示的新坐标值并返回。在转换过程中，坐标世界中原始点的物理位置保持不变，只有其参考坐标和相应的坐标值的表示方式会受到转换影响。

语法 1

```
float[] changeref(
    float[],
    float[],
    float[]
)
```

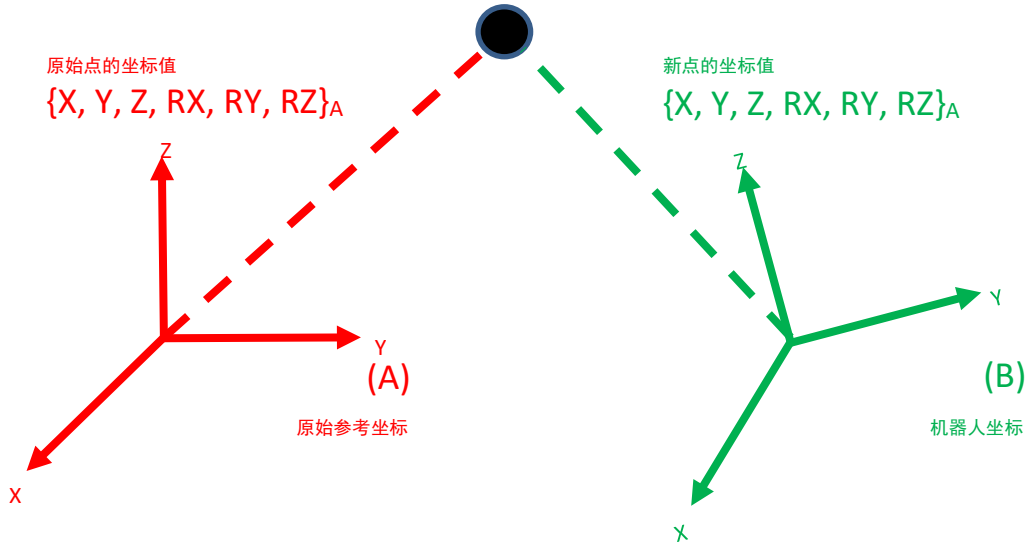
参数

float[]	原始点的坐标值 $\{X_o \quad Y_o \quad Z_o \quad RX_o \quad RY_o \quad RZ_o\}_A$
float[]	原始参考坐标系 $\{X_{oa} \quad Y_{oa} \quad Z_{oa} \quad RX_{oa} \quad RY_{oa} \quad RZ_{oa}\}_A$
float[]	新参考坐标系 $\{X_n \quad Y_n \quad Z_n \quad RX_n \quad RY_n \quad RZ_n\}_B$

返回值

float[]	新点的坐标值 $\{X_{nb} \quad Y_{nb} \quad Z_{nb} \quad RX_{nb} \quad RY_{nb} \quad RZ_{nb}\}_B$ 若无法计算，将返回空数组。

注




```
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}
//将"RobotBase"坐标系下的"P1"值转换为"base1"坐标系下的点值
```

语法 2

```
float[] changeref(
    float[],
    float[]
)
```

参数

`float[]` 原始点的坐标值 $\{X_o \quad Y_o \quad Z_o \quad RX_o \quad RY_o \quad RZ_o\}_A$

`float[]` 原始参考坐标系 $\{X_{oa} \quad Y_{oa} \quad Z_{oa} \quad RX_{oa} \quad RY_{oa} \quad RZ_{oa}\}_A$

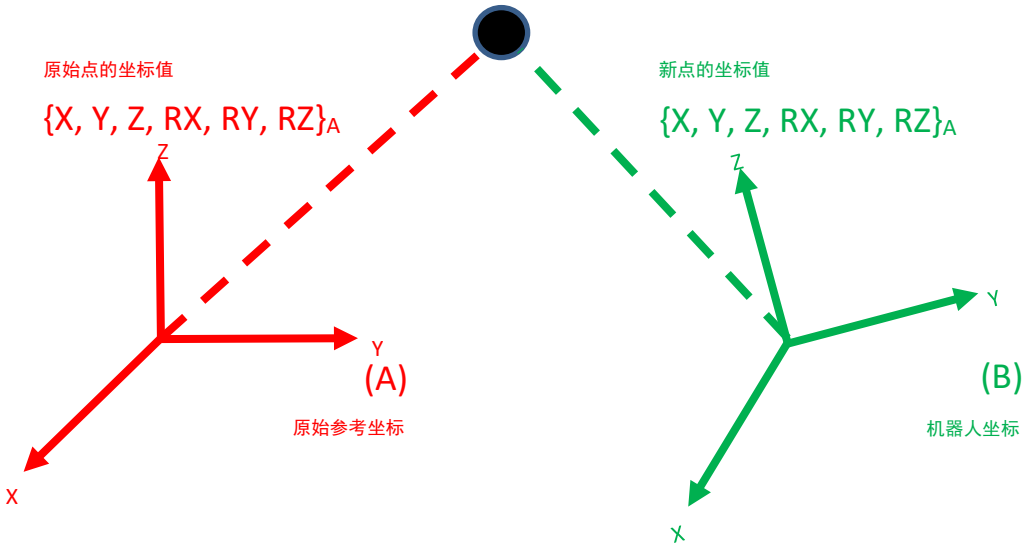
返回值

`float[]` 新点的坐标值 $\{X_{nr} \quad Y_{nr} \quad Z_{nr} \quad RX_{nr} \quad RY_{nr} \quad RZ_{nr}\}_R$

若无法计算，将返回空数组。

注

用法与语法 1 相同，但默认将机器人坐标系 $\{0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\}_R$ 作为新参考坐标系。




```
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()

基于输入的点，计算输入的点的坐标平面，并返回由平面上的三个点转换成的平面参数值。

语法 1

```
float[] points2coord(
    float[],
    float[],
    float[]
)
```

参数

`float[]` 坐标平面的原点坐标$X_{1(mm)}$ $Y_{1(mm)}$ $Z_{1(mm)}$ $RX_{1(°)}$ $RY_{1(°)}$ $RZ_{1(°)}$

`float[]` 平面X轴上的任意点$X_{2(mm)}$ $Y_{2(mm)}$ $Z_{2(mm)}$ $RX_{2(°)}$ $RY_{2(°)}$ $RZ_{2(°)}$

`float[]` 平面第一象限内的任意点$X_{3(mm)}$ $Y_{3(mm)}$ $Z_{3(mm)}$ $RX_{3(°)}$ $RY_{3(°)}$ $RZ_{3(°)}$

*必须以相同坐标系表示上述输入的点。

返回值

`float[]` 根据平面上的三个点得出的平面参数值

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

通过与平面原点重合的

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

和两个点组成的角计算，两个点为

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

和

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

注

假设存在以下三点：P1、P2和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)
```

语法 2

```
float[] points2coord(
    float, float, float,
    float, float, float,
    float, float, float
)
```

参数

`float, float, float` 待计算坐标平面的原点坐标$X_{1(mm)}$ $Y_{1(mm)}$ $Z_{1(mm)}$

`float, float, float` 待计算坐标平面X轴上的任意点的坐标$X_{2(mm)}$ $Y_{2(mm)}$ $Z_{2(mm)}$

`float, float, float` 待计算坐标平面上任意点的坐标$X_{3(mm)}$ $Y_{3(mm)}$ $Z_{3(mm)}$

*必须以相同坐标系表示上述输入的点。

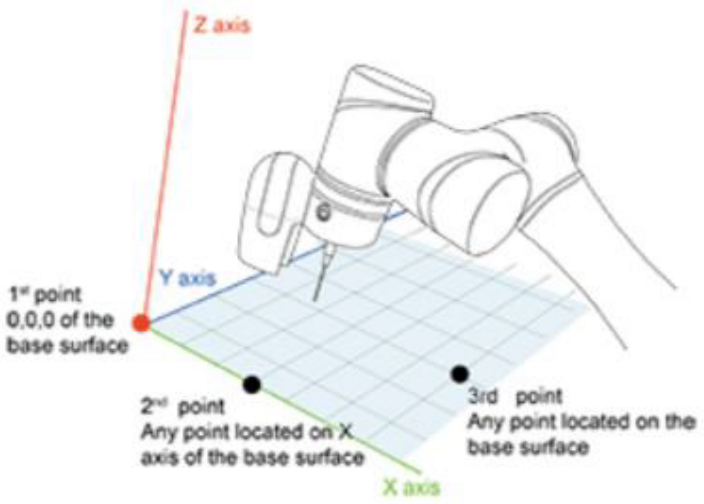
返回值

`float[]` 待计算坐标平面的定义参数。

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

注

```
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)
```



6.33 intercoord()

将输入的两个平面转换为一个新的平面。

语法 1

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

参数

`float[]` 输入的第一个平面$X_{1(mm)}\ Y_{1(mm)}\ Z_{1(mm)}\ RX_{1(°)}\ RY_{1(°)}\ RZ_{1(°)}$

`float[]` 输入的第二个平面$X_{2(mm)}\ Y_{2(mm)}\ Z_{2(mm)}\ RX_{2(°)}\ RY_{2(°)}\ RZ_{2(°)}$

*必须以相同坐标系表示上述输入的点。

返回值

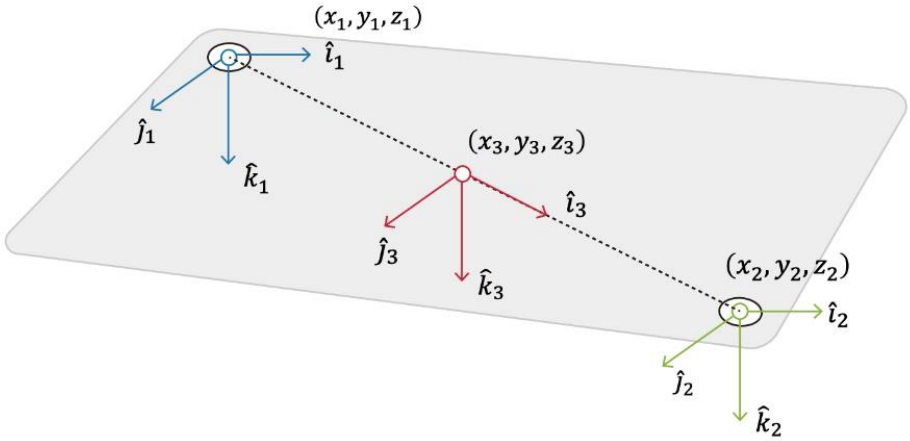
`float[]` 由输入的两个平面转换成的新平面。

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

注

$$X_3 = (X_1 + X_2)/2$$
$$Y_3 = (Y_1 + Y_2)/2$$
$$Z_3 = (Z_1 + Z_2)/2$$
$$\hat{\imath_3} = (X_2 - X_1, Y_2 - Y_1, Z_2 - Z_1,)$$
$$\hat{j_3} = (\hat{k_1} \times \hat{\imath_3})$$
$$\hat{k_3} = (\hat{\imath_1} \times \hat{j_3})$$

$x_3 = (x_1 + x_2)/2$
$y_3 = (y_1 + y_2)/2$
$z_3 = (z_1 + z_2)/2$
$\hat{\imath}_3 = (x_2 - x_1, y_2 - y_1, z_2 - z_1)$
$\hat{j}_3 = \hat{k}_1 \times \hat{\imath}_3$
$\hat{k}_3 = \hat{\imath}_3 \times \hat{j}_3$



假设存在视觉Landmark任务vb_1和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. 文件函数

- 文件函数可用于与文件的读取、写入和查询相关的操作。
- 文件路径
  1. 本地路径。只能使用在名为**TextFiles**、**XmlFiles**和**TMcraft**的目录下的路径。

FileName.txt	其目录默认为.\TextFiles （该文件路径与.\TextFiles\FileName.txt相同）
.\TextFiles\FileName.txt	文件位于名为**TextFiles**的本地目录下。
.\XmlFiles\FileName.xml	文件位于名为**XmlFiles**的本地目录下。
.\XmlFiles\文件夹\文件	子目录位于名为**XmlFiles**的本地目录下。
..\folder	不可使用。
.\TextFiles\..\..\folder	不可使用。

     不支持绝对路径。

C:\file1	无效。
D:\folder\file2	无效。
\TextFiles\FileName.txt	无效。

  2. 外部设备路径。适用于标为**TMROBOT**的U盘或SSD。

\USB\TMROBOT	外部USB的根目录。

  3. 远程路径。适用于TMflow中的网络服务。

\\127.0.0.1\shared	SMB / CIFS
ftp://127.0.0.1	FTP

- 路径不区分大小写。例如，以下路径全部指向同一文件。
  - .\TextFiles\FileName.txt
  - .\textfiles\fileName.txt
  - .\Textfiles\Filename.TXT
- 可使用路径指向子目录，例如：
  - subfolder\file
  - .\TextFiles\subfolder1\subfolder2\file
  - .\XmlFiles\subfolder\file
  - \USB\TMROBOT\subfolder\file
  - \\127.0.0.1\shared\subfolder\file
- 文件大小上限为 10 MB（10485760 字节）。
- 要读取或写入的数组的类型取决于数组的定义。

7.1 File_ReadBytes()

读取文件内容并以字节数组类型返回。

语法 1

```
byte[] File_ReadBytes(
    string
)
```

参数

string	文件路径

返回值

byte[]	以字节数组类型返回的文件内容。

注

```
.\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")//错误。不支持绝对路径。
byte[] var_bb4 = File_ReadBytes(".\SampleFile1.txt") //错误。文件必须位于名为 TextFiles 或 XmlFiles
                                                     的本地目录下。
byte[] var_bb5 = File_ReadBytes("SampleFileXX.txt")//错误。文件不存在。
```

7.2 File_ReadText()

读取文件内容并以字符串类型返回。

语法 1

```
string File_ReadText(
    string
)
```

参数

string	文件路径

返回值

string	以字符串类型返回的文件内容。

注

```
.\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 代表换行符，而非字符串值。

```
string var_s3 = File_ReadText("C:\SampleFile1.txt")
//错误。不支持绝对路径。
string var_s4 = File_ReadText(".\SampleFile1.txt")
//错误。文件必须位于名为 TextFiles 或 XmlFiles 的本地目录下。
```

7.3 File_ReadLines()

读取文件内容并以由换行符分隔的字符串类型返回。

语法 1

```
string[] File_ReadLines(
    string
)
```

参数

string	文件路径

返回值

string[]	以由换行符分隔的字符串类型返回的文件内容。

语法 2

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

参数

string	文件路径
int	要从第几行开始读取
int	要读取几行

返回值

string[]	以由换行符分隔的字符串类型返回的文件内容。 若“要从第几行开始读取”参数 <= 0，将返回空数组。 若“要从第几行开始读取”参数 > 总行数，将返回空数组。 若“要读取几行”参数 <= 0，将返回第一行到最后一行。 若“要读取几行”参数 > 总行数，将从开始处的第一行读取到最后一行并返回。

语法 3

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

注

与语法 2 相同，但“要读取几行”参数默认为 0，始终读取到最后一行并返回。

File_ReadLines(string,int,int) => **File_ReadLines**(string,int,0)

注

.\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!"} //超出总行数。读取到最后一行并返回。
var_ss = File_ReadLines("SampleFile2.txt", 0)    // {}        //空数组
var_ss = File_ReadLines("SampleFile2.txt", 4)    // {}        //空数组
int var_len = Length(var_ss)                     // 0
var_ss = File_ReadLines("SampleFile2.txt", 3, 0) // {"2Hi TM Robot!"} //返回第 3 行到最后一行。
```

```
.\TextFiles\SampleFile3.txt
1|
```

```
var_ss = File_ReadLines("SampleFile3.txt")       // {""}      // var_ss[0] = ""
var_len = Length(var_ss)                         // 1
```

7.4 File_NextLine()

记录上一次读取的文件路径，继续读取下一行文件内容或打开文件以读取。

语法 1

```
string File_NextLine(
    string
)
```

参数

string	文件路径

返回值

string	若与上一次读取的文件路径相同，将返回下一行文件内容。 若与上一次读取的文件路径不同，则打开文件并返回第一行文件内容。若已读取至文件末尾，将返回空字符串。

语法 2

```
string File_NextLine(
    string,
    bool
)
```

参数

string	文件路径
bool	是否打开文件以读取
false	测试文件路径。若相同，则继续读取下一行。若不同，则打开文件以读取。
true	打开文件并读取第一行。

返回值

string	“是否打开文件以读取”参数为false时 若与上一次读取的文件路径相同，将返回下一行文件内容。 若与上一次读取的文件路径不同，则打开文件并返回第一行文件内容。 “是否打开文件以读取”参数为true时 打开文件并返回第一行文件内容。 若已读取至文件末尾，将返回空字符串。

语法 3

```
string File_NextLine(
)
```

参数

void	无参数

返回值

string	返回记录中上一次读取的文件内容的下一行，若未读取过，则返回空字符串。

注

```
.\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()                      // ""                //没有打开文件以读取。
var_s = File_NextLine("SampleFile4.txt")     // "4Hello World!"
var_s = File_NextLine("SampleFile4.txt")     // ""                // 继续读取下一行。
var_s = File_NextLine("SampleFile5.txt")     // "5Hello World!"   // 文件路径不同。打开文件以读取。
```

```
var_s = File_NextLine("SampleFile4.txt")          // "4Hello World!"     // 文件路径不同。打开文件以读取。
var_s = File_NextLine("SampleFile4.txt")          // ""                  // 继续读取下一行
var_s = File_NextLine("SampleFile4.txt")          // "4Hello TM Robot!"
var_s = File_NextLine("SampleFile4.txt")          // ""                  // 继续读取下一行（已读取至文件末尾）
var_s = File_NextLine("SampleFile4.txt", true)    // "4Hello World!"     // 打开文件并读取第一行。
var_s = File_NextLine("SampleFile4.txt", true)    // "4Hello World!"     // 打开文件并读取第一行。
var_s = File_NextLine("SampleFile4.txt", false)   // ""                  // 继续读取下一行
var_s = File_NextLine()                           // "4Hello TM Robot!"
```

*如需分辨空行和文件末尾，请使用语法4，根据string[]的大小判断。
*或使用File_NextEOF()判断文件末尾。

语法 4

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

参数

string	文件路径
int	读取参数
0	测试文件路径。若相同，则继续读取下一行。若不同，则打开文件以读取。
1	打开文件并读取第一行。
2	打开文件但不读取。返回空数组。

返回值

string[]	以数组形式返回下一行文件内容中的字符串。 数组大小为1代表读取了下一行中的字符串。 数组大小为0代表已读取至文件末尾。

语法 5

```
string[] File_NextLine(
    int
)
```

参数

int	读取参数
0	测试文件路径。若相同，则继续读取下一行。若不同，则打开文件以读取。
1	打开文件并读取第一行。
2	打开文件但不读取。返回空数组。

返回值

string[]	以数组形式返回记录中上一次读取的文件内容的下一行中的字符串，若未读取过，则返回空字符串数组。 数组大小为1代表读取了下一行中的字符串。 数组大小为0代表已读取至文件末尾。

注

.\TextFiles\SampleFile4.txt

```
1|  4Hello World!
2|
3|  4Hello TM Robot!
```

.\TextFiles\SampleFile5.txt

```
1|  5Hello World!
2|  5Hello TM Robot!
3|  5Hi TM Robot!
```

```
string[] var_ss = {""}
var_ss = File_NextLine(0)                          // {}                    //没有打开文件以读取。
var_ss = File_NextLine("SampleFile4.txt", 0)       // {"4Hello World!"}
var_ss = File_NextLine("SampleFile4.txt", 0)       // {""}                  // 继续读取下一行。
var_ss = File_NextLine("SampleFile5.txt", 0)       // {"5Hello World!"}     //文件路径不同。打开文件以读
                                                                            取。
var_ss = File_NextLine("SampleFile4.txt", 0)       // {"4Hello World!"}     //文件路径不同。打开文件以读
                                                                            取。
var_ss = File_NextLine("SampleFile4.txt", 0)       // {""}                  // 继续读取下一行。
int var_len = Length(var_ss)                       // 1
var_ss = File_NextLine("SampleFile4.txt", 0)       // {"4Hello TM Robot!"}
var_ss = File_NextLine("SampleFile4.txt", 0)       // {}                    //继续读取下一行（已读取至文
                                                                            件末尾）
var_len = Length(var_ss)                           // 0
var_ss = File_NextLine("SampleFile4.txt", 1)       // {"4Hello World!"}     // 打 开 文 件 并 读 取 第 一 行 。
var_ss= File_NextLine("SampleFile4.txt", 2)        // {}                    //打开文件但不读取。
var_len = Length(var_ss)                           // 0
var_ss = File_NextLine("SampleFile4.txt")          // {"4Hello World!"}     // 继续读取下一行。
var_ss = File_NextLine(0)                          // {""}                  // 继续读取下一行。
var_ss = File_NextLine(0)                          // {"4Hello TM Robot!"}
var_ss = File_NextLine(0)                          // {}
```

7.5 File_NextEOF()

测试上一次读取的文件路径，判断是否已读取至文件末尾。

语法 1

```
bool File_NextEOF(
)
```

参数

void	无参数，但需要和File_NextLine()搭配使用

返回值

bool	若未读取过，将返回true。	
	false	未读取至文件末尾。
	true	没有打开文件或已读取至文件末尾。

注

```
.\TextFiles\SampleFile4.txt
1|  4Hello World!
2|
3|  4Hello TM Robot!
```

```
bool var_eof = File_NextEOF()                 // true      //没有打开文件以读取。
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)           //打开文件但不读取。
var_eof = File_NextEOF()                      // false
```

7.6 File_WriteBytes()

将数据内容纳入字节数组并写入文件。

语法 1

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

参数

string	文件路径	
?	要写入的值。支持整数、浮点数、布尔值、字符串和数组。 将以小端序转换值、以UTF8编码转换字符串。	
int	覆写文件还是附加至文件中。	
	0	覆写文件。若文件不存在，新建一个文件。
	1	附加至文件中。若文件不存在，新建一个文件。
int	要写入的值的起始索引（支持字符串和数组）	
	0 .. length-1	合法值
	< 0	非法值。起始索引将被设为0。
	>= length	非法值。起始索引将被设为0。
int	要写入的值的长度（支持字符串和数组）	
	<= 0	从起始索引开始写入，直至数据末尾。
	> 0	从起始索引开始写入特定长度的数据，最多至数据末尾。

返回值

bool	true	写入成功。
	false	写入失败。

语法 2

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

注

与语法 1 相同。“要写入的值的长度”参数被设为 0，始终写入至数据结束。

File_WriteBytes(string,?,int,int) => **File_WriteBytes**(string,?,int,int,0)

语法 3

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

注

与语法 1 相同。起始索引和“要写入的值的长度”参数被设为 0，始终写入至数据结束。

File_WriteBytes(string,?,int) => **File_WriteBytes**(string,?,int,0,0)

语法 4

```
bool File_WriteBytes(
    string,
    ?
)
```

注

与语法 1 相同。起始索引和“要写入的值的长度”参数被设为 0，始终覆写文件至数据结束。
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) //以 var_bb1 覆写文件

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) //以 var_bb2 覆写文件

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) //以 var_bb3 覆写文件

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) //将 var_bb1 附加至文件中

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) //将 var_bb2 附加至文件中

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) //将 var_bb1 从索引 3 开始附加至文件中，直至末尾。

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) //将 var_bb2 从索引 3 开始附加至文件中，附加内容长度为 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 为非法值//将 var_bb1 从索引 0 开始附加至文件中，直至末尾。

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)

//将 var_bb2 从索引 0 开始附加至文件中，附加内容长度为 100，即直至末尾。

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 //以小端序转换值。

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} //以小端序依次转换数组中的每个值。

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 //将 true 转换为 1、将 false 转换为 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" //将字符串转换为 UTF8 编码。

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()

将数据内容纳入字符串并写入文件。

语法 1

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

参数

string	文件路径	
?	要写入的值。支持整数、浮点数、布尔值、字符串和数组。	
int	覆写文件还是附加至文件中。	
	0	覆写文件。若文件不存在，新建一个文件。
	1	附加至文件中。若文件不存在，新建一个文件。
int	要写入的值的起始索引（支持字符串和数组）	
	0 .. length -1	合法值
	< 0	非法值。起始索引将被设为0。
	>= length	非法值。起始索引将被设为0。
int	要写入的值的长度（支持字符串和数组）	
	<= 0	从起始索引开始写入，直至数据末尾。
	> 0	从起始索引开始写入特定长度的数据，最多至数据末尾。

返回值

bool	true	写入成功。
	false	写入失败。

语法 2

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

注

与语法 1 相同。“要写入的值的长度”参数被设为 0，始终写入至数据结束。

File_WriteText(string,?,int,int) => **File_WriteText**(string,?,int,int,0)

语法 3

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

注

与语法 1 相同。起始索引和“要写入的值的长度”参数被设为 0，始终写入至数据结束。

File_WriteText(string,?,int) => **File_WriteText**(string,?,int,0,0)

语法 4

```
bool File_WriteText(
    string,
    ?
)
```

注

与语法 1 相同。起始索引和“要写入的值的长度”参数被设为 0，始终覆写文件至数据结束。
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)        //以"達明機器人\u0D0A"覆写文件
```

```
writeline.txt
1|  達明機器人
2|
```

```
var_flag = File_WriteLine("writeline.txt", var_s1)        //以"Hi TM Robot\u0D0A"覆写文件
```

```
writeline.txt
1|  Hi TM Robot
2|
```

```
var_flag = File_WriteLine("writeline.txt", var_s2, 1)     //将""達明機器人\u0D0A"附加至文件中
```

```
writeline.txt
1|  Hi TM Robot
2|  達明機器人
3|
```

```
var_flag = File_WriteLine("writeline.txt", var_s1, 1, 3) //将  "TM Robot\u0D0A"  附加至文件中的索引 3
```

```
writeline.txt
1|  Hi TM Robot
2|  達明機器人
3|  TM Robot
4|
```

```
? byte[] var_n2 = {100, 200}              //数组使用{, ,}格式。
File_WriteLine("writeline2.txt", var_n2, 1)
```

```
writeline2.txt
1|  {100,200}
2|
```

```
//对于其他格式，请使用 GetString()转换为字符串再写入。
? int var_n3 = 10000                      //将十进制值转换为字符串
File_WriteLine("writeline2.txt", var_n3, 1)
```

```
writeline2.txt
1|  {100,200}
2|  10000
3|
```

? float[] var_n6 = {1.23, 4.56, -7.89}

File_WriteLine("writeline2.txt", var_n6, 1)

writeline2.txt

```
1|  {100,200}
2|  10000
3|  {1.23,4.56,-7.89}
4|
```

? bool var_n9 = true

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

writeline2.txt

```
1|  {100,200}
2|  10000
3|  {1.23,4.56,-7.89}
4|  true
5|
```

? string var_n11 = "ABCDEFG"

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

writeline2.txt

```
1|  {100,200}
2|  10000
3|  {1.23,4.56,-7.89}
4|  true
5|  ABCDEFG
6|
```

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

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

writeline2.txt

```
1|  {100,200}
2|  10000
3|  {1.23,4.56,-7.89}
4|  true
5|  ABCDEFG
6|  {ABC,DEF,達明機器人}
7|
```

7.8 File_WriteLine()

将数据内容纳入以换行符（0x0D 0x0A）结尾的字符串并写入文件。

语法 1

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

参数

string	文件路径	
?	要写入的值。支持整数、浮点数、布尔值、字符串和数组。	
int	覆写文件还是附加至文件中。	
	0	覆写文件。若文件不存在，新建一个文件。
	1	附加至文件中。若文件不存在，新建一个文件。
int	要写入的值的起始索引（支持字符串和数组）	
	0 .. length -1	合法值
	< 0	非法值。起始索引将被设为0。
	>= length	非法值。起始索引将被设为0。
int	要写入的值的长度（支持字符串和数组）	
	<= 0	从起始索引开始写入，直至数据末尾。
	> 0	从起始索引开始写入特定长度的数据，最多至数据末尾。

返回值

bool	true	写入成功。
	false	写入失败。

语法 2

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

注

与语法 1 相同。“要写入的值的长度”参数被设为 0，始终写入至数据结束。

File_WriteLine(string,?,int,int) => **File_WriteLine**(string,?,int,int,0)

语法 3

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

注

与语法 1 相同。起始索引和“要写入的值的长度”参数被设为 0，始终写入至数据结束。

File_WriteLine(string,?,int) => **File_WriteLine**(string,?,int,0,0)

语法 4

```
bool File_WriteLine(
    string,
    ?
)
```

注

与语法 1 相同。起始索引和“要写入的值的长度”参数被设为 0，始终覆写文件至数据结束。
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)        //以"達明機器人\u0D0A"覆写文件
```

```
writeline.txt
1|  達明機器人
2|
```

```
var_flag = File_WriteLine("writeline.txt", var_s1)        //以"Hi TM Robot\u0D0A"覆写文件
```

```
writeline.txt
1|  Hi TM Robot
2|
```

```
var_flag = File_WriteLine("writeline.txt", var_s2, 1)     //将"達明機器人\u0D0A"附加至文件中
```

```
writeline.txt
1|  Hi TM Robot
2|  達明機器人
3|
```

```
var_flag = File_WriteLine("writeline.txt", var_s1, 1, 3) //将"TM Robot\u0D0A"从索引 3 开始至末尾附加
                                                          至文件中。
```

```
writeline.txt
1|  Hi TM Robot
2|  達明機器人
3|  TM Robot
4|
```

```
?   byte[] var_n2 = {100, 200}            //数组使用{, ,}格式。
    File_WriteLine("writeline2.txt", var_n2, 1)
```

```
writeline2.txt
1|  {100,200}
2|
```

```
//对于其他格式，请使用 GetString()转换为字符串再写入。
```

? int var_n3 = 10000 //将十进制值转换为字符串。

File_WriteLine("writeline2.txt", var_n3, 1)

```
writeline2.txt
1|  {100,200}
2|  10000
3|
```

? float[] var_n6 = {1.23, 4.56, -7.89}

File_WriteLine("writeline2.txt", var_n6, 1)

```
writeline2.txt
1|  {100,200}
2|  10000
3|  {1.23,4.56,-7.89}
4|
```

? bool var_n9 = true

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

```
writeline2.txt
1|  {100,200}
2|  10000
3|  {1.23,4.56,-7.89}
4|  true
5|
```

? string var_n11 = "ABCDEFG"

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

```
writeline2.txt
1|  {100,200}
2|  10000
3|  {1.23,4.56,-7.89}
4|  true
5|  ABCDEFG
6|
```

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

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

```
writeline2.txt
1|  {100,200}
2|  10000
3|  {1.23,4.56,-7.89}
4|  true
5|  ABCDEFG
6|  {ABC,DEF,達明機器人}
7|
```

7.9 File_WriteLines()

将数据内容纳入以换行符（0x0D 0x0A）结尾的字符串数组并写入文件。

语法 1

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

参数

string	文件路径	
?	要写入的值。支持整数、浮点数、布尔值、字符串和数组。	
int	覆写文件还是附加至文件中。	
	0	覆写文件。若文件不存在，新建一个文件。
	1	附加至文件中。若文件不存在，新建一个文件。
int	要写入的值的起始索引（支持字符串和数组）	
	0 .. length-1	合法值
	< 0	非法值。起始索引将被设为0。
	>= length	非法值。起始索引将被设为0。
int	要写入的值的长度（支持字符串和数组）	
	<= 0	从起始索引开始写入，直至数据末尾。
	> 0	从起始索引开始写入特定长度的数据，最多至数据末尾。

返回值

bool	true	写入成功。
	false	写入失败。

语法 2

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

注

与语法 1 相同。“要写入的值的长度”参数被设为 0，始终写入至数据结束。

File_WriteLines(string,?,int,int) => **File_WriteLines**(string,?,int,int,0)

语法 3

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

注

与语法 1 相同。起始索引和“要写入的值的长度”参数被设为 0，始终写入至数据结束。

File_WriteLines(string,?,int) => **File_WriteLines**(string,?,int,0,0)

语法 4

```
bool File_WriteLines(
    string,
    ?
)
```

注

与语法 1 相同。起始索引和“要写入的值的长度”参数被设为 0，始终覆写文件至数据结束。
File_WriteLines(string,?) => **File_WriteLines**(string,?,0,0,0)

* File_WriteText()：转换数据值并写入字符串，但不在字符串末尾添加换行符。
File_WriteLine()：转换数据值并写入字符串，并在字符串末尾添加换行符。
File_WriteLines()：转换数据值并写入字符串数组，并在字符串数组中的每个元素末尾添加换行符。

```
string var_ss1 = {"Hi TM Robot", "達明機器人"}

bool var_flag = false
var_flag = File_WriteLines("writelines.txt", var_ss1)        //以 ss1 覆写文件
```

writelines.txt

```
1|  Hi TM Robot
2|  達明機器人
3|
```

```
var_flag = File_WriteLines("writelines.txt", var_ss1, 1, 1) //将 ss1 从索引 1 开始至末尾附加至文件中。
```

writelines.txt

```
1|  Hi TM Robot
2|  達明機器人
3|  Hi TM Robot
4|
```

? `byte[] var_n2 = {100, 200}`
File_WriteLines("writelines2.txt", var_n2, 1)

writelines2.txt

```
1|  100
2|  200
3|
```

? `int var_n3 = 10000`
File_WriteLines("writelines2.txt", var_n3, 1)

writelines2.txt

```
1|  100
2|  200
3|  10000
4|
```

? float[] var_n6 = {1.23, 4.56, -7.89}

File_WriteLines("writelines2.txt", var_n6, 1)

writelines2.txt

```
1|  100
2|  200
3|  10000
4|  1.23
5|  4.56
6|  -7.89
7|
```

? string var_n11 = "ABCDEFG"

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

writelines2.txt

```
1|  100
2|  200
3|  10000
4|  1.23
5|  4.56
6|  -7.89
7|  ABCDEFG
8|
```

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

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

writelines2.txt

```
 1| 100
 2| 200
 3| 10000
 4| 1.23
 5| 4.56
 6| -7.89
 7| ABCDEFG
 8| ABC
 9| DEF
10| 達明機器人
11|
```

7.10 File_Exists()

检查文件路径是否可用。

语法 1

```
bool File_Exists(
    string
)
```

参数

string	文件路径	

返回值

bool	true	文件路径可用
	false	文件路径不可用
	*文件路径无效时返回 false（文件路径不可用）而非报错。	

注

```
bool var_exists = false
var_exists = File_Exists("sampleFile1.txt")        // true
var_exists = File_Exists("sampleFileX.txt")        // false    //文件路径不可用
var_exists = File_Exists("C:\SampleFile1.txt")     // false    //不支持绝对路径。
```

7.11 File_Length()

检查文件大小。

语法 1

```
int File_Length(
    string
)
```

参数

string	文件路径

返回值

int	以int32数据类型返回文件大小。文件大小上限为2147483647字节。
-1	文件路径不可用。
-2	超出文件大小上限。
	*文件路径无效时返回-1（文件路径不可用）而非报错。

注

```
Int var_len = 0
var_len = File_Length("sampleFile1.txt")          // 31
var_len = File_Length("sampleFileX.txt")          // -1  //文件路径不可用
var_len = File_Length("C:\SampleFile1.txt")       // -1  //不支持绝对路径。
```

7.12 File_Delete()

删除文件。

语法 1

```
bool File_Delete(
    string
    ...
)
```

参数

string	文件路径	
...	支持多个字符串。	

返回值

bool	true	删除成功。（或文件路径不可用或无效）
	false	删除失败。（无法删除被占用的文件）

语法 2

```
bool File_Delete(
    string[]
)
```

参数

string[]	文件路径

返回值

bool	true	删除成功。（或文件路径不可用或无效）
	false	删除失败。（无法删除被占用的文件）

注

```
bool var_flag = false
var_flag = File_Delete("sampleFile1.txt")          // true
var_flag = File_Delete("sampleFileX.txt")          // true      //文件路径不可用
var_flag = File_Delete("C:\SampleFile1.txt")       // true      //不支持绝对路径。
var_flag = File_Delete("sampleFile1.txt", "sampleFileX.txt")//支持多个文件路径。
var_flag = File_Delete("sampleFile1.txt", "sampleFileX.txt", "C:\SampleFile1.txt")
                                                   //支持多个文件路径。
string[] var_ss = {"sampleFile1.txt", "sampleFileX.txt", "C:\SampleFile1.txt""}
var_flag = File_Delete(var_ss)
var_flag = File_Delete(var_ss, "sampleFile2.txt") //错误。语法无效。
```

7.13 File_Copy()

复制文件。

语法 1

```
bool File_Copy(
    string,
    string,
    string
)
```

参数

string	源文件路径	
string	目标目录路径	
string	目标文件名。若为空，则默认使用与源文件路径相同的命名方式。	

返回值

bool	true	复制成功。
	false	复制失败。

*如目标路径中已存在同名文件，则覆写目标文件。

语法 2

```
bool File_Copy(
    string,
    string
)
```

注

与语法 1 相同。将目标文件名设为空字符串，使用与源文件路径相同的命名方式。

File_Copy(string,string) => **File_Copy**(string,string,"")

```
File_Copy("sampleFile1.txt", ".\TextFiles", "s1.txt") //将.\TextFiles\sampleFile1.txt 复制为.\TextFiles\s1.txt
File_Copy("sampleFile1.txt", ".\XmlFiles", "") //将.\TextFiles\sampleFile1.txt 复制为.\XmlFiles\sampleFile1.txt
File_Copy("sampleFile1.txt", "\USB\TMROBOT", "s1.txt") //将.\TextFiles\sampleFile1.txt 复制为 USB\s1.txt
File_Copy("sampleFile1.txt", "\USB\TMROBOT") //将.\TextFiles\sampleFile1.txt 复制为 USB\sampleFile1.txt
bool var_flag = false
var_flag = File_Copy("sampleFile1.txt", "C:\folder")    //错误。不支持绝对路径。
var_flag = File_Copy("sampleFile1.txt", ".")            //错误。文件路径中不含 TextFiles 或 XmlFiles。
```

7.14 File_CopyImage()

复制保存的视觉文件。

语法 1

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

参数

string	保存的视觉源文件的路径	
string	目标目录路径	
	可将图像复制并保存到外部路径，如外部设备路径目录或远程路径目录	
	不可将图像复制并保存到".\TextFiles"或".\XmlFiles"等本地路径，尝试时将报告错误。	
string	目标文件名。	
	若名称为空字符串，默认文件名将与源文件路径中的相同。	
int	复制选项	
	0	复制失败时不返回错误。不保留源图像的相对目录。（默认值）
	1	复制失败时不返回错误。保留源图像的相对目录。
	2	复制失败时返回错误。不保留源图像的相对目录。
	3	复制失败时返回错误。保留源图像的相对目录。

返回值

bool	true	复制成功。
	false	复制失败。
	*若指定路径中已存在同名文件，则覆写指定文件。	

语法 2

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

注

与语法 1 相同。将复制选项设为 0。复制失败时不返回错误。不保留源图像的相对目录。

File_CopyImage(string,string,string) => **File_CopyImage**(string,string,string,0)

语法 3

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

注

与语法 1 相同。将目标文件名设为空字符串，目标文件名将与源文件路径中的相同。将复制选项设为 0。复制失败时不返回错误。不保留源图像的相对目录。

File_CopyImage(string,string) => **File_CopyImage**(string,string,"",0)

语法 4

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

注

与语法 1 相同。将目标文件名设为空字符串，目标文件名将与源文件路径中的相同。
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 //不支持复制到本地目录。
var_flag = File_CopyImage(Job1_ImagePath_TM, ".\XmlFiles", "1.png")
// false //不支持复制到本地目录。
var_flag = File_CopyImage(Job1_ImagePath_TM, ".\XmlFiles", "1.png", 2)
// false //不支持复制到本地目录。
var_flag = File_CopyImage(Job1_ImagePath_TM, "\USB\TMROBOT", "1.png")
// true //将 Job1_ImagePath_TM（仅限于视觉 AOI 的变量）复制为 USB\1.png
var_flag = File_CopyImage(Job1_ImagePath_TM, "\USB\TMROBOT", "1.png", 3)
// true //将 Job1_ImagePath_TM（仅限于视觉 AOI 的变量）复制为 USB\ProjectName\Job1\Date\source\1.png
//保留要保存的图像的目录。
var_flag = File_CopyImage(Job1_ImagePath_TM, "\USB\TMROBOT")
// true //将 Job1_ImagePath_TM（仅限于视觉 AOI 的变量）复制为 USB\15-16-12_423.png
//保留要保存的图像的文件名。
var_flag = File_CopyImage(Job1_ImagePath_TM, "\USB\TMROBOT", 3)
// true //将 Job1_ImagePath_TM（仅限于视觉 AOI 的变量）复制为 USB\ProjectName\Job1\Date\source\15-16-12_423.png
//保留要保存的图像的目录和文件名。
```

7.15 File_GetImage()

在 TMvision 中执行任务作业后，系统会在图像存储路径缓冲区中保留源图像存储路径和结果图像存储路径。用户可使用此函数获取缓冲区内存储的文件路径并将其复制到外部。该函数优先从最早的图像存储路径获取图像（先进先出），随后自动删除该路径。

缓冲区内最多可存储 60 个图像路径。将新的图像存储路径添加至缓冲区内时，若容量不足，将自动从缓冲区内删除最早的图像存储路径，然后自动将新的图像存储路径添加至缓冲区内。

语法 1

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

参数

int	获取图像存储路径时的等待超时设置	
	< 0	无限期等待，直至成功获取图像存储路径。（默认值）
	= 0	获取一次。
	> 0	若使用等待，进程将停留在此命令，直至成功获取图像存储路径或等待超时，然后继续执行下面的命令

返回值

string[]	两个路径均为空字符串代表缓冲区内没有任何图像存储路径。	
	[0]	源图像路径
	[1]	结果图像路径

语法 2

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

注

与语法 1 相同。将等待超时参数设为-1，将无限期等待，直至成功获取图像存储路径。

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

```
string[] var_image = File_GetImage()        //等待，直至成功获取图像存储路径。
bool var_flag1 = File_CopyImage(var_image[0], "\USB\TMROBOT")
bool var_flag2 = File_CopyImage(var_image[1], "\USB\TMROBOT")
```

7.16 File_Replace()

以特定字符串替换并覆写文件内的字符串。

语法 1

```
bool File_Replace(
    string,
    string,
    string
)
```

参数

string	文件路径	
string	待替换字符串	
string	替换用字符串	

返回值

bool	true	成功	1.待替换字符串为空字符串。 2.待替换字符串不存在。 3.已找到并覆写文件内的待替换字符串。
	false	失败	

注

```
.\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()

以字符串模式读取文件，并获取字符串中的子字符串。

语法 1

```
string File_GetToken(
    string,
    string,
    string,
    int,
    int
)
```

参数

string		文件路径
string		要获取的字符串的前缀
string		要获取的字符串的后缀
int		要获取第几个匹配的子字符串
	>=1	获取第n个匹配的子字符串
	-1	获取最后一个匹配的子字符串
int		删除选项
	0	第一个匹配项无需位于输入的字符串的开头，且不删除前缀和后缀。（默认值）
	1	第一个匹配项无需位于输入的字符串的开头，且将删除前缀和后缀。
	2	第一个匹配项必须位于输入的字符串的开头，且不删除前缀和后缀。
	3	第一个匹配项必须位于输入的字符串的开头，且将删除前缀和后缀。

返回值

string	返回的获取的字符串。 若前缀和后缀均为空字符串，返回文件中的字符串内容。 若匹配次数<=0，返回空字符串。 若删除选项为2或3，则获取的第一个匹配项必须位于输入的字符串的开头，否则将返回空字符串。

语法 2

```
string File_GetToken(
    string,
    string,
    string,
    int
)
```

注

与语法 1 相同。删除选项默认为 0，不删除前缀和后缀。

File_GetToken(string,string,string,int) => **File_GetToken**(string,string,string,int,0)

语法 3

```
string File_GetToken(
    string,
    string,
    string
)
```

注

与语法 1 相同。获取第 1 个匹配项，删除选项默认为 0，不删除前缀和后缀。

File_GetToken(string,string,string) => **File_GetToken**(string,string,string,1,0)

.\TextFiles\SampleFile7.txt

```
1|  $Hello World!
2|  $Hello TM Robot!
3|  $Hi TM Robot!$
```

```
string var_n = "SampleFile7.txt"
string var_s = ""
var_s = File_GetToken(var_n, "", "", 0)              // "$Hello World!\u0D0A$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, "$", NewLine, 1, 1) // "Hello World!"        //删除前缀和后缀
var_s = File_GetToken(var_n, Ctrl("\r\n"), "$", 1)   // "\u0D0A$"
var_s = File_GetToken(var_n, newline, "$", 2)        // "\u0D0A$"
var_s = File_GetToken(var_n, NewLine, "$", 1, 1) // ""
```

*\u0D0A 代表换行符，而非字符串值。

.\TextFiles\SampleFile9.txt

```
1|  #Hello World!
2|  $Hello TM Robot!
3|  $Hi TM Robot!$
```

```
string var_n = "SampleFile9.txt"
string var_s = ""
var_s = File_GetToken(var_n, "", "")              // "#Hello World!\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, "$", newline, 1, 2)      // ""
                                                      //$不位于文件开头。返回空字符串。
var_s = File_GetToken(var_n, "$", "", 1, 2)           // ""
                                                      //$不位于文件开头。返回空字符串。
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)         // ""                  //超出匹配次数
var_s = File_GetToken(var_n, "", "#", 1, 2)           // "#"
```

语法 4

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

参数

string		文件路径
byte[]		要获取的字节数组中的字符串的前缀
byte[]		要获取的字节数组中的字符串的后缀
int		要获取第几个匹配的子字符串
	>=1	获取第n个匹配的子字符串
	-1	获取最后一个匹配的子字符串
int		删除选项
	0	第一个匹配项无需位于输入的字符串的开头，且不删除前缀和后缀。（默认值）
	1	第一个匹配项无需位于输入的字符串的开头，且将删除前缀和后缀。
	2	第一个匹配项必须位于输入的字符串的开头，且不删除前缀和后缀。
	3	第一个匹配项必须位于输入的字符串的开头，且将删除前缀和后缀。

返回值

string　返回的获取的字符串。

若前缀和后缀均为空字符串，返回文件中的字符串内容。

若匹配次数<=0，返回空字符串。

若删除选项为2或3，则获取的第一个匹配项必须位于输入的字符串的开头，否则将返回空字符串。

语法 5

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

注

与语法 4 相同。删除选项默认为 0，不删除前缀和后缀。

File_GetToken(string,byte[],byte[],int) => **File_GetToken**(string,byte[],byte[],int,0)

语法 6

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

注

与语法 4 相同。获取第 1 个匹配项，删除选项默认为 0，不删除前缀和后缀。

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 为$，0x0D 0x0A 为\u0D0A
var_s = File_GetToken(var_n, bb0, bb0, 0)       // "$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"          //删除前缀和后缀
```

```
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 代表换行符，而非字符串值。

7.18 File_GetAllTokens()

以字符串模式读取文件，并获取所有符合条件的子字符串。

语法 1

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

参数

string		文件路径
string		要获取的字符串的前缀
string		要获取的字符串的后缀
int		删除选项
	0	第一个匹配项无需位于输入的字符串的开头，且不删除前缀和后缀。（默认值）
	1	第一个匹配项无需位于输入的字符串的开头，且将删除前缀和后缀。
	2	第一个匹配项必须位于输入的字符串的开头，且不删除前缀和后缀。
	3	第一个匹配项必须位于输入的字符串的开头，且将删除前缀和后缀。

返回值

string[]	以数组形式返回符合条件的字符串。 若前缀和后缀均为空字符串，以字符串数组形式返回文件中的字符串内容。 若删除选项为2或3，则获取的第一个匹配项必须位于输入的字符串的开头，否则将返回空字符串。

语法 2

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

注

与语法 1 相同。删除选项默认为 0，不删除前缀和后缀。

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

```
1|  #Hello World!
2|  $Hello TM Robot!
3|  $Hi TM Robot!$
```

```
string var_n = "SampleFile9.txt"
string[] var_ss = {}
var_ss = File_GetAllTokens(var_n, "", "")          // {"#Hello World!\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. 串行通讯函数

8.1 SerialPort 类

可通过使用 SerialPort 类并声明变量创建 COM 端口设备。变量名将成为设备名称。

构造 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`

参数

`string`	连接说明		
`int`	位/秒，波特率		
`string`	奇偶校验	"none"、"odd"、"even"、"mark"、"space"（默认为"none"）	
`int`	数据位	5、6、7、8	（默认为8）
`float`	停止位	1、1.5、2	（默认为1）
`int`	读/写超时时间，单位为毫秒	0~10000	（默认为10000 ms）
`bool`	DTR/DSR	true、false	（默认为false）
`bool`	RTS/CTS	true、false	（默认为false）
`bool`	XON/XOFF	true、false	（默认为false）

注

```
SerialPort spd_c1 = "COM2",115200
  //构造一个设备，其波特率为 115200
SerialPort spd_c2 = "COM2",115200,"none",8,1
  //构造一个设备，其波特率为 115200、奇偶校验为 none、数据位为 8、停止位为 1
SerialPort spd_c3 = "COM2",115200,"none",8,1,10000
  //构造一个设备，其波特率为 115200、奇偶校验为 none、数据位为 8、停止位为 1、超时时间为 10000 ms
```

- 在流程项目中，将从串行端口列表创建设备并主动连接至该设备。
- 在脚本项目中，根据语法内容创建设备后不会连接至设备。需要使用开启设备函数连接。
- 无论使用设备读取还是写入，都会要求确认已连接至设备。

8.2 com_open()

开启一个串行端口设备。

语法 1

```
bool com_open(
    string
)
```

参数

`string` 串行端口设备名称

返回值

`bool`	`True`	开启成功。
	`False`	开启失败。（项目报告错误。）

注

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

8.3 com_close()

关闭一个串行端口设备。

语法 1

```
bool com_close(
    string
)
```

参数

`string` 串行端口设备名称

返回值

`bool`	`True`	关闭成功。
	`False`	关闭失败。

注

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

项目从开始节点开始运行时，会开启串行端口以连接，并持续从串行端口接收数据。对于接收至接收缓冲区内的数据，用户可使用 com_read 函数读取缓冲区内的数据。

开启串行端口后，将持续从串行端口接收数据，并将数据置于接收缓冲区内。用户可使用 com_read 或相关函数从缓冲区内获取数据。项目停止运行时，会关闭已开启的串行端口并清空接收缓冲区。

接收缓冲区的容量有限。若数据进入缓冲区时缓冲区空间已满，将自动删除最早的数据，使最新数据进入缓冲区。

8.4 com_read()

读取串行端口接收缓冲区内的数据并以字节数组形式返回。

语法 1

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

参数

`string` 串行端口上的设备的名称

返回值

`byte[]` 返回所有数据内容。若内容为空，将返回byte[0]。

注

```
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 = {}
```

*该函数会读取接收缓冲区内的所有数据，然后清空接收缓冲区。

语法 2

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

参数

`string` 串行端口上的设备的名称

`int` 要读取的元素数量（基于字节数组长度）

- `<= 0` 读取所有元素
- `> 0` 读取特定数量的元素（读取到特定数量后方可使用数据。）

`int` 读取时长，单位为毫秒

- `<= 0` 仅读取一次
- `> 0` 多次读取，直至取得数据或时间耗尽。

返回值

`byte[]` 以字节数组形式返回的特定数量的元素。若元素不足，将返回byte[0]。

语法 3

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

注

该语法与语法 2 相同。读取时长参数默认为 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[] = {}

// 接收缓冲区内的元素不足 100 个，故元素不足，返回 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} //读取所有元素

// ReceivedBuffer = {}

value = **com_read**("spd", 4, 100)

// value byte[] = {} //接收缓冲区内的元素不足 4 个，故元素不足，返回 byte[0]。

// 但读取时长被设为 100 ms，因此进程将停留在函数中，直至取得数据或时间耗尽，然后退出函数。

// ReceivedBuffer = {0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38} //假设 50 ms 后接收到数据，

// value byte[] = {0x31,0x32,0x33,0x34} //读取 4 个元素，然后退出函数。

// ReceivedBuffer = {0x35,0x36,0x37,0x38}

语法 4

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

参数

`string` 串行端口上的设备的名称

`byte[] or string`
读取时的前缀条件。若输入 byte[0]或空字符串""，则无前缀条件。

`byte[] or string`
读取时的后缀条件。若输入 byte[0]或空字符串""，则无后缀条件。

`int` 删除选项

- `0` 第一个匹配项无需位于输入的字符串的开头，且不删除前缀和后缀。（默认值）
- `1` 第一个匹配项无需位于输入的字符串的开头，且将删除前缀和后缀。
- `2` 第一个匹配项必须位于输入的字符串的开头，且不删除前缀和后缀。
- `3` 第一个匹配项必须位于输入的字符串的开头，且将删除前缀和后缀。

`int` 读取时长，单位为毫秒

- `<= 0` 仅读取一次
- `> 0` 多次读取，直至取得数据或时间耗尽。

返回值

`byte[]` 以字节数组形式返回满足前缀和后缀条件的第一个匹配项。
将获取所有内容匹配的数据中的第一项，然后保留其余的不获取。
若无匹配项，将返回byte[0]。

语法 5

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

注

该语法与语法 4 相同。读取时长参数默认为 0。

语法 6

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

注

该语法与语法 4 相同。默认不删除读取内容中的前缀和后缀，且读取时长为 0。

```
ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
value = com_read("spd", "He", newline)              //前缀为"He"、后缀为\u0D0A
  // value byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A}// Hello,\u0D0A
  //vReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}  //获取第一个匹配项并保留其余的
value = com_read("spd", "", newline, 1)             //前缀为""、后缀为\u0D0A。删除前缀和后缀。
  // value byte[] = {0x57,0x6F,0x72,0x6C,0x64}        // World
  // ReceivedBuffer = {}
value = com_read("spd", "", newline, 1, 100)        //前缀为""、后缀为\u0D0A。删除前缀和后缀。读取时
                                                    长为 100 ms。
  //   value byte[] = {}
  //无匹配项可读取。读取 byte[0]。等待 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)              //前缀为"lo"、后缀为\u0D0A
  // value byte[] = {0x6C,0x6F,0x2C,0x0D,0x0A}      // lo,\u0D0A
  //第一个匹配项前的数据{0x48,0x65,0x6C}将被删除。
  // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
  //获取第一个匹配项并保留其余的
byte[] bb = {}
value = com_read("spd", bb, newline)                //前缀为 byte[0]、后缀为\u0D0A
  //   value byte[] = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}      // World\u0D0A
  // ReceivedBuffer = {}
value = com_read("spd", bb, newline, 0, 100)        //前缀为 byte[0]、后缀为\u0D0A，读取时长为 100 ms
  //   value byte[] = {}
  //无匹配项可读取。读取 byte[0]。等待 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,                     // $TM
                  0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}      // #Robot
value = com_read("spd", "#", newline, 2)          //前缀为"#"、后缀为\u0D0A
  // value byte[] = {}                            //#不位于开头。返回空数组。
  // ReceivedBuffer = {0x24,0x48,0x65,0x6C,0x6C,0x6F,0x0D,0x0A,0x23,0x48,0x69,0x0D,0x0A,
                       0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}
value = com_read("spd", "$", newline, 2)          //前缀为"$"、后缀为\u0D0A
  // value byte[] = {0x24,0x48,0x65,0x6C,0x6C,0x6F,0x0D,0x0A} //第一个匹配项必须位于开头才能被获取。
  // ReceivedBuffer = {0x23,0x48,0x69,0x0D,0x0A,
                       0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}
value = com_read("spd", "#", newline, 2)          //前缀为"#"、后缀为\u0D0A
  // value byte[] = {0x23,0x48,0x69,0x0D,0x0A}     //第一个匹配项必须位于开头才能被获取。
  // ReceivedBuffer = {0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}
value = com_read("spd", "$", newline, 3)          //前缀为"$"、后缀为\u0D0A
  // value byte[] = {0x54,0x4D}
  // ReceivedBuffer = {0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}
value = com_read("spd", "#", newline, 3)          //前缀为"#"、后缀为\u0D0A
  // value byte[] = {0x52,0x6F,0x62,0x6F,0x74}
  // ReceivedBuffer = {}
```

语法 7

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

参数

`string` 串行端口上的设备的名称

`byte[] or string`
读取时的后缀条件。若输入 byte[0]或空字符串""，则无后缀条件。

`int` 删除选项

0	第一个匹配项无需位于输入的字符串的开头，且不删除前缀和后缀。（默认值）
1	第一个匹配项无需位于输入的字符串的开头，且将删除前缀和后缀。
2	第一个匹配项必须位于输入的字符串的开头，且不删除前缀和后缀。
3	第一个匹配项必须位于输入的字符串的开头，且将删除前缀和后缀。

`int` 读取时长，单位为毫秒

<= 0	仅读取一次
> 0	多次读取，直至取得数据或时间耗尽。

*读取时无前缀条件。

返回值

`byte[]` 以字节数组形式返回满足前缀和后缀条件的第一个匹配项。
将获取所有内容匹配的数据中的第一项，然后保留其余的不获取。
若无匹配项，将返回byte[0]。

语法 8

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

注

该语法与语法 7 相同。读取时长参数默认为 0。

语法 9

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

注

该语法与语法 7 相同。默认不删除读取内容中的前缀和后缀，且读取时长为 0。

注

```
ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
value = com_read("spd", newline)//后缀为\u0D0A
  //   value byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A}      // Hello,\u0D0A
  // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
  //获取第一个匹配项并保留其余的
value = com_read("spd", newline)//后缀为\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)          //后缀为\u0D0A
  //   value byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C}              // Hello,
  // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
  //获取第一个匹配项并保留其余的
value = com_read("spd", newline, 1)          //后缀为\u0D0A
  //   value byte[] = {0x57,0x6F,0x72,0x6C,0x64}              // World
  // ReceivedBuffer = {}
value = com_read("spd", newline, 1, 100)   //后缀为\u0D0A，读取时长为 100 ms
  //   value byte[] = {}                                     //无匹配项可读取。读取 byte[0]。等待 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()

读取串行端口缓冲区内的数据，然后将数据转换为 UTF8 字符串。

语法 1

```
string com_read_string(
    string
)
```

参数

`string` 串行端口上的设备的名称

返回值

`string` 返回所有数据内容。若内容为空，将返回空字符串。

语法 2

```
string com_read_string(
    string,
    int,
    int
)
```

参数

`string` 串行端口上的设备的名称

`int` 要读取的字符数量（基于字符串的字符数量）

- `<= 0` 读取所有字符
- `> 0` 读取特定数量的字符（读取到特定数量后方可使用数据。）

`int` 读取时长，单位为毫秒

- `<= 0` 仅读取一次
- `> 0` 多次读取，直至取得数据或时间耗尽。

返回值

`string` 以字符串形式返回特定数量的字符。若字符不足，将返回空字符串。

语法 3

```
string com_read_string(
    string,
    int
)
```

注

该语法与语法 2 相同。读取时长参数默认为 0。

语法 4

```
string com_read_string(
    string,
    byte[] or string,
    byte[] or string,
    int,
    int
)
```

参数

`string` 串行端口上的设备的名称

`byte[] or string`

读取时的前缀条件。若输入 byte[0]或空字符串""，则无前缀条件。

`byte[] or string`

读取时的后缀条件。若输入 byte[0]或空字符串""，则无后缀条件。

int 删除选项

0 第一个匹配项无需位于输入的字符串的开头，且不删除前缀和后缀。（默认值）

1 第一个匹配项无需位于输入的字符串的开头，且将删除前缀和后缀。

2 第一个匹配项必须位于输入的字符串的开头，且不删除前缀和后缀。

3 第一个匹配项必须位于输入的字符串的开头，且将删除前缀和后缀。

int 读取时长，单位为毫秒

<= 0 仅读取一次

> 0 多次读取，直至取得数据或时间耗尽。

返回值

string 将获取所有内容匹配的数据中的第一项

将获取所有内容匹配的数据中的第一项，然后保留其余的不获取。

若无匹配项，将返回空字符串。

语法 5

```
string com_read_string(
    string,
    byte[] or string,
    byte[] or string,
    int
)
```

注

该语法与语法 4 相同。读取时长参数默认为 0。

语法 6

```
string com_read_string(
    string,
    byte[] or string,
    byte[] or string
)
```

注

该语法与语法 4 相同。默认不删除读取内容中的前缀和后缀，且读取时长为 0。

语法 7

```
string com_read_string(
    string,
    byte[] or string,
    int,
    int
)
```

参数

string 串行端口上的设备的名称

byte[] or string

读取时的后缀条件。若输入 byte[0]或空字符串""，则无后缀条件。

int 删除选项

0 第一个匹配项无需位于输入的字符串的开头，且不删除前缀和后缀。（默认值）

1 第一个匹配项无需位于输入的字符串的开头，且将删除前缀和后缀。

2 第一个匹配项必须位于输入的字符串的开头，且不删除前缀和后缀。

3 第一个匹配项必须位于输入的字符串的开头，且将删除前缀和后缀。

int 读取时长，单位为毫秒

<= 0 仅读取一次

> 0　　多次读取，直至取得数据或时间耗尽。

*读取时无前缀条件。

返回值

string　以字符串形式返回满足前缀和后缀条件的第一个匹配项。
将获取所有内容匹配的数据中的第一项，然后保留其余的不获取。
若无匹配项，将返回空字符串。

语法 8

```
string com_read_string(
    string,
    byte[] or string,
    int
)
```

注

该语法与语法 7 相同。读取时长参数默认为 0。

语法 9

```
string com_read_string(
    string,
    byte[] or string
)
```

注

该语法与语法 7 相同。默认不删除读取内容中的前缀和后缀，且读取时长为 0。

注

```
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}
  //基于字符串长度，获取 4 个字符。
  // var _ReceivedBuffer = {0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA}
value = com_read_string("spd", 5, 100)
  //   value string = ""
  //基于字符串长度，接收缓冲区内的字符不足 5 个，故字符不足。等待 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)  //前缀为"He"、后缀为\u0D0A
  // value string = "Hello,\u0D0A"
  // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}   //获取第一个匹配项并保留其余的
value = com_read_string("spd", "", newline, 1)
  //前缀为""、后缀为\u0D0A。删除前缀和后缀。
```

```
  //    value string = "World"
  // ReceivedBuffer = {}
value = com_read_string("spd", "", newline, 1, 100)
  //    value string = ""              //无匹配项可读取。读取空字符串。等待 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)    //前缀为"lo"、后缀为\u0D0A
  //    value string = "lo,\u0D0A"
  //第一个匹配项前的数据"Hel"将被删除。
  // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
  //获取第一个匹配项并保留其余的
value = com_read_string("spd", newline, 1)       //后缀为\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)       //前缀为"#"、后缀为\u0D0A
  // value string = ""                                //#不位于开头。返回空数组。
  // 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", "$", newline, 2)       //前缀为"$"、后缀为\u0D0A
  // value string = "$Hello\u0D0A"                    //第一个匹配项必须位于开头才能被获取。
  // 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)       //前缀为"#"、后缀为\u0D0A
  // value string = "#Hi\u0D0A"                       //第一个匹配项必须位于开头才能被获取。
  // ReceivedBuffer = {0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

value = com_read_string("spd", "$", newline, 3)       //前缀为"$"、后缀为\u0D0A
  // value string = "TM"
  // ReceivedBuffer = {0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}
value = com_read_string("spd", "#", newline, 3)       //前缀为"#"、后缀为\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)               //后缀为\u0D0A
  //    value string = "Hello,\u0D0A"
  // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64}      //获取第一个匹配项并保留其余的
value = com_read_string("spd", newline, 0)            //后缀为\u0D0A
  //    value string = ""                            //无匹配项可读取。读取空字符串。
  // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64}
value = com_read_string("spd", newline, 1, 100)       //后缀为\u0D0A
  //    value string = ""
```

```
//无匹配项可读取。读取空字符串。等待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}          //获取第一个匹配项并保留其余的
value = com_read_string("spd", newline, 2)              //后缀为\u0D0A
// value string = "123\u0D0A"
// ReceivedBuffer = {}
```

8.6 com_write()

将数据写入串行端口

语法 1

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

参数

string	串行端口上的设备的名称		
?	要写入的值。支持的类型：int、float、bool、string 和 array。 将以小端序转换数值、以 UTF8 编码转换字符串值。		
int	要写入的值的起始索引（支持字符串和数组）		
	0 .. length-1	合法值	
	< 0	非法值。起始索引将被设为0。	
	>= length	非法值。起始索引将被设为0。	
int	要写入的值的长度（支持字符串和数组）		
	<= 0	从起始索引开始写入，直至数据末尾。	
	> 0	从起始索引开始写入特定长度的数据，最多至数据末尾。	

返回值

bool	True	写入成功	
	False	写入失败	1.要写入的值为空字符串或空数组。 2.无法正确发送至串行端口。

语法 2

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

参数

string	串行端口上的设备的名称		
?	要写入的值。支持的类型：int、float、bool、string 和 array。 将以小端序转换数值、以UTF8编码转换字符串值。		
int	数据起始索引，将从此处开始写入。（适用于字符串和数组）		
	0	字符串的长度 - 1	合法值
	< 0		非法值，起始索引将为0。
	>=	字符串的长度	非法值，起始索引将为0。
int	要写入的数据长度。（适用于字符串和数组）		
	<= 0		从起始索引到数据末尾
	> 0		从起始索引开始，写入特定长度的数据，最多至数据末尾。

返回值

bool	True	写入成功

False	写入失败	1.要写入的值为空字符串或空数组。 2.无法正确发送至串行端口。

语法 3

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

注

该语法与语法 2 相同。要写入的数据长度参数默认为 0。

```
flag = com_write("spd", 100)              //写入 0x64
flag = com_write("spd", 1000)             //写入 0xE8 0x03 0x00 0x00（int，小端序）
flag = com_write("spd", (float)1.234)     //写入 0xB6 0xF3 0x9D 0x3F（float，小端序）
flag = com_write("spd", (double)123.456)
  //写入0x77 0xBE 0x9F 0x1A 0x2F 0xDD 0x5E 0x40（double，小端序）
flag = com_write("spd", "Hello, World"+newline)
  //写入0x48 0x65 0x6C 0x6C 0x6F 0x2C 0x20 0x57 0x6F 0x72 0x6C 0x64 0x0D 0x0A（string，UTF8）
flag = com_write("spd", 1000, 1, 2)                  //值、起始索引和长度无效
  //写入 0xE8 0x03 0x00 0x00 （int，小端序）

byte[] bb = {100, 200}
flag = com_write("spd", bb)               //写入 0x64 0xC8
flag = com_write("spd", bb, 1, 1)         //写入 0xC8
//数组。从索引 1 开始获取 1 个元素。 [1]=200
flag = com_write("spd", bb, -1, 1)        //写入 0x64
//数组。从索引 0 开始获取 1 个元素。[0]=100
flag = com_write("spd", "達明機器人", 2)
//字符串。从索引 2 开始获取，直至索引末尾。 "機器人"
//写入 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA （string，UTF8）
string[] ss = {"TM", "", "達明機器人" }
flag = com_write("spd", ss)
  //写入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)))
  //写入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)
  //数组。从索引2开始获取100个元素（直至末尾）。 [2]=達明機器人
  //写入0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA
```

8.7 com_writeline()

在数据末尾自动添加换行符 0x0D 0x0A 并将数据写入串行端口

语法 1

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

参数

string	串行端口上的设备的名称	
?	要写入的值。支持的类型：int、float、bool、string 和 array。 将以小端序转换数值、以 UTF8 编码转换字符串值。	
int	要写入的值的起始索引（支持字符串和数组）	
	0 .. length-1	合法值
	< 0	非法值。起始索引将被设为0。
	>= length	非法值。起始索引将被设为0。
int	要写入的值的长度（支持字符串和数组）	
	<= 0	从起始索引开始写入，直至数据末尾。
	> 0	从起始索引开始写入特定长度的数据，最多至数据末尾。

返回值

bool	True	写入成功	
	False	写入失败	1.要写入的值为空字符串或空数组。 2.无法正确发送至串行端口。

语法 2

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

注

该语法与语法 1 相同。要写入的数据长度参数默认为 0。

语法 3

```
bool com_writeline(
    string,
    ?,
)
```

注

该语法与语法 1 相同。要写入的数据的起始索引默认为 0。要写入的数据长度参数默认为 0。

```
flag = com_writeline("spd", 100)              //写入 0x64 0x0D 0x0A
flag = com_writeline("spd", 1000)             //写入 0xE8 0x03 0x00 0x00 0x0D 0x0A（int，小端序）
flag = com_writeline("spd", (float)1.234)     //写入 0xB6 0xF3 0x9D 0x3F 0x0D 0x0A（float，小端序）
flag = com_writeline("spd", (double)123.456)
    //写入0x77 0xBE 0x9F 0x1A 0x2F 0xDD 0x5E 0x40 0x0D 0x0A（double，小端序）
```

```
flag = com_write("spd", "Hello, World"+newline)
    //写入0x48 0x65 0x6C 0x6C 0x6F 0x2C 0x20 0x57 0x6F 0x72 0x6C 0x64 0x0D 0x0A（string，UTF8）
flag = com_writeline("spd", "Hello, World")
    //写入0x48 0x65 0x6C 0x6C 0x6F 0x2C 0x20 0x57 0x6F 0x72 0x6C 0x64 0x0D 0x0A (string, UTF8)
flag = com_writeline("spd", 1000, 1, 2)      //值、起始索引和长度无效
    //写入 0xE8 0x03 0x00 0x00 0x0D 0x0A（int，小端序）

byte[] bb = {100, 200}
flag = com_writeline("spd", bb)              //写入 0x64 0xC8 0x0D 0x0A
flag = com_writeline("spd", bb, 1, 1)        //写入 0xC8 0x0D 0x0A
                                             //数组。从索引 1 开始获取 1 个元素。 [1]=200
flag = com_writeline("spd", bb, -1, 1)       //写入 0x64 0x0D 0x0A
                                             //数组。从索引 0 开始获取 1 个元素。[0]=100
flag = com_writeline("spd", "達明機器人", 2)
    //字符串。从索引 2 开始获取，直至索引末尾。 "機器人"
    //写入 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)
  //写入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)))
  //写入0x54 0x4D 0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA 0x0D 0x0A
flag = com_writeline("spd", ss)
  //写入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)
    //数组。从索引2开始获取100个元素（直至末尾）。 [2]=達明機器人
    //写入 0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA 0x0D 0x0A
```

9. Socket 函数

9.1 Socket 类

可通过使用 Socket 类并声明变量创建 TCP/IP 通信设备。变量名将成为设备名称。

构造

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

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

参数

`string`	远程主机的IP地址
`int`	远程主机的连接端口
`int`	读/写超时时间，单位为毫秒　0~10000　（默认为10000 ms）

注

```
Socket ntd_d1 = "192.168.1.10", 12345
  //构造一个设备，其IP为192.168.1.10、端口为12345、超时时间为10000 ms

Socket ntd_d2 = "192.168.1.11", 9999, 8000
  //构造一个设备，其IP为192.168.1.10、端口为9999、超时时间为8000 ms
```

- 在流程项目中，将从网络设备列表创建设备并将其开启。
- 在脚本项目中，根据语法内容创建设备后不会连接至设备。需要使用开启设备函数连接。
- 使用设备读取或写入时，将确认是否需要连接至设备。

9.2 socket_open()

开启一个 TCP/IP 设备。

语法 1

```
bool socket_open(
    string
)
```

参数

`string` TCP/IP 设备名称

返回值

`bool`	`True`	开启成功。
	`False`	开启失败。

注

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

9.3 socket_close()

关闭一个 TCP/IP 设备。

语法 1

```
bool socket_close(
    string
)
```

参数

`string` TCP/IP 设备名称

返回值

`bool`	`True`	开启成功。
	`False`	开启失败。

注

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

流程项目随着开始节点启动开始运行时，会启动 TCP/IP Socket Client 以连接至特定 IP 和端口。但是，对于脚本项目，用户必须使用 socket_open 函数开启指定的设备以连接。

连接至 TCP/IP 设备后，系统将持续接收连接数据，并将接收到的数据置于接收缓冲区内。可使用 Socket_read 等相应函数读取数据。项目停止运行时，将关闭现有的 TCP/IP Socket 连接，并清空接收缓冲区。

接收缓冲区的容量有限。若数据进入时接收缓冲区的容量不足，将自动删除最早的数据、添加最新的数据。

9.4 socket_read()

读取接收缓冲区内的数据并以字节数组形式返回。

语法 1

```
byte[] socket_read(
    string
)
```

参数

`string` 网络设备名称

返回值

`byte[]` 返回接收缓冲区内的所有数据。若缓冲区为空，将返回byte[0]。

注

```
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 = {}
```

*该函数会读取接收缓冲区内的所有数据，然后清空缓冲区。

语法 2

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

参数

`string` 网络设备名称

`int` 获取固定量的字节（基于字节数组的长度）

- `<= 0` 获取全部
- `> 0` 获取特定量（数据需达到特定量才能被获取）

`int` 读取时间（单位为毫秒）

- `<= 0` 读取一次
- `> 0` 多次读取，直至取得数据或时间耗尽。

返回值

`byte[]` 以字节数组形式返回接收缓冲区内特定量的数据。数据量不足时返回byte[0]。

语法 3

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

注

与语法 2 相同。读取时间参数默认为 0。

```
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[] = {}          //接收缓冲区内的数据不足 100 字节，故数量不足。将返回 Byte[0]。
  // 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}                    //获取所有数据
  // ReceivedBuffer = {}
var_value = socket_read("ntd_a", 4, 100)
  // byte[] = {}   //接收缓冲区内的数据不足 4 字节，故数量不足。将返回 Byte[0]。
                   //但读取时间被设为 100 ms，因此仍将停留在函数中等待获取数据或读取时间耗尽，然后退出。
   // ReceivedBuffer = {}{0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38}   //假设 50 ms 后接收到数据
  // byte[] = {0x31,0x32,0x33,0x34}                                        //获取 4 字节，然后退出函数。
  // ReceivedBuffer = {0x35,0x36,0x37,0x38}
```

语法 4

```
byte[] socket_read(
    string,
    byte[] or string,
    byte[] or string,
    int,
    int
)
```

参数

`string` 网络设备名称

`byte[] or string`
读取时的前缀条件。输入 byte[0]或空字符串""代表无前缀条件。

`byte[] or string`
读取时的后缀条件。输入 byte[0]或空字符串""代表无后缀条件。

`int` 获取选项

0	第一个匹配项无需位于开头，且不删除前缀和后缀（默认值）
1	第一个匹配项无需位于开头，且将删除前缀和后缀
2	第一个匹配项必须位于开头，且将删除前缀和后缀
3	第一个匹配项必须位于开头，且将删除前缀和后缀

`int` 读取时间（单位为毫秒）

<= 0	读取一次
> 0	多次读取，直至取得数据或时间耗尽。

返回值

`byte[]` 以字节形式返回第一个满足前缀条件和后缀条件的数组。
仅获取接收缓冲区内第一条满足条件的数据。其余数据留在接收缓冲区内。
若前缀条件和后缀条件均为byte[0]或空字符串，将获取接收缓冲区内的所有数据。
如未能找到满足条件的数据，将返回byte[0]。

语法 5

```
byte[] socket_read(
    string,
    byte[] or string,
    byte[] or string,
    int
)
```

注

与语法 4 相同。读取时间参数默认为 0。

语法 6

```
byte[] socket_read(
    string,
    byte[] or string,
    byte[] or string
)
```

注

与语法 4 相同。获取选项和读取时间参数均默认为 0。

```
ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
byte[] var_value = socket_read("ntd_a", "", "")                    //无前缀。无后缀。获取所有数据。
  // 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)             //前缀为"He"、后缀为\u0D0A
  // byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A}             // Hello,\u0D0A
  // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}          //获取第一个匹配项。保留后续数据。
var_value = socket_read("ntd_a", "", newline, 1)                   //前缀为""、后缀为\u0D0A，删除前缀和后缀。
  // byte[] = {0x57,0x6F,0x72,0x6C,0x64}                            // World
  // ReceivedBuffer = {}
var_value = socket_read("ntd_a", "", newline, 1, 100)              //前缀为""、后缀为\u0D0A，删除前缀和后缀。
                                                                   等待 100 ms
  // byte[] = {}                                                   //未满足获取选项，读取 byte[0]。等待 100 ms。
  // ReceivedBuffer = {}
  // ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A}    //假设 50 ms 后接收到数据
  // 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)             //前缀为"lo"、后缀为\u0D0A
  // byte[] = {0x6C,0x6F,0x2C,0x0D,0x0A}                           // lo,\u0D0A
  //第一个匹配项前的数据{0x48,0x65,0x6C}将被删除。
  // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
byte[] var_bb = {}
var_value = socket_read("ntd_a", var_bb, newline)                  //前缀为 byte[0]、后缀为\u0D0A
  // byte[] = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}                 // World\u0D0A
  // ReceivedBuffer = {}
var_value = socket_read("ntd_a", var_bb, newline, 0, 100)//前缀为 byte[0]、后缀为\u0D0A，等待 100 ms
  // byte[] = {}                                  //未满足获取选项，读取 byte[0]。等待 100 ms。
```

```
  // ReceivedBuffer = {}
  // ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A}  //假设 50 ms 后接收到数据
  // 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,                     // $TM
                  0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}      // #Robot
byte[] var_value = socket_read("ntd_a", "#", newline, 2)        //前缀为"#"、后缀为\u0D0A
  // byte[] = {}                                                //#不在前缀中，返回空字符串。
  // ReceivedBuffer = {0x24,0x48,0x65,0x6C,0x6C,0x6F,0x0D,0x0A,0x23,0x48,0x69,0x0D,0x0A,
                       0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}
var_value = socket_read("ntd_a", "$", newline, 2)               //前缀为"$"、后缀为\u0D0A
  // byte[] = {0x24,0x48,0x65,0x6C,0x6C,0x6F,0x0D,0x0A}         //获取第一个匹配项，该项需位于前缀中。
  // ReceivedBuffer = {0x23,0x48,0x69,0x0D,0x0A,
                       0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}
var_value = socket_read("ntd_a", "#", newline, 2)               //前缀为"#"、后缀为\u0D0A
  // byte[] = {0x23,0x48,0x69,0x0D,0x0A}                        //获取第一个匹配项，该项需位于前缀中。
  // ReceivedBuffer = {0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}
var_value = socket_read("ntd_a", "$", newline, 3)               //前缀为"$"、后缀为\u0D0A
  // byte[] = {0x54,0x4D}
  // ReceivedBuffer = {0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}
var_value = socket_read("ntd_a", "#", newline, 3)               //前缀为"#"、后缀为\u0D0A
  // byte[] = {0x52,0x6F,0x62,0x6F,0x74}
  // ReceivedBuffer = {}
```

语法 7

```
byte[] socket_read(
    string,
    byte[] or string,
    int,
    int
)
```

参数

`string` 网络设备名称

`byte[] or string`
读取时的后缀条件。输入 byte[0]或空字符串""代表无后缀条件。

`int` 获取选项

0	第一个匹配项无需位于开头，且不删除前缀和后缀（默认值）
1	第一个匹配项无需位于开头，且将删除前缀和后缀
2	第一个匹配项必须位于开头，且将删除前缀和后缀
3	第一个匹配项必须位于开头，且将删除前缀和后缀

`int` 读取时间（单位为毫秒）

<= 0	读取一次
> 0	多次读取，直至取得数据或时间耗尽。

返回值

`byte[]` 以字节形式返回第一个满足后缀条件的数组。（无前缀条件限制）
仅获取接收缓冲区内第一条满足条件的数据。其余数据留在接收缓冲区内。
若后缀条件为byte[0]或空字符串，将获取接收缓冲区内的所有数据。

如未能找到满足条件的数据，将返回byte[0]。

语法 8

```
byte[] socket_read(
    string,
    byte[] or string,
    int
)
```

注

与语法 7 相同。读取时间参数默认为 0。

语法 9

```
byte[] socket_read(
    string,
    byte[] or string
)
```

注

与语法 7 相同。获取选项和读取时间参数均默认为 0。

```
ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
byte[] var_value = socket_read("ntd_a", "")                    //无后缀。获取所有数据。
  // 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)               //后缀为\u0D0A
  // byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A}        // Hello,\u0D0A
  // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}     //获取第一个匹配项。保留后续数据。
var_value = socket_read("ntd_a", newline)                      //后缀为\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)            //后缀为\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)                   //后缀为\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)                   //后缀为\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)                   //后缀为\u0D0A
  // byte[] = {0x23,0x52,0x6F,0x62,0x6F,0x74}                  // #Robot
  // ReceivedBuffer = {}
```

9.5 socket_read_string()

读取接收缓冲区内的数据并将字节数组转换成 UTF8 字符串文本

语法 1

```
string socket_read_string(
    string
)
```

参数

`string` 网络设备名称

返回值

`string` 返回接收缓冲区内的所有数据。接收缓冲区为空时返回空字符串。

语法 2

```
string socket_read_string(
    string,
    int,
    int
)
```

参数

`string` 网络设备名称

`int` 获取固定量的字符串（基于字符串的长度）

`<= 0` 获取全部

`> 0` 获取特定量（数据需达到特定量才能被获取）

`int` 读取时间（单位为毫秒）

`<= 0` 读取一次

`> 0` 多次读取，直至取得数据或时间耗尽。

返回值

`string` 以字符串形式返回接收缓冲区内特定量的数据。数据量不足时返回空字符串。

语法 3

```
string socket_read_string(
    string,
    int
)
```

注

与语法 2 相同。读取时间参数默认为 0。

语法 4

```
string socket_read_string(
    string,
    byte[] or string,
    byte[] or string,
    int,
    int
)
```

参数

`string` 网络设备名称

`byte[] or string`

读取时的前缀条件。输入 byte[0]或空字符串""代表无前缀条件。

`byte[] or string`

读取时的后缀条件。输入 byte[0]或空字符串""代表无后缀条件。

`int` 获取固定量的字符串（基于字符串的长度）

`<= 0` 获取全部

`> 0` 获取特定量（数据需达到特定量才能被获取）

`int` 读取时间（单位为毫秒）

`<= 0` 读取一次

`> 0` 多次读取，直至取得数据或时间耗尽。

返回值

`string` 返回第一个满足前缀条件和后缀条件的字符串。

仅获取接收缓冲区内第一条满足条件的数据。其余数据留在接收缓冲区内。

若前缀条件和后缀条件均为byte[0]或空字符串，将获取接收缓冲区内的所有数据。

如未能找到满足条件的数据，将返回空字符串。

语法 5

```
string socket_read_string(
    string,
    byte[] or string,
    byte[] or string,
    int
)
```

注

与语法 4 相同。读取时间参数默认为 0。

语法 6

```
string socket_read_string(
    string,
    byte[] or string,
    byte[] or string
)
```

注

与语法 4 相同。获取选项和读取时间参数均默认为 0。

语法 7

```
string socket_read_string(
    string,
    byte[] or string,
    int,
    int
)
```

参数

`string` 网络设备名称

`byte[] or string`

读取时的后缀条件。输入 byte[0]或空字符串""代表无后缀条件。

`int` 获取固定量的字符串（基于字符串的长度）

`<= 0` 获取全部

`> 0` 获取特定量（数据需达到特定量才能被获取）

`int` 读取时间（单位为毫秒）

`<= 0` 读取一次

`> 0` 多次读取，直至取得数据或时间耗尽。

返回值

`string` 返回第一个满足后缀条件的字符串。（无前缀条件限制）
仅获取接收缓冲区内第一条满足条件的数据。其余数据留在接收缓冲区内。
若后缀条件为byte[0]或空字符串，将获取接收缓冲区内的所有数据。
如未能找到满足条件的数据，将返回空字符串。

语法 8

```
string socket_read_string(
    string,
    byte[] or string,
    int
)
```

注

与语法 7 相同。读取时间参数默认为 0。

语法 9

```
string socket_read_string(
    string,
    byte[] or string
)
```

注

与语法 7 相同。获取选项和读取时间参数均默认为 0。

```
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}      //根据字符串长度，获取 4。
  // ReceivedBuffer = {0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA}
var_value = socket_read_string("ntd_a", 5, 100)
  // string = ""                //接收缓冲区内的待读取数据量不足 5。（根据字符串长度）。等待 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) //前缀为"He"、后缀为\u0D0A
  // string = "Hello,\u0D0A"
  // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
var_value = socket_read_string("ntd_a", "", newline, 1)    //前缀为""、后缀为\u0D0A，删除前缀和后缀。
  // string = "World"
  // ReceivedBuffer = {}
```

```
var_value = socket_read_string("ntd_a", "", newline, 1, 100)
  // string = ""                                          //未满足获取选项，读取空字符串。等待 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)    //前缀为"lo"、后缀为\u0D0A
  // string = "lo,\u0D0A"
  //第一个匹配项前的数据"Hel"将被删除。
  // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}
var_value = socket_read_string("ntd_a", newline, 1)              //后缀为\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,                  // $TM
                  0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}   // #Robot
string var_value = socket_read_string("ntd_a", "#", newline, 2) //前缀为"#"、后缀为\u0D0A
  // string = ""                                          //#不在前缀中，返回空字符串。
  // ReceivedBuffer = {0x24,0x48,0x65,0x6C,0x6C,0x6F,0x0D,0x0A,0x23,0x48,0x69,0x0D,0x0A,
                       0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}
var_value = socket_read_string("ntd_a", "$", newline, 2)         //前缀为"$"、后缀为\u0D0A
  // string = "$Hello\u0D0A"                              //获取第一个匹配项，该项需位于前缀中。
  // 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)         //前缀为"#"、后缀为\u0D0A
  // string = "#Hi\u0D0A"                                 //获取第一个匹配项，该项需位于前缀中。
  // ReceivedBuffer = {0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

var_value = socket_read_string("ntd_a", "$", newline, 3)         //前缀为"$"、后缀为\u0D0A
  // string = "TM"
  // ReceivedBuffer = {0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}
var_value = socket_read_string("ntd_a", "#", newline, 3)         //前缀为"#"、后缀为\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)          //后缀为\u0D0A
  // string = "Hello,\u0D0A"
  // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64}
var_value = socket_read_string("ntd_a", newline, 0)              //后缀为\u0D0A
  // string = ""                                          //未满足获取选项，读取空字符串。
  // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64}
var_value = socket_read_string("ntd_a", newline, 1, 100)         //后缀为\u0D0A
  // string = ""                                          //未满足获取选项，读取空字符串。等待
                                                          100 ms。
  // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64}
  // ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A,0x31,0x32,0x33,0x0D,0x0A}
```

```
//假设数据进入，获取选项得到满足。
// string = "World"
// ReceivedBuffer = {0x31,0x32,0x33,0x0D,0x0A}
var_value = socket_read_string("ntd_a", newline, 2)                    //后缀为\u0D0A
// string = "123\u0D0A"
// ReceivedBuffer = {}
```

9.6 socket_send()

将数据值发送至远程设备。

语法 1

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

参数

string	网络设备名称	
?	要写入的值。支持的类型：int、float、bool、string 和 array。 将以小端序转换数值、以 UTF8 编码转换字符串值。	
int	要写入的值的起始索引（支持字符串和数组）	
	0 .. length-1	合法值
	< 0	非法值。起始索引将被设为0。
	>= length	非法值。起始索引将被设为0。
int	要写入的值的长度（支持字符串和数组）	
	<= 0	从起始索引开始写入，直至数据末尾。
	> 0	从起始索引开始写入特定长度的数据，最多至数据末尾。

返回值

int	发送结果	
	1	发送成功。
	0	数据值为空字符串或空数组，无法发送。
	-1	发送时出现Socket异常。
	-2	无法连接至远程设备。
	-3	设备名称不存在，或IP或端口不正确。

语法 2

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

注

与语法 1 相同。“要写入的值的长度”参数默认为 0。

语法 3

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

注

与语法 1 相同。起始索引和“要写入的值的长度”参数默认为 0。

```
int var_re = socket_send("ntd_a", 100)              //发送 0x64
var_re = socket_send("ntd_a", 1000)                 //发送 0xE8,0x03,0x00,0x00（int，小端序）
var_re = socket_send("ntd_a", (float)1.234)         //发送 0xB6,0xF3,0x9D,0x3F（float，小端序）
```

```
var_re = socket_send("ntd_a", (double)123.456)
  //发送0x77,0xBE,0x9F,0x1A,0x2F,0xDD,0x5E,0x40（double，小端序）
var_re = socket_send("ntd_a", "Hello, World"+newline)
  //发送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)
  //发送0x64,0x00,0x00,0x00,0xC8,0x00,0x00,0x00,0x2C,0x01,0x00,0x00,0x90,0x01,0x00,0x00（int[]，小端序）
string[] var_ss = {"TM", "", "Robot"}
var_re = socket_send("ntd_a", var_ss)
  //发送0x54,0x4D,0x52,0x6F,0x62,0x6F,0x74 (string[], UTF8)
  //var_ss[1]为空字符串。转换后的值亦为空。

var_re = socket_send("ntd_a", 1000, 1, 2)        //值、起始索引和长度无效
  //发送 0xE8,0x03,0x00,0x00（int，小端序）
var_re = socket_send("ntd_a", "Hello, World"+newline, 0, 7)
  //发送 0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x20 (string, UTF8)
byte[] var_bb = {100, 200}
var_re = socket_send("ntd_a", var_bb)            //发送 0x64,0xC8
var_re = socket_send("ntd_a", var_bb, 1, 1)      //发送 0xC8      //数组。从地址 1 开始读取 1，[1]=200
var_re = socket_send("ntd_a", var_bb, -1, 1)     //发送 0x64      //数组。从地址 0 开始读取 1。[0]=100
var_re = socket_send("ntd_a", "達明機器人", 2) //数组。从地址 2 开始读取至末尾。 "機器人"
  //发送 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)
  //发送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)))
  //发送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)
  //数组。从地址2开始读取100。（直至末尾）[2]=達明機器人
  //发送0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA
```

9.7 socket_sendline()

为数据值添加换行符 0x0D 0x0A 并发送至远程设备。

语法 1

```
int socket_sendline(
    string,
    ?,
    int,
    int
)
```

参数

string	网络设备名称	
?	要写入的值。支持的类型：int、float、bool、string 和 array。 将以小端序转换数值、以 UTF8 编码转换字符串值。	
int	要写入的值的起始索引（支持字符串和数组）	
	0 .. length-1	合法值
	< 0	非法值。起始索引将被设为0。
	>= length	非法值。起始索引将被设为0。
int	要写入的值的长度（支持字符串和数组）	
	<= 0	从起始索引开始写入，直至数据末尾。
	> 0	从起始索引开始写入特定长度的数据，最多至数据末尾。

返回值

int	发送结果	
	1	发送成功。
	0	数据值为空字符串或空数组，无法发送。
	-1	发送时出现Socket异常。
	-2	无法连接至远程设备。
	-3	设备名称不存在，或IP或端口不正确。

语法2

```
int socket_sendline(
    string,
    ?,
    int
)
```

注

与语法 1 相同。“要写入的值的长度”参数默认为 0。

语法 3

```
int socket_sendline(
    string,
    ?
)
```

注

与语法 1 相同。起始索引和“要写入的值的长度”参数默认为 0。

```
int var_re = socket_sendline("ntd_a", 200)        //发送 0xC8,0x0D,0x0A
var_re = socket_sendline("ntd_a", 2000)           //发送 0xD0,0x07,0x00,0x00,0x0D,0x0A（int，小端序）
var_re = socket_sendline("ntd_a", (float)0.234)   //发送 0xB2,0x9D,0x6F,0x3E,0x0D,0x0A（float，小端序）
```

```
var_re = socket_sendline("ntd_a", (double)0.234)
  //发送0xC1,0xCA,0xA1,0x45,0xB6,0xF3,0xCD,0x3F,0x0D,0x0A（double，小端序）
var_re = socket_sendline("ntd_a", "Hello, World"+newline)
  //发送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)
  //发送0x64,0x00,0x00,0x00,0xC8,0x00,0x00,0x00,0x2C,0x01,0x00,0x00,0x0D,0x0A（int[]，小端序）
string[] var_ss = {"TM", "", "Robot"}
var_re = socket_sendline("ntd_a", var_ss)
  //发送0x54,0x4D,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A (string[], UTF8)
  //var_ss[1]为空字符串。转换后的值亦为空。

var_re = socket_sendline("ntd_a", 1000, 1, 2)          //值、起始索引和长度无效
  //发送 0xE8,0x03,0x00,0x00,0x0D,0x0A（int，小端序）
var_re = socket_sendline("ntd_a", "Hello, World"+newline, 0, 7)
  //发送 0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x20,0x0D,0x0A (string, UTF8)
byte[] var_bb = {123, 234}
var_re = socket_sendline("ntd_a", var_bb)              //发送 0x7B,0xEA,0x0D,0x0A
var_re = socket_sendline("ntd_a", var_bb, 1, 1)      //发送 0xEA,0x0D,0x0A  //数组。从地址 1 开始读取 1。
var_re = socket_sendline("ntd_a", var_bb, -1, 1)     //发送 0x7B,0x0D,0x0A  //数组。从地址 0 开始读取 1。
var_re = socket_sendline("ntd_a", "達明機器人", 2) //"数组。从地址 2 开始读取至末尾。"機器人"
  //发送 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)
  //发送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)))
  //发送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)
//数组。从地址2开始读取至末尾。 [2]=達明機器人
//发送0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA,0x0D,0x0A
```

10. 参数化对象

使用参数化对象的方法与使用用户定义的变量的相同。无需声明即可使用参数化对象通过项目运行语法获取或修改点数据，使机器人更灵活地运行。其表达式包含项目、索引和属性三个部分，语法如下所示。

参数化项目[索引].属性

支持下列参数化项目：

1. 点
2. 基准
3. TCP
4. VPoint
5. IO
6. 机器人
7. FT

索引和属性的定义因参数化项目而异。

以读写点（项目）"P1"（索引）的坐标（属性）为例。索引被定义为点的名称，属性被定义为带读写模式的浮点数据类型（使用方法与数组的相同）。

Value	float[]	读/写	点坐标{X,Y,Z,RX,RY,RZ}

读取值

```
float[] f = Point["P1"].Value                 //在点项目中，索引被定义为点的名称，数据类型为字符串。
float f1 = Point["P1"].Value[0]               //可仅获取"P1"的 x 值
```

写入值

```
Point["P1"].Value = {0, 0, 90, 0, 90, 0}      //以{0,0,90,0,90,0}替换"P1"的坐标
Point["P1"].Value[2] = 120                    //或仅以 12 替换"P1"的 z 值
```

10.1 点

语法

基准

```
Point[string].属性
```

项目

```
Point
```

索引

```
string
```

点管理器中的点名称

属性

名称	类型	模式	说明	格式
Value	float[]	R/W	点的坐标	{X, Y, Z, RX, RY, RZ}，大小 = 6
Joint	float[]	R/W	关节角度	{J1, J2, J3, J4, J5, J6}，大小 = 6
Pose	int[]	R/W	机器人的姿态	{Config1, Config2, Config3}，大小 = 3
Flange	float[]	R	法兰中心坐标	{X, Y, Z, RX, RY, RZ}，大小 = 6
BaseName	string	R	基准名称	"Base Name"
TCPName	string	R	TCP 名称	"TCP Name"
TeachValue	float[]	R	示教点的原始坐标	{X, Y, Z, RX, RY, RZ}，大小 = 6
TeachJoint	float[]	R	关节角度（示教点处的原始角度）	{J1, J2, J3, J4, J5, J6}，大小 = 6
TeachPose	int[]	R	机器人在示教点处的原始姿态	{Config1, Config2, Config3}，大小 = 3

*设置Value时将重新计算Joint和Flange，设置Joint时将重新计算Value和Flange。因此，无法计算设置值时将报告错误。

注

```
//读取值
float[] f = Point["P1"].Value          //获取"P1"的坐标{X, Y, Z, RX, RY, RZ}
float f1 = Point["P1"].Value[0]        //或仅获取"P1"的 x 值
float f1 = Point["P1"].Value[6]        //超出数组访问范围，返回错误
string s =Point["P1"].BaseName         // s ="RobotBase"
//写入值
Point["P1"].Value = {0, 0, 90, 0, 90, 0}        //以{0,0,90,0,90,0}替换"P1"的坐标
Point["P1"].Value[2] = 120                      //或仅以 120 替换"P1"的 z 值
Point["P1"].Flange = {0, 0, 90, 0, 90, 0}       //只读，无效操作
Point["P1"].Value = {0, 0, 90, 0, 90}           //要写入数组的元素不为 6 个，返回错误（写入 5 个元素）
Point["P1"].Pose = {1, 2, 4, 0}                 //要写入数组的元素不为 3 个，返回错误（写入 4 个元素）
```

10.2 基准

语法

基准

Base[string].属性 或
Base[string, int].属性

项目

Base

索引

string　基准管理器中的基准名称
*基准名称带有只读不写模式属性。
"RobotBase"

int　基准索引，可用于指定通过视觉一次性获取全部构建的多重基准中的基准，可为 0 到 N，默认值为 0。

属性

名称	类型	模式	说明	格式
Value	float[]	R/W	基准值	{X, Y, Z, RX, RY, RZ}，大小 = 6
Type	string	R	基准类型	"R"：机器人基准 "V"：视觉基准 "C"：自定义基准
TeachValue	float[]	R	原始示教基准值	{X, Y, Z, RX, RY, RZ}，大小 = 6

注

```
//读取值
float[] f = Base["RobotBase"].Value          //获取基准"RobotBase"的基准值{0,0,0,0,0,0}
float f1 = Base["base1"].Value[0]            //或仅获取"base1"的 x 值
string s =Base["base1"].Type                 // s ="C"
s =Base[Point["P1"].BaseName].Type           // s ="R"   //假设"P1"的类型为"机器人基准"
float[] f = Base["vision_osga",1].Value      //获取"vision_osga"的第二个值
//写入值
Base["RobotBase"].Value = {0, 0, 90, 0, 90, 0}   //"RobotBase"为只读的系统坐标系，无效操作
Base["base1"].Value = {0, 90, 0, 0, 90, 0}       //以{0,90,0,0,90,0}替换"base1"的值
Base["base1"].Value[4] = 120                     //或仅以 120 替换"base1"的 RY 值
Base["base1"].Value[6] = 120                     //超出数组访问范围，返回错误
Base["base1"].Type = "C"                         //只读，无效操作
Base["base1"].Value = {0, 0, 90, 0, 90}          //要写入数组的元素不为 6 个，返回错误（写入 5 个元素）
Base["base1"].Value = {0, 0, 90, 0, 90, 0, 100}  //要写入数组的元素不为 6 个，返回错误（写入 7 个元素）
```

10.3 TCP

语法

TCP

```
TCP[string].属性
```

项目

```
TCP
```

索引

`string` TCP列表中的TCP名称

*TCP名称带有只读不写模式属性。

"NOTOOL"

"HandCamera"

属性

名称	类型	模式	说明	格式
Value	float[]	R/W	TCP 值	{X, Y, Z, RX, RY, RZ}，大小 = 6
Mass	float	R/W	质量值	质量（kg）
MOI	float[]	R/W	主转动惯量值	{Ixx, Iyy, Izz}，大小 = 3
MCF	float[]	R/W	主轴相对于工具框架的质心框架值	{X, Y, Z, RX, RY, RZ}，大小 = 6
TeachValue	float[]	R	TCP 原始值	{X, Y, Z, RX, RY, RZ}，大小 = 6
TeachMass	float	R	质量原始值	质量（kg）
TeachMOI	float[]	R	主转动惯量原始值	{Ixx, Iyy, Izz}，大小 = 3
TeachMCF	float[]	R	主轴相对于工具框架的质心框架原始值	{X, Y, Z, RX, RY, RZ}，大小 = 6

注

```
//读取值
float[] f = TCP["NOTOOL"].Value                 //获取 TCP"NOTOOL"的值{0,0,0,0,0,0}
float f1 = TCP["NOTOOL"].Value[0]               //或仅获取"NOTOOL"的 x 值
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}
//写入值
TCP["NOTOOL"].Value = {0, -10, 0, 0, 0, 0}      //"NOTOOL"为只读的系统 TCP，无效操作
TCP["T1"].Value = {0, -10, 0, 0, 0, 0}          //以{0,-10,0,0,0,0}替换"T1"的值
TCP["T1"].Value[0] = 10                         //或仅以 10 替换"T1"的 X 值
TCP["T1"].Mass = 2.4                            //以 2.4 kg 替换"T1"的质量值
TCP["T1"].MOI = {0, 0, 0, 1, 2}                 //要写入数组的元素不为 3 个，返回错误（写入 5 个元素）
TCP["T1"].MCF = {0, -20, 0, 0, 0, 0, 0}         //要写入数组的元素不为 6 个，返回错误（写入 7 个元素）
```

10.4 VPoint

语法

VPoint

VPoint[string].属性

项目

VPoint　　视觉任务初始位置

索引

string　　VPoint名称

属性

名称	类型	模式	说明	格式
Value	float[]	R/W	VPoint 初始坐标	{X, Y, Z, RX, RY, RZ}，大小 = 6
BaseName	string	R	VPoint 名称	"Base Name"
TeachValue	float[]	R	VPoint 原始任务初始坐标	{X, Y, Z, RX, RY, RZ}，大小 = 6

注

```
//读取值
float[] f = VPoint["Job1"].Value          //获取 VPoint"Job1"的初始坐标{X, Y, Z, RX, RY, RZ}
float f1 = VPoint["Job1"].Value[0]        //或仅获取"Job1"的 x 值
float f1 = VPoint["Job1"].Value[6]        //超出数组访问范围，返回错误
string s = VPoint["Job1"].BaseName        // s ="RobotBase"
//写入值
VPoint["Job1"].Value = {0, 0, 90, 0, 90, 0}          //以{0,0,90,0,90,0}替换 VPoint"Job1"的初始坐标
VPoint["Job1"].Value[2] = 120                        //或仅以 120 替换"Job1"的 Z 值
VPoint["Job1"].BaseName = "base1"                    //只读，无效操作
VPoint["Job1"].Value = {0, 0, 90, 0, 90}             //要写入数组的元素不为 6 个，返回错误（写入 5 个元素）
VPoint["Job1"].Value = {0, 0, 90, 0, 90, 0, 100}     //要写入数组的元素不为 6 个，返回错误（写入 7 个元素）
```

10.5 IO

语法

IO

```
IO[string].属性
```

项目

IO	输入/输出

索引

string	控制模块名称	
	ControlBox	控制柜
	EndModule	末端模块
	ExtModuleN	外部模块（N = 0~n）
	Safety	安全模块

属性

ControlBox / EndModule / ExtModuleN

名称	类型	模式	说明	格式
DI	byte[]	R	数字输入	[0] = DI0　0：低电平、1：高电平 [1] = DI1 [n] = DIn
DO	byte[]	R/W	数字输出	[0] = DO0　0：低电平、1：高电平 [1] = DO1 [n] = DOn
AI	float[]	R	模拟输入	[0] = AI0　-10.24V~+10.24V（电压） [1] = AI1 [n] = AIn
AO	float[]	R/W	模拟输出	[0] = AO0　-10.00V~+10.00V（电压） [1] = AO1 [n] = AOn
InstantDI	byte[]	R	数字输入（即时命令）	[0] = DI0　0：低电平、1：高电平 [1] = DI1 [n] = DIn
InstantDO	byte[]	R/W	数字输出（即时命令）	[0] = DO0　0：低电平、1：高电平 [1] = DO1 [n] = DOn
InstantAI	float[]	R	模拟输入（即时命令）	[0] = AI0　-10.24V~+10.24V（电压） [1] = AI1 [n] = AIn
InstantAO	float[]	R/W	模拟输出（即时命令）	[0] = AO0　-10.00V~+10.00V（电压） [1] = AO1 [n] = AOn

*DI[n]/DO[n]/AI[n]/AO[n]的设置不同于实际硬件设备标识。

Safety

名称	类型	模式	说明	格式
SI	byte[]	R	安全功能输入	0：低电平、1：高电平 SI[0] = SF1 用户连接的 ESTOP 输入 SI[1] = SF3 用户连接的外部防护装置输入 SI[2]根据安全输入端口指定 SI[3]根据安全输入端口指定

名称	类型	模式	说明	格式
				.. SI[7]根据安全输入端口指定
SO	byte[]	R	安全功能输出	0：低电平、1：高电平 SO[0]根据安全输出端口指定 SO[1]根据安全输出端口指定 SO[2]根据安全输出端口指定 SO[3]根据安全输出端口指定 .. SO[7]根据安全输出端口指定

DI/DO/AI/AO 与 InstantDI/InstantDO/InstantAI/InstantAO 的区别

DI/DO/AI/AO 是可在项目主流程中预约的队列命令。若 DI/DO/AI/AO 位于机器人运动函数（如带混合轨迹的点节点）之后，则 DI/DO/AI/AO 将于点节点结束后执行。若使用 InstantDI/InstantDO/InstantAI/InstantAO 命令，则命令将在点节点运行时执行，而不等待点节点结束。

此外，若在项目线程页面自动将 DI/DO/AI/AO 转为即时命令，则不会等待点节点结束后再运行，其结果与使用 InstantDI/InstantDO/InstantAI/InstantAO 相同。

注

```
//读取值
byte[] di = IO["ControlBox"].DI                 //获取控制柜的数字输入状态
int dilen = Length(di)                          //通过数组大小获取数字 PIN 量
byte di0 = IO["ControlBox"].DI[0]               //获取控制柜 DI[0]的状态
byte di32 = IO["ControlBox"].DI[32]             //超出数组访问范围（给定 DI 是一个长度为 16 的数组，其索引从 0 开始，到 15 结束），返回错误。
float[] ai = IO["ControlBox"].AI                //获取控制柜的模拟输入状态
float[] ao = IO["ControlBox"].AO                //获取控制柜的模拟输出状态
byte si0 = IO["Safety"].SI[0]                   //获取安全模块 SI[0]的安全输入状态
byte so4 = IO["Safety"].SO[4]                   //获取安全模块 SO[4]的安全输出状态
byte si1 = IO["ControlBox"].SI[1]               //控制柜不支持 SI 属性，返回错误。
byte di2 = IO["Safety"].DI[2]                   //安全模块不支持 DI 属性，返回错误。
byte di7 = IO["ControlBox"].InstantDI[7]        //获取控制柜 DI[7]的状态（立即执行）

//写入值
IO["ControlBox"].DI = {1,1,0,0}                 //只读，无效操作
IO["ControlBox"].DI[0] = 0                      //只读，无效操作
IO["ControlBox"].DO[2] = 1                      //将 DO 2 设为高电平
IO["ControlBox"].AO[0] = 3.3                    //将 AO0 设为 3.3V
IO["ControlBox"].DO = {1,1,0,0}                 //要写入的元素与数组大小不匹配（给定 DI 是一个长度为 16 的数组，包含 16 个元素），返回错误
IO["ControlBox"].InstantDO[0] = 1               //将 DO 0 设为高电平（立即执行）
IO["Safety"].SI[0] = 0                          //只读，无效操作
IO["Safety"].SO[4] = 1                          //只读，无效操作
IO["ControlBox"].SO[1] = 1                      //控制柜不支持 SO 属性，返回错误。
IO["Safety"].DO[2] = 1                          //安全模块不支持 DO 属性，返回错误。
```

10.6 机器人

语法

机器人

Robot[int].属性

项目

Robot

索引

int　　机器人的索引，固定为0

属性

名称	类型	模式	说明	格式
CoordRobot	float[]	R	与机器人的机器人基准相反的机器人端点的 TCP 坐标	{X, Y, Z, RX, RY, RZ}，大小 = 6
CoordBase	float[]	R	与机器人的当前基准相反的机器人端点的 TCP 坐标。	{X, Y, Z, RX, RY, RZ}，大小 = 6
Joint	float[]	R	当前的机器人关节角度	{J1, J2, J3, J4, J5, J6}，大小 = 6
BaseName	string	R	当前基准名称	"Base Name"
TCPName	string	R	当前 TCP 名称	"TCP Name"
CameraLight	byte	R/W	机器人相机照明	0：低电平（关）、1：高电平（开）
InstantCameraLight	byte	R/W	机器人相机照明（即时命令）	0：低电平（关）、1：高电平（开）
TCPForce3D	float	R	当前的 TCP 力，即机器人基准 x、y 和 z 的合力。	N
TCPSpeed3D	float	R	当前的 TCP 速度，即机器人基准 x、y 和 z 的合成速度。	mm/s

注

```
//读取值
float[] rtool = Robot[0].CoordRobot      //获取与机器人的机器人基准相反的机器人端点的当前 TCP 坐标
float[] ftool = Robot[0].CoordBase       //获取与机器人的当前基准相反的机器人端点的当前 TCP 坐标
float f = Robot[0].CoordBase[0]          //或仅获取与机器人的当前基准相反的机器人端点的当前 TCP 坐标中的 X 值。

f = Robot[0].CoordBase[6]                //超出数组访问范围，返回错误
float[] joint = Robot[0].Joint           //获取当前的机器人关节角度
float j = Robot[0].Joint[0]              //或仅获取机器人的第一个关节的当前角度
string b = Robot[0].BaseName             // b = "RobotBase"
string t = Robot[0].TCPName              // t = "NOTOOL"
byte light = Robot[0].CameraLight        // light = 0（关）
float tf3d = Robot[0].TCPForce3D         // tf3d = 1.234
float ts3d = Robot[0].TCPSpeed3D         // ts3d = 1.234

//写入值
Robot[0].CoordRobot = {0, 90, 0, 0, 0, 0}    //只读，无效操作
Robot[0].CoordBase = {0, 0, 90, 0, 90, 0}    //只读，无效操作
Robot[0].BaseName = "Base1"                  //只读，无效操作
Robot[0].TCPName = "Tool1"                   //只读，无效操作
Robot[0].CameraLight = 1                     //开启机器人相机照明
Robot[0].CameraLight = 0                     //关闭机器人相机照明
```

```
Robot[0].TCPSpeed3D = 1.234                    //只读，无效操作
```

10.7 FT

语法

FT

FT[string].属性

项目

FT　　力/扭矩传感器状态

索引

string　　F/T传感器列表中的F/T传感器名称

属性

名称	类型	模式	说明	格式
X	float	R	X 轴力值	
Y	float	R	Y 轴力值	
Z	float	R	Z 轴力值	
TX	float	R	X 轴扭矩值	
TY	float	R	Y 轴扭矩值	
TZ	float	R	Z 轴扭矩值	
F3D	float	R	XYZ 合力值	
T3D	float	R	XYZ 扭矩值	
Value	float[]	R	XYZ 合力值和扭矩值数组。	{X, Y, Z, TX, TY, TZ}，大小 = 6
ForceValue	float[]	R	XYZ 合力值数组	{X, Y, Z}，大小 = 3
TorqueValue	float[]	R	XYZ 扭矩值数组	{TX, TY, TZ}，大小 = 3
RefCoordX	float	R	基于节点中设置的参考坐标系测得的 X 轴力值	
RefCoordY	float	R	基于节点中设置的参考坐标系测得的 Y 轴力值	
RefCoordZ	float	R	基于节点中设置的参考坐标系测得的 Z 轴力值	
RefCoordTX	float	R	基于节点中设置的参考坐标系测得的 X 轴扭矩值	
RefCoordTY	float	R	基于节点中设置的参考坐标系测得的 Y 轴扭矩值	
RefCoordTZ	float	R	基于节点中设置的参考坐标系测得的 Z 轴扭矩值	
RefCoordF3D	float	R	基于节点中设置的参考坐标系测得的 XYZ 合力值	
RefCoordT3D	float	R	基于节点中设置的参考坐标系测得的 XYZ 扭矩	
RefCoordForceValue	float[]	R	基于节点中设置的参考坐标系测得的 XYZ 合力值矩阵	{RefCoordX, RefCoordY, RefCoordZ}，大小 = 3
RefCoordTorqueValue	float[]	R	基于节点中设置的参考坐标系测得的 XYZ 扭矩值矩阵	{RefCoordTX, RefCoordTY, RefCoordTZ}，大小 = 3
Model	string	R	F/T 传感器型号名称	
Zero	byte	R/W	开启或关闭 F/T 传感器偏移	0：关闭零点偏移、 1：开启零点偏移

*进行力控制时，与RefCoord*相关的属性均有值。

注

```
//读取值
float x = FT["fts1"].X                          //获取 F/T 传感器"fts1"当前的 X 轴力值
float tx = FT["fts1"].TX                        //获取 F/T 传感器"fts1"当前的 X 轴扭矩值
float f3d = FT["fts1"].F3D                      //获取 F/T 传感器"fts1"当前的 XYZ 合力值
float[] force = FT["fts1"].ForceValue           //获取 F/T 传感器"fts1"当前的 XYZ 合力值数组
string mode = FT["fts1"].Model                  //获取 F/T 传感器"fts1"的型号名称
//写入值
FT["fts1"].Y = 3.14                             //只读，无效操作
FT["fts1"].TY = 1.34                            //只读，无效操作
FT["fts1"].T3D = 4.13                           //只读，无效操作
FT["fts1"].TorqueValue = {1.1, 2.2, 3.3}        //只读，无效操作
FT["fts1"].Zero = 1                             //记录 F/T 传感器当前的偏移
```

11. 机器人示教类
11.1 TPoint 类

可通过使用 TPoint 类并声明变量创建点名称和点值。变量名将成为点名称。*构造点时需要计算坐标和角度值。若无法根据参数值计算出坐标和角度值，将返回错误。

构造 1

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

参数

`float[]` 根据RobotBase和NOTOOL TCP记录的机器人端TCP坐标：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)。

注

```
TPoint p1 = {0,-282.75,1094.9,90,0,0}
                        //根据RobotBase和NOTOOL，坐标值为0,-282.75,1094.9,90,0,0
TPoint p2 = 517.5,-147.8,442.45,180,0,90
                        //根据RobotBase和NOTOOL，坐标值为517.5,-147.8,442.45,180,0,90
```

构造 2

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

参数

`string` 点的定义。默认为"C"。

- `"C"` 点坐标。项目运行结束后，不会将值写回记录文件。项目下一次运行时使用声明的值。（与构造1相同）
- `"D"` 点坐标。项目运行结束后，将值写回记录文件。项目下一次运行时优先使用记录文件。若记录文件为空，则使用声明的值。
- `"J"` 关节角度。项目运行结束后，不会将值写回记录文件。项目下一次运行时使用声明的值。
- `"JD"` 关节角度。项目运行结束后，将值写回记录文件。项目下一次运行时优先使用记录文件。若记录文件为空，则使用声明的值。

`float[]` 若以坐标表示，则为机器人端TCP坐标的六个元素：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)；若以角度表示，则为机器人关节的六个元素：关节1(°)、关节2(°)、关节3(°)、关节4(°)、关节5(°)、关节6(°)。根据RobotBase基准和NOTOOL TCP记录。

注

```
TPoint p3 = "D",{522.07,130.75,442.45,180,0,120}
TPoint p4 = "D",134.95,-147.8,1094.9,90,0,90    //无法根据坐标值计算角度值，返回错误
TPoint p5 = "J",0,0,90,0,90,0                   //根据RobotBase和NOTOOL，角度值为0,0,90,0,90,0
TPoint p6 = "JD",{30,0,90,0,90,0}               //根据RobotBase和NOTOOL，角度值为30,0,90,0,90,0
```

构造 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`

参数

`string`	点的定义。默认为"C"。
`"C"`	点坐标。项目运行结束后，不会将值写回记录文件。项目下一次运行时使用声明的值。
`"D"`	点坐标。项目运行结束后，将值写回记录文件。项目下一次运行时优先使用记录文件。若记录文件为空，则使用声明的值。
`"J"`	关节角度。项目运行结束后，不会将值写回记录文件。项目下一次运行时使用声明的值。
`"JD"`	关节角度。项目运行结束后，将值写回记录文件。项目下一次运行时优先使用记录文件。若记录文件为空，则使用声明的值。
`float[]`	若以坐标表示，则为机器人端TCP坐标的六个元素：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)；若以角度表示，则为机器人关节的六个元素：关节1(°)、关节2(°)、关节3(°)、关节4(°)、关节5(°)、关节6(°)。根据基准名称和TCP名称记录。
`string`	基准名称。若为空字符串，则将使用记录中的当前基准名称。
`string`	TCP名称。若为空字符串，则将使用记录中的当前TCP名称。

注

```
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"                    //根据base1和NOTOOL
TPoint p9 = "JD",{30,0,90,0,90,0},"RobotBase","tcp1"              //根据RobotBase和tcp1
TPoint p0 = "C",{517.5,-147.8,342.45,180,0,90},"base1","tcp1"     //根据base1和tcp1
//以下语法在定义部分外。
ChangeBase("base1")
TPoint p00 = {517.5,-147.8,342.45,180,0,90},"","tcp1"             //根据当前基准base1和指定的tcp1
```

构造 4

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

参数

`string`	点的定义。默认为"C"。
`"C"`	项目运行结束后，不会将值写回记录文件。项目下一次运行时使用声明的值。
`"D"`	项目运行结束后，将值写回记录文件。项目下一次运行时优先使用记录文件。若记录文件为空，则使用声明的值。
`float[]`	以坐标表示，此为机器人端TCP坐标的六个元素：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)。
`float[]`	以角度表示，此为机器人关节的六个元素：关节1(°)、关节2(°)、关节3(°)、关节4(°)、关节5(°)、关节6(°)。根据基准名称和TCP名称记录。
`string`	基准名称。若为空字符串，则将使用记录中的当前基准名称。
`string`	TCP名称。若为空字符串，则将使用记录中的当前TCP名称。

注

```
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}   //根据RobotBase和NOTOOL
TPoint tp3 = {0,-292.75,1004.9,90,0,0},{0,0,0,0,0,0},"base1","tcp1"   //根据base1和tcp1
TPoint tp4 = {134.95,-147.8,1094.9,90,0,90},{0,0,0,0,90,0}   //根据RobotBase和NOTOOL
```

```
//以下语法在定义部分外。
ChangeBase("base1")
ChangeTCP("tcp1")
TPoint tp5 = {0,-292.75,1004.9,90,0,0},{0,0,0,0,0,0},"","" //根据当前基准base1和当前TCP tcp1
```

成员属性

名称	类型	模式	说明	格式
Value	float[]	R/W	点坐标	{X, Y, Z, RX, RY, RZ}，大小 = 6
Joint	float[]	R/W	关节角度	{J1, J2, J3, J4, J5, J6}，大小 = 6
Pose	int[]	R/W	机器人的姿态	{Config1, Config2, Config3}，大小 = 3
Flange	float[]	R	法兰中心坐标	{X, Y, Z, RX, RY, RZ}，大小 = 6
BaseName	string	R	基准名称	"Base Name"
TCPName	string	R	TCP 名称	"TCP Name"
TeachValue	float[]	R	点坐标（原始示教点坐标）	{X, Y, Z, RX, RY, RZ}，大小 = 6
TeachJoint	float[]	R	关节角度（原始示教点角度）	{J1, J2, J3, J4, J5, J6}，大小 = 6
TeachPose	int[]	R	机器人的姿态（原始示教点姿态）	{Config1, Config2, Config3}，大小 = 3

*设置Value时将重新计算Joint和Flange，设置Joint时将重新计算Value和Flange。因此，无法计算时将返回错误。

*只能在变量定义部分为变量赋予原始示教点值。

注

```
TPoint P1 = {0,-282.75,1094.9,90,0,0}
                                    //根据RobotBase和NOTOOL，"P1"的坐标值为0,-282.75,1094.9,90,0,0
//读取值
float[] f = P1.Value                //获取点"P1"的点坐标{0,-282.75,1094.9,90,0,0}
float f1 = P1.Value[1]              //或仅获取点"P1"的 Y 值
float f2 = P1.Value[6]              //返回错误。超出数组访问范围
float[] f3 = P1.Joint               //获取点"P1"的角度值{0,0,0,0,0,0}
string s = P1.BaseName              // s = "RobotBase"
//写入值
P1.Value = {517.5,-147.8,442.45,180,0,90}  //将点"P1"的点坐标改为
                                           {517.5,-147.8,442.45,180,0,90}
P1.Value[2] = 450                          //或仅将点"P1"的 z 值改为 450
P1.Flange = {0,0,90,0,90,0}                //只读，无效操作
P1.Value = {0,0,90,0,90}            //返回错误。要写入的数组元素不为 6 个。（为 5 个）
P1.Pose = {1,2,4,0}                 //返回错误。要写入的数组元素不为 3 个。（为 4 个）
```

11.2 TBase 类

可通过使用 TBase 类并声明变量创建点名称和点值。变量名将成为点名称。
*系统基准名称带有只读模式属性而无写入模式属性。
"RobotBase"
因此，作为系统基准名称的变量名只能用于变量声明。在构造内，输入的值无效。

构造 1

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

参数

`float[]` 基准值：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)

注

```
TBase base1 = {0,0,90,0,0,0}          //名称：base1、类型：C、值：0,0,90,0,0,0
TBase base2 = 0,90,90,0,0,0           //名称：base2、类型：C、值：0,90,90,0,0,0
TBase RobotBase = 0,0,90,0,0,90
                  //由于RobotBase为系统基准，输入值除用于变量声明外均无效。
```

构造 2

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

参数

`string` 基准类型。默认为"C"。

"C" 项目运行结束后，不会将值写回记录文件。项目下一次运行时使用声明的值。（与构造1相同）

"D" 项目运行结束后，将值写回记录文件。项目下一次运行时优先使用记录文件。若记录文件为空，则使用声明的值。

"V" 定义与类型"D"相同。该类型主要用于视觉任务。其名称必须符合以*vision_*开头的命名规则。

*仅在变量定义部分，声明变量才会带有写回记录文件机制。

`float[]` 基准值：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)

注

```
TBase base5 = "C",{0,0,90,0,0,0}          //名称：base5、类型：C、值：0,0,90,0,0,0
TBase base6 = "D",0,90,90,0,0,0           //名称：base6、类型：D、值：0,90,90,0,0,0
TBase vision_base7 = "V",90,90,90,0,0,0   //名称：vision_base7、类型：V、值：90,90,90,0,0,0
TBase base8 = "V",90,90,90,0,0,0
                  //返回错误。名称base8不符合V类型的命名规则。
```

成员属性

名称	类型	模式	说明	格式
Value	float[]	R/W	基准值	{X, Y, Z, RX, RY, RZ}，大小 = 6
Type	string	R	基准类型	"R"：机器人基准 "C"：自定义基准 "D"：自定义基准，带写回机制 "V"：视觉基准
TeachValue	float[]	R	基准值（原始示教点基准值）	{X, Y, Z, RX, RY, RZ}，大小 = 6

*只能在变量定义部分为变量赋予原始示教基准值。

成员方法

名称	说明
GetValue()	获取基准值
SetValue()	设置基准值

11.2.1 GetValue()

获取基准值。（适用于在视觉任务中创建的多重基准）

语法 1

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

参数

int	基准索引。用户可用其指定通过视觉一次性获取全部创建的多重基准中的基准。索引可为0到N，默认值为0。基准索引。用户可用其指定通过视觉一次性获取全部创建的多重基准中的基准。索引可为0到N，默认值为0。

返回值

float[]	基准值：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°) 若基准索引值大于多重基准的基准数量，将返回空数组。

11.2.2 SetValue()

设置基准值。（适用于在视觉任务中创建的多重基准）

语法 1

```
bool SetValue(
    int,
    float[]
)
```

参数

int	基准索引。用户可用其指定通过视觉一次性获取全部创建的多重基准中的基准。索引可为0到N，默认值为0。
float[]	基准值：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)

返回值

bool	若成功，返回True。若失败，返回False。 若基准索引值大于多重基准的基准数量，设置将失败。

注

```
TBase base1 = {0,0,90,0,0,0}                    //名称：base1、类型：C、值：0,0,90,0,0,0
TBase RobotBase = 0,0,90,0,0,90
                    //由于RobotBase为系统基准，输入值除用于变量声明外均无效。
TBase base5 = "C",{0,0,90,0,0,0}                 //名称：base5、类型：C、值：0,0,90,0,0,0
TBase base6 = "D",0,90,90,0,0,0                  //名称：base6、类型：D、值：0,90,90,0,0,0
TBase vision_base8 = "V",90,90,90,0,0,0          //名称：vision_base8、类型：V、值：90,90,90,0,0,0
//读取值
float[] b1 = base1.Value                         //基准："base1"、值：{0,0,90,0,0,0}
float[] b2 = RobotBase.Value                     //基准："RobotBase"、值：{0,0,0,0,0,0}
float b3 = base1.Value[2]                        //基准"base1"的Z值：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} 获取基准"base5"中的第一个基准值。
float[] b51 = base5.GetValue(1)         //{}空数组  获取基准"base5"中的第二个基准值。
float[] vb = vision_base8.GetValue(1)   //获取基准"vision_base8"中的第二个基准值。
                                        //假设存在创建并更新了vision_base8的基准值的视觉任务。
//写入值
RobotBase.Value = {0,0,90,0,90,0}       //"RobotBase"为只读的系统基准，无效操作。
base1.Value = {0,90,0,0,90,0}           //将基准"base1"改为{0,90,0,0,90,0}
base1.Value[4] = 120                    //或仅将基准"base1"的 RY 值改为 120。
base1.Value[6] = 120                    //返回错误。超出数组访问范围。
base1.Type = "C"                        //只读，无效操作
base1.Value = {0,0,90,0,90}
                        //返回错误。要写入的数组元素不为 6 个。（为 5 个）
base1.Value = {0,0,90,0,90,0,100}
                        //返回错误。要写入的数组元素不为6个。（为7个）
base6.Value = {30,90,90,0,0,0}          //由于"base6"为D类型，写回记录文件。
base5.SetValue(0, {90,90,90,0,0,0})     //True。设置基准"base5"中的第一个坐标值。
base5.SetValue(1, {0,0,0,90,90,90})     //False。设置失败。
```

11.3 TTCP 类

可通过使用 TTCP 类并声明变量创建点名称和点值。变量名将成为点名称。
*系统 TCP 名称带有只读模式属性而无写入模式属性。
"NOTOOL"
"HandCamera"
因此，作为系统基准名称或在机器人全局 TCP 设置内的变量名只能用于变量声明。在构造内，输入的值无效。

构造 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`

参数

`float[]` TCP值：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)
`float` 工具质量（kg）。默认为0。

注

TTCP tcp1 = {0,0,90,0,0,0} //名称：tcp1、类型：C、值：0,0,90,0,0,0、质量：0 kg
TTCP tcp2 = {0,0,90,0,0,0},2 //名称：tcp2、类型：C、值：0,0,90,0,0,0、质量：2 kg
TTCP tcp3 = 0,0,90,0,0,0 //名称：tcp3、类型：C、值：0,0,90,0,0,0、质量：0 kg
TTCP tcp4 = 0,0,90,0,0,0,2 //名称：tcp4、类型：C、值：0,0,90,0,0,0、质量：2 kg
TTCP NOTOOL = 0,0,90,0,0,90
//由于NOTOOL为系统工具名称，输入值除用于变量声明外均无效。

构造 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`

参数

`string` 工具类型。默认为"C"。
"C" 项目运行结束后，不会将值写回记录文件。项目下一次运行时使用声明的值。（与构造1相同）
"D" 项目运行结束后，将值写回记录文件。项目下一次运行时优先使用记录文件。若记录文件为空，则使用声明的值。
"V" 定义与类型"D"相同。该类型主要用于视觉任务。其名称必须符合以*vision_*开头的命名规则。
*仅在变量定义部分，声明变量才会带有写回记录文件机制。
`float[]` TCP值：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)
`float` 工具质量（kg）。默认为0。

注

TTCP tcp5 = "C",{0,0,90,0,0,0} //名称：tcp5、类型：C、值：0,0,90,0,0,0、质量：0 kg
TTCP tcp6 = "D",{0,90,90,0,0,0},2 //名称：tcp6、类型：D、值：0,90,90,0,0,0、质量：2 kg
TTCP visionTCP_tcp7 = "V",90,90,90,0,0,0 //名称：visionTCP_tcp7、类型：V、值：90,90,90,0,0,0
TTCP tcp7v = "V",90,90,90,0,0,0
//返回错误。名称tcp7v不符合V类型的命名规则。
TTCP tcp8 = "D",90,90,90,0,0,0,2 //名称：tcp8、类型：D、值：90,90,90,0,0,0、质量：2 kg

构造 3

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

TTCP *VariableName* = `float[], float, float[], float[]`

参数

`string`	工具类型。默认为"C"。
"C"	项目运行结束后，不会将值写回记录文件。项目下一次运行时使用声明的值。（与构造1相同）
"D"	项目运行结束后，将值写回记录文件。项目下一次运行时优先使用记录文件。若记录文件为空，则使用声明的值。
"V"	定义与类型"D"相同。该类型主要用于视觉任务。其名称必须符合以*vision_*开头的命名规则。
	*仅在变量定义部分，声明变量才会带有写回记录文件机制。
`float[]`	TCP值：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)
`float`	工具质量（kg）。默认为0。
`float[]`	主转动惯量值：Ixx（kg-mm^2）、Iyy（kg-mm^2）、Izz（kg-mm^2）
`float[]`	主轴相对于工具框架的质心框架：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)

注

```
TTCP tcp9 = {0,0,90,0,0,0},2,{2,0.5,0.5},{0,0,-80,0,0,0}
  //名称：tcp9、类型：C、值：0,0,90,0,0,0、质量：2 kg、惯性：2,0.5,0.5、参考坐标：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}
  //名称：tcp0、类型：D、值：0,0,150,0,0,90、质量：1 kg、惯性：1,0.5,0.5、参考坐标：0,0,-80,0,0,0
```

成员属性

名称	类型	模式	说明	格式
Value	float[]	R/W	TCP 值	{X, Y, Z, RX, RY, RZ}，大小 = 6
Mass	float	R/W	质量	质量（kg）
MOI	float[]	R/W	主转动惯量	{Ixx, Iyy, Izz}，大小 = 3
MCF	float[]	R/W	主轴相对于工具框架的质心框架	{X, Y, Z, RX, RY, RZ}，大小 = 6
TeachValue	float[]	R	TCP 值（原始 TCP 设置值）	{X, Y, Z, RX, RY, RZ}，大小 = 6
TeachMass	float	R	质量（原始 TCP 设置值）	质量（kg）
TeachMOI	float[]	R	主转动惯量（原始 TCP 设置值）	{Ixx, Iyy, Izz}，大小 = 3
TeachMCF	float[]	R	主轴相对于工具框架的质心框架（原始 TCP 设置值）	{X, Y, Z, RX, RY, RZ}，大小 = 6

*仅在变量定义部分声明变量时，才能赋予原始TCP设置值。

注

```
TTCP tcp1 = {0,0,90,0,0,0},3        //名称：tcp1、类型：C、值：0,0,90,0,0,0、质量：3 kg
TTCP NOTOOL = 0,0,90,0,0,90
                //由于NOTOOL为系统工具名称，输入值除用于变量声明外均无效。
TTCP tcp9 = "D",{0,0,90,0,0,0},2,{2,0.5,0.5},{0,0,-80,0,0,0}
//读取值
float[] t0 = NOTOOL.Value           //获取名为 NOTOOL 的工具的 TCP 值{0,0,0,0,0,0}
float[] t1 = tcp1.Value             //获取名为 tcp1 的工具的 TCP 值{0,0,90,0,0,0}
float t2 = tcp1.Value[2]            //或仅获取名为 tcp1 的工具的 Z 值。
float mass = tcp1.Mass              // mass = 3
float[] mcf = tcp1.MCF              // mcf = {0,0,0,0,0,0}
float mass9 = tcp9.Mass             // mass9 = 2     //若在 tcp9.Mass = 2.1 之后运行，将取得 2.1 kg
float[] moi9 = tcp9.MOI             // moi9 = {2,0.5,0.5}
```

```
//写入值
NOTOOL.Value = {0, -10, 0, 0, 0, 0}     //NOTOOL 为只读的系统 TCP，无效操作。
tcp1.Value = {0, -10, 0, 0, 0, 0}       //将名为 tcp1 的 TCP 改为{0,-10,0,0,0,0}
tcp1.Value[0] = 10                      //或仅将 tcp1 的 X 值改为 10
tcp1.Mass = 2.4                         //将名为 tcp1 的 TCP 的质量改为 2.4 kg。
tcp9.Mass = 2.1                         //将名为 tcp9 的 TCP 的质量改为 2.1 kg。
tcp1.MOI = {0, 0, 0, 1, 2}      //返回错误。要写入的数组元素不为 3 个。（为 5 个）
tcp1.MCF = {0, -20, 0, 0, 0, 0, 0}
                                //返回错误。要写入的数组元素不为 6 个。（为 7 个）
```

12. 机器人运动及视觉任务函数

使用流程项目时，如需使用机器人运动函数和视觉任务函数，需要通过外部命令使项目流程进入 Listen 节点（外部命令控制模式）或脚本节点。若使用脚本项目编程，则可直接在项目编程中使用这些函数。

只能在主线程中调用机器人运动函数和视觉任务函数，将函数移动至初始位置。主线程即流程项目中的主流程或脚本项目中的主函数。不得在其他线程中调用运动函数。所有运动进程都将暂时排队并依次被处理。用户可按需使用队列标签号了解当前运动命令过程。

12.1 QueueTag()

为机器人运动设置队列标签号，以表示当前正在进行的机器人运动。可使用 TMSTA SubCmd 01 监控各队列标签的状态。

语法 1

```
bool QueueTag(
    int,
    int
)
```

参数

int	标签号。可为1到15的整数。
int	是否等待标签结束再继续运行。 0　不等待（默认值） 1　等待 该值被设为1时，进程将停留在函数中，等待标签完成，然后继续运行。

返回值

bool	标记成功时返回True。标记失败时返回False。

语法 2

```
bool QueueTag(
    int
)
```

注

与语法 1 相同。等待参数默认为 0，不等待标签结束即继续运行。

12.2 WaitQueueTag()

等待机器人运动队列标签号完成。

语法 1

```
int WaitQueueTag(
    int,
    int
)
```

参数

int	排队等待的标签号。	
	1..15	有效标签号。
	0	无效标签号，但会等待至超时。（若超时时间被设为无限，则不等待超时。）
	<0	不可用标签号。不会等待超时。
	>15	不可用标签号。不会等待超时。
int	设置超时时间	
	< 0	无限期等待。标签号为1到15间的合法值时有效。（默认值）
	= 0	等待检查一次
	> 0	超时前等待的毫秒数

使用排队等待时，进程将停留在函数中，直至标签完成、标签不存在或超时，然后继续运行。

返回值

int	返回等待结果	
	1	标签已完成
	0	标签未完成或超时
	-1	标签不存在

*标签号可复用。

*标签号将保留最后四个标签的状态。若等待的标签号未出现或已超出最后四个标签，则返回不存在。

语法 2

```
int WaitQueueTag(
    int
)
```

注

该语法与语法 1 相同。默认为无超时，需要等待标签完成（或不存在）

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

- **运动函数队列标签**

运动函数队列标签用于与机器人运动函数搭配使用。由于所有运动函数都在缓冲区内排队并按顺序执行，因此可以使用协同队列标签了解当前正在执行哪个运动函数。

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)不等待并继续运行
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)不等待并继续运行
,*49\r\n
```

执行脚本内容时，由于 QueueTag()不等待，执行后，进程返回

> $***TMSCT***,4,2,OK,*5F\r\n

机器人运动执行 PTP() targetP1 时，由于 QueueTag(1)，将返回

> $***TMSTA***,10,01,01,true,*64\r\n //TMSTA SubCmd 01、标签号 01，完成

机器人运动执行 PTP() targetP2 时，由于 QueueTag(2)，将返回

> $***TMSTA***,10,01,02,true,*67\r\n // TMSTA SubCmd 01、标签号 02，完成

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)等待并停留在函数中，直至标签完成
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)不等待并继续运行
,*56\r\n

执行脚本内容时，由于 QueueTag(3,1)被设为需等待，标签完成后，进程返回

> $***TMSTA***,10,01,03,true,*66\r\n //TMSTA SubCmd 01、标签号 03，完成

QueueTag(3)完成后，进程继续运行，由于 QueueTag(4)被设为不等待，执行后，进程返回

> $***TMSCT***,4,2,OK,*5F\r\n

机器人运动执行 PTP() targetP2 时，由于 QueueTag(4)，将返回

> $***TMSTA***,10,01,04,true,*61\r\n //TMSTA SubCmd 01、标签号 04，完成

*进程执行脚本时，将返回$TMSCT,4,2,OK。因此，若使用 QueueTag 或 WaitQueueTag 等待，也会在执行后返回。

12.3 StopAndClearBuffer()

令机器人停止运动，然后清除缓冲区内现有的机器人命令。

语法

```
bool StopAndClearBuffer(
)
```

参数

`void` 无参数

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令

注

StopAndClearBuffer()

12.4 PTP()

定义 PTP 运动命令并发送至缓冲区以待执行。

语法 1

```
bool PTP(
    string,
    float[],
    int,
    int,
    int,
    bool
)
```

参数

`string` 数据格式定义，由三个字母组成
#1：运动目标格式：
`"C"`：以笛卡尔坐标表示
`"J"`：以关节角度表示
#2：速度格式：
`"P"`：以百分比表示
#3：混合格式
`"P"`：以百分比表示

`float[]` 运动目标度数。若以笛卡尔坐标定义，则包含工具中心点的笛卡尔坐标：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)。若以关节角度定义，则包含六个关节的角度：关节1(°)、关节2(°)、关节3(°)、关节4(°)、关节5(°)、关节6(°)。

`int` 速度设置，以百分比（%）表示

`int` 加速至最高速度的时间（ms）

`int` 混合值，以百分比（%）表示

`bool` 禁用精确定位
`true` 禁用精确定位
`false` 启用精确定位

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令（格式错误）

注

数据格式参数包括：(1) `"CPP"`、(2) `"JPP"`

```
float[] targetP1= {0,0,90,0,90,0}          //声明一个浮点型数组以存储目标坐标
PTP("JPP", targetP1,10,200,0,false)        //通过PTP运动移动至targetP1，速度 = 10%，加速至最高
                                           速度的时间 = 200 ms。
```

语法 2

```
bool PTP(
    string,
    float[],
    int,
    int,
    int,
    bool,
    int[]
)
```

参数

`string` 数据格式定义，由三个字母组成
- #1：运动目标格式：
  - `"C"`：以笛卡尔坐标表示
- #2：速度格式：
  - `"P"`：以百分比表示
- #3：混合格式
  - `"P"`：以百分比表示

`float[]` 运动目标。若以笛卡尔坐标定义，则包含工具中心点的笛卡尔坐标：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)

`int` 速度设置，以百分比（%）表示

`int` 加速至最高速度的时间（ms）

`int` 混合值，以百分比（%）表示

`bool` 禁用精确定位
- `true` 禁用精确定位
- `false` 启用精确定位

`int[]` 机器人的姿态：[Config1, Config2, Config3]，更多信息请参见附录

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令（格式错误）

注

数据格式参数包括：(1) `"CPP"`

```
float[] targetP1 = {417.50,-122.30,343.90,180.00,0.00,90.00}
        //声明一个浮点型数组以存储目标坐标。
float[] pose = {0,2,4}                          //声明一个浮点型数组以存储姿态。
PTP("CPP", targetP1,50,200,0,false, pose)       //通过PTP运动移动至targetP1，速度 = 50%，
                                                加速至最高速度的时间 = 200 ms。
```

语法 3

```
bool PTP(
    string,
    float, float, float, float, float, float,
    int,
    int,
    int,
    bool
```

)

参数

`string` 数据格式定义，由三个字母组成
#1：运动目标格式：
"C"：以笛卡尔坐标表示
"J"：以关节角度表示
#2：速度格式：
"P"：以百分比表示
#3：混合格式：
"P"：以百分比表示

`float, float, float, float, float, float`
运动目标。若以笛卡尔坐标表示，则包含工具中心点的笛卡尔坐标：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)。若以关节角度表示，则包含六个关节的角度：关节1(°)、关节2(°)、关节3(°)、关节4(°)、关节5(°)、关节6(°)。

`int` 速度设置，以百分比（%）表示

`int` 加速至最高速度的时间（ms）

`int` 混合值，以百分比（%）表示

`bool` 禁用精确定位
`true` 禁用精确定位
`false` 启用精确定位

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令（格式错误）

注

数据格式参数包括：(1) "CPP"和(2) "JPP"

```
PTP("JPP",0,0,90,0,90,0,35,200,0,false)          //通过PTP运动移动至关节角度0,0,90,0,90,0，速度 = 35%，加速至最高速度的时间 = 200 ms。
```

语法 4

```
bool PTP(
    string,
    float, float, float, float, float, float,
    int,
    int,
    int,
    bool,
    int, int, int
)
```

参数

`string` 数据格式定义，由三个字母组成
#1：运动目标格式：
"C"：以笛卡尔坐标表示
#2：速度格式：
"P"：以百分比表示
#3：混合格式：
"P"：以百分比表示

`float, float, float, float, float, float`
运动目标。包含工具中心点的笛卡尔坐标：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)

`int` 速度设置，以百分比（%）表示
`int` 加速至最高速度的时间（ms）
`int` 混合值，以百分比（%）表示
`bool` 禁用精确定位
`true` 禁用精确定位
`false` 启用精确定位
`int, int, int`
机器人的姿态：Config1, Config2, Config3，更多信息请参见附录

返回值

`bool` True 已接受命令；False 已拒绝命令（格式错误）

注

数据格式参数包括：(1) `"CPP"`

```
PTP("CPP",417.50,-122.30,343.90,180.00,0.00,90.00,10,200,0,false,0,2,4)  //通过PTP运动移动至坐标417.50,-122.30,343.90,180.00,0.00,90.00，速度 = 10%，加速至最高速度的时间 = 200 ms，姿态 = 024。
```

12.5 Line()

定义 Line 运动命令并发送至缓冲区以待执行。

语法 1

```
bool Line(
    string,
    float[],
    int,
    int,
    int,
    bool
)
```

参数

`string` 数据格式定义，由三个字母组成
#1：运动目标格式：
"C"：以笛卡尔坐标表示
#2：速度格式：
"P"：以百分比表示
"A"：以绝对速度表示，与项目速度同步
"D"：以绝对速度表示，与项目速度不同步
#3：混合格式：
"P"：以百分比表示
"R"：以半径表示

`float[]` 运动目标。包含工具中心点的笛卡尔坐标：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)

`int` 速度设置，以百分比（%）或速度（mm/s）表示

`int` 加速至最高速度的时间（ms）

`int` 混合值，以百分比（%）或半径（mm）表示

`bool` 禁用精确定位
`true` 禁用精确定位
`false` 启用精确定位

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令（格式错误）

注

数据格式参数包括：
(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}
    //声明一个浮点型数组以存储目标坐标
Line("CAR", Point1,100,200,50,false)        //以直线运动移动至Point1，速度 = 100 mm/s，加速至最高速度的时间 = 200 ms，混合半径 = 50 mm
```

语法2

```
bool Line(
    string,
    float, float, float, float, float, float,
    int,
    int,
    int,
    bool
```

)

参数

`string` 数据格式定义，由三个字母组成

#1：运动目标格式：

"C"：以笛卡尔坐标表示

#2：速度格式：

"P"：以百分比表示

"A"：以绝对速度表示，与项目速度同步

"D"：以绝对速度表示，与项目速度不同步

#3：混合格式：

"P"：以百分比表示

"R"：以半径表示

`float, float, float, float, float, float`

运动目标。包含工具中心点的笛卡尔坐标：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)

`int` 速度设置，以百分比（%）或速度（mm/s）表示

`int` 加速至最高速度的时间（ms）

`int` 混合值，以百分比（%）或半径（mm）表示

`bool` 禁用精确定位

`true` 禁用精确定位

`false` 启用精确定位

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令（格式错误）

注

数据格式参数包括：

(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)

//以直线运动移动至417.50,-122.30,343.90,180.00,0.00,90.00，速度 = 100 mm/s，加速至最高速度的时间 = 200 ms，混合半径 = 50 mm

12.6 Circle()

定义 Circle 运动命令并发送至缓冲区以待执行。

语法 1

```
bool Circle(
    string,
    float[],
    float[],
    int,
    int,
    int,
    int,
    bool
)
```

参数

`string` 数据格式定义，由三个字母组成
#1：运动目标格式：
"C"：以笛卡尔坐标表示
#2：速度格式：
"P"：以百分比表示
"A"：以绝对速度表示，与项目速度同步
"D"：以绝对速度表示，与项目速度不同步
#3：混合格式：
"P"：以百分比表示

`float[]` 弧上的点。包含工具中心点的笛卡尔坐标：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)

`float[]` 弧的终点，包含工具中心点的笛卡尔坐标：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)

`int` 速度设置，以百分比（%）或速度（mm/s）表示

`int` 加速至最高速度的时间（ms）

`int` 混合值，以百分比（%）表示

`int` 弧角（°），若给定非零值，TCP将保持姿态不变，并通过弧上的给定点和终点从当前点移动至指定的弧角；若给定值为零，TCP将通过弧上的点从当前点移动至终点，期间对姿态进行线性插补以从当前姿态转为结束姿态。

`bool` 禁用精确定位
`true` 禁用精确定位
`false` 启用精确定位

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令（格式错误）

注

数据格式参数包括：(1) "CPP"、(2) "CAP"、(3) "CDP"

```
float[] PassP = {417.50,-122.30,343.90,180.00,0.00,90.00}
                    //声明一个浮点型数组以存储通过点值
float[] EndP = {381.70,208.74,343.90,180.00,0.00,135.00}
                    //声明一个浮点型数组以存储终点值
Circle("CAP", PassP, EndP,100,200,50,270,false)
                    //沿270°的圆弧移动，速度 = 100 mm/s，加速至最高速度的时间 = 200 ms，混合值 = 50%
```

语法 2

```
bool Circle(
    string,
    float, float, float, float, float, float,
    float, float, float, float, float, float,
    int,
    int,
    int,
    int,
    bool
)
```

参数

`string` 数据格式定义，由三个字母组成

#1：运动目标格式：

`"C"`：以笛卡尔坐标表示

#2：速度格式：

`"P"`：以百分比表示

`"A"`：以绝对速度表示，与项目速度同步

`"D"`：以绝对速度表示，与项目速度不同步

#3：混合格式：

`"P"`：以百分比表示

`float, float, float, float, float, float` 弧上的点。包含工具中心点的笛卡尔坐标：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)

`float, float, float, float, float, float` 弧的终点。包含工具中心点的笛卡尔坐标：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)

`int` 速度设置，以百分比（%）或速度（mm/s）表示

`int` 加速至最高速度的时间（ms）

`int` 混合值，以百分比（%）表示

`int` 弧角（°），若给定非零值，TCP将保持姿态不变，并通过弧上的给定点和终点从当前点移动至指定的弧角；若给定值为零，TCP将通过弧上的点从当前点移动至终点，期间对姿态进行线性插补以从当前姿态转为结束姿态。

`bool` 禁用精确定位

`true` 禁用精确定位

`false` 启用精确定位

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令（格式错误）

注

数据格式参数包括：(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)
      //沿270°的圆弧移动，速度 = 100 mm/s，加速至最高速度的时间 = 200 ms，混合值 = 50%，通过点 = 417.50,-122.30,343.90,180.00,0.00,90.00，终点 = 381.70,208.74,343.90,180.00,0.00,135
```

12.7 PLine()

定义 PLine 运动命令并发送至缓冲区以待执行。

语法 1

```
bool PLine(
    string,
    float[],
    int,
    int,
    int
)
```

参数

`string` 数据格式定义，由三个字母组成
#1：运动目标格式：
"J"：以关节角度表示
"C"：以笛卡尔坐标表示
#2：速度格式：
"A"：以绝对速度表示，与项目速度同步
"D"：以绝对速度表示，与项目速度不同步
#3：混合格式：
"P"：以百分比表示

`float[]` 运动目标。若以关节角度表示，则包含六个关节的角度：关节1(°)、关节2(°)、关节3(°)、关节4(°)、关节5(°)、关节6(°)；若以笛卡尔坐标表示，则包含工具中心点的笛卡尔坐标：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)

`int` 速度设置，以速度（mm/s）表示

`int` 加速至最高速度的时间（ms）

`int` 混合值，以百分比（%）表示

返回值

`bool` True 已接受命令；False 已拒绝命令（格式错误）

注

数据格式参数包括：(1) "CAP"、(2) "CDP"、(3) "JAP"、(4) "JDP"

```
float[] targetP1 = {417.50,-122.30,343.90,180.00,0.00,90.00}   //声明一个浮点型数组以存储目标坐标
PLine("CAP",targetP1,100,200,50)                                //以PLine运动移动至targetP1，速度 = 100 mm/s，加速至最高速度的时间 = 200 ms，混合值 = 50%
```

语法 2

```
bool PLine(
    string,
    float, float, float, float, float, float,
    int,
    int,
    int
)
```

参数

`string` 数据格式定义，由三个字母组成
#1：运动目标格式：
"C"：以笛卡尔坐标表示
"J"：以关节角度表示

#2：速度格式：
"A"：以绝对速度表示，与项目速度同步
"D"：以绝对速度表示，与项目速度不同步
#3：混合格式：
"P"：以百分比表示

`float, float, float, float, float, float,`
运动目标。若以关节角度表示，则包含六个关节的角度：关节1(°)、关节2(°)、关节3(°)、关节4(°)、关节5(°)、关节6(°)；若以笛卡尔坐标表示，则包含工具中心点的笛卡尔坐标：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)

`int` 速度设置，以速度（mm/s）表示

`int` 加速至最高速度的时间（ms）

`int` 混合值，以百分比（%）表示

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令（格式错误）

注

数据格式参数包括：(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)
//以PLine运动移动至417.50,-122.30,343.90,180.00,0.00,90.00，速度 = 100 mm/s，加速至最高速度的时间 = 200 ms，混合值 = 50%

12.8 Move_PTP()

定义并发送 PTP 相对运动命令以执行。

语法 1

```
bool Move_PTP(
    string,
    float[],
    int,
    int,
    int,
    bool
)
```

参数

`string` 数据格式定义，由三个字母组成
#1：相对运动目标格式：
"C"：相对于当前基准表示
"T"：相对于工具坐标表示
"J"：以关节角度表示
#2：速度格式：
"P"：以百分比表示
#3：混合格式：
"P"：以百分比表示

`float[]` 相对运动参数。若以坐标表示（相对于当前基准或工具坐标），则包含相对于特定坐标的工具端TCP相对运动值：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)；若以关节角度定义，则包含六个关节的角度：关节1(°)、关节2(°)、关节3(°)、关节4(°)、关节5(°)、关节6(°)

`int` 速度设置，以百分比（%）表示

`int` 加速至最高速度的时间（ms）

`int` 混合值，以百分比（%）表示

`bool` 禁用精确定位
`true` 禁用精确定位
`false` 启用

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令（格式错误）

注

运动命令参数包括：(1) "CPP"、(2) "TPP"或(3) "JPP"

```
float[] relmove = {0,0,10,45,0,0}                    //声明一个浮点型数组以存储相对运动目标
Move_PTP("TPP", relmove,10,200,0,false)              //通过PTP运动移动至相对运动目标，速度 = 10%，加速至最高速度的时间 = 200 ms
```

语法2

```
bool Move_PTP(
    string,
    float, float, float, float, float, float,
    int,
    int,
    int,
    bool
)
```

参数

`string` 数据格式定义，由三个字母组成
#1：相对运动目标格式：
"C"：相对于当前基准表示
"T"：相对于工具坐标表示
"J"：以关节角度表示
#2：速度格式：
"P"：以百分比表示
#3：混合格式：
"P"：以百分比表示

`float, float, float, float, float, float`
相对运动参数。若以坐标表示（相对于当前基准或工具坐标），则包含相对于特定坐标的工具端TCP相对运动值：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)；若以关节角度定义，则包含六个关节的角度：关节1(°)、关节2(°)、关节3(°)、关节4(°)、关节5(°)、关节6(°)

`int` 速度设置，以百分比（%）表示

`int` 加速至最高速度的时间（ms）

`int` 混合值，以百分比（%）表示

`bool` 禁用精确定位
`true` 禁用精确定位
`false` 启用

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令（格式错误）

注

运动命令参数包括：(1) `"CPP"`、(2) `"TPP"`和(3) `"JPP"`

```
Move_PTP("TPP",0,0,10,45,0,0,10,200,0,false)    //通过PTP运动移动至0,0,10,45,0,0（相对于工具坐标），速度 = 10%，加速至最高速度的时间 = 200 ms
```

12.9 Move_Line()

定义并发送 Line 相对运动命令以执行。

语法 1

```
bool Move_Line(
    string,
    float[],
    int,
    int,
    int,
    bool
)
```

参数

`string` 数据格式定义，由三个字母组成
#1：相对运动目标格式：
"C"：相对于当前基准表示
"T"：相对于工具坐标表示
#2：速度格式：
"P"：以百分比表示
"A"：以绝对速度表示，与项目速度同步
"D"：以绝对速度表示，与项目速度不同步
#3：混合格式：
"P"：以百分比表示
"R"：以半径表示

`float[]` 相对运动参数。包含相对于特定坐标（当前基准或工具坐标）的工具端TCP相对运动值：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)。

`int` 速度设置，以百分比（%）或速度（mm/s）表示

`int` 加速至最高速度的时间（ms）

`int` 混合值，以百分比（%）或半径（mm）表示

`bool` 禁用精确定位
`true` 禁用精确定位
`false` 启用

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令（格式错误）

注

运动命令参数包括：(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}                    //声明一个浮点型数组以存储相对运动目标
Move_Line("TAP", relmove,125,200,0,false)            //通过直线运动移动至相对运动目标，速度 = 125 mm/s，加速至最高速度的时间 = 200 ms
```

语法2

```
bool Move_Line(
    string,
    float, float, float, float, float, float,
    int,
    int,
    int,
    bool
)
```

参数

`string` 数据格式定义，由三个字母组成

#1：相对运动目标格式：

"C"：相对于当前基准表示

"T"：相对于工具坐标表示

#2：速度格式：

"P"：以百分比表示

"A"：以绝对速度表示，与项目速度同步

"D"：以绝对速度表示，与项目速度不同步

#3：混合格式：

"P"：以百分比表示

"R"：以半径表示

`float, float, float, float, float, float` 相对运动参数。包含相对于特定坐标（当前基准或工具坐标）的工具端TCP相对运动值：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)。

`int` 速度设置，以百分比（%）或速度（mm/s）表示

`int` 加速至最高速度的时间（ms）

`int` 混合值，以百分比（%）或半径（mm）表示

`bool` 禁用精确定位

`true` 禁用精确定位

`false` 启用

返回值

`bool` True 已接受命令；False 已拒绝命令（格式错误）

注

运动命令参数包括：(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)    //通过直线运动移动至相对运动目标0,0,10,45,0,0，速度 = 125 mm/s，加速至最高速度的时间 = 200 ms。
```

12.10 Move_PLine()

定义并发送 PLine 相对运动命令以执行。

语法 1

```
bool Move_PLine(
    string,
    float[],
    int,
    int,
    int
)
```

参数

`string` 数据格式定义，由三个字母组成
#1：相对运动目标格式：
"C"：相对于当前基准表示
"T"：相对于工具坐标表示
"J"：以关节角度表示
#2：速度格式：
"A"：以绝对速度表示，与项目速度同步
"D"：以绝对速度表示，与项目速度不同步
#3：混合格式：
"P"：以百分比表示

`float[]` 若以坐标表示（相对于当前基准或工具坐标），则包含相对于特定坐标的工具端TCP相对运动值：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)；若以关节角度定义，则包含六个关节的角度：关节1(°)、关节2(°)、关节3(°)、关节4(°)、关节5(°)、关节6(°)

`int` 速度设置，以速度（mm/s）表示

`int` 加速至最高速度的时间（ms）

`int` 混合值，以百分比（%）表示

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令（格式错误）

注

运动命令参数包括：
(1) "CAP"、(2) "CDP"、(3) "TAP"、(4) "TDP"、(5) "JAP"、(6) "JDP"

```
float[] target = {0,0,10,45,0,0}          //声明一个浮点型数组以存储相对运动目标
Move_PLine("CAP", target,125,200,0)       //通过PLine运动移动至相对运动目标，速度 = 125 mm/s，
                                          加速至最高速度的时间 = 200 ms。
```

语法 2

```
bool Move_PLine(
    string,
    float, float, float, float, float, float,
    int,
    int,
    int,
)
```

参数

`string` 数据格式定义，由三个字母组成

#1：相对运动目标格式：

"C"：相对于当前基准表示

"T"：相对于工具坐标表示

"J"：以关节角度表示

#2：速度格式：

"A"：以绝对速度表示，与项目速度同步

"D"：以绝对速度表示，与项目速度不同步

#3：混合格式：

"P"：以百分比表示

`float, float, float, float, float, float`

相对运动参数。若以坐标表示（相对于当前基准或工具坐标），则包含相对于特定坐标的工具端TCP相对运动值：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)；若以关节角度定义，则包含六个关节的角度：关节1(°)、关节2(°)、关节3(°)、关节4(°)、关节5(°)、关节6(°)

`int` 速度设置，以速度（mm/s）表示

`int` 加速至最高速度的时间（ms）

`int` 混合值，以百分比（%）表示

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令（格式错误）

注

运动命令参数包括：

(1) "CAP"、(2) "CDP"、(3) "TAP"、(4) "TDP"、(5) "JAP"、(6) "JDP"

```
Move_PLine("CAP",0,0,10,45,0,0,125,200,0)    //通过PLine运动移动至0,0,10,45,0,0，速度 = 125 mm/s，加速至最高速度的时间 = 200 ms
```

12.11 ChangeBase()

设置连续运动参考的基准。

语法 1

```
bool ChangeBase(
    string,
    int
)
```

参数

string	基准名称
int	基准索引。用户可用其指定通过视觉一次性获取全部创建的多重基准中的基准。索引可为0到N，默认值为0。

返回值

bool	True 已接受命令；False 已拒绝命令

语法 2

```
bool ChangeBase(
    string
)
```

注

与语法 1 相同。基准索引默认为 0。

```
ChangeBase("RobotBase")
                        //将基准变更为"RobotBase"
ChangeBase("vision_job1", 1)
                        //将基准变更为"vision_job1"的第二个基准值。若指定的基准名称或基准索引不存在，将返回错误。
ChangeBase("RobotBase", 10)
                        //将基准变更为"RobotBase"。由于此为系统基准名称，基准索引无效。
```

语法 3

```
bool ChangeBase(
    float[]
)
```

参数

float[]	基准参数，由X、Y、Z、RX、RY、RZ组成

返回值

True 已接受命令；False 已拒绝命令

注

```
float[] Base1 = {20,30,10,0,0,90}        //声明一个浮点型数组以存储基准值
ChangeBase(Base1)                        //将基准变更为指定的基准。
```

语法 4

```
bool ChangeBase(
    float, float, float, float, float, float
)
```

参数

float, float, float, float, float, float	基准参数，由X、Y、Z、RX、RY、RZ组成

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令

注

ChangeBase(20,30,10,0,0,90) //将基准值变更为{20,30,10,0,0,90}

12.12 ChangeTCP()

将变更后续运动的 TCP 的命令发送至缓冲区以待执行。

语法 1

```
bool ChangeTCP(
    string
)
```

参数

`string` TCP名称

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令（格式错误）

注

```
ChangeTCP("NOTOOL")          //将 TCP 变更为"NOTOOL"。
```

语法 2

```
bool ChangeTCP(
    float[]
)
```

参数

`float[]` TCP参数，由X、Y、Z、RX、RY、RZ组成

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令（格式错误）

注

```
float[] Tool1 = {0,0,150,0,0,90}      //声明一个浮点型数组以存储 TCP 值
ChangeTCP(Tool1)                      //将 TCP 值变更为 Tool1
```

语法 3

```
bool ChangeTCP(
    float[],
    float
)
```

参数

`float[]` TCP参数，由X、Y、Z、RX、RY、RZ组成

`float` 工具重量

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令（格式错误）

注

```
float[] Tool1 = {0,0,150,0,0,90}      //声明一个浮点型数组以存储 TCP 值
ChangeTCP(Tool1,2)                    //将 TCP 值变更为 Tool1，其重量 = 2 kg
```

语法 4

```
bool ChangeTCP(
    float[],
    float,
    float[]
)
```

参数

`float[]` TCP参数，由X、Y、Z、RX、RY、RZ组成

`float` 工具重量

`float[]` 工具的转动惯量：(1)Ixx、(2)Iyy、(3)Izz，及其参考框架：(4)X、(5)Y、(6)Z、(7)RX、(8)RY、(9)RZ

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令（格式错误）

注

```
float[] Tool1 = {0,0,150,0,0,90}                //声明一个浮点型数组以存储 TCP 值
float[] COM1 = {2,0.5,0.5,0,0,-80,0,0,0}        //声明一个浮点型数组以存储转动惯量及其参考框架
ChangeTCP(Tool1,2, COM1)                        //将 TCP 值变更为 Tool1，其重量 = 2 kg，转动惯量为 COM1
```

语法 5

```
bool ChangeTCP(
    float, float, float, float, float, float
)
```

参数

`float, float, float, float, float, float`
TCP参数，由X、Y、Z、RX、RY、RZ组成

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令（格式错误）

注

```
ChangeTCP(0,0,150,0,0,90)                       //将 TCP 值变更为{0,0,150,0,0,90}
```

语法 6

```
bool ChangeTCP(
    float, float, float, float, float, float,
    float
)
```

参数

`float, float, float, float, float, float`
TCP参数，由X、Y、Z、RX、RY、RZ组成

`float` TCP重量

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令（格式错误）

注

```
ChangeTCP(0,0,150,0,0,90,2)                     //将 TCP 值变更为{0,0,150,0,0,90}，重量 = 2 kg
```

语法 7

```
bool ChangeTCP(
    float, float, float, float, float, float,
    float,
    float, float, float, float, float, float, float, float, float
)
```

参数

`float, float, float, float, float, float`
TCP参数，由X、Y、Z、RX、RY、RZ组成

`float` 工具重量

`float, float, float, float, float, float, float, float, float`
工具的转动惯量：(1)Ixx、(2)Iyy、(3)Izz，及其参考框架：(4)X、(5)Y、(6)Z、(7)RX、(8)RY、(9)RZ

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令（格式错误）

注

ChangeTCP(0,0,150,0,0,90,2, 2,0.5,0.5,0,0,-80,0,0,0) //将 TCP 值变更为{0,0,150,0,0,90}，其重量 = 2 kg，转动惯量 = {2,0.5,0.5}，参考框架 = {0,0,-80,0,0,0}

12.13 ChangeLoad()

将变更后续运动的有效载荷值的命令发送至缓冲区以待执行。

语法 1

```
bool ChangeLoad(
    float
)
```

参数

`float` 有效载荷（kg）

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令（格式错误）

注

```
ChangeLoad(5.3)                                    //将有效载荷设为 5.3 kg
```

12.14 PVTEnter()

设置以启动关节/笛卡尔命令 PVT 模式

语法 1

```
bool PVTEnter(
    int
)
```

参数

int	PVT模式	
	0	关节
	1	笛卡尔

返回值

bool　True 已接受命令；False 已拒绝命令

注

语法 2

```
bool PVTEnter(
)
```

参数

void　无参数。默认使用关节命令PVT模式。

返回值

bool　True 已接受命令；False 已拒绝命令

注

PVTEnter(1)

12.15 PVTExit()

设置以退出 PVT 模式运动

语法 1

```
bool PVTExit(
)
```

参数

`void` 无参数

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令

注

PVTExit()

12.16 PVTPoint()

以位置、速度和持续时间设置 PVT 模式运动参数。

语法 1

```
bool PVTPoint(
    float[],
    float[],
    float
)
```

参数

float[]	目标位置 对于关节PVT模式，为{J1, J2, J3, J4, J5, J6} 对于笛卡尔PVT模式，为{X, Y, Z, RX, RY, RZ}
float[]	目标速度 对于关节PVT模式，为{J1, J2, J3, J4, J5, J6}' 对于笛卡尔PVT模式，为{X, Y, Z, RX, RY, RZ}'
float	时长（单位为秒）

返回值

bool	True 已接受命令；False 已拒绝命令

注

语法 2

```
bool PVTPoint(
    float, float, float, float, float, float,
    float, float, float, float, float, float,
    float
)
```

参数

float, float, float, float, float, float,	目标位置。 对于关节PVT模式，为{J1, J2, J3, J4, J5, J6} 对于笛卡尔PVT模式，为{X, Y, Z, RX, RY, RZ}
float, float, float, float, float, float,	目标速度。 对于关节PVT模式，为{J1, J2, J3, J4, J5, J6}' 对于笛卡尔PVT模式，为{X, Y, Z, RX, RY, RZ}'
float	时长（单位为秒）

返回值

bool	True 已接受命令；False 已拒绝命令

注

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()

设置以暂停 PVT 模式运动

语法 1

```
bool PVTPause(
)
```

参数

`void` 无参数

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令

注

PVTPause()

12.18 PVTResume()

设置以恢复 PVT 模式运动

语法 1

```
bool PVTResume(
)
```

参数

`void` 无参数

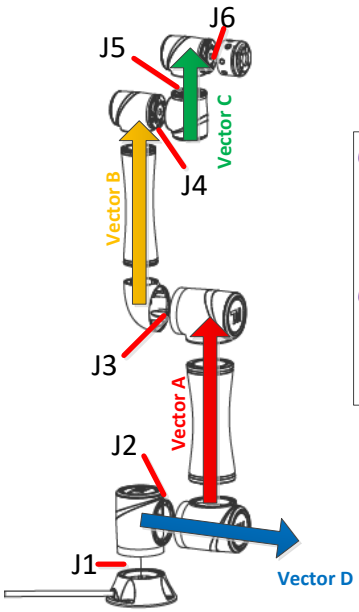
返回值

`bool` `True` 已接受命令；`False` 已拒绝命令

注

PVTResume()

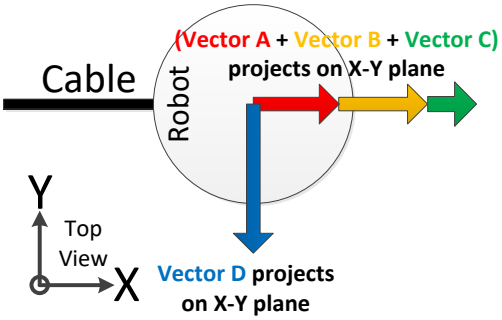
姿态配置参数：[Config1, Config2, Config3]



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()

执行项目中现有的视觉任务，但不执行具有初始位置的视觉任务。
*必须先在项目中创建视觉任务，且不勾选**从初始位置开始**。
*若使用脚本项目，则需要在定义部分中编写相应的变量定义。
*视觉任务执行后，相应的变量定义值将更新。

语法 1

```
bool Vision_DoJob(
    string
)
```

参数

string	视觉任务名称。

返回值

bool	True	视觉任务完成且结果为成功。	
	False	视觉任务失败。	1.结果为失败。 2.执行失败。 3.视觉任务有初始点。

注

bool var_result = **Vision_DoJob**("job1")

12.20 Vision_DoJob_PTP()

执行项目中现有的视觉任务，通过 PTP 运动移动至初始位置，并使用运动命令排队。
*必须先在项目中创建视觉任务。如需移动至初始位置，请勾选**从初始位置开始**，否则请取消勾选。
*若使用脚本项目，则需要在定义部分中编写相应的变量定义。
*视觉任务执行后，相应的变量定义值将更新。

语法 1

```
bool Vision_DoJob_PTP(
    string,
    int,
    int,
    bool
)
```

参数

string	视觉任务名称。	
int	速度设置，以百分比（%）表示	
int	加速至最高速度的时间（ms）	
bool	是否使用机器人选择模式智能姿态	
	true	根据系统使用机器人姿态。
	false	使用对视觉任务进行示教时记录的机器人姿态。

返回值

bool	True	视觉任务完成且结果为成功。	
	False	视觉任务失败。	1.结果为失败。 2.执行失败。

语法 2

```
bool Vision_DoJob_PTP(
    string,
    int,
    int
)
```

注

与语法 1 相同。“是否使用机器人选择模式智能姿态”参数默认为 false。

bool var_result = **Vision_DoJob_PTP**("job1", 100, 500)
var_result = **Vision_DoJob_PTP**("job1", 100, 500, true)

12.21 Vision_DoJob_Line()

执行项目中现有的视觉任务，通过直线运动移动至初始位置，并使用运动命令排队。
*必须先在项目中创建视觉任务。如需移动至初始位置，请勾选**从初始位置开始**，否则请取消勾选。
*若使用脚本项目，则需要在定义部分中编写相应的变量定义。
*视觉任务执行后，相应的变量定义值将更新。

语法 1

```
bool Vision_DoJob_Line(
    string,
    int
)
```

参数

string	视觉任务名称	
int	速度设置，以百分比（%）表示	

返回值

bool	True	视觉任务完成且结果为成功。	
	False	视觉任务失败。	1.结果为失败。 2.执行失败。

注

bool var_result = **Vision_DoJob_Line**("job1", 100)

语法 2

```
bool Vision_DoJob_Line(
    string,
    int,
    int
    bool
)
```

参数

string	视觉任务名称	
int	速度设置，以速度（mm/s）表示	
int	加速至最高速度的时间（ms）	
bool	是否关联项目速度。	
	true	关联项目速度。
	false	取消关联项目速度。

返回值

bool	True	视觉任务完成且结果为成功。	
	False	视觉任务失败。	1.结果为失败。 2.执行失败。

语法 3

```
bool Vision_DoJob_Line(
    string,
    int,
    int
)
```

注

与语法2相同。“是否关联项目速度”参数默认为true。

bool var_result = **Vision_DoJob_Line**("job1", 100, 500)
var_result = **Vision_DoJob_Line**("job1", 100, 500, false)

13. 视觉函数

13.1 Vision_IsJobAvailable()

检查当前项目中的视觉任务是否存在且有效

语法 1

```
bool Vision_IsJobAvailable(
    string
)
```

参数

string		视觉任务名称

返回值

bool	True	视觉任务存在且有效
	False	视觉任务不存在或无效

注

请在调用视觉任务前使用该函数检查其是否存在且有效，以免调用时因不存在或无效出现错误。

bool var_result = **Vision_IsJobAvailable**("job1")

13.2 Vision_GetOutputArraySize()

在视觉任务执行后获取二维输出变量的数组大小。

语法 1

```
int Vision_GetOutputArraySize(
    string
)
```

参数

string	视觉任务输出的变量名，数据类型为字符串数组。

返回值

int	查找对象的索引。（二维数组的行数）

语法 2

```
int Vision_GetOutputArraySize(
    string,
    int
)
```

参数

string	视觉任务输出的变量名，数据类型为字符串数组。
int	查找对象的索引。（二维数组的行索引）

返回值

int	指定行的索引。（行索引）。获取要定位的指定对象的输出结果数组大小。

13.3 Vision_GetOutputArrayValue()

在视觉任务执行后获取二维输出变量的数组值。

语法 1

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

参数

string	视觉任务输出的变量名，数据类型为字符串数组。
int	查找对象的索引。（二维数组的行索引）

返回值

string[]	指定行索引的输出结果数组值（行索引）。

语法 2

```
string Vision_GetOutputArrayValue(
    string,
    int,
    int
)
```

参数

string	视觉任务输出的变量名，数据类型为字符串数组。
int	查找对象的索引。（二维数组的行索引）
int	指定列索引的输出结果数组值

返回值

string	指定对象和位置的输出结果值。

注

视觉任务执行后，若以二维数据数组形式输出结果，则需要使用该函数获取相关信息。即通过以参数代入变量名获取视觉任务和模块的结果数据。例如，"job1_Count_Blob_1_DetectObjectX_TM"为视觉任务 job1 的输出变量，由计数（斑点）模块 1 输出。若输出值是二维数组，则数组中的每行代表一个查找对象结果，而每列代表多个结果。

例如，结果数组为{ {"0", "1", "2"}, {"3", "4"}, {"5"} }时

获取查找对象数

```
int var_count = Vision_GetOutputArraySize("job1_Count_Blob_1_DetectObjectX_TM")
  //获取变量 job1_Count_Blob_1_DetectObjectX_TM 的行数（即查找对象数）
  // var_count = 3
int var_length = Vision_GetOutputArraySize("job1_Count_Blob_1_DetectObjectX_TM", 1)
  //获取位于变量 job1_Count_Blob_1_DetectObjectX_TM 的行 1（第二个对象）的输出结果数组大小
  // var_length = 2
```

获取查找对象的结果值

```
string[] var_ss = Vision_GetOutputArrayValue("job1_Count_Blob_1_DetectObjectX_TM", 0)
  //获取位于变量 job1_Count_Blob_1_DetectObjectX_TM 的行 0（第一个对象）的输出结果
  // var_ss = {"0", "1", "2"}
string var_s = Vision_GetOutputArrayValue("job1_Count_Blob_1_DetectObjectX_TM", 2, 0)
  //获取位于变量 job1_Count_Blob_1_DetectObjectX_TM 的行 2（第三个对象）、列 0（项目 1）的输出结果
  // var_s = "5"
```

```
string var_se = Vision_GetOutputArrayValue("job1_Count_Blob_1_DetectObjectX_TM", 3, 0)
  //返回错误。超出访问范围（查找对象数：3，索引可为：0~2）
```

13.4 Vision_GetTriggerJobOutputCount()

在视觉 IO 触发任务执行后获取缓冲区内的输出变量数。

语法 1

```
int Vision_GetTriggerJobOutputCount(
    string
)
```

参数

string	IO触发任务输出的变量名，数据类型为字符串数组。

返回值

int	缓冲区内指定变量的输出变量数。

13.5 Vision_GetTriggerJobOutputValue()

在视觉 IO 触发任务执行后获取输出变量值。（获取并从缓冲区内删除。）

语法 1

```
string[] Vision_GetTriggerJobOutputValue(
    string
)
```

参数

`string`　　IO触发任务输出的变量名，数据类型为字符串数组。

返回值

`string[]`　　缓冲区内指定变量的输出值。

注

视觉 IO 触发任务执行后（视觉编辑时必须启用 IO 触发模式），结果将随每次 IO 触发更新至变量中。由于 IO 触发任务的输出是多路输出，为防止输出数据因离开进程被覆写，输出数据将被逐个存储在缓冲区内（先进先出），用户可通过调用函数获取缓冲区内的计数或值。例如，job1_Count_Blob_1_DetectObjectX_TM 为视觉任务 job1 的输出变量，由计数（斑点）模块 1 输出。视觉任务执行后，该变量将随每次 IO 触发更新并将结果存储至缓冲区内，因此用户可通过调用函数获取缓冲区内的变量 job1_Count_Blob_1_DetectObjectX_TM 的计数或值。

缓冲区的容量有限。若结果即将进入时接收缓冲区的容量不足，将自动删除最早的数据、将最新的数据添加至缓冲区内。

假设当前缓冲区内的内容为{ {"0", "1", "2"}, {"3", "4"}, {"5"} }

获取缓冲区内视觉触发任务的输出变量数。

```
int var_count = Vision_GetTriggerJobOutputCount ("job1_Count_Blob_1_DetectObjectX_TM")
  // var_count = 3
```

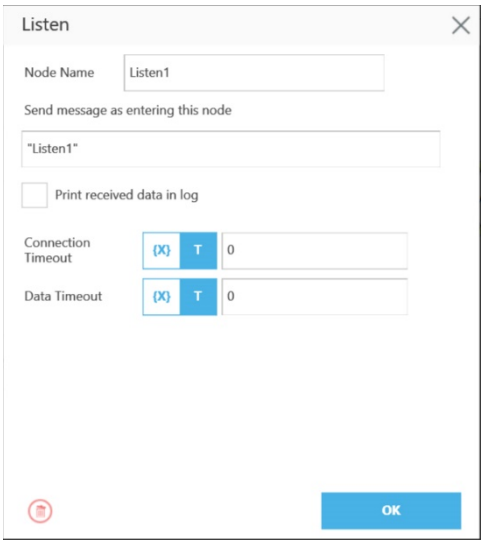
获取缓冲区内的输出变量值。

```
string[] var_s = Vision_GetTriggerJobOutputValue ("job1_Count_Blob_1_DetectObjectX_TM")
  // var_s = {"0", "1", "2"}
  //获取后，缓冲区内的内容变为{ {"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"}
  //获取后，缓冲区内的内容变为{ {"5"} }。
var_s = Vision_GetTriggerJobOutputValue ("job1_Count_Blob_1_DetectObjectX_TM")
  // var_s = {"5"}
  //获取后，缓冲区内的内容变为 { }。
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. 外部命令

14.1 Listen 节点

用户可在 Listen 节点中建立 SocketTCPListener（服务器站点），以连接外部设备并基于数据包格式通信。TM_Robot_Function 中的所有功能均可在 Listen 节点中运行。



1. **发送消息：**进入此节点时发送一条消息
2. **打印日志：**启用通信日志（显示在右侧）
3. **连接超时：**进入此节点时，若超过一定时间（单位为毫秒）未连接，就会超时。
   若 <= 0，则不会超时
4. **数据超时：**连接后，无通信数据包时将超时
   若 <= 0，则不会超时

Socket TCPListener 于项目执行后启动，于项目停止时关闭。Socket TCPListener 启动后，IP 和监听端口将显示在右侧的通知日志窗口中。

IP　　HMI → 系统 → 网络 → IP 地址

端口　　5890

进入 Listen 节点后，流程将停留在 Listen 节点，直至满足两个退出条件中的任意一个。

成功：　*ScriptExit()*已执行或项目已停止

失败：　1.连接超时
　　2.数据超时
　　3.在 TCP Listener 启动前，流程已进入该 Listen 节点

将按顺序执行 Listen 节点接收到的命令。若命令无效，将返回一条包含存在错误的行的编号的错误消息。有效命令将被执行。

命令分为两类。第一类是可立即完成的命令，如变量赋值。第二类是需要按顺序执行的命令，如运动命令和 IO 赋值。第二类命令将被置于队列中并按顺序执行。

14.2 ScriptExit()

退出外部命令控制模式。

语法 1

```
bool ScriptExit(
)
```

参数

`void` 无参数

返回值

`bool` `True` 已接受命令；`False` 已拒绝命令（格式错误）

注

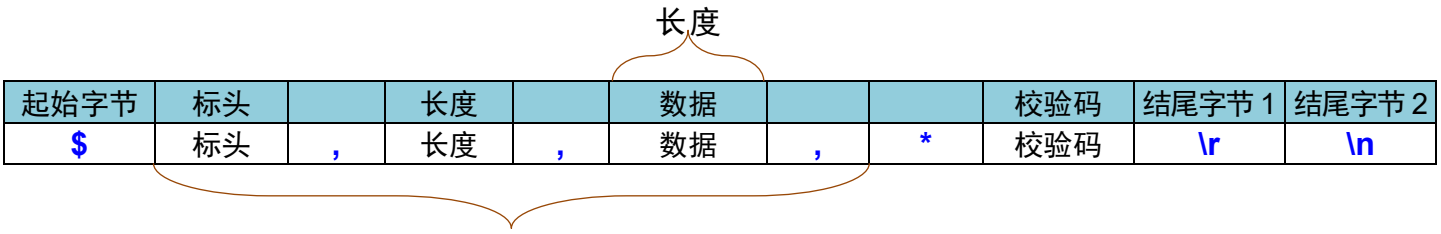
退出外部命令控制模式，等待命令执行完毕，然后退出Listen节点并沿成功路线继续。

*通过TMSCT通信数据包执行
*将不会执行ScriptExit()之后的函数，例如

```
<  $TMSCT,78,2,ChangeBase("RobotBase")\r\n
   ChangeTCP("NOTOOL")\r\n
   ScriptExit()\r\n                    //退出外部命令控制模式。
   ChangeLoad(10.1),*6C\r\n            //不会执行 ChangeLoad。
```

*退出脚本模式后，需要等待所有命令和函数执行完毕，然后才会退出Listen节点并沿成功路线继续。等待退出Listen节点时，Listen节点不处于外部命令控制模式，因此不会再接受外部命令，也不会回复CPEER错误数据包。

14.3 通信协议



名称	大小	ASCII	十六进制	说明
起始字节	1	$	0x24	通信起始字节
标头	*X*			通信标头
分隔符	1	,	0x2C	标头和长度间的分隔符
长度	*Y*			数据长度
分隔符	1	,	0x2C	长度和数据间的分隔符
数据	*Z*			通信数据
分隔符	1	,	0x2C	数据和校验码间的分隔符
符号	1	*	0x2A	校验码开始符号
校验码	*2*			通信校验码
结尾字节 1	1	\r	0x0D	
结尾字节 2	1	\n	0x0A	通信结尾字节

1. 标头

定义通信数据包的用途。数据定义可能因标头而异。

- *TMSCT* 外部命令
- *TMSTA* 获取状态或属性
- *CPERR* 通信数据错误（如封包错误、校验码错误、标头错误等）

2. ***长度***

长度以 UTF8 字节形式定义长度。可以十进制、十六进制或二进制表示，上限为 int 32 位

示例：

```
$TMSCT,100,Data,*CS\r\n          //十进制 100，即数据长度为 100 字节
$TMSCT,0x100,Data,*CS\r\n        //十六进制 0x100，即数据长度为 256 字节
$TMSCT,0b100,Data,*CS\r\n        //二进制 0b100，即数据长度为 4 字节
$TMSCT,8,1,達明,*58\r\n          //数据 1,達明的长度为 8 字节（UTF8）
```

3. ***数据***

通信数据包的内容。支持任意字符（包括 UTF8 中的 0x00~0xFF）。长度中定义了数据的长度，标头中定义了数据的用途

4. ***校验码***

通信数据包的校验码。校验码通过 XOR（异或）计算，校验码计算范围为从$到*（不包括$和*），如下所示：

```
$TMSCT,100,Data,*CS\r\n
```

Checksum = Byte[1] ^ Byte[2] … ^ Byte[N-6]

校验码的表示形式固定为 2 个十六进制格式字节（不含 0x）。

例如：

```
$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

起始字节	标头		长度		数据			校验码	结尾字节 1	结尾字节 2
$	TMSCT	,	长度	,	数据	,	*	校验码	\r	\n

ID		脚本
脚本 ID	,	脚本语言

TMSCT 将通信数据包定义为外部命令语言。外部命令语言数据包含用逗号分隔的两个部分。其中一个部分是 ID，另一个是脚本

ID　　脚本ID，可为任意英文字母或数字（遇到非字母数字的字节时将报告CPERR 04错误）。ID用于指定返回消息的目标脚本。

,　　分隔符

SCRIPT　以脚本语言定义的内容。在通信数据包中，可利用分隔符（0x0D 0x0A）将多行脚本纳入脚本部分

注

只能在外部命令控制模式下使用 TMSCT，否则将回复 CPEER 错误数据包。

返回值（机器人→外部设备）

1. 进入 Listen 节点时，机器人将向所有连接的设备发送一条消息。ID 被设为 0。

   $***TMSCT***,9,0,Listen1,*4C\r\n

   9　　*0,Listen1* 的长度为 9 字节

   0　　脚本 ID 为 0

   ,　　分隔符

   Listen1　要发送的消息

2. 将根据脚本内容回复 OK 或 ERROR 消息。对于带有;N 的消息，;N 代表存在错误或警告的行的编号。接收消息后，若脚本有效，机器人将执行消息，然后发回返回消息。若脚本无效，则不会执行，立即发回返回消息。

   $***TMSCT***,4,1,OK,*5C\r\n　　//对 ID 1 的响应
   //OK 表示脚本有效。

   $***TMSCT***,8,2,OK;2;3,*52\r\n　　//对 ID 2 的响应
   //OK;2;3 表示脚本有效，但其行 2 和行 3 中存在警告。

   $***TMSCT***,13,3,ERROR;1;2;3,*3F\r\n　　//对 ID 3 的响应
   //ERROR;1;2;3 表示脚本无效，其行 1、行 2 和行 3 中存在错误。

接收（机器人←外部设备）

1. 进入 Listen 节点时，机器人将开始接收、检查并执行外部命令。若机器人未进入 Listen 节点（不处于外部命令控制模式），则接收的脚本将被弃置，并回复 CPEER 错误数据包。
2. 来自外部设备的消息应定义脚本 ID，作为用于机器人回复的消息中的 ID。

   < $***TMSCT***,25,1,ChangeBase("RobotBase"),*08\r\n　　//定义为 ID 1
   > $***TMSCT***,4,1,OK,*5C\r\n　　//对 ID 1 的响应

3. 在通信数据包中，可利用分隔符\r\n 将多行脚本纳入脚本部分

   < $***TMSCT***,64,2,ChangeBase("RobotBase")\r\n
   ChangeTCP("NOTOOL")\r\n
   ChangeLoad(10.1),*68\r\n　　//通信数据包中三行脚本（行间以\r\n 分隔）

> $***TMSCT***,4,2,OK,*5F\r\n

4. Listen 节点中的局部变量仅在退出 Listen 节点前可用且有效。

< $***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 //由于已声明 int var_i，发生错误。

5. 在 Listen 节点中，可访问或修改项目变量，但不能声明新变量，因为在 Listen 节点中创建的变量为局部变量。

14.5 TMSTA

起始字节	标头		长度		数据			校验码	结尾字节 1	结尾字节 2
$	*TMSTA*	,	长度	,	数据	,	*	校验码	\r	\n

SubCmd		
SubCmd	…	…

（基于 SubCmd）

TMSTA 将通信数据包定义为获取状态或属性。数据包的数据部分包含不同的子命令（SubCmd）。数据包格式可能因子命令而异。定义如下所示。

SubCmd

00　　是否处于外部命令控制模式

01　　配置的队列标签编号是否已完成

90..99　要发送的日期消息（可自定义数据格式）

注

执行 TMSTA 无需进入 Listen 节点

***SubCmd　00*　是否处于外部命令控制模式**

格式

响应（机器人→外部设备）

SubCmd		是否进入		消息
00	,	false	,	
00	,	true	,	消息

接收（机器人←外部设备）

SubCmd
00

响应（机器人→外部设备）

1. 若不处于外部命令控制模式，将返回 false。

   $***TMSTA***,9,00,false,,*37\r\n

   9　　表示 00,false 的长度为 9 字节

   00　　表示 SubCmd 为 00

   ,　　分隔符

   false　　流程未进入 Listen 节点

   ,　　分隔符

   　　空字符串（未进入 Listen 节点）

2. 若处于外部命令控制模式，将返回 true。

   $***TMSTA***,15,00,true,Listen1,*79\r\n

   15　　表示 00,true,Listen1 的长度为 15 字节

   00　　表示 SubCmd 为 00

   ,　　分隔符

   true　　流程已进入 Listen 节点

   ,　　分隔符

   Listen1　　要发送的消息，如处于 Listen 节点中（表示流程处于 Listen1 中）

接收（机器人←外部设备）

1. 从外部设备发送至机器人

   $***TMSTA***,2,00,*41\r\n

2　　表示 00 的长度为 2 字节。
00　　表示 SubCmd 为 00（无论是否处于外部命令控制模式）。

SubCmd　01 配置的队列标签编号是否已完成

格式

发送（机器人→外部设备）

SubCmd		标签号		状态
01	,	01 .. 15	,	true/false/none

接收（机器人←外部设备）

SubCmd		标签号
01	,	01 .. 15

注

使用 TMSTA 01 查询时，用户可查看最后 4 个标签号的状态。

发送（机器人→外部设备）

1. 从机器人发送至外部设备。队列标签编号完成后自动发送。

   $***TMSTA***,10,01,08,true,*6D\r\n

   10　　表示 01,00,true 的长度为 10 字节
   01　　表示 SubCmd 为 01（发送标签号状态）
   ,　　分隔符
   08　　标签号 08
   ,　　分隔符
   true　　*true*　　表示标签号已完成
   　　false　　表示标签号未完成
   　　none　　表示标签号不存在

接收（机器人←外部设备）

1. 从外部设备发送至机器人。用户可查看最后 4 个标签号的状态。

   $***TMSTA***,5,01,15,*6F\r\n

   5　　表示 01,88 的长度为 5 字节
   01　　表示 SubCmd 为 01（发送标签号状态）
   ,　　分隔符
   15　　标签号 15

   >　$***TMSTA***,10,01,15,none,*7D\r\n　　//标签号 15 不存在

2. 标签号使用从 1 到 15 的整数值。若值无效，将回复 none 表示不存在。

   $***TMSTA***,5,01,88,*6B\r\n

   >　$***TMSTA***,10,01,88,none,*79\r\n　　//标签号 88 不存在

SubCmd　90~99 发送数据消息

格式

发送（机器人→外部设备）

SubCmd		数据
90~99	,	…

接收（机器人←外部设备）

无

注

1. 使用 TMSTA 90~99 发送消息时，用户可使用自定义格式。
2. 自定义格式表示格式由项目流程和外部设备共同定义。
3. 为提升使用灵活性，用户可使用 90~99 的多个 SubCmd 定义不同发送格式，例如：

将 SubCmd 90 定义为字符串；
将 SubCmd 91 定义为 float[]；
将 SubCmd 92 定义为 byte[]
…

诸如此类，供外部设备基于 SubCmd 使用不同方法分析和解析。

发送（机器人→外部设备）

1. 从机器人发送至外部设备。于外部命令执行 ListenSend()函数时发送数据。

```
string var_s = "Hello World"
float[] var_f = {1,2,3,4}
byte[] var_b = {0x10, 0x11, 0x12, 0x13}
ListenSend(90, var_s)
//通信内容 $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)
//通信内容 $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)
//通信内容 $TMSTA,7,92,…,*63\r\n
// 0x39,0x32,0x2C,0x10,0x11,0x12,0x13
```

14.6 CPERR

起始字节	标头		长度		数据			校验码	结尾字节 1	结尾字节 2
$	CPERR	,	长度	,	数据	,	*	校验码	\r	\n

错误代码
代码（00~FF）

CPERR 将通信数据包定义为发送通信协议错误。数据部分被定义为错误代码。

错误代码	错误代码，以2个十六进制格式字节（不含0x）表示
00	数据包正确。无错误。（返回的消息通常会回复数据包的内容，而非返回无错误）
01	封包错误。
02	校验码错误。
03	标头错误。
04	数据包数据错误。
F1	未进入Listen节点

注

机器人用于响应外部设备

响应（机器人→外部设备）

01 封包错误

```
<   $TMSCT,-100,1,ChangeBase("RobotBase"),*13\r\n   //长度不可为负
>   $CPERR,2,01,*49\r\n                              //CPERR 错误代码 01
```

02 校验码错误

```
<   $TMSCT,25,1,ChangeBase("RobotBase"),*09\r\n      //09 不是正确的校验码
>   $CPERR,2,02,*4A\r\n                              //CPERR 错误代码 02
```

03 标头错误

```
<   $TMsct,25,1,ChangeBase("RobotBase"),*28\r\n      //TMsct 不是正确的标头
>   $CPERR,2,03,*4B\r\n                              //CPERR 错误代码 03
```

04 数据包数据错误

```
<   $TMSTA,4,XXXX,*47\r\n                            //TMSTA 下没有 XXXX SubCmd
>   $CPERR,2,04,*4C\r\n                              //CPERR 错误代码 04
```

F1 非外部命令模式

```
<   $TMSCT,25,1,ChangeBase("RobotBase"),*0D\r\n
    //假设当前不处于外部命令控制模式
>   $CPERR,2,F1,*3F\r\n                              //CPERR 错误代码 F1
```

14.7 优先命令

鉴于 TMscript 语法具有串行执行特性，若使用 QueueTag(1, 1)或 WaitQueueTag(1)等队列语法等待标签号到达，程序将停滞不前，直至条件满足后再继续执行。因此，若在等待时接收到外部发送的 ScriptExit()语法，则无法退出外部命令控制模式，因为程序仍在等待条件满足。

程序进入 Listen 节点（外部命令控制模式）时，除串行执行 TMscript 语法外，还可使用优先命令。在 Listen 节点中，优先命令的执行优先级高于语法，优先命令将按以下定义立即运行。

1. ***ScriptExit(0)***
   立即令机器人运动停止，清除缓冲区内的机器人运动指令，退出外部命令控制模式。离开 Listen 节点后，转至失败路径。

2. ***ScriptExit(1)***
   立即令机器人运动停止，清除缓冲区内的机器人运动指令，退出外部命令控制模式。离开 Listen 节点后，转至成功路径。

3. ***StopAndClearBuffer(0)***
   立即令机器人运动停止，清除缓冲区内的机器人运动指令。

4. ***StopAndClearBuffer(1)***
   立即令机器人运动停止，清除缓冲区内的机器人运动指令，退出正在执行的当前脚本程序，继续执行下一个脚本程序。

5. ***StopAndClearBuffer(2)***
   立即令机器人运动停止，清除缓冲区内的机器人运动指令，退出正在执行的当前脚本程序，清除接收脚本缓冲区内的所有脚本程序。

- 优先命令仅支持命令定义的常规使用，而不支持与变量和函数搭配使用，例如：

```
<   $TMSCT,42,1,int var_st=2\r\n
    StopAndClearBuffer(var_st),*3A\r\n
>   $TMSCT,9,1,ERROR;2,*04\r\n          //StopAndClearBuffer(var_st)语法无效
```

- 并用优先命令和 TMscript 语法将导致仅执行优先命令，而不执行语法。

```
<   $TMSCT,94,2,float[] targetP1= {0,0,90,0,90,0}\r\n     //不会执行。
    PTP("JPP",targetP1,10,200,0,false)\r\n                //不会执行。
    StopAndClearBuffer(0),*75\r\n                         //优先执行 StopAndClearBuffer(0)
>   $TMSCT,4,2,OK,*5F\r\n
```

- 对于每个外部命令数据包，系统只会处理其中的一条优先命令。若数据包含有多条优先命令，系统将先处理 ScriptExit(0/1)，然后再处理 StopAndClearBuffer(0/1/2)。若分别存在多条 ScriptExit 和 StopAndClearBuffer，系统将只处理第一条。

```
< $TMSCT,46,3,StopAndClearBuffer(2)\r\n     //将会执行。
                                            //存在多条 StopAndClearBuffer，系统将仅处理其中的第一条。
    StopAndClearBuffer(1),*68\r\n           //不会执行。
>   $TMSCT,4,3,OK,*5E\r\n
<   $TMSCT,61,4,StopAndClearBuffer(2)\r\n   //不会执行。
    StopAndClearBuffer(1)\r\n               //不会执行。
    ScriptExit(1),*52\r\n                   //将会执行。
                                            //系统将先处理优先级更高的 ScriptExit，然后处理 StopAndClearBuffer。
```

> $***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)等待。
程序将停留在该命令，直至标签 1 执行完毕。
< $***TMSCT***,15,2,ScriptExit(0),*55\r\n //接收到 ScriptExit(0)命令时，若标签 1 未执行完毕，
> $***TMSCT***,4,1,OK,*5C\r\n //响应为 1 OK
//由于运动命令和 QueueTag()被清除，不会因执行完毕发送 TMSTA 标签号。
> $***TMSCT***,4,2,OK,*5F\r\n //响应为 2 OK
将退出外部命令控制模式，并在离开 Listen 节点后转至失败路径。

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)等待。
程序将停留在该命令，直至标签 1 执行完毕。
< $***TMSCT***,23,2,StopAndClearBuffer(0),*55\r\n
//接收到 StopAndClearBuffer(0)命令时，若标签 1 未执行完毕
> $***TMSCT***,4,1,OK,*5C\r\n //响应为 1 OK
//由于运动命令和 QueueTag()被清除，不会因执行完毕发送 TMSTA 标签号。
> $***TMSCT***,4,2,OK,*5F\r\n //响应为 2 OK
不会退出外部命令控制模式，仍然在 Listen 节点内。

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 //标签 0 将等待 60 秒的超时时间。
PTP("JPP",targetP1,40,200,0,false),*64\r\n //将等待 60 秒，然后执行。
< $***TMSCT***,23,2,StopAndClearBuffer(0),*55\r\n
//若在 60 秒内接收到 StopAndClearBuffer(0)命令，将清除第一个 PTP()函数。
> $***TMSCT***,4,2,OK,*5F\r\n //响应为 2 OK
将等待 60 秒的超时时间，然后执行第二个 PTP()函数。响应为 1 OK。
> $***TMSCT***,4,1,OK,*5C\r\n //响应为 1 OK

*StopAndClearBuffer(0)只会清除运动命令，不会清除函数语法和逻辑运算。虽然 WaitQueueTag(0)被设为数字 0，但会等待超时，因此无法在清除运动命令后退出。必须使用 StopAndClearBuffer(1/2)命令才能退出当前正在执行的脚本程序。

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)等待。
程序将停留在该命令，直至标签 1 执行完毕。
< $***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 //上一个数据包仍在等待，不会执行此数据包。
< $***TMSCT***,23,3,StopAndClearBuffer(1),*55\r\n
//接收到 StopAndClearBuffer(1)命令时，若标签 1 未执行完毕
> $***TMSCT***,4,1,OK,*5C\r\n //响应为 1 OK
//由于运动命令和 QueueTag()被清除，不会因执行完毕发送 TMSTA 标签号。
> $***TMSCT***,4,3,OK,*5E\r\n //响应为 3 OK

将清除并退出当前正在执行的脚本 1，然后继续执行接下来的脚本 2。

> $***TMSCT***,4,2,OK,*5F\r\n　　//响应为 2 OK

> $***TMSTA***,10,01,02,true,*67\r\n　　//标签 2 执行完毕

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)等待。
程序将停留在该命令，直至标签 1 执行完毕。

< $***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　　//上一个数据包仍在等待，不会执行此数据包。

< $***TMSCT***,23,3,StopAndClearBuffer(2),*56\r\n
//接收到 StopAndClearBuffer(2)命令时，若标签 1 未执行完毕

> $***TMSCT***,4,1,OK,*5C\r\n　　//响应为 1 OK
//由于运动命令和 QueueTag()被清除，不会因执行完毕发送 TMSTA 标签号。

> $***TMSCT***,4,3,OK,*5E\r\n　　//响应为 3 OK

将清除并退出当前正在执行的脚本 1，然后清除已接收脚本缓冲区内的所有脚本程序。因此，将清除脚本 2 而不响应脚本 2。

15. Modbus 函数
15.1 ModbusTCP 类

可通过使用 Modbus TCP 类并声明变量创建 Modbus TCP 设备。变量名将成为设备名称。

构造 1

ModbusTCP *VariableName* = `string, int, int`
ModbusTCP *VariableName* = `string, int`
ModbusTCP *VariableName* = `string`

参数

`string`	远程主机IP地址	
`int`	远程主机连接端口	（默认为502）
`int`	读/写超时时间，单位为毫秒	0~10000 （默认为10000 ms）

注

```
ModbusTCP mtcp1 = "192.168.1.10"
  //构造一个设备，其IP为192.168.1.10
ModbusTCP mtcp2 = "192.168.1.10", 502
  //构造一个设备，其IP为192.168.1.10、端口为502
ModbusTCP mtcp3 = "192.168.1.10", 502, 8000
  //构造一个设备，其IP为192.168.1.10、端口为502、超时时间为8000 ms
```

*在流程项目或脚本项目中构造设备后，不会主动连接设备，直至开始读取或写入。

成员方法

名称	说明
Preset()	配置预设 ModbusTCP 参数。
IODDPreset()	读取 IODD 文件并配置预设 ModbusTCP 参数。

15.1.1 Preset()

配置预设 ModbusTCP 参数。

语法 1

```
bool Preset(
    string,
    byte,
    string,
    int,
    string,
    int
)
```

参数

`string`	预设设备名称	
`byte`	从站ID	
`string`	信号类型	"DO"、"DI"、"RO"、"RI"
`int`	起始地址	
`string`	类型	"bool"、"byte"、"int16"、"int32"、"float"、"double"、"string"
`int`	后缀参数，	
	类型为"int32"时	`0`：小端序（CD AB） `1`：大端序（AB CD） （默认值）
	类型为"float"时	`0`：小端序（CD AB） `1`：大端序（AB CD） （默认值）

类型为"double"时　　0：小端序（CD AB）　1：大端序（AB CD）　（默认值）
类型为"string"时　　地址数（默认为0）
对于其他类型无效。

返回值

`bool`　预设成功为`True`，预设失败为`False`

语法 2

```
bool Preset(
    string,
    byte,
    string,
    int,
    string
)
```

注

与语法 1 相同。默认将后缀参数设为默认值。

注

```
ModbusTCP mbus1 = "127.0.0.1"
    //构造一个设备，其IP为127.0.0.1、端口为502、超时时间为8000 ms

mbus1.Preset("light", 1, "DO", 7206, "bool")        //将预设设备名称设为"light"
mbus1.Preset("9000", 1, "RO", 9000, "string")       //将预设设备名称设为"9000"
    //返回错误，必须以字母和数字的组合命名。

mbus1.Preset("preset_9000", 1, "RO", 9000, "int", 0)
    //将预设设备名称设为"preset_9000"//小端序
mbus1.Preset("preset_9000", 1, "RO", 9000, "int", 1)
    //将预设设备名称设为"preset_9000"//大端序
mbus1.Preset("preset_9000", 1, "RO", 9000, "int")
    //将预设设备名称设为"preset_9000"//大端序
mbus1.Preset("preset_9000", 1, "RO", 9000, "string", 5)
    //将预设设备名称设为"preset_9000"//字符串类型
    //若名称 preset_9000 存在，将通过处理序列以内容覆写同名配置。
bool flag = modbus_read("mbus1", "light")        // flag = false    //假设相机照明已关闭
modbus_write("mbus1", "light", true)             // write true      //相机照明已开启
flag = modbus_read("mbus1", "light")             // flag = true

modbus_write("mbus1 ", "preset_9000", Ctrl("\0\0\0\0\0\0\0\0\0\0"))
  //清空 preset_9000，使其成为占用 5 个地址（5 RO = 10 字节）的字符串类型

modbus_write("mbus1", "preset_9000", 1234)              //写入"1234"      //由于 preset_9000 为字符串类型
int var_i = modbus_read("mbus1", "preset_9000")     // var_i = 1234   //将尝试将"1234"转换为整数。
  //由于 preset_9000 为字符串类型且占用 5 个地址，将读取 10 字节并按字符串转换。（遇到 0x00 时结束。）

modbus_write("mbus1", "preset_9000", "HelloWorld")      //写入"HelloWorld"
string var_s = modbus_read("mbus1", "preset_9000")      // var_s = "HelloWorld"
var_i = modbus_read("mbus1", "preset_9000")
//返回错误，无法将 HelloWorld 转换为整数。
```

```
modbus_write("mbus1", "preset_9000", 1234)                    //写入"1234"
  //继数据的最后部分之后，写入"HelloWorld"          //因此，当前地址 9000 内的数据为"1234oWorld"
var_i = modbus_read("mbus1", "preset_9000")
//返回错误，无法将 1234oWorld 转换为整数。
```

15.1.2 IODDPreset()

读取 IODD 文件并配置预设 ModbusTCP 参数。

语法 1

```
bool IODDPreset(
    string,
    byte,
    int,
    int
)
```

参数

string	IODD 文件名（读取存储在本地主机.\XmlFiles\IODD 目录下的 IODD 文件）
byte	从站ID
int	起始地址输入
int	起始地址输出

返回值

bool	预设成功为True，预设失败为False

注

```
ModbusTCP mbus1 = "192.168.1.10"
//构造一个设备，其IP为192.168.1.10、端口为502、超时时间为10000 ms

mbus1.IODDPreset("OMRON-E2EQ-X3B4-IL2-20170301-IODD1.1.xml", 1, 100, 200)
  //加载文件.\XmlFiles\IODD\OMRON-E2EQ-X3B4-IL2-20170301-IODD1.1.xml 并添加预设配置参数。
  //定义预设名称的方法与流程项目规则相同。
```

15.2 ModbusRTU 类

可通过使用 Modbus TCP 类并声明变量创建 Modbus TCP 设备。变量名将成为设备名称。

构造

```
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
```

参数

string	连接说明		
int	位/秒，波特率		
string	奇偶校验	"none"、"odd"、"even"、"mark"、"space"（默认为"none"）	
int	数据位	5、6、7、8	（默认为8）
float	停止位	1、1.5、2	（默认为1）
int	读/写超时时间，单位为毫秒	0~10000	（默认为10000 ms）
bool	DTR/DSR	true、false	（默认为false）
bool	RTS/CTS	true、false	（默认为false）
bool	XON/XOFF	true、false	（默认为false）

注

```
ModbusRTU mrtu1 = "COM2",115200
  //构造一个设备，其波特率为 115200
ModbusRTU mrtu2 = "COM2",115200,"none",8,1
  //构造一个设备，其波特率为 115200、奇偶校验为 none、数据位为 8、停止位为 1
ModbusRTU mrtu3 = "COM2",115200,"none",8,1,10000
  //构造一个设备，其波特率为 115200、奇偶校验为 none、数据位为 8、停止位为 1、超时时间为 10000 ms
```

*在流程项目或脚本项目中构造设备后，不会主动连接设备，直至开始读取或写入。

成员方法

名称	说明
Preset()	配置预设 ModbusTCP 参数。
IODDPreset()	读取 IODD 文件并配置预设 ModbusTCP 参数。

15.2.1 Preset()

配置预设 ModbusTCP 参数。

语法 1

```
bool Preset(
    string,
    byte,
    string,
    int,
    string,
    int
)
```

参数

string	预设设备名称	
byte	从站ID	
string	信号类型	"DO"、"DI"、"RO"、"RI"
int	起始地址	
string	类型	"bool"、"byte"、"int16"、"int32"、"float"、"double"、"string"
int	后缀参数，	
	类型为"int32"时	0：小端序（CD AB） 1：大端序（AB CD）（默认值）
	类型为"float"时	0：小端序（CD AB） 1：大端序（AB CD）（默认值）
	类型为"double"时	0：小端序（CD AB） 1：大端序（AB CD）（默认值）
	类型为"string"时	地址数（默认为0）
	对于其他类型无效。	

返回值

bool	预设成功为True，预设失败为False

语法 2

```
bool Preset(
    string,
    byte,
    string,
    int,
    string
)
```

注

与语法 1 相同。默认将后缀参数设为默认值。

注

```
ModbusRTU mbus1 = "COM2",115200
  //构造一个设备，其波特率为 115200、奇偶校验为 none、数据位为 8、停止位为 1、超时时间为 10000 ms

mbus1.Preset("light", 1, "DO", 7206, "bool")       //将预设设备名称设为"light"
mbus1.Preset("9000", 1, "RO", 9000, "string")      //将预设设备名称设为"9000"
    //返回错误，必须以字母和数字的组合命名。

mbus1.Preset("preset_9000", 1, "RO", 9000, "int", 0)
    //将预设设备名称设为"preset_9000"//小端序
mbus1.Preset("preset_9000", 1, "RO", 9000, "int", 1)
    //将预设设备名称设为"preset_9000"//大端序
mbus1.Preset("preset_9000", 1, "RO", 9000, "int")
    //将预设设备名称设为"preset_9000"//大端序
mbus1.Preset("preset_9000", 1, "RO", 9000, "string", 5)
    //将预设设备名称设为"preset_9000"//字符串类型
    //若名称 preset_9000 存在，将通过处理序列以内容覆写同名配置。

bool flag = modbus_read("mbus1", "light")       // flag = false    //假设相机照明已关闭
modbus_write("mbus1", "light", true)            // write true      //相机照明已开启
flag = modbus_read("mbus1", "light")            // flag = true

modbus_write("mbus1 ", "preset_9000", Ctrl("\0\0\0\0\0\0\0\0\0\0"))
  //清空 preset_9000，使其成为占用 5 个地址（5 RO = 10 字节）的字符串类型
```

```
modbus_write("mbus1", "preset_9000", 1234)   //写入"1234"     //由于 preset_9000 为字符串类型
int var_i = modbus_read("mbus1", "preset_9000")     // var_i = 1234   //将尝试将"1234"转换为整数。
  //由于 preset_9000 为字符串类型且占用 5 个地址，将读取 10 字节并按字符串转换。（遇到 0x00 时结束。）

modbus_write("mbus1", "preset_9000", "HelloWorld")      //写入"HelloWorld"
string var_s = modbus_read("mbus1", "preset_9000")      // var_s = "HelloWorld"
var_i = modbus_read("mbus1", "preset_9000")
//返回错误，无法将 HelloWorld 转换为整数。

modbus_write("mbus1", "preset_9000", 1234)              //写入"1234"
  //继数据的最后部分之后，写入"HelloWorld"    //因此，当前地址 9000 内的数据为"1234oWorld"
var_i = modbus_read("mbus1", "preset_9000")
//返回错误，无法将 1234oWorld 转换为整数。
```

15.2.2 IODDPreset()

读取 IODD 文件并配置预设 ModbusTCP 参数。

语法 1

```
bool IODDPreset(
    string,
    byte,
    int,
    int
)
```

参数

`string`	IODD 文件名（读取存储在本地主机.\XmlFiles\IODD 目录下的 IODD 文件）
`byte`	从站ID
`int`	起始地址输入
`int`	起始地址输出

返回值

`bool`	预设成功为`True`，预设失败为`False`

注

```
ModbusRTU mbus1 = "COM2",115200
  //构造一个设备，其波特率为 115200、奇偶校验为 none、数据位为 8、停止位为 1、超时时间为 10000 ms

mbus1.IODDPreset("OMRON-E2EQ-X3B4-IL2-20170301-IODD1.1.xml", 1, 100, 200)
  //加载文件.\XmlFiles\IODD\OMRON-E2EQ-X3B4-IL2-20170301-IODD1.1.xml 并添加预设配置参数。
  //定义预设名称的方法与流程项目规则相同。
```

15.3 modbus_open()

开启至 Modbus TCP/RTU 设备的连接。

语法 1

```
bool modbus_open(
    string
)
```

参数

`string` TCP/RTU 设备名称

返回值

`bool`	`True`	开启成功
	`False`	开启失败

注

```
ModbusTCP mbus1 = "127.0.0.1"      //构造一个设备，其IP为127.0.0.1、端口为502、超时时间为10000 ms
modbus_open("mbus1")               //连接至IP为127.0.0.1、端口为502的设备。
```

15.4 modbus_close()

关闭自 Modbus TCP/RTU 设备的连接。

语法 1

```
bool modbus_close(
    string
)
```

参数

`string` TCP/RTU 设备名称

返回值

`bool`	`True`	关闭成功
	`False`	关闭失败

注

```
ModbusTCP mbus1 = "127.0.0.1"      //构造一个设备，其IP为127.0.0.1、端口为502、超时时间为10000 ms
modbus_open("mbus1")               //连接至IP为127.0.0.1、端口为502的设备。
modbus_close("mbus1")              //关闭连接。
```

15.5 modbus_read()

Modbus TCP/RTU 读取函数

语法 1（TCP/RTU）

```
? modbus_read(
    string,
    string
)
```

参数

`string` TCP/RTU 设备名称

`string` 属于TCP/RTU设备的预定义参数

返回值

`?` 返回值的数据类型取决于预定义参数

信号类型	功能代码	类型	地址编号	返回值的数据类型
数字输出	01	byte	1	byte (H: 1)(L: 0)
		bool	1	bool (H: true)(L: false)
数字输入	02	byte	1	byte (H: 1)(L: 0)
		bool	1	bool (H: true)(L: false)
寄存器输出	03	byte	1	byte
		int16	1	int
		int32	2	int
		float	2	float
		double	4	double
		string	?	string
		bool	1	bool
寄存器输入	04	byte	1	byte
		int16	1	int
		int32	2	int
		float	2	float
		double	4	double
		string	?	string
		bool	1	bool

*将自动根据小端序（CD AB）或大端序（AB CD）设置转换 int32、float、double 数据。

*字符串将遵循 UTF8 数据格式转换（止于 0x00）

注

Modbus 地址数据大小

数字 1 个地址 = 1 位大小

寄存器 1 个地址 = 2 字节大小

若在预设设置中应用默认值

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 个地址 = 1 位
value = modbus_read("TCP_1", "preset_7202")     // value = true         // DI 1 个地址 = 1 位
value = modbus_read("TCP_1", "preset_9000")     // value = ab1234cd     // RO 4 个地址 = 8 字节大小
```

```
value = modbus_read("TCP_1", "preset_7001")      // value = -314.1593  // RI 2 个地址 = 4 字节大小（float）
```

语法 2（TCP/RTU）

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

参数

string	TCP/RTU 设备名称	
byte	从站ID	
string	读取类型	
DO	数字输出	（FC 01读取线圈状态）
DI	数字输入	（FC 02读取输入状态）
RO	寄存器输出	（FC 03读取保持寄存器）
RI	寄存器输入	（FC 04读取输入寄存器）
int	起始地址	
int	数据长度	

返回值

byte[] 从Modbus服务器返回的字节数组
*用户定义的modbus_read读取字节数组时仅遵循大端序（AB CD）格式

注

Modbus 地址数据大小

数字	1 个地址 = 1 位大小
寄存器	1 个地址 = 2 字节大小

若将用户定义的值应用于用户设置，如：

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 个地址 = 4 位（对于字节数组）
value = modbus_read("TCP_1", 0, "DI", 7202, 3)
    // value = {1,0,0}            // DI 3 个地址 = 3 位（对于字节数组）
value = modbus_read("TCP_1", 0, "RO", 9000, 6)
    // value = {0x54,0x65,0x63,0x68,0x6D,0x61,0x6E,0xE9,0x81,0x94,0xE6,0x98}
    // RO 6 个地址 = 12 字节大小
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 个地址 = 24 字节大小
```

```
value = modbus_read("TCP_1", 0, "RI", 7301, 6)
    // value = {0x07,0xE2,0x00,0x05,0x00,0x12,0x00,0x0F,0x00,0x0A,0x00,0x39}
    // RI 6 个地址  = 12 字节大小
```

15.6 modbus_read_int16()

Modbus TCP/RTU 读取函数，并将 Modbus 地址数据数组转换为 int16 数组

语法 1（TCP/RTU）

```
int[] modbus_read_int16(
    string,
    byte,
    string,
    int,
    int,
    int
)
```

参数

`string` TCP/RTU 设备名称

`byte` 从站ID

`string` 读取类型

`DO`	数字输出	（FC 01读取线圈状态）
`DI`	数字输入	（FC 02读取输入状态）
`RO`	寄存器输出	（FC 03读取保持寄存器）
`RI`	寄存器输入	（FC 04读取输入寄存器）

`int` 起始地址

`int` 数据长度

`int` 将地址数据转换为int16数组时遵循小端序（CD AB）还是大端序（AB CD）。*无效参数。仅支持int32、float、double类型

`0` 小端序

`1` 大端序（默认值）

返回值

`int[]` 从Modbus服务器返回的整型数组

语法 2（TCP/RTU）

```
int[] modbus_read_int16(
    string,
    byte,
    string,
    int,
    int
)
```

注

与语法 1 类似，采用大端序（AB CD）设置

modbus_read_int16("TCP_1", 0, "DI", 7200, 2) => **modbus_read_int16**("TCP_1", 0, "DI", 7200, 2, 1)

Modbus 地址数据大小

数字	1 个地址 = 1 位大小
寄存器	1 个地址 = 2 字节大小

若将用户定义的值应用于用户设置，如：

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}    转换为 int16[]    value = {0,0}     // byte[0][1] , byte[2][3]
value = modbus_read_int16("TCP_1", 0, "DI", 7202, 3)
    // byte[] = {1,0,0}              转换为 int16[]    value = {256,0}
    // byte[0][1] , byte[2][3] //自动填充至[3]
value = modbus_read_int16("TCP_1", 0, "RO", 9000, 6)
    // byte[] = {0x54,0x65,0x63,0x68,0x6D,0x61,0x6E,0xE9,0x81,0x94,0xE6,0x98}
    //转换为 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}
    //转换为 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}
    //转换为 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}
    //转换为 int16[]  value = {2018,5,18,15,49,35}
```

15.7 modbus_read_int32()

Modbus TCP/RTU 读取函数，并将 Modbus 地址数据数组转换为 int32 数组

语法 1（TCP/RTU）

```
int[] modbus_read_int32(
    string,
    byte,
    string,
    int,
    int,
    int
)
```

参数

string	TCP/RTU 设备名称	
byte	从站ID	
string	读取类型	
	DO	数字输出 （FC 01读取线圈状态）
	DI	数字输入 （FC 02读取输入状态）
	RO	寄存器输出 （FC 03读取保持寄存器）
	RI	寄存器输入 （FC 04读取输入寄存器）
int	起始地址	
int	数据长度	
int	将地址数据转换为int32数组时遵循小端序（CD AB）还是大端序（AB CD）。	
	0	小端序
	1	大端序（默认值）

返回值

int[]	从Modbus服务器返回的整型数组

语法 2（TCP/RTU）

```
int[] modbus_read_int32(
    string,
    byte,
    string,
    int,
    int
)
```

注

与语法 1 类似，采用大端序（AB CD）设置。

modbus_read_int32("TCP_1", 0, "DI", 7200, 2) => **modbus_read_int32**("TCP_1", 0, "DI", 7200, 2, 1)

Modbus 地址数据大小

数字	1 个地址 = 1 位大小
寄存器	1 个地址 = 2 字节大小

若将用户定义的值应用于用户设置，如：

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}    转换为 int32[]    value = {0}// byte[0][1][2][3]
value = modbus_read_int32("TCP_1", 0, "DI", 7202, 3)
    // byte[] = {1,0,0}              转换为 int32[]    value = {16777216}    // byte[0][1][2][3] //自动填充至[3]。
value = modbus_read_int32("TCP_1", 0, "RO", 9000, 6)
    // byte[] = {0x54,0x65,0x63,0x68,0x6D,0x61,0x6E,0xE9,0x81,0x94,0xE6,0x98}
    //转换为 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}
    //转换为 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}
    //转换为 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}
    //转换为 int32[]  value = {0x000507E2,0x000F0012,0x00230031} = {329698,983058,2293809}
```

15.8 modbus_read_float()

Modbus TCP/RTU 读取函数，并将 Modbus 地址数据数组转换为浮点型数组

语法 1（TCP/RTU）

```
float[] modbus_read_float(
    string,
    byte,
    string,
    int,
    int,
    int
)
```

参数

string	TCP/RTU 设备名称	
byte	从站ID	
string	读取类型	
	DO	数字输出 （FC 01读取线圈状态）
	DI	数字输入 （FC 02读取输入状态）
	RO	寄存器输出 （FC 03读取保持寄存器）
	RI	寄存器输入 （FC 04读取输入寄存器）
int	起始地址	
int	数据长度	
int	将地址数据转换为浮点型数组时遵循小端序（CD AB）还是大端序（AB CD）。	
	0	小端序
	1	大端序（默认值）

返回值

float[]	从Modbus服务器返回的浮点型数组

语法 2（TCP/RTU）

```
float[] modbus_read_float(
    string,
    byte,
    string,
    int,
    int
)
```

注

与语法 1 类似，采用大端序（AB CD）设置。

modbus_read_float("TCP_1", 0, "DI", 7200, 2) => **modbus_read_float**("TCP_1", 0, "DI", 7200, 2, 1)

Modbus 地址数据大小

数字	1 个地址 = 1 位大小
寄存器	1 个地址 = 2 字节大小

若将用户定义的值应用于用户设置，如：

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}          转换为 float[]    value = {0}// byte[0][1][2][3]
value = modbus_read_float("TCP_1", 0, "DI", 7202, 3)
    // byte[] = {1,0,0}            转换为 float[]    value = {2.350989E-38}    // byte[0][1][2][3]
    //自动填充至[3]。
value = modbus_read_float("TCP_1", 0, "RO", 9000, 6)
    // byte[] = {0x54,0x65,0x63,0x68,0x6D,0x61,0x6E,0xE9,0x81,0x94,0xE6,0x98}
    //转换为 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}
    //转换为 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]
    //转换为 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}
    //转换为 float[]          value = {3.400471E-34,1.65306E-39,5.326512E-39}
```

15.9 modbus_read_double()

Modbus TCP/RTU 读取函数，并将 Modbus 地址数据数组转换为双精度浮点型数组

语法 1（TCP/RTU）

```
double[] modbus_read_double(
    string,
    byte,
    string,
    int,
    int,
    int
)
```

参数

string	TCP/RTU 设备名称	
byte	从站ID	
string	读取类型	
	DO	数字输出 （FC 01读取线圈状态）
	DI	数字输入 （FC 02读取输入状态）
	RO	寄存器输出 （FC 03读取保持寄存器）
	RI	寄存器输入 （FC 04读取输入寄存器）
int	起始地址	
int	数据长度	
int	将地址数据转换为双精度浮点型数组时遵循小端序（CD AB）还是大端序（AB CD）。	
	0	小端序
	1	大端序（默认值）

返回值

double[]	从Modbus服务器返回的双精度浮点型数组

语法 2（TCP/RTU）

```
double[] modbus_read_double(
    string,
    byte,
    string,
    int,
    int
)
```

注

与语法 1 类似，采用大端序（AB CD）设置。

modbus_read_double("TCP_1", 0, "DI", 7200, 2) => **modbus_read_double**("TCP_1", 0, "DI", 7200, 2, 1)

Modbus 地址数据大小

数字	1 个地址 = 1 位大小
寄存器	1 个地址 = 2 字节大小

若将用户定义的值应用于用户设置，如：

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}    转换为 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}    转换为 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}
    //转换为 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}
    //转换为 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]
    //转换为 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}
    //转换为 double[]    value = {1.06475148078395E-270,1.52982527955113E-308}
```

15.10 modbus_read_string()

Modbus TCP/RTU 读取函数，并将 Modbus 地址数据数组转换为 UTF8 字符串文本

语法 1（TCP/RTU）

```
string modbus_read_string(
    string,
    byte,
    string,
    int,
    int,
    int
)
```

参数

`string` TCP/RTU 设备名称

`byte` 从站ID

`string` 读取类型

- `DO` 数字输出 （FC 01读取线圈状态）
- `DI` 数字输入 （FC 02读取输入状态）
- `RO` 寄存器输出 （FC 03读取保持寄存器）
- `RI` 寄存器输入 （FC 04读取输入寄存器）

`int` 起始地址

`int` 数据长度

`int` 将地址数据转换为字符串时遵循小端序（CD AB）还是大端序（AB CD）。*无效参数。仅支持int32、float、double类型。字符串遵循UTF8，并根据地址依次传输。

- `0` 小端序
- `1` 大端序（默认值）

返回值

`string` 从Modbus服务器返回的UTF8字符串（止于0x00）

语法 2（TCP/RTU）

```
string modbus_read_string(
    string,
    byte,
    string,
    int,
    int
)
```

注

与语法 1 类似，采用大端序（AB CD）设置。

modbus_read_string("TCP_1", 0, "RO", 9000, 2) => **modbus_read_string**("TCP_1", 0, "RO", 9000, 2, 1)

Modbus 地址数据大小

数字	1 个地址 = 1 位大小
寄存器	1 个地址 = 2 字节大小

若将用户定义的值应用于用户设置，如：

TCP device	0	RO	9000	12

```
modbus_write("TCP_1", 0, "RO", 9000) = "1234 達明机器手臂"
        //未定义要写入的地址数，默认值 0 表示写入长度为 22 字节的完整数据。
        //写入 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 个地址  = 6 字节大小
        //转换为 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}
        //转换为 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}
        //转换为 string = 1234 達明机器手臂 AB          //UTF8 格式转换
        //写入数据时，将不会写入结尾 0x00。读取 12 个地址时，将超出范围读取。
modbus_write("TCP_1", 0, "RO", 9000) = "1234"+Ctrl("\0")
        //写入 byte[] = {0x31,0x32,0x33,0x34,0x00}      //需要写入 3 个寄存器地址
value = modbus_read_string("TCP_1", 0, "RO", 9000, 5)
        // byte[] = {0x31,0x32,0x33,0x34,0x00,0x00, 0x94,0xE6,0x98,0x8E}    //最后 4 个值为这些地址中的
        原始数据
        //转换为 string = 1234      //UTF8 格式转换，止于 0x00
```

15.11 modbus_write()

Modbus TCP/RTU 写入函数

语法 1（TCP/RTU）

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

参数

`string` TCP/RTU 设备名称

`string` 属于TCP/RTU设备的TCP/RTU预定义参数

`?` 输入的数据。将根据下表应用预定义参数。

信号类型	功能代码	类型	输入类型	输入的值
数字输出	05	byte	byte	(H: 1)(L: 0)
		bool	bool	(H: true)(L: false)
寄存器输出	06	byte	byte	
		bool	bool	
		int16	int	
寄存器输出	16	int32	int	
		float	float	
		double	double	
		string	string	

*将根据用户设置按小端序（CD AB）或大端序（AB CD）转换 int32、float、double 数据。

*将按 UTF8 格式转换字符串

*不支持使用预定义参数写入数组值。如需写入数组值，应使用用户定义的方法（语法 3/4）

`int` 要写入的最大地址数，仅对字符串类型数据有效

`> 0` 有效地址长度。按定义的地址长度写入

`<= 0` 无效地址长度。写入全部数据

若跳过该参数（如语法2所示），将应用预定义地址长度。

返回值

`bool` `True` 写入成功

`False` 写入失败 1.输入的数据?为空字符串或空数组
2.Modbus通信发生错误

语法 2（TCP/RTU）

```
bool modbus_write(
    string,
    string,
    ?,
)
```

注

与语法 1 类似，按预定义地址长度写入。若预定义地址长度 <= 0，将写入全部数据。

Modbus 地址数据大小

数字 1 个地址 = 1 位大小

寄存器 1 个地址 = 2 字节大小

若将用户定义的值应用于用户设置，如：

```
preset_800      DO      800     bool
preset_9000     RO      9000    string    4
```

```
modbus_write("TCP_1", "preset_800", 1)          //写入 1（true）
modbus_write("TCP_1", "preset_800", 0)          //写入 0（false）
bool flag = true
modbus_write("TCP_1", "preset_800", flag)       //写入 1（true）
modbus_write("TCP_1", "preset_800", false)      //写入 0（false）

string ss = "ABCDEFGHIJKLMNOPQRST"              //由于无地址数，应用预定义地址长度 4。即可写入 4 RO = 8 字节大小。
modbus_write("TCP_1", "preset_9000", ss)        //写入 ABCDEFGH    //超出范围部分将被跳过
                                                //由于无地址数，将应用预定义地址长度 4。即可写入 4 RO = 8 字节大小。
modbus_write("TCP_1", "preset_9000", "1234567")         //写入 1234567\0      //使用 0x00 填充至 4 个地址
                                                        //由于地址长度 = 0，写入全部数据。"09AB123"需要 4 个地址。
modbus_write("TCP_1", "preset_9000", "09AB123", 0)      //写入 09AB123\0      //使用 0x00 填充至 4 个地址
                                                        //由于地址长度 =5, 写入 5 个地址的数据。即可写入 5 RO = 10 字节大小。
modbus_write("TCP_1", "preset_9000", "09AB1234", 5)     //写入 09AB1234       //输入的数据仅需 4 个地址。
```

语法 3（TCP/RTU）

```
bool modbus_write(
    string,
    byte,
    string,
    int,
    ?,
    int
)
```

参数

`string` TCP/RTU 设备名称

`byte` 从站ID

`string` 写入类型

- `DO` 数字输出 （FC 05/15写入单个/多个线圈）
- `RO` 寄存器输出 （FC 06/16写入单个/多个寄存器）

`int` 起始地址

`?` 输入的数据

信号类型	功能代码	输入?的类型	输入的值
数字输出	05	byte	(H: 1)(L: 0)
		bool	(H: true)(L: false)
数字输出	15	byte[]	(H: 1)(L: 0)
		bool[]	(H: true)(L: false)
寄存器输出	06	byte	

		bool	
寄存器输出	16	int	
		float	
		double	
		string	
		byte[]	
		int[]	
		float[]	
		double[]	
		string[]	
		bool[]	

*用户定义的modbus_write写入时将遵循大端序（AB CD）格式

*这里的int代表int32。对于int16类型数据，必须先使用GetBytes()将int16转换为字节数组

`int` 要写入的最大地址数，仅对字符串类型数据有效

> 0 有效地址长度。按定义的地址长度写入

<= 0 无效地址长度。写入全部数据

返回值

`bool` `True` 写入成功

`False` 写入失败 1.输入的数据?为空字符串或空数组

2.Modbus通信发生错误

语法 4（TCP/RTU）

```
bool modbus_write(
    string,
    byte,
    string,
    int,
    ?
)
```

注

与语法 3 类似，地址长度 <= 0，将写入全部数据。

modbus_write("TCP_1", 0, "RO", 9000, bb) => **modbus_write**("TCP_1", 0, "RO", 9000, bb, 0)

Modbus 地址数据大小

数字 1 个地址 = 1 位大小

寄存器 1 个地址 = 2 字节大小

若将用户定义的值应用于用户设置，如：

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)          //写入 1,1,1
                                                   //零值，写入 0。非零值，写入 1。
modbus_write("TCP_1", 0, "DO", 800, bb, 2)         //写入 1,1
                                                   //地址数 = 2，仅写入 2 个地址。
modbus_write("TCP_1", 0, "DO", 800, true)          //写入 1
int i = 10000
```

```
modbus_write("TCP_1", 0, "RO", 9000, i)          //写入 0x00,0x00,0x27,0x10
                                                 //默认为 int32 大端序（AB CD）
bb = GetBytes(i, 0, 1)                           // bb = {0x10,0x27}
                                                 //转换为 int16 小端序（CD AB）
modbus_write("TCP_1", 0, "RO", 9000, bb)         //写入 0x10,0x27
string[] n = {"ABC", "12", "34"}
modbus_write("TCP_1", 0, "RO", 9000, n, 2)       //写入 ABC1
                                                 //仅有 2 个地址可用，无法应用超出范围的值。
modbus_write("TCP_1", 0, "RO", 9000, n, 5)       //写入 ABC12340
                                                 //数据需要 4 个地址（0xAB 0xC1 0x23 0x40）
```

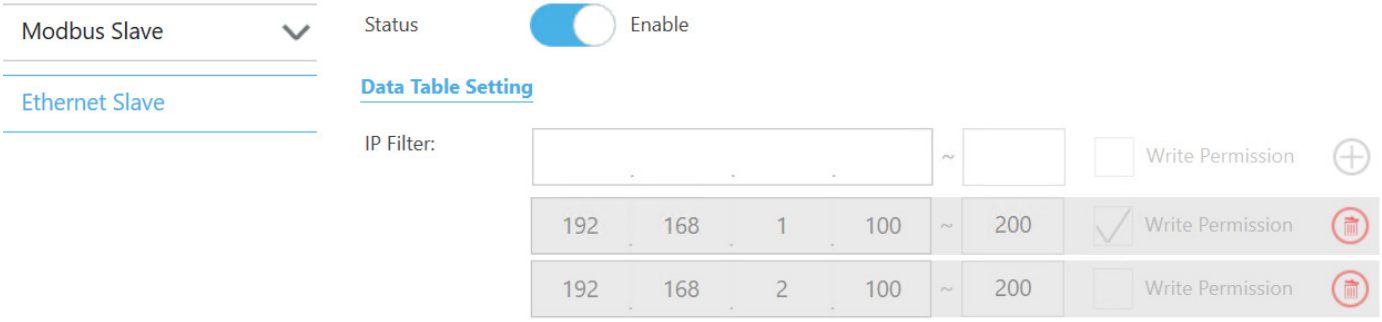
16. TM Ethernet Slave

以太网从站具有基于客户端/服务器连接框架通过 Socket TCP 建立的功能。启用后，机器人将建立一个 Socket TCP Listener 服务器，向所有连接的客户端发送机器人状态和数据，或接收客户端发送的内容，以执行相应指令并定期、及时地更新相应信息（不保证实时性）。

与 Modbus 从站类似，预先设为**启用**的以太网从站会在重新接通电源后自动启动。已建立的 IP 和端口将显示在通知窗口中。

IP　　TMflow → 系统 → 网络 → IP 地址
端口　　5891

16.1 GUI 设置



启用/禁用　　启用或禁用以太网从站
IP 筛选器　　IP 白名单
设置允许连接至以太网从站的合格 IP 地址范围。若不设置筛选器，网络上的所有设备均可连接至以太网从站。
写入权限　　若勾选，则允许相应 IP 范围内的设备使用 TMSVR 命令写入以太网服务器。

例如如此设置 IP 筛选器：

组 1 192.168.1.100～200 代表 IP 192.168.1.100、192.168.1.101、……、192.168.1.200 可以连接。
组 2 192.168.2.100～200 代表 IP 192.168.2.100、192.168.2.101、……、192.168.2.200 可以连接。
若客户端的 IP 地址不在以上列出的 IP 范围内，将拒绝客户端连接。

组 1（192.168.1.100 ~ 200）拥有写入权限，因此在该组范围内的客户端于连接后向以太网从站发送数据时，以太网从站会写入数据。组 2（192.168.2.100 ~ 200）没有写入权限，向以太网从站发送数据时，以太网从站不会写入数据，并以写入权限错误代码响应。

16.2 svr_read()

读取本地主机机器人设置连接选项卡中以太网从站通信数据表的项目值。

语法 1

```
? svr_read(
    string
)
```

参数

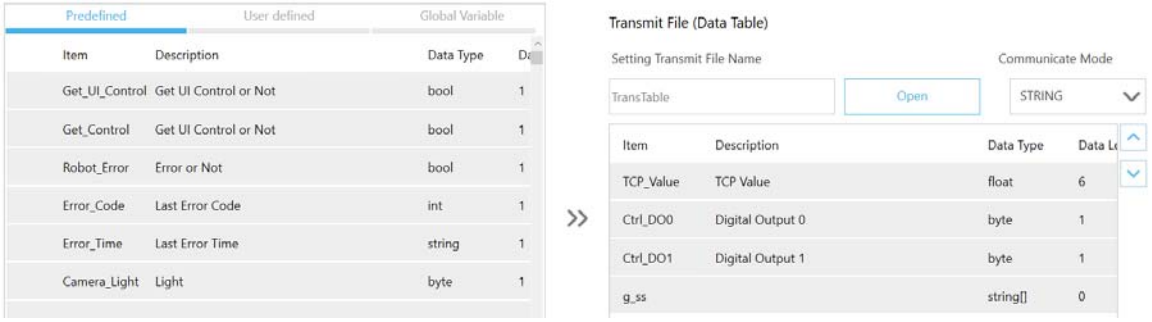
`string` 项目名称

返回值

`?` 按设置数据类型返回值

注

假设将 TCP_Value float[]、Ctrl_DO0 byte、Ctrl_DO1 byte 和 g_ss string[]作为通信数据表。




```
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                                    //直接用变量名g_ss为变量赋值
byte st = svr_read("Robot_Link")             //0（机器人未连接）1（机器人已连接）

float[] fva1 = svr_read("TCP_Value")         //报告错误。假设以太网从站未启动。
float[] fva2 = svr_read("TCP_Value1")        //报告错误。项目名称TCP_Value1不存在。

float[] fva3 = svr_read("g_ff")              //报告错误。通信表中不存在项目名称g_ff。
//假设g_ff存在于全局变量中，由于没有加载所有全局变量，无法访问。
float[] fva4 = svr_read("Coord_Base_Flange")    // {0.01,-252.6,891.7,90,0,0}
//虽然是作为通信数据添加的，但可访问系统定义中的项目名称Coord_Base_Flange。
```

16.3 svr_write()

将项目值写入本地主机机器人设置连接选项卡中以太网从站通信数据表。

语法 1

```
bool svr_write(
    string,
    ?
)
```

参数

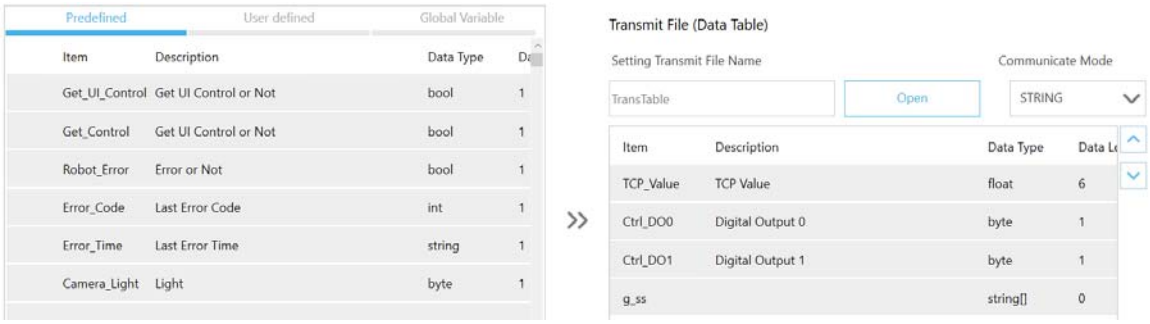
string	项目名称	
?	项目值	

返回值

bool	True	写入成功	
	False	写入失败	可能原因 1.项目名称不存在。 2.无法写入只读项目名称。 3.要写入的项目值与项目数据类型不匹配。

注

假设通信数据表内包含 TCP_Value float[]、Ctrl_DO0 byte、Ctrl_DO1 byte 和 g_ss string[]。



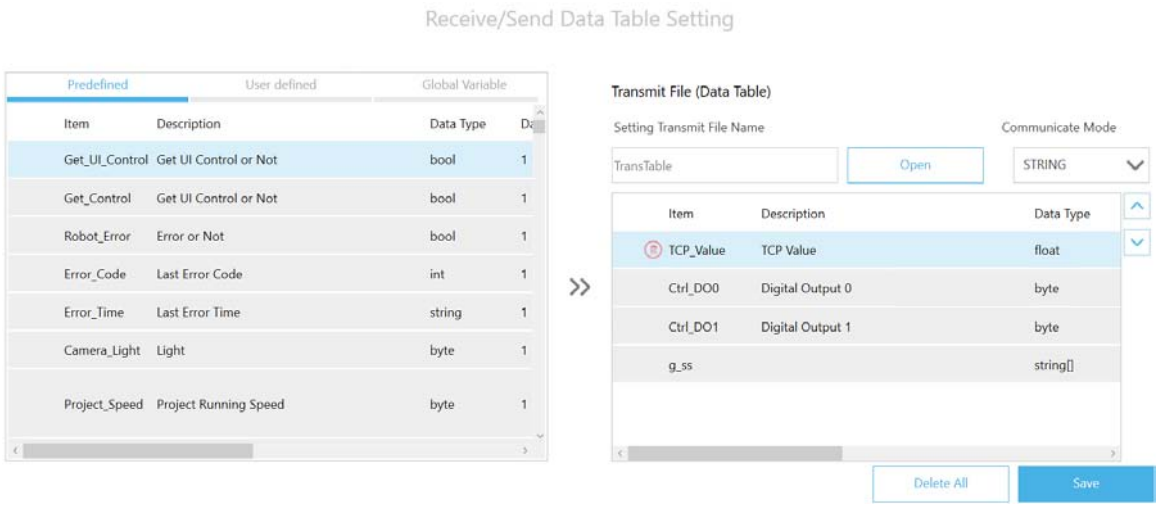

```
float[] tvalue = {1,2,3,0.1,0.2,0.3}
bool flag = false
flag = svr_write("TCP_Value", tvalue)      // flag = false    只读，无效处理（而非错误）
flag = svr_write("Ctrl_DO0", 1)            // flag = true，Ctrl_DO0 = 1
flag = svr_write("Ctrl_DO1", 0)            // flag = true，Ctrl_DO1 = 0

flag = svr_write("TCP_Value", tvalue)      //错误。假设以太网从站未启动。
flag = svr_write("TCP_Value1", tvalue)     //错误。项目名称TCP_Value1不存在。
flag = svr_write("Ctrl_DO0", "True")       //错误。写入项目名称Ctrl_DO0的值为字符串（数据类型被设为字节）
```

16.4 数据表

用户可使用数据表中的项目自定义所需数据内容、配置以太网从站和客户端间的通信传输协议，并将设置保存为通信文件。启用以太网从站后，通信文件中的数据项目将按相关数据内容建立，并被定期发送至连接的客户端（不保证实时性）。数据格式的类型由通信文件中的设置定义。客户端可使用任意支持的数据格式类型向服务器发送数据。

在协议中，支持的数据格式类型包括：

二进制	二进制格式，按字节数组转换（小端序/UTF8）
字符串	字符串格式，与外部命令格式类似
JSON	JSON 字符串格式



配置界面采用从左到右机制。用户可将左侧的项目添加至右侧的通信数据表中，并调整右侧通讯数据表中各项目的排列顺序。待发送内容中始终存在一个以太网从站中预定义的项目 ***Robot_Link***，其类型为字节，具有只读属性，用于表示是否连接至机器人。

1. 预定义

此部分中的项目和设置由 TMflow 定义，项目的数据内容由 TMflow 更新。定义的项目为机器人的一般状态，如机器人的坐标、项目状态、电气控制柜状态，或与 IO 相关的状态，如数字输入/数字输出、模拟输入/模拟输出。

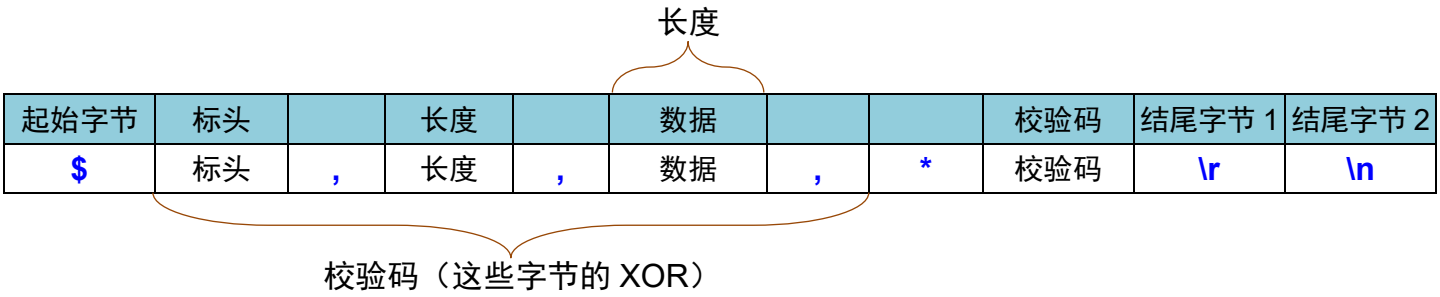
2. 用户定义

此部分中的项目和设置由 TMflow 用户定义，以供项目程序通过表达式编辑器读/写项目数据，或供外部用户通过 TCP/IP 连接使用 TMSVR 命令读/写项目数据。借助用户定义选项卡，项目程序可作为数据交换协议与外部通信设备协同工作。可将用户定义选项卡中的项目列表保存为自定义文件，以便将来编辑或交换数据。

3. 全局变量

在全局变量选项卡中，由 TMflow 用户创建的变量列表提供了在项目编程中直接使用变量名进行读/写操作的方法，外部通信设备可通过通信协议读/写全局变量。

16.5 通信协议



名称	大小	ASCII	十六进制	说明
起始字节	1	$	0x24	通信起始字节
标头	*X*			通信标头
分隔符	1	,	0x2C	标头和长度间的分隔符
长度	*Y*			数据长度
分隔符	1	,	0x2C	长度和数据间的分隔符
数据	*Z*			通信数据
分隔符	1	,	0x2C	数据和校验码间的分隔符
符号	1	*	0x2A	校验码开始符号
校验码	*2*			通信校验码
结尾字节 1	1	\r	0x0D	
结尾字节 2	1	\n	0x0A	通信结尾字节

*与外部命令使用相同的通信协议。

1. 标头

定义通信数据包的用途。不同标头对通信数据包和数据的定义不同。

- *TMSVR* 定义 TM Ethernet Slave 的功能
- *CPERR* 定义通信数据包的错误，如封包错误、校验码错误、标头错误等。
  *与外部命令使用相同的内容定义。

2. 长度

长度表示数据占用的 UTF8 字节长度。用户可使用十进制、十六进制或二进制格式。最大长度为 32 位。例如：

```
$TMSVR,100,Data,*CS\r\n          //十进制 100，表示数据长度为 100 字节
$TMSVR,0x100,Data,*CS\r\n        //十六进制 0x100，表示数据长度为 256 字节
$TMSVR,0b100,Data,*CS\r\n        //二进制 0b100，表示数据长度为 4 字节
$TMSVR,8,1,達明,*CS\r\n          //表示数据"1,達明"的长度为 8 字节（UTF8）
```

3. 数据

通信数据包的内容支持任意字符（包括 0x00~0xFF，使用 UTF8 编码），数据长度由长度决定。数据中定义的用途和说明必须由标头定义。

4. 校验码

通信数据包的校验码。计算方法为 XOR（异或）。计算范围为$和*之间的所有字节（不包括$和*），如下所示。

```
$TMSVR,100,Data,*CS\r\n
```

Checksum = Byte[1] ^ Byte[2] … ^ Byte[N-6]

校验码格式设为 2 个十六进制字节（不含 0x），例如

```
$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

起始字节	标头		长度		数据			校验码	结尾字节 1	结尾字节 2
$	*TMSVR*	,	长度	,	数据	,	*	校验码	\r	\n

ID		模式		内容
事务 ID	,	0/1/2/3/ 11/12/13	,	项目和值

TMSVR 的定义为 TM Ethernet Slave 协议。数据包的数据部分又分为三部分：ID（事务 ID）、模式（内容模式）和内容（项目和值），以逗号隔开，说明如下。

ID　以任意字母数字字符表示的事务编号。（若遇到非字母数字字节，将报告 CPERR 04 错误）。用作通信数据包响应时，为用于标识要响应的命令组的事务编号。

,　分隔符

模式　数据内容格式模式

- 0　表示服务器以字符串格式响应客户端命令。
- 1　表示内容数据类型为二进制格式
- 2　表示内容数据类型为字符串格式
- 3　表示内容数据类型为JSON格式
- 11　表示内容数据类型为二进制格式（请求读取）
- 12　表示内容数据类型为字符串格式（请求读取）
- 13　表示内容数据类型为JSON格式（请求读取）
- 1/2/3用于客户端写入服务器和客户端从服务器读取。客户端从服务器读取指服务器定期向连接的客户端发送内容。
- 11/12/13用于客户端通过请求读取从服务器读取，即客户端发送读取项目请求，服务器回复客户端项目值。

,　分隔符

内容　数据内容。按模式定义格式化。

注

TMSVR 命令用于客户端和服务器的双向通信。正常情况下，服务器将定期向连接的客户端广播来自传输和用户定义的通信文件的数据项。服务器向客户端发送数据时，数据将被发送至客户端，以从服务器读取，无需响应服务器。客户端向服务器发送数据时，服务器从客户端接收数据以写入，需要响应客户端。

服务器发送数据时，事务编号随每次迭代从 0 到 9 循环。客户端向服务器发送数据时，事务编号可为客户端一侧自定义的任意字母数字字符。若通信数据包格式经检查无误，服务器将根据数据包中的事务编号回复客户端命令处理状态。

客户端向服务器发送数据时，服务器会检查是否符合所有写入条件，然后再写入数据。若写入命令存在任何错误，则不会覆写数据。将检查的写入条件包括：

1. 数据内容格式模式的有效性
2. 连接的客户端基于 IP 筛选器的写入权限。
3. 数据内容与模式匹配。
4. 要写入的项目存在于传输或用户定义的通信文件中。
5. 要写入的项目的属性不为只读。

6. 机器人处于适当模式下（M/A）。
7. 写入的数据与各项目的数据类型匹配。

模式为 11/12/13 时，使用请求读取方法。客户端发送项目请求，服务器回复项目值。请求的项目可为预定义区域内的项目，无需检查定期交付，但在用户定义区域和全局变量区域中，对于自定义定义，仍需检查定期交付以获取请求。

客户端发送项目请求时，并非只能获取一个项目，而是可以同时获取多个项目。项目请求成功时，服务器将按照与模式 11/12/13 匹配的格式向客户端发送项目值。但是，若项目请求失败，如项目名称不存在，服务器将按照模式 0 格式发送命令处理状态。

0. 模式 = 0（服务器响应客户端命令处理的状态）

服务器接收并处理来自客户端的写入命令后，将以模式 0 响应另一条 TMSVR 命令。模式 0 的详情如下所示。

数据

ID		模式		错误代码		错误说明
事务 ID	,	***0***	,	00~07	,	

事务 ID　　客户端发送命令时定义的事务 ID，供服务器用于响应。

模式　　0（服务器响应客户端）

错误代码　　错误代码定义。固定为2个十六进制字节（不含0x）

- 00　正确写入。无错误。
- 01　不支持的通信格式或模式。（例如：模式 = 99）
- 02　不允许连接的客户端写入。（IP筛选器无写入权限）
- 03　通信格式和数据内容格式不匹配。（例如：模式 = 3，但数据内容不为JSON格式）
- 04　要写入或读取的项目不存在。
- 05　无法写入只读项目。
- 06　写入时M/A模式不正确。
- 07　要写入的值与配置的类型或大小不匹配。

错误说明 错误代码后的错误说明。

- 00　OK
- 01　NotSupport
- 02　WritePermission
- 03　InvalidData
- 04　NotExist;XXX　　//;XXX表示是哪个数据项
- 05　ReadOnly;XXX
- 06　ModeError;XXX
- 07　ValueError;XXX

```
<   $TMSVR,15,S0,2,Ctrl_DO0=1,*76\r\n        //事务 ID 为 S0，字符串格式，设置 Ctrl_DO0=1
>   $TMSVR,10,S0,0,00,OK,*18\r\n
                    //服务器响应事务 ID S0，模式为 0，错误代码为 00，正确写入

<   $TMSVR,16,S1,99,Ctrl_DO0=1,*46\r\n        //事务 ID 为 S1，模式为 99
>   $TMSVR,18,S1,0,01,NotSupport,*0E\r\n
                    //服务器响应事务 ID S1，模式为 0，错误代码为 01，不支持的模式

<   $TMSVR,15,S2,2,Ctrl_DO0=1,*74\r\n         //事务 ID 为 S2，字符串格式，设置 Ctrl_DO0=1
>   $TMSVR,23,S2,0,02,WritePermission,*6A\r\n
    //服务器响应事务 ID S2，模式为 0，错误代码为 02，连接的客户端没有写入权限。

<   $TMSVR,15,S3,3,Ctrl_DO0=1,*74\r\n         //事务 ID 为 S3，JSON 格式，设置 Ctrl_DO0=1
>   $TMSVR,19,S3,0,03,InvalidData,*74\r\n
    //服务器响应事务 ID S3，模式为 0，错误代码为 03，JSON 格式，数据格式（JSON）与内容数据格式（字符串）不匹配

<   $TMSVR,16,S4,2,Ctrl_DO32=1,*40\r\n         //事务 ID 为 S4，字符串格式，设置 Ctrl_DO32=1
>   $TMSVR,26,S4,0,04,NotExist;Ctrl_DO32,*58\r\n
    //服务器响应事务 ID S4，模式为 0，错误代码为 04，项目 Ctrl_DO32 不存在。
```

< $*TMSVR*,17,S5,*2*,Robot_Link=1,*07\r\n //事务 ID 为 S5，字符串格式，设置 Robot_Link=1
> $*TMSVR*,27,S5,*0*,05,ReadOnly;Robot_Link,*1E\r\n
//服务器响应事务 ID S5，模式为 0，错误代码为 05，Robot_Link 项目为只读。

假设用户定义了项目：adata，类型：int，大小：4，写入模式：自动。
< $*TMSVR*,20,S6,*2*,adata={1,2,3,4},*55\r\n //事务 ID 为 S6，字符串格式，设置 adata={1,2,3,4}
> $*TMSVR*,23,S6,*0*,06,ModeError;adata,*2D\r\n
//服务器响应事务 ID S6，模式为 0，错误代码为 06，写入时 M/A 模式不匹配（假设写入时为手动模式）。

< $*TMSVR*,18,S7,*2*,adata={1,2,3},*47\r\n //事务 ID 为 S7，字符串格式，设置 adata={1,2,3}
> $*TMSVR*,24,S7,*0*,07,ValueError;adata,*42\r\n
//服务器响应事务 ID S7，模式为 0，错误代码为 07，要写入的值与数据的大小或类型不匹配。（配置的大小为 4，但只需要写入 3。）

1. 模式 = 1 二进制

使用小端序值和 UTF8 值将数据项目名称相应地转换为字节数组，进而以二进制模式传输数据内容。格式如下所示。

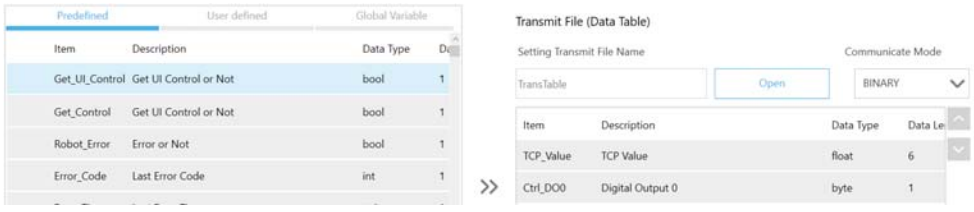
数据

ID		模式		内容
事务 ID	,	*1*	,	项目和值

项目长度 *2 字节小端序*	***项目*** *UTF8*	***值的长度*** *2 字节小端序*	***值*** *小端序/UTF8*	…

项目长度 小端序，2 字节，值在 0 至 65535 之间，表示随后的项目的长度
项目 项目名称
值的长度 小端序，2 字节，值在 0 至 65535 之间，表示随后的数据的长度
值 数据值

假设将 Check TCP_Value float[]和 Ctrl_DO0 byte 作为通信数据，并以二进制模式传输。



> 24 54 4D 53 56 52 2C // $TMSVR, //标头
36 39 2C // 69, //长度
30 2C *31* 2C // 0,1, //事务 ID 为 0，模式为 1（二进制）
0A 00 *52 6F 62 6F 74 5F 4C 69 6E 6B* 01 00 *00* // Robot_Link=0 //名称占用 10 字节，值占用 1 字节
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} //名称占用 9 字节，值占用 24 字节
08 00 *43 74 72 6C 5F 44 4F 30* 01 00 *00* // Ctrl_DO0=0 //名称占用 8 字节，值占用 1 字节
2C 2A 39 36 0D 0A // ,*96\r\n //校验码

```
<    24 54 4D 53 56 52 2C          // $TMSVR,          //标头
     31 38 2C                      // 18,              //长度
     54 31 2C 31 2C                // T1,1,            //事务 ID 为 T1，模式为 1（二进制）
     08 00 43 74 72 6C 5F 44 4F 30 01 00 01
                                   // Ctrl_DO0=1   //名称占用 8 字节，值占用 1 字节
     2C 2A 37 41 0D 0A             // ,*7A\r\n         //校验码
>    $TMSVR,10,T1,0,00,OK,*1E\r\n  //服务器响应 ID T1，模式为 1，错误代码为 00，正确写入
```

若要发送的项目的数据类型为 string []，则将在字符串元素间插入两个字节 *0x00 0x00* 作为分隔符。

```
>    24 54 4D 53 56 52 2C          // $TMSVR,          //标头
     39 30 2C                      // 90,              //长度
     30 2C 31 2C                   // 0,1              //事务 ID 为 0，模式为 1（二进制）
     0A 00 52 6F 62 6F 74 5F 4C 69 6E 6B 01 00 00
                                   // Robot_Link=0 //名称占用 10 字节，值占用 1 字节
     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}//名称占用 9 字节，值占用 24 字节
     08 00 43 74 72 6C 5F 44 4F 30 01 00 01
                                   // Ctrl_DO0=1   //名称占用 8 字节，值占用 1 字节
     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"}      //名称占用 4 字节，值占用 13 字节
     2C 2A 44 43 0D 0A             // ,*DC\r\n         //校验码
```

此外，若要接收的项目的数据类型为 string []，则将其转换成字节数组时，会在字符串元素间插入两个字节 00 00 作为分隔符。

```
<    24 54 4D 53 56 52 2C          // $TMSVR,          //标头
     32 35 2C                      // 25,              //长度
     54 32 2C 31 2C                // T2,1,            //事务 ID 为 T2，模式为 1（二进制）
     04 00 67 5F 73 73 0C 00 48 65 6C 6C 6F 00 00 57 6F 72 6C 64
                                   // g_ss={"Hello", "World"}   //名称占用 4 字节，值占用 12 字节
     2C 2A 30 32 0D 0A             // ,*02\r\n         //校验码
>    $TMSVR,10,T2,0,00,OK,*1D\r\n  //服务器响应 ID T2，模式为 0，错误代码为 00，正确写入
```

2. 模式 = 2 字符串

以字符串形式传输数据内容，数据项目的名称和值位于外部命令的脚本字符串中。格式如下所示。

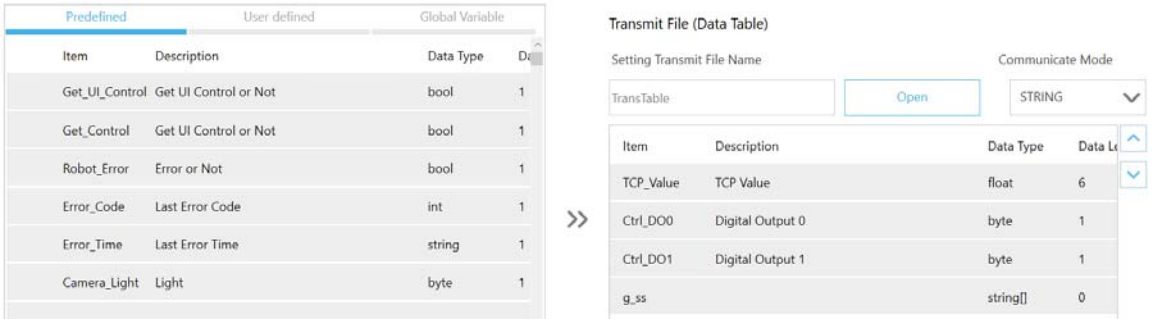
数据

ID		模式		内容
事务 ID	,	2	,	项目和值

项目	=	*值*	\r\n	…

项目　　项目名称
　　=　　等于
　　值　　数据值
　　\r\n　　若需要分隔下一个项目，按需使用回车符号。

假设将 Check TCP_Value float[]和 Ctrl_DO0 byte、Ctrl_DO1 byte 和 g_ss string[]作为通信数据，并以字符串模式传输。




```
>  $TMSVR,97,9,2,Robot_Link=0\r\n          // Robot_Link=0      //事务 ID 为 9，模式为 2（字符串）
   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     //事务 ID 为 T2，模式为 2（字符串）
   Ctrl_DO1=1,*34\r\n                      // set Ctrl_DO1=1
>  $TMSVR,10,T2,0,00,OK,*1D\r\n            //服务器响应 ID T2，模式为 0，错误代码为 00，正确写入
```

3. 模式 = 3 JSON

按 JSON 格式序列化数据项目的名称和值，进而以 JSON 字符串形式传输数据内容，如下所示。

数据

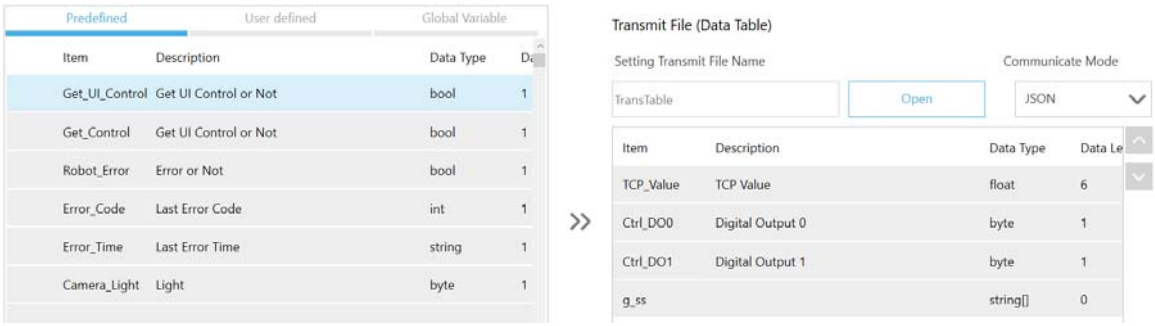
ID		模式		内容
事务 ID	,	3	,	项目和值

项目 项目名称
值 数据值

```
public class TMSVRJsonData
{
    public string Item;
    public object Value;
}
```

*[]数组用于涉及多个项目时。

假设将 TCP_Value float[]和 Ctrl_DO0 byte、Ctrl_DO1 byte 和 g_ss string[]作为通信数据，并以 JSON 模式传输。




```
>  $TMSVR,196,5,3,[{"Item":"Robot_Link","Value":0},          // Robot_Link=0
                                                            //事务 ID 为 5，模式为 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          //服务器响应 ID T9，模式为 0，错误代码为 0，正确写入
```

11. 模式 = 11 二进制（请求读取）

使用小端序值和 UTF8 值将数据项目名称相应地转换为字节数组，进而以二进制模式传输数据内容。格式如下所示。

数据（客户端至服务器）

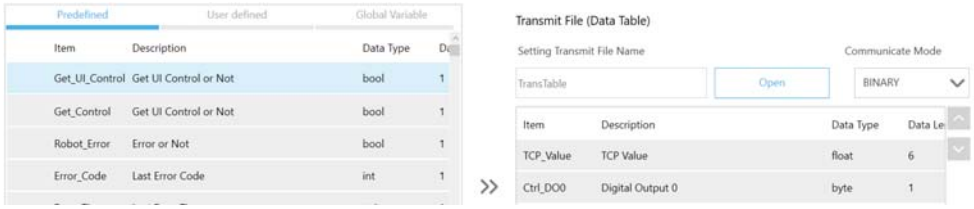
ID		模式		内容
事务 ID	,	*11*	,	项目

项目长度 *2 字节小端序*	*项目* *UTF8*	…

读取请求与模式 1 的区别在于不需要值。

项目长度　　小端序，2 字节，值在 0 至 65535 之间，表示随后的项目的长度
项目　　项目名称

假设将 TCP_Value float[]和 Ctrl_DO0 byte 作为通信数据，并以二进制模式传输。



服务器定期交付

```
>   24 54 4D 53 56 52 2C          // $TMSVR,          //标头
    36 39 2C                      // 69,              //长度
    30 2C 31 2C                   // 0,1,             //事务 ID 为 0，模式为 1（二进制）
    0A 00 52 6F 62 6F 74 5F 4C 69 6E 6B 01 00 00  // Robot_Link=0 //名称占用 10 字节，值占用 1 字节
    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}//名称占用 9 字节，值占用
    24 字节
    08 00 43 74 72 6C 5F 44 4F 30 01 00 00    // Ctrl_DO0=0   //名称占用 8 字节，值占用 1 字节
    2C 2A 39 36 0D 0A             // ,*96\r\n         //校验码
```

客户端请求读取

```
<   24 54 4D 53 56 52 2C          // $TMSVR,          //标头
    32 36 2C                      // 26,              //长度
    51 31 2C 31 31 2C             // Q1,11,           //事务 ID 为 Q1，模式为 11（二进制（请求读取））
    08 00 43 74 72 6C 5F 44 4F 30          // Ctrl_DO0          //名称占用 8 字节
    08 00 54 43 50 5F 4D 61 73 73          // TCP_Mass          //名称占用 8 字节
    2C 2A 37 46 0D 0A             // ,*7F\r\n         //校验码
```

服务器回复项目值

```
>   24 54 4D 53 56 52 2C          // $TMSVR,          //标头
    33 35 2C                      // 35,              //长度
    51 31 2C 31 31 2C             // Q1,11,           //服务器响应 ID Q1，模式为 11（二进制）
    08 00 43 74 72 6C 5F 44 4F 30 01 00 00
                                  // Ctrl_DO0=0   //名称占用 8 字节，值占用 1 字节
    08 00 54 43 50 5F 4D 61 73 73 04 00 00 00 00 00
                                  // TCP_Mass=0 //名称占用 8 字节，值占用 4 字节
```

2C 2A 37 38 0D 0A // ,*78\r\n //校验码

*服务器回复内容的格式与模式 1（二进制）相同

客户端请求读取

< 24 54 4D 53 56 52 2C // $TMSVR, //标头

32 36 2C // 26, //长度

51 32 2C *31 31* 2C // Q2,11, //事务 ID 为 Q1，模式为 11（二进制（请求读取））

08 00 *43 74 72 6C 5F 44 4F 30* // Ctrl_DO0 //名称占用 8 字节

08 00 *54 43 50 5F 4D 61 58 58* // TCP_MaXX //名称占用 8 字节

2C 2A 37 43 0D 0A // ,*7C\r\n //校验码

服务器回复项目值

> $***TMSVR***,25,Q2,*0*,04,NotExist;TCP_MaXX,*17\r\n

//服务器响应 ID Q2，模式为 0，错误代码为 04，项目不存在

12. 模式 = 12 字符串（请求读取）

以字符串形式传输数据内容，数据项目的名称和值位于外部命令的脚本字符串中。格式如下所示。

数据（客户端至服务器）

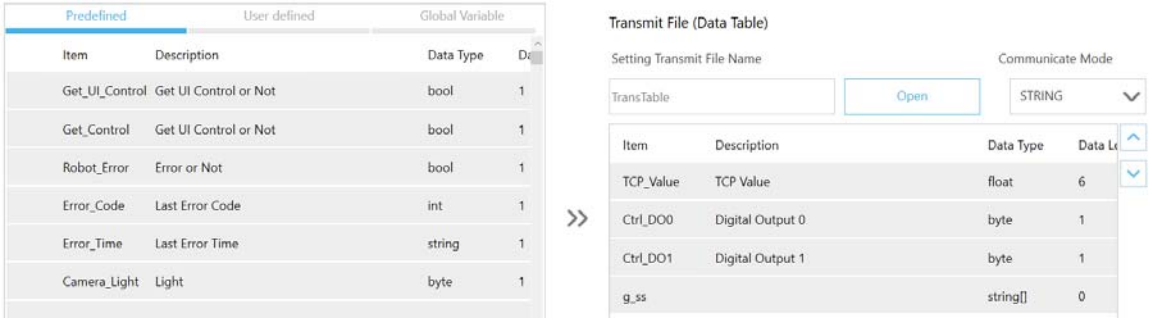
ID		模式		内容
事务 ID	,	*12*	,	项目和值

项目	\r\n	…

不需要值。

项目　　项目名称
\r\n　　换行符。仅在有下一个项目时用作分隔符。

假设将 TCP_Value float[]和 Ctrl_DO0 byte、Ctrl_DO1 byte 和 g_ss string[]作为通信数据，并以字符串模式传输。



服务器定期交付

```
>   $TMSVR,97,9,2,Robot_Link=0\r\n      // Robot_Link=0      //事务 ID 为 9，模式为 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"}
```

客户端请求读取

```
<   $TMSVR,28,Q2,12,Robot_Link\r\n      //项目 Robot_Link
                                        //事务 ID 为 Q2，模式为 12（JSON（请求读取））
    TCP_Mass,*0E\r\n                    //项目 TCP_Mass
```

服务器回复项目值

```
>   $TMSVR,30,Q2,12,Robot_Link=0\r\n    //服务器响应 ID Q2，模式为 12
    TCP_Mass=0,*09\r\n
```

*服务器回复内容的格式与模式 2（字符串）相同

13. 模式 = 13 JSON（请求读取）

按 JSON 格式序列化数据项目的名称和值，进而以 JSON 字符串形式传输数据内容，如下所示。

数据（客户端至服务器）

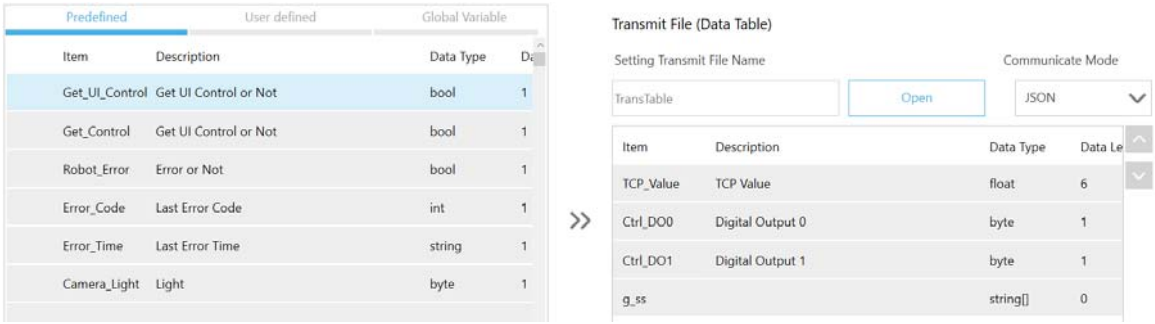
ID		模式		内容
事务 ID	,	13	,	项目和值

项目 项目名称
值 数据值

```
public class TMSVRJsonData
{
    public string Item;
    public object Value;
}
```

*[]数组用于涉及多个项目时。
*与模式 3（JSON）共用相同的类进行序列化/反序列化，但值属性可能不存在

假设将 TCP_Value float[]和 Ctrl_DO0 byte、Ctrl_DO1 byte 和 g_ss string[]作为通信数据，并以 JSON 模式传输。



服务器定期交付

```
>   $TMSVR,196,5,3,[{"Item":"Robot_Link","Value":0},          // Robot_Link=0   //事务 ID 为 5，模式为 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"}
```

客户端请求读取

```
<   $TMSVR,27,Q3,13,[{"Item":"TCP_Mass"}],*3C\r\n
                                  //事务 ID 为 Q3，模式为 13（JSON（请求读取））
```

服务器回复项目值

```
>   $TMSVR,39,Q3,13,[{"Item":"TCP_Mass","Value":0.0}],*40\r\n    //服务器响应 ID Q3，模式为 13
```

*服务器回复内容的格式与模式 3（JSON）相同

17. Profinet 函数

机器人可通过Profinet通信协议与外部控制器通信。在Profinet通信协议机制中，机器人作为Profinet IO设备供外部设备读写机器人数据。同时，TMflow通过Profinet函数监控从外部设备接收的数据表和向外部设备发送的数据表，并更改向外部设备发送的数据表中的自定义定义部分。

通信数据表

数据表由输入数据和输出数据组成。输入数据表供外部设备向机器人发布数据，输出数据表供机器人向外部设备发送数据。两类数据表均分为系统定义数据部分和自定义定义数据部分。

1. 系统定义部分：项目和设置由机器人定义，数据内容由机器人或外部设备更新。其中定义的项目与机器人状态相关，如机器人基准、项目状态、控制柜状态，或与输入/输出状态相关，如数字输入/输出和模拟输入/输出。用户可使用Profinet函数读取系统定义部分中的输入数据表和输出数据表。
2. 自定义定义部分：项目和设置由用户定义，数据内容由用户或外部设备更新。在编辑项目的同时，用户可使用Profinet函数读写自定义定义部分中的输出数据表或读取自定义定义部分中的输入数据表，并将自定义定义部分用作项目与外部设备间的数据交换寄存器。

通信数据表（从机器人视角）	数据部分	TMflow Profinet 函数权限	外部设备权限
输入数据表	系统定义部分	读取	写入
	自定义定义部分	读取	写入
输出数据表	系统定义部分	读取	读取
	自定义定义部分	读取/写入	读取

17.1 profinet_read_input()

读取输入表的内容。

语法 1

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

参数

int	起始地址
int	要读取的地址量

返回值

byte[]	以字节数组形式返回的数据。

注

```
byte[] var_ba = profinet_read_input(148,16)
    // {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
```

语法 2

```
byte profinet_read_input(
    int,
)
```

参数

int	起始地址

返回值

byte	以字节形式返回的数据。

注

```
byte var_b = profinet_read_input(148)
    // 0x30
```

语法 3

```
? profinet_read_input(
    string,
    int,
    int
)
```

参数

string	项目名称
int	项目的起始移位地址
int	要读取的地址量

返回值

?	返回值的数据类型取决于通信数据表中的项目定义。 *数据类型包括**byte**、**byte[]**、**int**、**int[]**、**float**、**float[]**、**string**

语法 4

```
? profinet_read_input(
    string,
    int,
)
```

参数

string 项目名称
int 项目的起始移位地址

返回值

? 返回值的数据类型取决于通信数据表中的项目定义。
*数据类型包括**byte**、**byte[]**、**int**、**int[]**、**float**、**float[]**、**string**

注

*与语法 3 相同。默认读取至项目末尾。
*根据配置文件按小端序（DCBA）或大端序（ABCD）读取数据。

语法 5

```
? profinet_read_input(
    string
)
```

参数

string 项目名称

返回值

? 返回值的数据类型取决于通信数据表中的项目定义。
*数据类型包括 **byte**、**byte[]**、**int**、**int[]**、**float**、**float[]**、**string**

注

*与语法 3 相同。项目的起始移位地址默认为 0，默认读取至项目末尾。
*根据配置文件按小端序（DCBA）或大端序（ABCD）读取数据。

```
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}（小端序）  转换为 int[]
    // int[] = {0x00000001,0x00000002,0x00000003}（小端序）
    // 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,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}（小端序）  转换为 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}（小端序）
    // 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}（小端序）          转换为 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}（小端序）
    // 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}（大端序）     转换为 float[]
    // float[] = {0x3F99999A,0x3FA66666,0x40066666}（大端序）
    // 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}（大端序）     转换为 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}（大端序）
    // 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}（大端序）     转换为 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}（大端序）
    // 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()

读取输入表的内容并将数据转换为 32 位整数。

语法 1

```
int[] profinet_read_input_int(
    int,
    int,
    int
)
```

参数

int	起始地址
int	要读取的地址量
int	基于小端序（DCBA）还是大端序（ABCD）将读取的数据转换为int数组。
0	小端序
1	大端序
2	根据配置文件。

返回值

int[]	以整型数组形式返回的数据。

注

```
int[] value = profinet_read_input_int(164, 12, 0)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF}（小端序）  转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000}（小端序）
    // 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}（小端序）  转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0x00FF8000}（小端序）
    // int[] = {32767,99999,16744448}
int[] value = profinet_read_input_int(164, 10, 0)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80}（小端序）  转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0x00008000}（小端序）
    // 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}（小端序）  转换为 int[]
    // int[] = {0xFF7F0000,0x9F860100,0x0080FFFF}（大端序）
    // 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}（小端序）  转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000}（小端序）
    // int[] = {32767.99999,-32768}
```

语法 2

```
int[] profinet_read_input_int(
    int,
    int
)
```

参数

int	起始地址
int	要读取的地址量

注

与语法 1 相同，读取数据转换参数默认为 2。

*基于小端序（DCBA）还是大端序（ABCD）将读取的数据转换为 int 数组。

```
int[] var_ia = profinet_read_input_int(164,12,0)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF}（小端序）  转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000}（小端序）
    // 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}（小端序）  转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0x00FF8000}（小端序）
    // int[] = {32767.99999,16744448}
var_ia = profinet_read_input_int(164,10,0)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80}（小端序）  转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0x00008000}（小端序）
    // 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}（大端序）  转换为 int[]
    // int[] = {0xFF7F0000,0x9F860100,0x0080FFFF}（大端序）
    // 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}（小端序）  转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000}（小端序）
    // 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}（小端序）  转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000}（小端序）
    // int[] = {32767.99999,-32768}
```

语法 3

```
int profinet_read_input_int(
    int
)
```

参数

`int`　起始地址

*基于小端序（DCBA）还是大端序（ABCD）将读取的数据转换为int数组。

返回值

`Int`　以整数形式返回的数据。

注

```
int var_i = profinet_read_input_int(164)
    // byte[] = {0xE4,0x07,0x00,0x00}（小端序）  转换为 int
    // int = 0x000007E4（小端序）
    // int = 2020
var_i = profinet_read_input_int(164)
    // byte[] = {0x00,0x00,0x07,0xE4}（大端序）  转换为 int
    // int = 0x000007E4（大端序）
    // int = 2020
```

17.3 profinet_read_input_float()

读取输入表的内容并将数据转换为 32 位浮点数。

语法 1

```
float[] profinet_read_input_float(
    int,
    int,
    int
)
```

参数

int	起始地址
int	要读取的地址量
int	基于小端序（DCBA）还是大端序（ABCD）将读取的数据转换为float数组。
0	小端序
1	大端序
2	根据配置文件。

返回值

float[]以浮点型数组形式返回的数据。

语法 2

```
float[] profinet_read_input_float(
    int,
    int
)
```

注

与语法 1 相同，读取数据转换参数默认为 2。

*基于小端序（DCBA）还是大端序（ABCD）将读取的数据转换为 float 数组。

```
float[] var_fa = profinet_read_input_float(284,12,0)
    // byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40}（小端序）转换为 float[]
    // float[] = {0x3F800000,0x40000000,0x40400000}（小端序）
    // 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}（小端序）    转换为 float[]
    // float[] = {0x3F800000,0x40000000,0x00400000}（小端序）
    // 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}（小端序）    转换为 float[]
    // float[] = {0x3F800000,0x40000000,0x00000000}（小端序）
    // 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}（小端序）转换为 float[]
    // float[] = {0x0000803F,0x00000040,0x00004040}（大端序）
    // 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}（小端序）转换为 float[]
    // float[] = {0x3F800000,0x40000000,0x40400000}（小端序）
    // 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}（小端序） 转换为 float[]

// float[] = {0x3F800000,0x40000000,0x40400000}（小端序）

// float[] = {1.0,2.0,3.0}

语法 3

```
float profinet_read_input_float(
    int
)
```

参数

`int` 起始地址

*基于小端序（DCBA）还是大端序（ABCD）将读取的数据转换为float数组。

返回值

`float` 以浮点数形式返回的数据

注

float var_f = **profinet_read_input_float**(284)

// byte[] = {0x00,0x00,0x80,0x3F}（小端序）转换为 float

// float = 0x3F800000（小端序）

// float = 1.0

var_f = **profinet_read_input_float**(284)

// byte[] = {0x3F,0x80,0x00,0x00}（大端序）转换为 float

// float = {0x3F800000}（大端序）

// float = {1.0}

17.4 profinet_read_input_string()

读取输入表的内容并将数据转换为 UTF8 编码字符串。

语法 1

```
string profinet_read_input_string(
    int,
    int
)
```

参数

int	起始地址
int	要读取的地址量

返回值

string	以UTF8字符串形式返回的数据（遇到0x00时结束）。

注

```
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()

读取输入表的内容并获取数据字节中某一位或某几位的值。

语法 1

```
byte profinet_read_input_bit(
    int,
    int
)
```

参数

int	起始地址
int	要获取数据字节中第几位的值

返回值

byte	以字节形式返回的数据。 位值 == 1时返回1。 位值 == 0时返回0。

注

```
byte var_b = profinet_read_input_bit(148,0)
    // 0x30    要获取的位："0"
    // 0
var_b = profinet_read_input_bit(148,5)
    // 0x30    要获取的位："5"
    // 1
```

语法 2

```
byte profinet_read_input_bit(
    string,
    int
)
```

参数

string	项目名称
int	要获取第几位的值

返回值

byte	以字节形式返回的数据。 位值 == 1时返回1。 位值 == 0时返回0。 *将以字节形式返回位数据。

注

```
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,...}   //全部数据
    // 1   要获取的位："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,...}   //全部数据
    // 0   要获取的位："17"
```

语法 3

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

参数

int	起始地址
int	起始位
int	要读取几位

返回值

byte[]	以byte[]形式返回的数据。 位值 == 1时返回1。 位值 == 0时返回0。 *将以字节形式返回位数据，例如对于bit[0]将返回byte[0]、对于bit[1]将返回byte[1]。

注

```
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,...}   //全部数据
    // 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,...}   //全部数据
    // byte[] = {1,1,0,1,0,0,1,1}
```

语法 4

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

参数

string	项目名称
int	起始位
int	要读取几位

返回值

byte[]	以byte[]形式返回的数据。 位值 == 1时返回1。 位值 == 0时返回0。 *将以字节形式返回位数据，例如对于bit[0]将返回byte[0]、对于bit[1]将返回byte[1]。

注

```
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,...}   //全部数据
    // 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,...}   //全部数据
    // byte[] = {1,1,0,1,0,0,1,1}
```

17.6 profinet_read_output()

读取输出表的内容。

语法 1

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

参数

int	起始地址
int	要读取的地址长度

返回值

byte[]	以字节数组形式返回的数据。

注

```
byte[] var_ba = profinet_read_output(540,16)
    // {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
```

语法 2

```
byte profinet_read_output(
    int
)
```

参数

int	起始地址

返回值

byte	以字节形式返回的数据

注

```
byte var_b = profinet_read_output(540)
    // 0x30
```

语法 3

```
? profinet_read_output(
    string,
    int,
    int
)
```

参数

string	项目名称
int	项目的起始移位地址
int	要读取的地址量

返回值

?	返回值的数据类型取决于通信数据表中定义的项目。 *数据类型包括**byte**、**byte[]**、**int**、**int[]**、**float**、**float[]**、**string**

语法 4

```
? profinet_read_output(
    string,
    int,
)
```

参数

string	项目名称

int 项目的起始移位地址

返回值

? 返回值的数据类型取决于通信数据表中定义的项目。
*数据类型包括**byte**、**byte[]**、**int**、**int[]**、**float**、**float[]**、**string**

注

*与语法 3 相同。默认读取至项目末尾。
*根据配置文件按小端序（DCBA）或大端序（ABCD）读取数据。

语法 5

```
? profinet_read_output(
    string
)
```

参数

string 项目名称

返回值

? 返回值的数据类型取决于通信数据表中定义的项目。
*数据类型包括**byte**、**byte[]**、**int**、**int[]**、**float**、**float[]**、**string**

注

*与语法 3 相同。项目的起始移位地址默认为 0，默认读取至项目末尾。
*根据配置文件按小端序（DCBA）或大端序（ABCD）读取数据。

```
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}（小端序）      转换为 int
    // int = 0x000007E4（小端序）
    // 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}（小端序）  转换为 int[]
    // int[] = { 0x00000001,0x00000002,0x00000003}（小端序）
    // 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}（小端序）    转换为 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}（小端序）
        // 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}（小端序）          转换为 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}（小端序）
        // 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}

    float var_f = profinet_read_output("Current_TCP_Mass")
        // byte[] = {0x40,0x40,0x00,0x00}（大端序）          转换为 float
        // float = 0x40400000（大端序）
        // 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}（大端序）  转换为 float[]
        // float[] = {0x3F99999A,0x3FA66666,0x40066666}（大端序）
        // 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}（大端序）  转换为 float[]
        // float[] = {0x40066666,0x400CCCCD,0x40133333}（大端序）
        // 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}（大端序）  转换为 float[]
        // float[] = {0x3F8CCCCD,0x3F99999A,0x3FA66666,0x40066666,0x400CCCCD,0x40133333}（大端序）
        // 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()

读取输出表的内容并将数据转换为 32 位整数。

语法 1

```
int[] profinet_read_output_int(
    int,
    int,
    int
)
```

参数

int	起始地址
int	要读取的地址量
int	基于小端序（DCBA）还是大端序（ABCD）将读取的数据转换为int数组。
0	小端序
1	大端序
2	根据配置文件。

返回值

int[]	以整型数组形式返回的数据。

语法 2

```
int[] profinet_read_output_int(
    int,
    int
)
```

注

与语法 1 相同，读取数据转换参数默认为 2。

*基于小端序（DCBA）还是大端序（ABCD）将读取的数据转换为 int 数组。

```
int[] var_ia = profinet_read_output_int(556,12,0)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF}（小端序）  转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000}（小端序）
    // 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}（小端序）  转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0x00FF8000}（小端序）
    // int[] = {32767.99999,16744448}
var_ia = profinet_read_output_int(556,10,0)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80}（小端序）  转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0x00008000}（小端序）
    // 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}（小端序）  转换为 int[]
    // int[] = {0xFF7F0000,0x9F860100,0x0080FFFF}（大端序）
    // 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}（小端序）  转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000}（小端序）
    // 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}（小端序）  转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000}（小端序）
    // int[] = {32767.99999,-32768}
```

语法 3

```
int profinet_read_output_int(
    int
)
```

参数

`int` 起始地址

*基于小端序（DCBA）还是大端序（ABCD）将读取的数据转换为整数。

返回值

`int` 以整数形式返回的数据

注

```
int var_i = profinet_read_output_int(556)
    // byte[] = {0xE4,0x07,0x00,0x00}（小端序）          转换为 int
    // int = 0x000007E4（小端序）
    // int = 2020
var_i = profinet_read_output_int(556)
    // byte[] = {0x00,0x00,0x07,0xE4}（大端序）          转换为 int
    // int = 0x000007E4（大端序）
    // int = 2020
```

17.8 profinet_read_output_float()

读取输出表的内容并将数据转换为 32 位浮点数。

语法 1

```
float[] profinet_read_output_float(
    int,
    int,
    int
)
```

参数

int	起始地址
int	要读取的地址量
int	基于小端序（DCBA）还是大端序（ABCD）将读取的数据转换为float数组。
0	小端序
1	大端序
2	根据配置文件。

返回值

float[] 以浮点型数组形式返回的数据。

语法 2

```
float[] profinet_read_output_float(
    int,
    int
)
```

注

与语法 1 相同，读取数据转换参数默认为 2。

*基于小端序（DCBA）还是大端序（ABCD）将读取的数据转换为float数组。

```
float[] var_fa = profinet_read_output_float(676,12,0)
    // byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40}（小端序）转换为 float[]
    // float[] = {0x3F800000,0x40000000,0x40400000}（小端序）
    // 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}（小端序）    转换为 float[]
    // float[] = {0x3F800000,0x40000000,0x00400000}（小端序）
    // 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}（小端序）    转换为 float[]
    // float[] = {0x3F800000,0x40000000,0x00000000}（小端序）
    // 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}（小端序）  转换为 float[]
    // float[] = {0x0000803F,0x00000040,0x00004040}（大端序）
    // 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}（小端序）  转换为 float[]
    // float[] = {0x3F800000,0x40000000,0x40400000}（小端序）
    // 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}（小端序）  转换为 float[]
    // float[] = {0x3F800000,0x40000000,0x40400000}（小端序）
    // float[] = {1.0,2.0,3.0}
    }
```

语法 3

```
float profinet_read_output_float(
    int
)
```

参数

`int`　　起始地址

*基于小端序（DCBA）还是大端序（ABCD）将读取的数据转换为浮点数。

返回值

`float`　以浮点数形式返回的数据

注

```
float var_f = profinet_read_output_float(676)
    // byte[] = {0x00,0x00,0x80,0x3F}（小端序）转换为 float
    // float = 0x3F800000（小端序）
    // float = 1.0
var_f = profinet_read_output_float(676)
    // byte[] = {0x3F,0x80,0x00,0x00}（大端序）转换为 float
    // float = 0x3F800000（大端序）
    // float = 1.0
```

17.9 profinet_read_output_string()

读取输出表的内容并将数据转换为 UTF8 编码字符串。

语法 1

```
string profinet_read_output_string(
    int,
    int
)
```

参数

int	起始地址
int	要读取的地址量

返回值

string	以UTF8字符串形式返回的数据（遇到0x00时结束）。

注

```
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()

读取输出表的内容并获取数据字节中某一位或某几位的值。

语法 1

```
byte profinet_read_output_bit(
    int,
    int
)
```

参数

int	起始地址
int	要获取数据字节中第几位的值

返回值

byte	以字节形式返回的数据。 位值 == 1时返回1。 位值 == 0时返回0。

注

```
byte var_b = profinet_read_output_bit(540,0)
    // 0x30    要获取的位："0"
    // 0
var_b = profinet_read_output_bit(540,5)
    // 0x30    要获取的位："5"
    // 1
```

语法 2

```
byte profinet_read_output_bit(
    string,
    int
)
```

参数

string	项目名称
int	要获取第几位的值

返回值

byte	以字节形式返回的数据。 位值 == 1时返回1。 位值 == 0时返回0。 *将以字节形式返回位数据。

注

```
byte[] var_data = {57,184,12}
profinet_write_output(540,var_data,3)
    // {00111001,10111000,00001100}（二进制）
byte var_b = profinet_read_output_bit("Register_Bit",0)
    // 1
var_b = profinet_read_output_bit("Register_Bit",17)
    // 0
```

语法 3

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

参数

int	起始地址
int	起始位
int	要读取几位

返回值

byte[]	以byte[]形式返回的数据。 位值 == 1时返回1。 位值 == 0时返回0。 *将以字节形式返回位数据，例如对于bit[0]将返回byte[0]、对于bit[1]将返回byte[1]。

注

```
byte[] var_data = {57,184,12}
profinet_write_output(540,var_data,3)
    // {00111001,10111000,00001100}（二进制）
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}
```

语法 4

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

参数

string	项目名称
int	起始位
int	要读取几位

返回值

byte[]	以byte[]形式返回的数据。 位值 == 1时返回1。 位值 == 0时返回0。 *将以字节形式返回位数据，例如对于bit[0]将返回byte[0]、对于bit[1]将返回byte[1]。

注

```
byte[] var_data = {57,184,12}
profinet_write_output(540,var_data,3)
    // {00111001,10111000,00001100}（二进制）
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()

将数据写入输出表。

语法 1

```
bool profinet_write_output (
    int,
    ?,
    int
)
```

参数

int	起始地址	
?	要写入的数据	
	*支持的数据类型包括byte、byte[]、int、int[]、float、float[]、string	
int	要写入的最大地址量	
	> 0	合法的数据长度。按地址量写入。
	<= 0	非法的数据长度。按要写入的数据的完整长度写入。

返回值

bool	True	写入成功。
	False	写入失败。 1.要写入的数据为空字符串或空数组 2.无法正确地发送和接收。

注

*在配置文件中按小端序（DCBA）或大端序（ABCD）写入数据。

语法 2

```
bool profinet_write_output(
    int,
    ?
)
```

参数

int	起始地址	
?	要写入的数据	
	*支持的数据类型包括byte、byte[]、int、int[]、float、float[]、string	

返回值

bool	True	写入成功。
	False	写入失败。 1.要写入的数据为空字符串或空数组 2.无法正确地发送和接收。

注

与语法 1 相同。默认按要写入的数据的完整长度写入。

*在配置文件中按小端序（DCBA）或大端序（ABCD）写入数据。

语法 3

```
bool profinet_write_output(
    int,
    ?,
    int,
    int
)
```

参数

int	起始地址

?		要写入的数据
		*支持的数据类型包括byte、byte[]、int、int[]、float、float[]、string
int		要写入的数据的起始地址
int		要写入的地址量

返回值

bool	True	写入成功。
	False	写入失败。

1.要写入的数据为空字符串或空数组
2.无法正确地发送和接收。

注

*在配置文件中按小端序（DCBA）或大端序（ABCD）写入数据。

```
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}（小端序）      转换为 int
    // int = 0x00007FFF（小端序）
    // int = 32767
profinet_write_output(556,var_data,1)
var_i = profinet_read_output_int(556)
    // byte[] = {0xFF,0x00,0x00,0x00}（小端序）      转换为 int
    // int = 0x000000FF（小端序）
    // 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}（小端序）   转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000}（小端序）
    // 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}（小端序）   转换为 int[]
    // int[] = {0x00007FFF,0x00000000,0x00000000}（小端序）
```

```
        // 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}（小端序）  转换为 int[]
        // int[] = {0x00007FFF,0x0001869F,0x00FF8000}（小端序）
        // 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}（小端序）  转换为 int[]
        // int[] = {0x0001869F,0x00000000,0x00000000}（小端序）
        // 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}（小端序）      转换为 float
        // float = 0xC1200000（小端序）
        // float = -10.0
profinet_write_output(676,var_data,1)
var_f = profinet_read_output_float(676)
        // byte[] = {0x00,0x00,0x00,0x00}（小端序）      转换为 float
        // float = 0x00000000（小端序）
        // 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}（小端序）  转换为 float[]
        // float[] = {0xC1200000,0x40533333,0x42F6E666}（小端序）
        // 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}（小端序）  转换为 float[]
        // float[] = {0x00200000,0x00000000,0x00000000}（小端序）
        // 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}（小端序）  转换为 float[]
        // float[] = {0xC1200000,0x40533333,0x00F6E666}（小端序）
        // 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}（小端序）  转换为 float[]
        // float[] = {0x40533333,0x00000000,0x00000000}（小端序）
        // 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"
```

语法 4

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

参数

string	项目名称
int	项目的起始移位地址
?	要写入的数据 *支持的数据类型包括**byte**、**byte[]**、**int**、**int[]**、**float**、**float[]**、**string**
int	要写入的数据的起始地址
int	要写入的地址量

返回值

bool	True	写入成功。
	False	写入失败。 1.要写入的数据为空字符串或空数组 2.无法正确地发送和接收。

注

*在配置文件中按小端序（DCBA）或大端序（ABCD）写入数据。

语法 5

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

参数

string	项目名称
int	项目的起始移位地址

?	要写入的数据 *支持的数据类型包括byte、byte[]、int、int[]、float、float[]、string	
int	要写入的数据的起始地址	

返回值

bool	True	写入成功。
	False	写入失败。 1.要写入的数据为空字符串或空数组 2.无法正确地发送和接收。

注

与语法 4 相同。默认按要写入的数据的完整长度写入。

*在配置文件中按小端序（DCBA）或大端序（ABCD）写入数据。

语法 6

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

参数

string	项目名称
int	项目的起始移位地址
?	要写入的数据 *支持的数据类型包括byte、byte[]、int、int[]、float、float[]、string

返回值

bool	True	写入成功。
	False	写入失败。 1.要写入的数据为空字符串或空数组 2.无法正确地发送和接收。

注

与语法 4 相同。写入起始地址默认为 0，默认按要写入的数据的完整长度写入。

*在配置文件中按小端序（DCBA）或大端序（ABCD）写入数据。

语法 7

```
bool profinet_write_output(
    string,
    ?
)
```

参数

string	项目名称
?	要写入的数据 *支持的数据类型包括byte、byte[]、int、int[]、float、float[]、string

返回值

bool	True	写入成功。
	False	写入失败。 1.要写入的数据为空字符串或空数组 2.无法正确地发送和接收。

注

与语法 4 相同。起始移位地址和写入起始地址默认为 0，默认按要写入的数据的完整长度写入。

*在配置文件中按小端序（DCBA）或大端序（ABCD）写入数据。

```
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}（小端序）  转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000}（小端序）
    // 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}（小端序）  转换为 int[]
    // int[] = {0x00000000,0x0001869F,0x00000000}（小端序）
    // 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}（小端序）  转换为 int[]
    // int[] = {0x00000000,0x00007FFF,0x0001869F,0xFFFF8000,0x00000000}（小端序）
    // 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}（小端序）  转换为 float[]
    // float[] = {0xC1200000,0x40533333,0x42F6E666}（小端序）
    // 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}（小端序）  转换为 float[]
    // float[] = {0x00000000,0x40533333,0x42F6E666}（小端序）
    // 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}（小端序）  转换为 float[]
    // float[] = {0x00000000,0x00000000,0xC1200000,0x40533333,0x42F6E666}（小端序）
    // float[] = {0,0,-10,3.3,123.45}
```

17.12 profinet_write_output_bit()

将内容写入输出表中数据字节中某一位或某几位的值。

语法 1

```
bool profinet_write_output_bit(
    int,
    int,
    int
)
```

参数

int	起始地址	
int	要写入数据字节中第几位的值	
int	要写入的数据	

返回值

bool	True	写入成功。	
	False	写入失败。	1.无法正确地发送和接收。

注

```
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    要获取的位："1"
    // 1
profinet_write_output_bit(540,7,0)
var_b = profinet_read_output_bit(540,7)
    // 0x72    要获取的位："7"
    // 0
```

语法 2

```
bool profient_write_output_bit(
    string,
    int,
    int
)
```

参数

int	项目名称	
int	要写入第几位的值	
int	要写入的数据	
	*将以int形式写入位数据。	

返回值

bool	True	写入成功。	
	False	写入失败。	1.无法正确地发送和接收。

注

```
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    要获取的位："1"
    // 1
profinet _write_output_bit("Register_Bit",7,0)
var_b = profinet _read_output_bit(540,7)
    // 0x72    要获取的位："7"
    // 0
```

语法 3

```
bool profinet_write_output_bit(
    int,
    int,
    byte[],
    int,
    int
)
```

参数

int	起始地址
int	起始位
byte[]	要写入的数据。 *将以字节形式写入位数据，例如对于bit[0]将写入byte[0]、对于bit[1]将写入byte[1]。
int	要写入的数据的起始位
int	要写入几位数据

返回值

bool	True	写入成功。	
	False	写入失败。	1.无法正确地发送和接收。

注

字节值 >= 1时位值 = 1
字节值 == 0时位值 = 0

语法 4

```
bool profinet_write_output_bit(
    int,
    int,
    byte[],
    int
)
```

参数

int	起始地址
int	要写入第几位的值
byte[]	要写入的数据 *将以字节形式返回位数据，例如对于bit[0]将返回byte[0]、对于bit[1]将返回byte[1]。
int	要写入的数据的起始位

返回值

bool	True	写入成功。	
	False	写入失败。	1.无法正确地发送和接收。

注

*与语法 3 相同。按要写入的剩余数据的完整长度写入。
字节值 >= 1时位值 = 1
字节值 == 0时位值 = 0

语法 5

```
bool profinet_write_output_bit(
    int,
    int,
    byte[]
)
```

参数

int	起始地址
int	要写入第几位的值
byte[]	要写入的数据
	*将以字节形式返回位数据，例如对于bit[0]将返回byte[0]、对于bit[1]将返回byte[1]。

返回值

bool	True	写入成功。
	False	写入失败。 1.无法正确地发送和接收。

注

*与语法 3 相同。写入起始位默认为 0，默认按要写入的数据的完整长度写入。

字节值 >= 1时位值 = 1

字节值 == 0时位值 = 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}
```

语法 6

```
bool profinet_write_output_bit(
    string,
    int,
    byte[],
    int,
    int
)
```

参数

string	项目名称
int	起始位
byte[]	要写入的数据
	*将以字节形式返回位数据，例如对于bit[0]将返回byte[0]、对于bit[1]将返回byte[1]。
int	要写入的数据的起始位
int	要写入几位数据

返回值

bool	True	写入成功。
	False	写入失败。 1.无法正确地发送和接收。

注

字节值 >= 1时位值 = 1
字节值 == 0时位值 = 0

语法 7

```
bool profinet_write_output_bit(
    string,
    int,
    byte[],
    int
)
```

参数

string	项目名称	
int	起始位	
byte[]	要写入的数据	
	*将以字节形式返回位数据，例如对于bit[0]将返回byte[0]、对于bit[1]将返回byte[1]。	
int	要写入的数据的起始位	

返回值

bool	True	写入成功。	
	False	写入失败。	1.无法正确地发送和接收。

注

*与语法 6 相同。按要写入的剩余数据的完整长度写入。
字节值 >= 1时位值 = 1
字节值 == 0时位值 = 0

语法 8

```
bool profinet_write_output_bit(
    string,
    int,
    byte[]
)
```

参数

string	项目名称	
int	起始位	
byte[]	要写入的数据	
	*将以字节形式返回位数据，例如对于bit[0]将返回byte[0]、对于bit[1]将返回byte[1]。	

返回值

bool	True	写入成功。	
	False	写入失败。	1.无法正确地发送和接收。

注

*与语法 6 相同。写入起始位默认为 0，默认按要写入的数据的完整长度写入。
字节值 >= 1时位值 = 1
字节值 == 0时位值 = 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 函数

机器人可通过EtherNet/IP通信协议与外部控制器通信。在EtherNet/IP通信协议机制中，机器人作为EtherNet/IP IO设备供外部设备读写机器人数据。同时，TMflow通过EtherNet/IP函数监控从外部设备接收的数据表和向外部设备发送的数据表，并更改向外部设备发送的数据表中的自定义定义部分。

通信数据表

数据表由输入数据和输出数据组成。输入数据表供外部设备向机器人发布数据，输出数据表供机器人向外部设备发送数据。两类数据表均分为系统定义数据部分和自定义定义数据部分。

1. 系统定义部分：项目和设置由机器人定义，数据内容由机器人或外部设备更新。其中定义的项目与机器人状态相关，如机器人基准、项目状态、控制柜状态，或与输入/输出状态相关，如数字输入/输出和模拟输入/输出。用户可使用EtherNet/IP函数读取系统定义部分中的输入数据表和输出数据表。
2. 自定义定义部分：项目和设置由用户定义，数据内容由用户或外部设备更新。在编辑项目的同时，用户可使用EtherNet/IP函数读写自定义定义部分中的输出数据表或读取自定义定义部分中的输入数据表，并将自定义定义部分用作项目与外部设备间的数据交换寄存器。

通信数据表 （从机器人视角）	数据部分	TMflow EtherNet/IP 函数权限	外部设备权限
输入数据表	系统定义部分	读取	写入
	自定义定义部分	读取	写入
输出数据表	系统定义部分	读取	读取
	自定义定义部分	读取/写入	读取

18.1 eip_read_input()

读取输入表的内容。

语法 1

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

参数

int	起始地址
int	要读取的地址量

返回值

byte[]	以字节数组形式返回的数据。

注

```
byte[] var_ba = eip_read_input(104,8)
    // {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32}
```

语法 2

```
byte eip_read_input(
    int,
)
```

参数

int	起始地址

返回值

byte	以字节形式返回的数据。

注

```
byte var_b = eip_read_input(104)
    // 0x30
```

语法 3

```
? eip_read_input(
    string,
    int,
    int
)
```

参数

string	项目名称
int	项目的起始移位地址
int	要读取的地址量

返回值

?	返回值的数据类型取决于通信数据表中的项目定义。 *数据类型包括**byte**、**byte[]**、**int**、**int[]**、**float**、**float[]**、**string**

语法 4

```
? eip_read_input(
    string,
    int,
)
```

参数

`string` 项目名称

`int` 项目的起始移位地址

返回值

`?` 返回值的数据类型取决于通信数据表中的项目定义。

*数据类型包括**byte**、**byte[]**、**int**、**int[]**、**float**、**float[]**、**string**

注

*与语法 3 相同。默认读取至项目末尾。

*根据配置文件按小端序（DCBA）或大端序（ABCD）读取数据。

语法 5

```
? eip_read_input(
    string
)
```

参数

`string` 项目名称

返回值

`?` 返回值的数据类型取决于通信数据表中的项目定义。

*数据类型包括 **byte**、**byte[]**、**int**、**int[]**、**float**、**float[]**、**string**

注

*与语法 3 相同。项目的起始移位地址默认为 0，默认读取至项目末尾。

*根据配置文件按小端序（DCBA）或大端序（ABCD）读取数据。

```
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}（小端序）转换为 int[]
    // int[] = {0x00000001,0x00000002,0x00000003}（小端序）
    // 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}（小端序）
        转换为 int[]
    // int[] = {0x00000004,0x00000005,0x00000006,0x00000007,0x00000008,0x00000009,
        0x0000000A,0x0000000B,0x0000000C,0x00000000,0x00000000,0x00000000}（小端序）
    // 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}（小端序）    转换为 int[]
    // int[] = {0x00000001,0x00000002,0x00000003,0x00000004,0x00000005,0x00000006,
        0x00000007,0x00000008,0x00000009,0x0000000A,0x0000000B,0x0000000C,
        0x00000000,0x00000000,0x00000000}（小端序）
    // 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}（大端序）  转换为 float[]
    // float[] = {0x3F99999A,0x3FA66666,0x40066666}（大端序）
    // 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}（大端序）
        转换为 float[]
    // float[] = {0x40066666,0x400CCCCD,0x40133333,0x00000000,0x00000000,0x00000000,
        0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000}（大端序）
    // float[] = {2.1,2.2,2.3,0,0,0,0,0,0,0,0}

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}（大端序）    转换为 float[]
    // float[] = {0x3F8CCCCD,0x3F99999A,0x3FA66666,0x40066666,0x400CCCCD,0x40133333,
        0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,
        0x00000000,0x00000000,0x00000000}（大端序）
    // float[] = { 1.1,1.2,1.3,2.1,2.2,2.3,0,0,0,0,0,0,0,0}
```

18.2 eip_read_input_int()

读取输入表的内容并将数据转换为 32 位整数。

语法 1

```
int[] eip_read_input_int(
    int,
    int,
    int
)
```

参数

int	起始地址
int	要读取的地址量
int	基于小端序（DCBA）还是大端序（ABCD）将读取的数据转换为int数组。
	0 小端序
	1 大端序
	2 根据配置文件。

返回值

int[]	以整型数组形式返回的数据。

语法 2

```
int[] eip_read_input_int(
    int,
    int
)
```

参数

int	起始地址
int	要读取的地址量

注

与语法 1 相同，读取数据转换参数默认为 2。

*基于小端序（DCBA）还是大端序将读取的数据转换为 int 数组。

```
int[] var_ia = eip_read_input_int(112,12,0)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x00,0x80,0xFF,0xFF}（小端序）转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000}（小端序）
    // int[] = {32767.99999,-32768}
var_ia = eip_read_input_int(112,11,0)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x00,0x80,0xFF}（小端序）    转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0x00FF8000}（小端序）
    // int[] = {32767.99999,16744448}
var_ia = eip_read_input_int(112,10,0)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x00,0x80}（小端序）    转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0x00008000}（小端序）
    // int[] = {32767.99999,32768}

var_ia = eip_read_input_int(112,12,1)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x00,0x80,0xFF,0xFF}（大端序）转换为 int[]
    // int[] = {0xFF7F0000,0x9F860100,0x0080FFFF}（大端序）
    // 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}（小端序）转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000}（小端序）
    // 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}（小端序）转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000}（小端序）
    // int[] = {32767.99999,-32768}
```

语法 3

```
int eip_read_input_int(
    int
)
```

参数

`int` 起始地址

*基于小端序（DCBA）还是大端序（ABCD）将读取的数据转换为int数组。

返回值

`int` 以整数形式返回的数据。

注

```
int var_i = eip_read_input_int(112)
    // byte[] = {0xE4,0x07,0x00,0x00}（小端序）    转换为 int
    // int = 0x000007E4（小端序）
    // int = 2020
var_i = eip_read_input_int(112)
    // byte[] = {0x00,0x00,0x07,0xE4}（大端序）    转换为 int
    // int = 0x000007E4（大端序）
    // int = 2020
```

18.3 eip_read_input_float()

读取输入表的内容并将数据转换为 32 位浮点数。

语法 1

```
float[] eip_read_input_float(
    int,
    int,
    int
)
```

参数

int	起始地址
int	要读取的地址量
int	基于小端序（DCBA）还是大端序（ABCD）将读取的数据转换为float数组。
0	小端序
1	大端序
2	根据配置文件。

返回值

float[] 以浮点型数组形式返回的数据。

语法 2

```
float[] eip_read_input_float(
    int,
    int
)
```

参数

int	起始地址
int	要读取的地址量

注

与语法 1 相同，读取数据转换参数默认为 2。

*基于小端序（DCBA）还是大端序（ABCD）将读取的数据转换为 float 数组。

```
float[] var_fa = eip_read_input_float(172,12,0)
    // byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40}（小端序）  转换为 float[]
    // float[] = {0x3F800000,0x40000000,0x40400000}（小端序）
    // 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}（小端序）  转换为 float[]
    // float[] = {0x3F800000,0x40000000,0x00400000}（小端序）
    // 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}（小端序）  转换为 float[]
    // float[] = {0x3F800000,0x40000000,0x00000000}（小端序）
    // 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}（小端序）转换为 float[]
    // float[] = {0x0000803F,0x00000040,0x00004040}（大端序）
    // 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}（小端序）转换为 float[]
    // float[] = {0x3F800000,0x40000000,0x40400000}（小端序）
    // 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}（小端序）转换为 float[]
    // float[] = {0x3F800000,0x40000000,0x40400000}（小端序）
    // float[] = {1.0,2.0,3.0}
```

语法 3

```
float eip_read_input_float(
    int
)
```

参数

`int` 起始地址

*基于小端序（DCBA）还是大端序（ABCD）将读取的数据转换为float数组。

返回值

`float` 以浮点数形式返回的数据。

注

```
float var_f = eip_read_input_float(172)
    // byte[] = {0x00,0x00,0x80,0x3F}（小端序）      转换为 float
    // float = 0x3F800000（小端序）
    // float = 1.0
var_f = eip_read_input_float(172)
    // byte[] = {0x3F,0x80,0x00,0x00}（大端序）      转换为 float
    // float = 0x3F800000（大端序）
    // float = 1.0
```

18.4 eip_read_input_string()

读取输入表的内容并将数据转换为 UTF8 编码字符串。

语法 1

```
string eip_read_input_string(
    int,
    int
)
```

参数

int	起始地址
int	要读取的地址量

返回值

string	以UTF8字符串形式返回的数据（遇到0x00时结束）。

注

```
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()

读取输入表的内容并获取数据字节中某一位或某几位的值。

语法 1

```
byte eip_read_input_bit(
    int,
    int
)
```

参数

int	起始地址
int	要获取数据字节中第几位的值

返回值

byte	以字节形式返回的数据。 位值 == 1时返回1。 位值 == 0时返回0。 *将以字节形式返回位数据。

注

```
byte var_b = eip_read_input_bit(104,0)
    // 0x30    要获取的位："0"
    // 0
var_b = eip_read_input_bit(104,5)
    // 0x30    要获取的位："5"
    // 1
```

语法 2

```
byte eip_read_input_bit(
    string,
    int
)
```

参数

string	项目名称
int	要获取第几位的值

返回值

byte	以字节形式返回的数据。 位值 == 1时返回1。 位值 == 0时返回0。 *将以字节形式返回位数据。

注

```
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,…}   //全部数据
    // 1   要获取的位："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,…}   //全部数据
    // 0   要获取的位："17"
```

语法 3

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

参数

int	起始地址
int	起始位
int	要读取几位

返回值

byte[]	以byte[]形式返回的数据。 位值 == 1时返回1。 位值 == 0时返回0。 *将以字节形式返回位数据，例如对于bit[0]将返回byte[0]、对于bit[1]将返回byte[1]。

注

```
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,...}   //全部数据
    // 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,...}   //全部数据
    // byte[] = {1,1,0,1,0,0,1,1}
```

语法 4

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

参数

string	项目名称
int	起始位
int	要读取几位

返回值

byte[]	以byte[]形式返回的数据。 位值 == 1时返回1。 位值 == 0时返回0。 *将以字节形式返回位数据，例如对于bit[0]将返回byte[0]、对于bit[1]将返回byte[1]。

注

```
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,...}   //全部数据
    // 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,...}   //全部数据
    // byte[] = {1,1,0,1,0,0,1,1}
```

18.6 eip_read_output()

读取输出表的内容。

语法 1

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

参数

int	起始地址
int	要读取的地址长度

返回值

byte[]	以字节数组形式返回的数据。

注

```
byte[] var_ba = eip_read_output(300,8)
    // {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32}
```

语法 2

```
byte eip_read_output(
    int
)
```

参数

int	起始地址

返回值

byte	以字节形式返回的数据。

注

```
byte var_b = eip_read_output(300)
    // 0x30
```

语法 3

```
? eip_read_output(
    string,
    int,
    int
)
```

参数

string	项目名称
int	项目的起始移位地址
int	要读取的地址量

返回值

?	返回值的数据类型取决于通信数据表中定义的项目。 *数据类型包括byte、byte[]、int、int[]、float、float[]、string

语法 4

```
? eip_read_output(
    string,
    int,
)
```

参数

`string` 项目名称

`int` 项目的起始移位地址

返回值

`?` 返回值的数据类型取决于通信数据表中定义的项目。

*数据类型包括**byte**、**byte[]**、**int**、**int[]**、**float**、**float[]**、**string**

注

*与语法 3 相同。默认读取至项目末尾。

*根据配置文件按小端序（DCBA）或大端序（ABCD）读取数据。

语法 5

```
? eip_read_output(
    string
)
```

参数

`string` 项目名称

返回值

`?` 返回值的数据类型取决于通信数据表中定义的项目。

*数据类型包括**byte**、**byte[]**、**int**、**int[]**、**float**、**float[]**、**string**

注

*与语法 3 相同。项目的起始移位地址默认为 0，默认读取至项目末尾。

*根据配置文件按小端序（DCBA）或大端序（ABCD）读取数据。

```
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}（小端序）      转换为 int
    // int = 0x000007E4（小端序）
    // 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}（小端序）转换为 int[]
    // int[] = { 0x00000001,0x00000002,0x00000003}（小端序）
    // 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}（小端序）
        转换为 int[]
    // int[] = {0x00000004,0x00000005,0x00000006,0x00000007,0x00000008,0x00000009,
        0x0000000A,0x0000000B,0x0000000C,0x00000000,0x00000000,0x00000000}（小端序）
    // 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}（小端序）    转换为 int[]
    // int[] = {0x00000001,0x00000002,0x00000003,0x00000004,0x00000005,0x00000006,
        0x00000007,0x00000008,0x00000009,0x0000000A,0x0000000B,0x0000000C,
        0x00000000,0x00000000,0x00000000}（小端序）
    // 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}（大端序）            转换为 float
    // float = 0x40400000（大端序）
    // 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}（大端序）  转换为 float[]
    // float[] = {0x3F99999A,0x3FA66666,0x40066666}（大端序）
    // 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}（大端序）  转换为 float[]
    // float[] = {0x40066666,0x400CCCCD,0x40133333}（大端序）
    // 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}（大端序）  转换为 float[]
    // float[] = {0x3F8CCCCD,0x3F99999A,0x3FA66666,0x40066666,0x400CCCCD,0x40133333}（大端序）
    // 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()

读取输出表的内容并将数据转换为 32 位整数。

语法 1

```
int[] eip_read_output_int(
    int,
    int,
    int
)
```

参数

int	起始地址
int	要读取的地址量
int	基于小端序（DCBA）还是大端序（ABCD）将读取的数据转换为int数组。
0	小端序
1	大端序
2	根据配置文件。

返回值

int[]	以整型数组形式返回的数据。

语法 2

```
int[] eip_read_output_int(
    int,
    int
)
```

参数

int	起始地址
int	要读取的地址量

注

与语法 1 相同，读取数据转换参数默认为 2。

*基于小端序（DCBA）还是大端序（ABCD）将读取的数据转换为 int 数组。

```
int[] var_ia = eip_read_output_int(308,12,0)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x00,0x80,0xFF,0xFF}（小端序）转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000}（小端序）
    // int[] = {32767.99999,-32768}
var_ia = eip_read_output_int(308,11,0)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x00,0x80,0xFF}（小端序）    转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0x00FF8000}（小端序）
    // int[] = {32767.99999,16744448}
var_ia = eip_read_output_int(308,10,0)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x00,0x80}（小端序）    转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0x00008000}（小端序）
    // int[] = {32767.99999,32768}
var_ia = eip_read_output_int(308,12,1)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x00,0x80,0xFF,0xFF}（小端序）转换为 int[]
    // int[] = {0xFF7F0000,0x9F860100,0x0080FFFF}（大端序）
    // 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}（小端序） 转换为 int[]

// int[] = {0x00007FFF,0x0001869F,0xFFFF8000}（小端序）

// 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}（小端序） 转换为 int[]

// int[] = {0x00007FFF,0x0001869F,0xFFFF8000}（小端序）

// int[] = {32767.99999,-32768}

语法 3

```
int eip_read_output_int(
    int
)
```

参数

`int` 起始地址

*基于小端序（DCBA）还是大端序（ABCD）将读取的数据转换为整数。

返回值

`int` 以整数形式返回的数据

注

int var_i = **eip_read_output_int**(308)

// byte[] = {0xE4,0x07,0x00,0x00}（小端序） 转换为 int

// int = 0x000007E4（小端序）

// int = 2020

var_i = **eip_read_output_int**(308)

// byte[] = {0x00,0x00,0x07,0xE4}（大端序） 转换为 int

// int = 0x000007E4（大端序）

// int = 2020

18.8 eip_read_output_float()

读取输出表的内容并将数据转换为 32 位浮点数。

语法 1

```
float[] eip_read_output_float(
    int,
    int,
    int
)
```

参数

int	起始地址
int	要读取的地址量
int	基于小端序（DCBA）还是大端序（ABCD）将读取的数据转换为float数组。
0	小端序
1	大端序
2	根据配置文件。

返回值

float[] 以浮点型数组形式返回的数据。

语法 2

```
float[] eip_read_output_float(
    int,
    int
)
```

参数

int	起始地址
int	要读取的地址量

注

与语法 1 相同，读取数据转换参数默认为 2。

*基于小端序（DCBA）还是大端序（ABCD）将读取的数据转换为float数组。

```
float[] var_fa = eip_read_output_float(368,12,0)
    // byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40}（小端序）转换为 float[]
    // float[] = {0x3F800000,0x40000000,0x40400000}（小端序）
    // 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}（小端序）    转换为 float[]
    // float[] = {0x3F800000,0x40000000,0x00400000}（小端序）
    // 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}（小端序）    转换为 float[]
    // float[] = {0x3F800000,0x40000000,0x00000000}（小端序）
    // 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}（小端序）转换为 float[]
    // float[] = {0x0000803F,0x00000040,0x00004040}（大端序）
    // 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}（小端序）转换为 float[]
    // float[] = {0x3F800000,0x40000000,0x40400000}（小端序）
    // 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}（小端序）转换为 float[]
    // float[] = {0x3F800000,0x40000000,0x40400000}（小端序）
    // float[] = {1.0,2.0,3.0}
```

语法 3

```
float eip_read_output_float(
    int
)
```

参数

`int` 起始地址

*基于小端序（DCBA）还是大端序（ABCD）将读取的数据转换为浮点数

返回值

`float` 以浮点数形式返回的数据

注

```
float var_f = eip_read_output_float(368)
    // byte[] = {0x00,0x00,0x80,0x3F}（小端序）    转换为 float
    // float = 0x3F800000（小端序）
    // float = 1.0
var_f = eip_read_output_float(368)
    // byte[] = {0x3F,0x80,0x00,0x00}（大端序）    转换为 float
    // float = 0x3F800000（大端序）
    // float = 1.0
```

18.9 eip_read_output_string()

读取输出表的内容并将数据转换为 UTF8 编码字符串。

语法 1

```
string eip_read_output_string(
    int,
    int
)
```

参数

int	起始地址
int	要读取的地址量

返回值

string	以UTF8字符串形式返回的数据（遇到0x00时结束）。

注

```
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()

读取输出表的内容并获取数据字节中某一位或某几位的值。

语法 1

```
byte eip_read_output_bit(
    int,
    int
)
```

参数

int	起始地址
int	要获取数据字节中第几位的值

返回值

byte	以字节形式返回的数据。 位值 == 1时返回1。 位值 == 0时返回0。 *将以字节形式返回位数据。

注

```
byte var_b = eip_read_output_bit(300,0)
    // 0x30    要获取的位："0"
    // 0
var_b = eip_read_output_bit(300,5)
    // 0x30    要获取的位："5"
    // 1
```

语法 2

```
byte eip_read_output_bit(
    string,
    int
)
```

参数

string	项目名称
int	要获取第几位的值

返回值

byte	以字节形式返回的数据。 位值 == 1时返回1。 位值 == 0时返回0。 *将以字节形式返回位数据。

注

```
byte[] var_data = {57,184,12}
eip_write_output(300,var_data,3)
    // {00111001,10111000,00001100}（二进制）
byte var_b = eip_read_output_bit("T2O_Register_Bit",0)
    // 1
var_b = eip_read_output_bit("T2O_Register_Bit",17)
    // 0
```

语法 3

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

参数

int	起始地址
int	起始位
int	要读取几位

返回值

byte[]	以byte[]形式返回的数据。 位值 == 1时返回1。 位值 == 0时返回0。 *将以字节形式返回位数据，例如对于bit[0]将返回byte[0]、对于bit[1]将返回byte[1]。

注

```
byte[] var_data = {57,184,12}
eip_write_output(300,var_data,3)
    // {00111001,10111000,00001100}（二进制）
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}
```

语法 4

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

参数

string	项目名称
int	起始位
int	要读取几位

返回值

byte[]	以byte[]形式返回的数据。 位值 == 1时返回1。 位值 == 0时返回0。 *将以字节形式返回位数据，例如对于bit[0]将返回byte[0]、对于bit[1]将返回byte[1]。

注

```
byte[] var_data = {57,184,12}
eip_write_output(300,var_data,3)
    // {00111001,10111000,00001100}（二进制）
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()

将数据写入输出表。

语法 1

```
bool eip_write_output(
    int,
    ?,
    int
)
```

参数

int	起始地址	
?	要写入的数据	
	*支持的数据类型包括byte、byte[]、int、int[]、float、float[]、string	
int	要写入的最大地址量	
	> 0	合法的数据长度。按地址量写入。
	<= 0	非法的数据长度。按要写入的数据的完整长度写入。

返回值

bool	True	写入成功。
	False	写入失败。 1.要写入的数据为空字符串或空数组 2.无法正确地发送和接收。

注

*在配置文件中按小端序（DCBA）或大端序（ABCD）写入数据。

语法 2

```
bool eip_write_output(
    int,
    ?
)
```

参数

int	起始地址	
?	要写入的数据	
	*支持的数据类型包括byte、byte[]、int、int[]、float、float[]、string	

返回值

bool	True	写入成功。
	False	写入失败。 1.要写入的数据为空字符串或空数组 2.无法正确地发送和接收。

注

与语法 1 相同。默认按要写入的数据的完整长度写入。

*在配置文件中按小端序（DCBA）或大端序（ABCD）写入数据。

语法 3

```
bool eip_write_output(
    int,
    ?,
    int,
    int
)
```

参数

int		起始地址
?		要写入的数据
		*支持的数据类型包括**byte**、**byte[]**、**int**、**int[]**、**float**、**float[]**、**string**
int		要写入的数据的起始地址
int		要写入的地址量

返回值

bool	True	写入成功。
	False	写入失败。

1.要写入的数据为空字符串或空数组
2.无法正确地发送和接收。

注

**在配置文件中按小端序（DCBA）或大端序（ABCD）写入数据。

```
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}（小端序）      转换为 int
    // int = 0x00007FFF（小端序）
    // int = 32767
eip_write_output(308,var_data,1)
var_i = eip_read_output_int(308)
    // byte[] = {0xFF,0x00,0x00,0x00}（小端序）      转换为 int
    // int = 0x000000FF（小端序）
    // 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}（小端序）转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000}（小端序）
    // 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}（小端序）转换为 int[]
        // int[] = {0x00007FFF,0x00000000,0x00000000}（小端序）
        // 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}（小端序）转换为 int[]
        // int[] = {0x00007FFF,0x0001869F,0x00FF8000}（小端序）
        // 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}（小端序）转换为 int[]
        // int[] = {0x0001869F,0x00000000,0x00000000}（小端序）
        // 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}（小端序）      转换为 float
        // float = 0xC1200000（小端序）
        // float = -10.0
eip_write_output(368,var_data,1)
var_f = eip_read_output_float(368)
        // byte[] = {0x00,0x00,0x00,0x00}（小端序）      转换为 float
        // float = 0x00000000（小端序）
        // 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}（小端序） 转换为 float[]
        // float[] = {0xC1200000,0x40533333,0x42F6E666}（小端序）
        // 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}（小端序）转换为 float[]
        // float[] = {0x00200000,0x00000000,0x00000000}（小端序）
        // 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}（小端序） 转换为 float[]
        // float[] = {0xC1200000,0x40533333,0x00F6E666}（小端序）
        // 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}（小端序） 转换为 float[]
        // float[] = {0x40533333,0x00000000,0x00000000}（小端序）
        // 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"
```

语法 4

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

参数

string	项目名称
int	项目的起始移位地址
?	要写入的数据 *支持的数据类型包括**byte**、**byte[]**、**int**、**int[]**、**float**、**float[]**、**string**
int	要写入的数据的起始地址
int	要写入的地址量

返回值

bool	True	写入成功。
	False	写入失败。 1.要写入的数据为空字符串或空数组 2.无法正确地发送和接收。

注

**在配置文件中按小端序（DCBA）或大端序（ABCD）写入数据。

语法 5

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

参数

string	项目名称
int	项目的起始移位地址
?	要写入的数据
	*支持的数据类型包括byte、byte[]、int、int[]、float、float[]、string
int	要写入的数据的起始地址

返回值

bool	True	写入成功。
	False	写入失败。
		1.要写入的数据为空字符串或空数组 2.无法正确地发送和接收。

注

与语法 4 相同。默认按要写入的数据的完整长度写入。

*在配置文件中按小端序（DCBA）或大端序（ABCD）写入数据。

语法 6

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

参数

string	项目名称
int	项目的起始移位地址
?	要写入的数据
	*支持的数据类型包括byte、byte[]、int、int[]、float、float[]、string

返回值

bool	True	写入成功。
	False	写入失败。
		1.要写入的数据为空字符串或空数组 2.无法正确地发送和接收。

注

与语法 4 相同。写入起始地址默认为 0，默认按要写入的数据的完整长度写入。

*在配置文件中按小端序（DCBA）或大端序（ABCD）写入数据。

语法 7

```
bool eip_write_output(
    string,
    ?
)
```

参数

string	项目名称
?	要写入的数据
	*支持的数据类型包括byte、byte[]、int、int[]、float、float[]、string

返回值

bool	True	写入成功。
	False	写入失败。
		1.要写入的数据为空字符串或空数组 2.无法正确地发送和接收。

注

与语法 4 相同。起始移位地址和写入起始地址默认为 0，默认按要写入的数据的完整长度写入。
*在配置文件中按小端序（DCBA）或大端序（ABCD）写入数据。

```
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}（小端序）转换为 int[]
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000}（小端序）
    // 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}（小端序）转换为 int[]
    // int[] = {0x00000000,0x0001869F,0x00000000}（小端序）
    // 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}（小端序）  转换为 int[]
    // int[] = {0x00000000,0x00007FFF,0x0001869F,0xFFFF8000,0x00000000}（小端序）
    // 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}（小端序） 转换为 float[]
    // float[] = {0xC1200000,0x40533333,0x42F6E666}（小端序）
    // 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}（小端序） 转换为 float[]
    // float[] = {0x00000000,0x40533333,0x42F6E666}（小端序）
    // 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}（小端序）  转换为 float[]
    // float[] = {0x00000000,0x00000000,0xC1200000,0x40533333,0x42F6E666}（小端序）
    // float[] = {0,0,-10,3.3,123.45}
```

18.12 eip_write_output_bit()

将内容写入输出表中数据字节中某一位或某几位的值。

语法 1

```
bool eip_write_output_bit(
    int,
    int,
    int
)
```

参数

int	起始地址	
int	要写入第几位的值	
int	要写入的数据	
	*将以int形式写入位数据。	

返回值

bool	True	写入成功。	
	False	写入失败。	1.无法正确地发送和接收。

注

```
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    要获取的位："1"
    // 1
eip_write_output_bit(300,7,0)
var_b = eip_read_output_bit(300,7)
    // 0x72    要获取的位："7"
    // 0
```

语法 2

```
bool eip_write_output_bit(
    string,
    int,
    int
)
```

参数

string	项目名称	
int	要写入第几位的值	
int	要写入的数据	
	*将以int形式写入位数据。	

返回值

bool	True	写入成功。	
	False	写入失败。	1.无法正确地发送和接收。

注

```
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    要获取的位："1"
    // 1
eip_write_output_bit("T2O_Register_Bit",7,0)
var_b = eip_read_output_bit(300,7)
    // 0x72    要获取的位："7"
    // 0
```

语法 3

```
bool eip_write_output_bit(
    int,
    int,
    byte[],
    int,
    int
)
```

参数

int	起始地址
int	要写入第几位的值
byte[]	要写入的数据
	*将以字节形式返回位数据，例如对于bit[0]将返回byte[0]、对于bit[1]将返回byte[1]。
int	要写入的数据的起始位
int	要写入几位数据

返回值

bool	True	写入成功。	
	False	写入失败。	1.无法正确地发送和接收。

注

字节值 >= 1时位值 = 1
字节值 == 0时位值 = 0

语法 4

```
bool eip_write_output_bit(
    int,
    int,
    byte[],
    int
)
```

参数

int	起始地址
int	要写入第几位的值
byte[]	要写入的数据
	*将以字节形式返回位数据，例如对于bit[0]将返回byte[0]、对于bit[1]将返回byte[1]。
int	要写入的数据的起始位

返回值

bool	True	写入成功。	
	False	写入失败。	1.无法正确地发送和接收。

注

*与语法 3 相同。按要写入的剩余数据的完整长度写入。

字节值 >= 1时位值 = 1
字节值 == 0时位值 = 0

语法 5

```
bool eip_write_output_bit(
    int,
    int,
    byte[]
)
```

参数

int	起始地址	
int	要写入第几位的值	
byte[]	要写入的数据	
	*将以字节形式返回位数据，例如对于bit[0]将返回byte[0]、对于bit[1]将返回byte[1]。	

返回值

bool	True	写入成功。	
	False	写入失败。	1.无法正确地发送和接收。

注

*与语法 3 相同。写入起始位默认为 0，默认按要写入的数据的完整长度写入。
字节值 >= 1时位值 = 1
字节值 == 0时位值 = 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}
```

语法 6

```
bool eip_write_output_bit(
    string,
    int,
    byte[],
    int,
    int
)
```

参数

string	项目名称	
int	起始位	
byte[]	要写入的数据	
	*将以字节形式返回位数据，例如对于bit[0]将返回byte[0]、对于bit[1]将返回byte[1]。	
int	要写入的数据的起始位	
int	要写入几位数据	

返回值

bool	True	写入成功。
	False	写入失败。 1.无法正确地发送和接收。

注

字节值 >= 1时位值 = 1
字节值 == 0时位值 = 0

语法 7

```
bool eip_write_output_bit(
    string,
    int,
    byte[],
    int
)
```

参数

string	项目名称
int	起始位
byte[]	要写入的数据
	*将以字节形式返回位数据，例如对于bit[0]将返回byte[0]、对于bit[1]将返回byte[1]。
int	要写入的数据的起始位

返回值

bool	True	写入成功。
	False	写入失败。 1.无法正确地发送和接收。

注

*与语法 6 相同。按要写入的剩余数据的完整长度写入。
字节值 >= 1时位值 = 1
字节值 == 0时位值 = 0

语法 8

```
bool eip_write_output_bit(
    string,
    int,
    byte[]
)
```

参数

string	项目名称
int	起始位
byte[]	要写入的数据
	*将以字节形式返回位数据，例如对于bit[0]将返回byte[0]、对于bit[1]将返回byte[1]。

返回值

bool	True	写入成功。
	False	写入失败。 1.无法正确地发送和接收。

注

*与语法 6 相同。写入起始位默认为 0，默认按要写入的数据的完整长度写入。
字节值 >= 1时位值 = 1
字节值 == 0时位值 = 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. 力控制函数
19.1 FTSensor 类

可通过使用 FTSensor 类并声明变量创建力/扭矩传感器设备。变量名将成为设备名称。

构造 1

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

参数

`string`	传感器供应商提供的支持的型号
	`"ATI_Axia80"`
	`"OnRobot_HEX-E"`
	`"OnRobot_HEX-H"`
	`"ROBOTIQ_FT300"`

`string`	使用串行端口通信时需要的连接说明
`float[]`	位置设置：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)
`float[]`	TCP值：X(mm)、Y(mm)、Z(mm)
`float`	工具重量（kg）

注

```
FTSensor fts_1 = "ATI_Axia80","COM2"     //构造设备ATI_Axia80。使用EtherCAT通信时，连接参数无效。
FTSensor fts_2 = "ATI_Axia80"            //构造设备ATI_Axia80。
FTSensor fts_3 = "ROBOTIQ_FT300","COM2"
    //构造设备ROBOTIQ_FT300。需指定用于串行端口通信的连接端口。
FTSensor fts_4 = "ROBOTIQ_FT300","COM2",{0,0,0,0,0,0},{0.12,0.24,0.36},1
    //构造设备ROBOTIQ_FT300。需指定用于串行端口通信的连接端口，并配置传感器位置和工具质心设置。
```

*若未填写位置设置、TCP 值或工具质量，将使用项目中力传感器设置中的参数。
*在流程项目或脚本项目中构造设备后，不会主动连接设备，直至开始读取或写入。

成员属性

名称	类型	模式	说明	格式
X	float	R	X 轴受力强度值	
Y	float	R	Y 轴受力强度值	
Z	float	R	Z 轴受力强度值	
TX	float	R	X 轴扭矩值	
TY	float	R	Y 轴扭矩值	
TZ	float	R	Z 轴扭矩值	
F3D	float	R	XYZ 受力强度值	
T3D	float	R	XYZ 扭矩值	
Value	float[]	R	XYZ 受力强度值和扭矩值数组。	{X, Y, Z, TX, TY, TZ}，大小 = 6
ForceValue	float[]	R	XYZ 受力强度值数组	{X, Y, Z}，大小 = 3
TorqueValue	float[]	R	XYZ 扭矩值数组	{TX, TY, TZ}，大小 = 3
RefCoordX	float	R	基于节点中设置的参考坐标系测得的 X 轴受力强度值	
RefCoordY	float	R	基于节点中设置的参考坐标系测得的 Y 轴受力强度值	
RefCoordZ	float	R	基于节点中设置的参考坐标系测得的 Z 轴受力强度值	
RefCoordTX	float	R	基于节点中设置的参考坐标系测得的 X 轴扭矩值	
RefCoordTY	float	R	基于节点中设置的参考坐标系测得的 Y 轴扭矩值	
RefCoordTZ	float	R	基于节点中设置的参考坐标系测得的 Z 轴扭矩值	
RefCoordF3D	float	R	基于节点中设置的参考坐标系测得的 XYZ 受力强度	
RefCoordT3D	float	R	基于节点中设置的参考坐标系测得的 XYZ 扭矩	
RefCoordForceValue	float[]	R	基于节点中设置的参考坐标系测得的 XYZ 受力强度值矩阵	{RefCoordX, RefCoordY, RefCoordZ}，大小 = 3
RefCoordTorqueValue	float[]	R	基于节点中设置的参考坐标系测得的 XYZ 扭矩值矩阵	{RefCoordTX, RefCoordTY, RefCoordTZ}，大小 = 3
Model	string	R	F/T 传感器型号名称	
Zero	byte	R/W	开启或关闭 F/T 传感器偏移	0：关闭零点偏移、1：开启零点偏移

*进行力控制运动时，与RefCoord*相关的属性均有值。

成员方法

名称	说明
Open()	连接至传感器设备。
Close()	从传感器设备断开连接。

19.1.1 Open()

开启至力/扭矩传感器的连接。

语法 1

```
bool Open(
)
```

参数

void		无参数

返回值

bool	True	开启成功
	False	开启失败（项目返回错误）

*开启设备后，将继续连接设备以通信。由于传感器型号不同，获取值可能需要一些时间。

19.1.2 Close()

关闭至力/扭矩传感器的连接。

语法 1

```
bool Close(
)
```

参数

void		无参数

返回值

bool	True	关闭成功
	False	关闭失败

注

```
FTSensor fts_1 = "ROBOTIQ_FT300","COM2"
fts_1.Open()        //连接至设备。
fts_1.Close()       //从设备断开连接。
```

19.2 Force 类

用户可通过使用 Force 类并声明变量设置机器人目标力和扭矩参数，以实现各种相关力控制运动。

构造 1

Force *VariableName* = string

参数

string 传感器名称

注

```
FTSensor fts1 = "ATI_Axia80"      //构造设备ATI_Axia80。
Force fc1 = "fts1"                //声明力控制变量并将fts1固定为传感器名称。
```

成员方法

名称	默认值	说明
Reset()		将除传感器名称外的所有力控制运动参数复位为默认值。
Frame()	1	设置力控制运动的参考坐标。
StopDuration()	200	合规停止时长。设置在力控制和位置控制间切换的时长。
Distance()	-	力控制运动的移动距离（仅在 SetPoint F/T 运行模式下可用。）
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	力控制运动的力和扭矩。正负值表示力控制方向。用户可调整PID控制参数。
Trajectory()	-	力控制运动的轨迹路径。（适用于轨迹F/T运行模式下。）
Timeout()	-	超时停止条件（仅在单 F/T 运行模式下可用。）
AllowPosTol()	-	容许位置误差停止条件
DInput()	-	数字输入停止条件
AInput()	-	模拟输入停止条件
FTReached()	-	监控传感器上的力、扭矩或合力值的停止条件。
Condition()	-	条件表达式停止条件
Start()	true、false	开始力控制运动。
Stop()		停止力控制运动。

19.2.1 Reset()

将除传感器名称外的所有力控制运动参数复位为默认值。

语法 1

```
void Reset(
)
```

参数

void 无参数

返回值

void 无返回值

19.2.2 Frame()

设置力控制运动的参考坐标。

语法 1

```
void Frame(
    int
)
```

参数

int	与力控制运动关联的基准 0 机器人基准 1 工具：与工具坐标方向关联的基准。（默认值） 2 当前基准 3 轨迹：随路径变化的基准。

返回值

void	无返回值

语法 2

```
void Frame(
    string
)
```

参数

string	点名称。使用点的TCP坐标作为与力控制运动关联的基准。

返回值

void	无返回值

语法 3

```
void Frame(
    float[],
    string
)
```

参数

float[]	机器人端点的TCP坐标：X(mm)、Y(mm)、Z(mm)、RX(°)、RY(°)、RZ(°)
string	基准名称。若为空字符串，则将使用当前基准名称。

返回值

void	无返回值

语法 4

```
void Frame(
    float[]
)
```

注

与语法 3 相同。基准名称为 RobotBase。

语法 5

```
void Frame(
    float, float, float, float, float, float,
    string
)
```

注

与语法 3 相同。float[]类型被替换为 float 参数。

语法 6

```
void Frame(
    float, float, float, float, float, float
)
```

注

与语法 4 相同。float[]类型被替换为 float 参数。

19.2.3 StopDuration()

设置合规停止时长。设置在力控制和位置控制间切换的时长。

语法 1

```
void StopDuration(
    int
)
```

参数

int	合规停止时长，单位为毫秒

返回值

void	无返回值

19.2.4 Distance()

设置力控制运动的移动距离。（仅在SetPoint F/T运行模式下可用。）

语法 1

```
void Distance(
    int
)
```

参数

int	距离（mm）
< 0	不限制移动距离。
>= 0	移动距离限制。（从力控制运动开始的起点算起的距离。）

返回值

void	无返回值

19.2.5 FTSet()

设置力控制运动的力和扭矩。正负值表示力控制方向。用户可调整PID控制参数。

语法 1

```
void FTSet(
    int or string,
    bool,
    float,
    int
)
```

参数

int or string	轴力或轴扭矩

	0	或	"FX"	FX (N)	3	或	"TX"	TX (Nm)
	1	或	"FY"	FY (N)	4	或	"TY"	TY (Nm)
	2	或	"FZ"	FZ (N)	5	或	"TZ"	TZ (Nm)

`bool` 是否启用对指定的轴力或轴扭矩的控制

false	禁用
true	启用

`float` 力或扭矩的控制值。正负值表示力控制方向。

`int` PID控制参数

（弱）				（强）
0	1	2	3	4

返回值

`void` 无返回值

语法 2

```
void FTSet(
    int or string,
    bool,
    float
)
```

注

与语法 1 相同。PID 控制参数为 2。

语法 3

```
void FTSet(
    int or string,
    bool
)
```

注

参数定义与语法 1 相同。用于设置是否启用对指定的轴力或轴扭矩的控制

语法 4

```
void FTSet(
    int or string,
    float
)
```

注

参数定义与语法 1 相同。用于设置力或扭矩的控制值。

语法 5

```
void FTSet(
    int or string,
    int
)
```

注

参数定义与语法 1 相同。用于设置 PID 控制参数。

语法 6

```
void FTSet(
    int or string,
    float[]
)
```

参数

`int or string`

轴力或轴扭矩

0	或	"FX"	FX (N)	3	或	"TX"	TX (Nm)
1	或	"FY"	FY (N)	4	或	"TY"	TY (Nm)
2	或	"FZ"	FZ (N)	5	或	"TZ"	TZ (Nm)

`float[]` PID控制参数{Kp, Ki, Kd}

19.2.6 Trajectory()

设置力控制运动的轨迹路径。（适用于轨迹F/T运行模式下。）

语法 1

```
void Trajectory(
    string
)
```

参数

`string` 子流程名称。只能在流程项目中使用。若在脚本项目中使用，将返回错误。

返回值

`void` 无返回值

语法 2

```
void Trajectory(
    ?
)
```

参数

`?` 轨迹路径。为语句或自定义函数

返回值

`void` 无返回值

语法 3

```
void Trajectory(
)
```

参数

`void`

无参数，用于解除轨迹F/T运行模式。（改为SetPoint F/T运行模式。）

返回值

`void` 无返回值

19.2.7 Timeout()

设置超时停止条件。（仅在SetPoint F/T运行模式下可用。）

语法 1

```
void Timeout(
    int
)
```

参数

`int` 超时时间，单位为毫秒

`< 0` 禁用

	>= 0	超时时长

返回值

void	无返回值

语法 2

```
void Timeout(
)
```

参数

void	无参数，用于取消停止条件。

返回值

void	无返回值

19.2.8 AllowPosTol()

设置容许位置误差停止条件。

语法 1

```
void AllowPosTol(
    int
)
```

参数

int	误差距离，单位为mm	
	< 0	禁用
	>= 0	容许误差距离。

返回值

void	无返回值

语法 2

```
void AllowPosTol(
)
```

参数

void	无参数，用于取消停止条件。

返回值

void	无返回值

19.2.9 DInput()

设置数字输入停止条件。

语法 1

```
void DInput(
    string,
    int,
    int or string
)
```

参数

string	控制模块名称	
	ControlBox	控制柜
	EndModule	末端模块
	ExtModuleN	外部模块（N = 0..n）

int	输入通道，0..n	
int or string		
	将停止条件设为低电平/高电平。	
	0 或 "L"	低电平
	1 或 "H"	高电平

返回值

void	无返回值

语法 2

```
void DInput(
)
```

参数

void	无参数，用于取消停止条件。

返回值

void	无返回值

19.2.10 AInput()

设置模拟输入停止条件。

语法 1

```
void AInput(
    string,
    int,
    int or string,
    float
)
```

参数

string	控制模块名称	
	ControlBox	控制柜
	EndModule	末端模块
	ExtModuleN	外部模块（N = 0..n）
int	输入通道，0..n	
int or string		
	设置判断条件	
	0 或 ">"	大于
	1 或 ">="	大于等于
	2 或 "=="	
		等于（由于难以与模拟输入保持相等，不建议使用。）
	3 or "<="	小于等于
	4 or "<"	小于
float	条件值	

返回值

void	无返回值

语法 2

```
void AInput(
)
```

参数

void	无参数，用于取消停止条件。

返回值

`void` 无返回值

19.2.11 FTReached()

设置监控传感器上的力、扭矩或合力值的停止条件。

语法 1

```
void FTReached(
    int or string,
    bool,
    float
)
```

参数

`int` 力、扭矩或合力值。

0	或	"FX"	FX (N)	3	或	"TX"	TX (Nm)
1	或	"FY"	FY (N)	4	或	"TY"	TY (Nm)
2	或	"FZ"	FZ (N)	5	或	"TZ"	TZ (Nm)
6	或	"F3D"	F3D (N)	7	或	"T3D"	T3D (Nm)

`bool` 是否启用对指定的力、扭矩或合力值的监控。

false	禁用
true	启用

`float` 监控值

返回值

`void` 无返回值

语法 2

```
void FTReached(
    int or string,
    bool
)
```

注

与语法 1 相同。用于启用或禁用对指定的力、扭矩或合力值的监控。

语法 3

```
void FTReached(
    bool
)
```

参数

`bool` 是否启用绝对值监控。

false	禁用
true	启用

语法 4

```
void FTReached(
)
```

参数

`void` 无参数，用于取消停止条件。

返回值

`void` 无返回值

19.2.12 Condition()

设置条件表达式停止条件。

语法 1

```
void Condition(
    bool or ?
)
```

参数

bool or ?	条件。为true/false或返回bool值的语句。

返回值

void	无返回值

语法 2

```
void Condition(
)
```

参数

void	无参数，用于取消停止条件。

返回值

void	无返回值

19.2.13 Start()

开始力控制运动。

语法 1

```
int Start(
    bool,
    bool
)
```

参数

bool	执行前是否将力传感器清零。	
	true	启用（默认值）
	false	禁用
bool	是否启用工具重力补偿。	
	true	启用
	false	禁用（默认值）

返回值

int	力控制运动停止后返回的结果值。	
	0	未在工作
	1	正在工作
	2	超时
	3	已达到距离
	4	已触发IO
	5	受到阻挡
	6	错误
	14	超速
	201	已触发数字IO
	202	已触发模拟IO

203	变量
204	已满足力条件
205	容许位置公差
206	运动结束

语法 2

```
int Start(
    bool
)
```

注

与语法 1 相同。启用工具重力补偿参数后，将从项目的力传感器设置中获取相关设备名称的参数值。若未取得相关设备名称，则填充 false。

语法 3

```
int Start(
)
```

注

与语法 1 相同。“执行前是否将力传感器清零”参数为 true。启用工具重力补偿参数后，将从项目的力传感器设置中获取相关设备名称的参数值。若未取得相关设备名称，则填充 false。

19.2.14 Stop()

停止力控制运动。

语法 1

```
int Stop(
)
```

参数

void	无参数

返回值

int	力控制运动停止后返回的结果值。

设置参数

```
FTSensor fts1 = "ATI_Axia80”      //构造设备ATI_Axia80。
Force fc1 = "fts1”                //声明力控制变量并将fts1固定为传感器名称。

(1)
    fc1.Frame(3)                                  //将力控制坐标设为轨迹。
    fc1.Frame({517.5,-147.8,442.45,180,0,90})     //将力控制坐标设为点（将覆写之前的设置）
    fc1.Frame("P1")                               //将力控制坐标设为点（将覆写之前的设置）
    fc1.Reset()                                   //复位所有参数（保留固定的"fts1"作为传感器名称）
(2)
    fc1.Frame(3)
    fc1.Frame({517.5,-147.8,442.45,180,0,90})
    fc1.Frame("P1")
    fc1.FTSet(0, true, 5, 0)                  //设置轴FX，使力控制为5 N、PID为0。
    fc1.FTSet(1, true, 6, 1)                  //设置轴FY，使力控制为6 N、PID为1。
    fc1.FTSet(2, true, 7, 3)                  //设置轴FZ，使力控制为7 N、PID为3。
    fc1.FTSet("FZ", true, 8)                  //设置轴FZ，使力控制为8 N、PID为2。（将覆写之前的FZ设置）
    fc1.Reset()                               //复位所有参数（保留固定的"fts1"作为传感器名称）
(3)
    fc1.Frame(1)                              //将力控制坐标设为工具。
    fc1.StopDuration(300)                     //将合规停止时长设为300 ms
    //力和扭矩控制
    fc1.Distance(1000)                        //将移动距离设为1000 mm
    fc1.FTSet("FZ", true, 5)                  //设置轴FZ，使力控制为5 N
    //停止条件
    fc1.Timeout(10000)                        //将超时时间设为10000 ms
    fc1.AllowPosTol(100)                      //将容许误差设为100 mm
    fc1.DInput("ControlBox", 0, "H")          //当ControlBox DI0为高电平时
    fc1.AInput("ControlBox", 0, ">=", 3.3)    //当ControlBox AI0大于等于3.3 V时
    fc1.FTReached(2, true, 1)                 //轴FZ满足1 N条件
    fc1.FTReached("FZ", true, 2)              //轴FZ满足2 N条件（将覆写之前的FZ设置）
    fc1.FTReached(true)                       //启用绝对值监控
    int count = 0
    fc1.Condition(count > 100)                //条件表达式
    fc1.Reset()                               //复位所有参数（保留固定的"fts1"作为传感器名称）
```

SetPoint F/T 运行模式

SetPoint 模式主要适用于通过力控制使机器人接触目标。

```
FTSensor fts1 = "ATI_Axia80"
Force fc1 = "fts1"
(1)
    PTP("JPP",{0,0,90,0,90,0},50,200,0,false)
    fc1.Distance(100)                          //将移动距离设为100 mm
    fc1.FTSet("FZ", true, 5)                   //设置轴FZ，使力控制为5 N
    fc1.Start()                                //开始力控制运动。执行前清零力传感器。
    //    设置轴FZ，使力控制为5 N、移动距离为100 mm，且机器人将沿Z方向移动。机器人一旦离开起点超过100 mm，就会停止移动。但运动仍处于力控制下，即在Start()函数中。

(2)
    PTP("JPP",{0,0,90,0,90,0},50,200,0,false)
    fc1.FTSet("FZ", true, 5)                   //设置轴FZ，使力控制为5 N
    fc1.AllowPosTol(80)                        //容许位置误差停止条件。
    int re = fc1.Start()                       //开始力控制运动。执行前清零力传感器。
    //    设置轴FZ，使力控制为5 N，且机器人将沿Z方向移动。在同时监控停止条件下，机器人一旦离开起点超过80 mm，就会因满足停止条件而停止移动，退出Start()函数，并返回值205。

(3)
    PTP("JPP",{0,0,90,0,90,0},50,200,0,false)
    fc1.Frame(1)                               //将力控制坐标设为工具
    fc1.StopDuration(300)                      //将合规停止时长设为300 ms
    //力和扭矩控制
    fc1.Distance(200)                          //将移动距离设为200 mm
    fc1.FTSet("FZ", true, 5)                   //设置轴FZ，使力控制为5 N
    //停止条件
    fc1.Timeout(10000)                         //将超时时间设为10000 ms
    fc1.AllowPosTol(1000)                      //将容许误差设为1000 mm
    fc1.DInput("ControlBox", 0, "H")           //当ControlBox DI0为高电平时
    fc1.AInput("ControlBox", 0, ">=", 3.3)     //当ControlBox AI0 >= 3.3 V时
    int count = 0
    fc1.Condition(count > 100)                 //条件表达式
    int re = fc1.Start()                       //开始力控制运动。执行前清零力传感器。
    //    设置轴FZ，使力控制为5 N、移动距离为200 mm，且机器人将沿Z方向移动。在同时监控停止条件下，机器人一旦离开起点超过200 mm，就会停止移动。但运动仍处于力控制下，仍将继续监控停止条件。满足任意停止条件后，才会停止力控制运动、退出Start()函数，并返回停止后的结果值。
```

轨迹 F/T 运行模式

轨迹模式主要适用于通过机器人运动控制和力控制使机器人接触目标。

```
FTSensor fts1 = "ATI_Axia80"
Force fc1 = "fts1"
(1)
    fc1.FTSet("FZ", true, 2)                    //设置轴FZ，使力控制为2 N
    fc1.Trajectory("FTSubflow0")                //将轨迹路径设为FTSubflow0。（只能在流程项目中使用。）
    int re = fc1.Start()                        //开始力控制运动。执行前清零力传感器。
    //    假设FTSubflow0中有数个待完成的点节点。
    //    设置轴FZ，使力控制为2 N、轨迹路径为子流程FTSubflow0。机器人将按照子流程中的节点进行控
          制，同时还将进行力控制。子流程结束时，将停止力控制运动、退出Start()函数，并返回值206。
(2)
    fc1.FTSet("FZ", true, 2)                    //设置轴FZ，使力控制为2 N
    fc1.Frame(3)                                //将力控制坐标设为轨迹
    fc1.Trajectory("FTSubflow0")                //将轨迹路径设为FTSubflow0。（只能在流程项目中使用。）
    int re = fc1.Start()                        //开始力控制运动。执行前清零力传感器。
    //    假设FTSubflow0中有数个待完成的点节点。
    //    设置轴FZ，使力控制为2 N、轨迹路径为子流程FTSubflow0。机器人将按照子流程中的节点进行控
          制，同时还将进行力控制。子流程结束时，将停止力控制运动、退出Start()函数，并返回值206。

(3)
    fc1.FTSet("FZ", true, 2)                    //设置轴FZ，使力控制为2 N
    fc1.Frame(1)                                //将力控制坐标设为工具
    fc1.Trajectory("FTSubflow1")                //将轨迹路径设为FTSubflow1。（只能在流程项目中使用。）
    //停止条件
    fc1.Timeout(10000)                          //超时时间在轨迹模式下无效。
    fc1.AllowPosTol(1000)                       //将容许误差设为1000 mm
    fc1.DInput("ControlBox", 0, "H")            //当ControlBox DI0为高电平时
    fc1.FTReached("FZ", true, 2)                //轴FZ满足2 N条件
    fc1.FTReached(true)                         //启用绝对值监控
    int re = fc1.Start()                        //开始力控制运动。执行前清零力传感器。
    //    假设FTSubflow1中有数个循环执行的点节点。
    //    设置轴FZ，使力控制为2 N、轨迹路径为子流程FTSubflow1。机器人将按照子流程中的节点进行控
          制，同时还将进行力控制。由于子流程会循环执行，将持续进行机器人运动、力控制和停止条件监
          控。满足任意停止条件后，才会停止力控制运动、退出Start()函数，并返回停止后的结果值。

(4)
    define
    {
        FTSensor fts1 = "ATI_Axia80"
        Force fc1 = "fts1"
        int count = 0
    }
    main
    {
        fc1.FTSet("FZ", true, 1, 0)             //设置轴FZ，使力控制为8 N、PID为0
        fc1.Frame(1)                            //将力控制坐标设为工具
        fc1.Trajectory(FTMotion())              //将轨迹路径设为自定义函数FTMotion()。
        //停止条件
```

```
    fc1.Timeout(10000)                  //超时时间在轨迹模式下无效。
    fc1.AllowPosTol(1000)               //将容许误差设为1000 mm
    fc1.DInput("ControlBox", 0, "H")    //当ControlBox DI0为高电平时
    fc1.FTReached("FZ", true, 2)        //轴FZ满足2 N条件
    fc1.FTReached(true)                 //启用绝对值监控
    fc1.Condition(count++ > 200000)     //条件表达式
    int re = fc1.Start()
                                        //开始力控制运动。执行前清零力传感器。
    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)
    }
}
//  设置轴FZ，使力控制为1 N、PID为0、轨迹路径为自定义函数FTMotion()。机器人将按照FTMotion()的
    内容进行控制，同时还将进行力控制。由于该函数会循环执行，将持续进行机器人运动、力控制和停
    止条件监控。满足任意停止条件后，才会停止力控制运动、退出Start()函数，并返回停止后的结果值。
```

购买欧姆龙产品的客户须知

承诺事项

承蒙对欧姆龙株式会社（以下简称“本公司”）产品的一贯厚爱和支持，藉此机会再次深表谢意。

如果未特别约定，无论贵司从何处购买的产品，都将适用本承诺事项中记载的事项。

请在充分了解这些注意事项基础上订购。

1. 定义

本承诺事项中的术语定义如下。

(1)“本公司产品”：是指“本公司”的FA系统机器、通用控制器、传感器、电子/结构部件。

(2)“产品目录等”：是指与“本公司产品”有关的欧姆龙综合产品目录、FA系统设备综合产品目录、安全组件综合产品目录、电子/机构部件综合产品目录以及其他产品目录、规格书、使用说明书、操作指南等，包括以电子数据方式提供的资料。

(3)“使用条件等”：是指在“产品目录等”资料中记载的“本公司产品”的使用条件、额定值、性能、运行环境、操作使用方法、使用时的注意事项、禁止事项以及其他事项。

(4)“客户用途”：是指客户使用“本公司产品”的方法，包括将“本公司产品”组装或运用到客户生产的部件、电子电路板、机器、设备或系统等产品中。

(5)“适用性等”：是指在“客户用途”中“本公司产品”的(a)适用性、(b)动作、(c)不侵害第三方知识产权、(d)法规法令的遵守以及(e)满足各种规格标准。

2. 关于记载事项的注意事项

对“产品目录等”中的记载内容，请理解如下要点。

(1) 额定值及性能值是在单项试验中分别在各条件下获得的值，并不构成对各额定值及性能值的综合条件下获得值的承诺。

(2) 提供的参考数据仅作为参考，并非可在该范围内一直正常运行的保证。

(3) 应用示例仅作参考，不构成对“适用性等”的保证。

(4) 如果因技术改进等原因，“本公司”可能会停止“本公司产品”的生产或变更“本公司产品”的规格。

3. 使用时的注意事项

选用及使用本公司产品时请理解如下要点。

(1) 除了额定值、性能指标外，使用时还必须遵守“使用条件等”。

(2) 客户应事先确认“适用性等”，进而再判断是否选用“本公司产品”。“本公司”对“适用性等”不做任何保证。

(3) 对于“本公司产品”在客户的整个系统中的设计用途，客户应负责事先确认是否已进行了适当配电、安装等事项。

(4) 使用“本公司产品”时，客户必须采取如下措施：(i)相对额定值及性能指标，必须在留有余量的前提下使用“本公司产品”，并采用冗余设计等安全设计(ii)所采用的安全设计必须确保即使“本公司产品”发生故障时也可将“客户用途”中的危险降到最小程度、(iii)构建随时提示使用者危险的完整安全体系、(iv)针对“本公司产品”及“客户用途”定期实施各项维护保养。

(5) 因DDoS攻击（分布式DoS攻击）、计算机病毒以及其他技术性有害程序、非法侵入，即使导致“本公司产品”、所安装软件、或者所有的计算机器材、计算机程序、网络、数据库受到感染，对于由此而引起的直接或间接损失、损害以及其他费用，“本公司”将不承担任何责任。

对于(i)杀毒保护、(ii)数据输入输出、(iii)丢失数据的恢复、(iv)防止“本公司产品”或者所安装软件感染计算机病毒、(v)防止对“本公司产品”的非法侵入，请客户自行负责采取充分措施。

(6)“本公司产品”是作为应用于一般工业产品的通用产品而设计生产的。如果客户将“本公司产品”用于以下所列用途，则本公司对产品不作任何保证。但“本公司”已表明可用于特殊用途，或已与客户有特殊约定时，另行处理。

(a) 必须具备很高安全性的用途（例：核能控制设备、燃烧设备、航空/宇宙设备、铁路设备、升降设备、娱乐设备、医疗设备、安全装置、其他可能危及生命及人身安全的用途）

(b) 必须具备很高可靠性的用途（例：燃气、自来水、电力等供应系统、24小时连续运行系统、结算系统、以及其他处理权利、财产的用途等）

(c) 具有苛刻条件或严酷环境的用途（例：安装在室外的设备、会受到化学污染的设备、会受到电磁波影响的设备、会受到振动或冲击的设备等）

(d)“产品目录等”资料中未记载的条件或环境下的用途

(7) 除了不适用于上述3.(6)(a)至(d)中记载的用途外，“本产品目录等资料中记载的产品”也不适用于汽车（含二轮车，以下同）。请勿配置到汽车上使用。关于汽车配置用产品，请咨询本公司销售人员。

4. 保修条件

“本公司产品”的保修条件如下。

(1) 保修期限　自购买之日起1年。（但是，“产品目录等”资料中有明确说明时除外。）

(2) 保修内容　对于发生故障的“本公司产品”，由“本公司”判断并可选择以下其中之一方式进行保修。

(a) 在本公司的维修保养服务点对发生故障的“本公司产品”进行免费修理（但是对于电子、结构部件不提供修理服务。）

(b) 对发生故障的“本公司产品”免费提供同等数量的替代品

(3) 当故障因以下任何一种情形引起时，不属于保修的范围。

(a) 将“本公司产品”用于原本设计用途以外的用途

(b) 超过“使用条件等”范围的使用

(c) 违反本注意事项“3.使用时的注意事项”的使用

(d) 非因“本公司”进行的改装、修理导致故障时

(e) 非因“本公司”出品的软件导致故障时

(f)“本公司”生产时的科学、技术水平无法预见的原因

(g) 除上述情形外的其它原因，如“本公司”或“本公司产品”以外的原因（包括天灾等不可抗力）

5. 责任限制

本承诺事项中记载的保修是关于“本公司产品”的全部保证。对于因“本公司产品”而发生的其他损害，“本公司”及“本公司产品”的经销商不负任何责任。

6. 出口管理

客户若将“本公司产品”或技术资料出口或向境外提供时，请遵守中国及各国关于安全保障进出口管理方面的法律、法规。否则，“本公司”有权不予提供“本公司产品”或技术资料。

IC320GC-zh
202404

注：规格如有变更，恕不另行通知。请以最新产品说明书为准。

欧姆龙自动化（中国）有限公司

http://www.fa.omron.com.cn　　咨询热线：**400-820-4535**