



Expression Editor

and

Listen Node

軟體版本: 1.82
文件版本: 1.00
發布日期: 2021-02-08

本文記述 Techman Robot 機器人產品系列（以下簡稱 TM Robot ）資訊，所有資訊屬於達明機器人（股）公司（以下簡稱本公司）財產，未經本公司事先授權不得以任何形式或方式轉載及複製任何資料。本文任何資訊不應視為任何要約或是承諾，日後如有變更，恕不另行通知。本說明書應定期審查，本公司不會對本文任何錯誤或是遺漏承擔責任。

 和  標誌為達明機器人（股）公司註冊商標，本公司保留本說明書及其拷貝的所有權及其著作權。

 達明機器人股份有限公司
TECHMAN ROBOT INC.

Contents

修訂表	11
1. 運算式.....	12
1.1 型別.....	12
1.2 變數與常數.....	13
1.3 陣列.....	16
1.4 運算子.....	17
1.5 型別轉換.....	19
1.6 警告訊息.....	21
2. 函式.....	23
2.1 Byte.ToInt16().....	23
2.2 Byte.ToInt32().....	25
2.3 Byte.ToDouble().....	26
2.4 Byte.ToDouble().....	27
2.5 Byte.ToInt16Array().....	28
2.6 Byte.ToInt32Array().....	30
2.7 Byte.ToFloatArray()	31
2.8 Byte.ToDoubleArray()	32
2.9 Byte.ToString().....	33
2.10 Byte.Concat().....	34
2.11 String.ToInteger()	38
2.12 String.ToFloat()	40
2.13 String.ToDouble()	42
2.14 String.ToByte()	44
2.15 String.IndexOf()	45
2.16 String.LastIndexOf().....	46
2.17 String.Substring().....	47

2.18	<i>String_Split()</i>	50
2.19	<i>String_Replace()</i>	51
2.20	<i>String_Trim()</i>	52
2.21	<i>String_ToLower()</i>	54
2.22	<i>String_ToUpper()</i>	55
2.23	<i>Array_Append()</i>	56
2.24	<i>Array_Insert()</i>	57
2.25	<i>Array_Remove()</i>	58
2.26	<i>Array_Equals()</i>	59
2.27	<i>Array_IndexOf()</i>	61
2.28	<i>Array_LastIndexOf()</i>	62
2.29	<i>Array_Reverse()</i>	63
2.30	<i>Array_Sort()</i>	66
2.31	<i>Array_SubElements()</i>	67
2.32	<i>ValueReverse()</i>	69
2.33	<i>GetBytes()</i>	73
2.34	<i>GetString()</i>	78
2.35	<i>GetToken()</i>	86
2.36	<i>GetAllTokens()</i>	95
2.37	<i>GetNow()</i>	97
2.38	<i>GetNowStamp()</i>	99
2.39	<i>GetVarValue()</i>	102
2.40	<i>Length()</i>	103
2.41	<i>Ctrl()</i>	105
2.42	<i>XOR8()</i>	107
2.43	<i>SUM8()</i>	109
2.44	<i>SUM16()</i>	111
2.45	<i>SUM32()</i>	113

2.46	<i>CRC16()</i>	115
2.47	<i>CRC32()</i>	118
2.48	<i>ListenPacket()</i>	120
2.49	<i>ListenSend()</i>	121
2.50	<i>VarSync()</i>	123

3. 數學函式 125

3.1	<i>abs()</i>	125
3.2	<i>pow()</i>	127
3.3	<i>sqrt()</i>	129
3.4	<i>ceil()</i>	130
3.5	<i>floor()</i>	131
3.6	<i>round()</i>	132
3.7	<i>random()</i>	134
3.8	<i>d2r()</i>	136
3.9	<i>r2d()</i>	137
3.10	<i>sin()</i>	138
3.11	<i>cos()</i>	139
3.12	<i>tan()</i>	140
3.13	<i>asin()</i>	141
3.14	<i>acos()</i>	142
3.15	<i>atan()</i>	143
3.16	<i>atan2()</i>	144
3.17	<i>log()</i>	145
3.18	<i>log10()</i>	147
3.19	<i>norm2()</i>	148
3.20	<i>dist()</i>	149
3.21	<i>trans()</i>	150
3.22	<i>inversetrans()</i>	152

3.23	<i>applytrans()</i>	154
3.24	<i>interpoint()</i>	156
3.25	<i>changeref()</i>	157
3.26	<i>points2coord()</i>	159
3.27	<i>intercoord()</i>	161

4. 檔案函式 163

4.1	<i>File_ReadBytes()</i>	164
4.2	<i>File_ReadText()</i>	165
4.3	<i>File_ReadLines()</i>	166
4.4	<i>File_NextLine()</i>	168
4.5	<i>File_NextEOF()</i>	172
4.6	<i>File_WriteBytes()</i>	173
4.7	<i>File_WriteText()</i>	178
4.8	<i>File_WriteLine()</i>	181
4.9	<i>File_WriteLines()</i>	184
4.10	<i>File_Exists()</i>	187
4.11	<i>File_Length()</i>	188
4.12	<i>File_Delete()</i>	189
4.13	<i>File_Copy()</i>	190
4.14	<i>File_CopyImage()</i>	191
4.15	<i>File_Replace()</i>	193
4.16	<i>File_GetToken()</i>	194
4.17	<i>File_GetAllTokens()</i>	199

5. SERIAL PORT 函式 201

5.1	<i>com_read()</i>	201
5.2	<i>com_read_string()</i>	208
5.3	<i>com_write()</i>	214

5.4	<i>com_writeline()</i>	216
6.	SOCKET 函式.....	218
6.1	<i>socket_read()</i>	218
6.2	<i>socket_read_string()</i>	225
6.3	<i>socket_send()</i>	231
6.4	<i>socket_sendline()</i>	233
7.	參數化物件	235
7.1	<i>Point</i>	236
7.2	<i>Base</i>	237
7.3	<i>TCP</i>	238
7.4	<i>VPoint</i>	240
7.5	<i>IO</i>	241
7.6	<i>Robot</i>	244
7.7	<i>FT</i>	246
8.	外部命令	248
8.1	<i>Listen</i> 節點	248
8.2	<i>ScriptExit()</i>	249
8.3	通訊協定.....	250
8.4	<i>TMSCT</i>	252
8.5	<i>TMSTA</i>	254
8.6	<i>CPERR</i>	258
8.7	優先命令.....	260
9.	機器人運動及視覺任務函式	264
9.1	<i>QueueTag()</i>	264
9.2	<i>WaitQueueTag()</i>	265
9.3	<i>StopAndClearBuffer()</i>	267

9.4	<i>Pause()</i>	268
9.5	<i>Resume()</i>	269
9.6	<i>PTP()</i>	270
9.7	<i>Line()</i>	274
9.8	<i>Circle()</i>	276
9.9	<i>PLine()</i>	279
9.10	<i>Move_PTP()</i>	281
9.11	<i>Move_Line()</i>	283
9.12	<i>Move_PLine()</i>	285
9.13	<i>ChangeBase()</i>	287
9.14	<i>ChangeTCP()</i>	289
9.15	<i>ChangeLoad()</i>	292
9.16	<i>PVTEnter()</i>	293
9.17	<i>PVTExit()</i>	294
9.18	<i>PVTPoint()</i>	295
9.19	<i>PVTPause()</i>	297
9.20	<i>PVTResume()</i>	298
	手臂姿態定義 Config1, Config2, Config3	299
9.21	<i>Vision_DoJob()</i>	300
9.22	<i>Vision_DoJob_PTP()</i>	301
9.23	<i>Vision_DoJob_Line()</i>	302
9.24	<i>Vision_IsJobAvailable()</i>	304

10. MODBUS 函式 305

10.1	<i>modbus_read()</i>	305
10.2	<i>modbus_read_int16()</i>	308
10.3	<i>modbus_read_int32()</i>	310
10.4	<i>modbus_read_float()</i>	312
10.5	<i>modbus_read_double()</i>	314

10.6	<i>modbus_read_string()</i>	316
10.7	<i>modbus_write()</i>	318
11.	TM ETHERNET SLAVE	322
11.1	界面設定.....	322
11.2	<i>svr_read()</i>	323
11.3	<i>svr_write()</i>	324
11.4	資料表.....	325
11.5	通訊協定.....	327
11.6	<i>TMSVR</i>	329
0.	<i>Mode = 0</i> 由 Server 回應 Client 命令處理狀態.....	331
1.	<i>Mode = 1 BINARY</i>	333
2.	<i>Mode = 2 STRING</i>	335
3.	<i>Mode = 3 JSON</i>	336
11.	<i>Mode = 11 BINARY (Request read)</i>	337
12.	<i>Mode = 12 STRING (Request read)</i>	339
13.	<i>Mode = 13 JSON (Request read)</i>	340
12.	PROFINET 函式	341
12.1	<i>profinet_read_input()</i>	342
12.2	<i>profinet_read_input_int()</i>	347
12.3	<i>profinet_read_input_float()</i>	350
12.4	<i>profinet_read_input_string()</i>	353
12.5	<i>profinet_read_input_bit()</i>	354
12.6	<i>profinet_read_output()</i>	357
12.7	<i>profinet_read_output_int()</i>	362
12.8	<i>profinet_read_output_float()</i>	365
12.9	<i>profinet_read_output_string()</i>	368
12.10	<i>profinet_read_output_bit()</i>	369

12.11	<i>profinet_write_output()</i>	372
12.12	<i>profinet_write_output_bit()</i>	381
13.	ETHERNET/IP 函式	387
13.1	<i>eip_read_input()</i>	388
13.2	<i>eip_read_input_int()</i>	393
13.3	<i>eip_read_input_float()</i>	396
13.4	<i>eip_read_input_string()</i>	399
13.5	<i>eip_read_input_bit()</i>	400
13.6	<i>eip_read_output()</i>	403
13.7	<i>eip_read_output_int()</i>	408
13.8	<i>eip_read_output_float()</i>	411
13.9	<i>eip_read_output_string()</i>	414
13.10	<i>eip_read_output_bit()</i>	415
13.11	<i>eip_write_output()</i>	418
13.12	<i>eip_write_output_bit()</i>	427

修訂表

Revision	Date	Revised Content
00	Dec, 2020	<p>除了先前軟體版本，於此版本：</p> <ul style="list-style-type: none"> ● Add option 2 and 3 for GetToken() and GetAllTokens() ● Add option 2 and 3 for File_GetToken() and File_GetAllTokens() ● Add option 2 and 3 for com_read() and com_read_string() ● Add socket chapter ● Add ScriptExit(0) and ScriptExit(1) ● Add StopAndClearBuffer(0), StopAndClearBuffer(1), and StopAndClearBuffer(2) ● 於 Profinet chapter <ul style="list-style-type: none"> ➤ [新增] 函式多型 (可用 Item Name (string) 進行存取) ➤ [新增] read_output_bit()、read_input_bit() 可以處理陣列資料 ➤ [新增] write_output_bit() 可以處理陣列資料 ● Add EtherNet/IP chapter ● Add Vision Do Job function in Robot Motion chapter ● Add File_CopyImage() ● Add points2coord() and intercoord() ● Add GetVarValue() <p>其他：</p> <ul style="list-style-type: none"> ● Modify WaitQueueTag tag number argument, value = 0 is only check timeout ● Modify WaitQueueTag timeout argument, value < 0 is infinite, and 0 is check one, and > 0 is timeout ● Remove socket_send to IP:Port in Motion functions ● 修改 GetToken(), GetAllTokens(), File_GetToken(), File_GetAllTokens(), com_read(), com_read_string() 對應的範例說明 ● 調整 com_write(), com_writeline() 的語法說明 ● PVTPoint 時間單位由 ms 改為 second ● 將全形的雙引號("")修改為英文字母的半形雙引號('")

1. 運算式

1.1 型別

在 變數 Manager 內，可建立變數型別

byte	8 位元整數值	無正負號	0 至 255	有效位數 3
int	32 位元整數值	正負號	-2147483648 至 2147483647	有效位數 10
float	32 位元浮點數值	正負號	-3.40282e+038f 至 3.40282e+038f	有效位數 7
double	64 位元浮點數值	正負號	-1.79769e+308 至 1.79769e+308	有效位數 15
bool	布林值		true 或 false	
string	字串值			

其中 int 型別下，在 Functions 的使用上，可依需求再細分為 int16 及 int32，預設為 int32

int16	16 位元整數值	正負號	-32768 至 32767	有效位數 5
int32	32 位元整數值	正負號	-2147483648 至 2147483647	有效位數 10

1.2 變數與常數

1. 變數

變數名稱命名方式，僅可使用數字或英文字元組合，不可使用特殊字元或其他字元

數字 0123456789

字元 a-z、A-Z、_

如

```
int i = 0
```

```
string s = "ABC"
```

```
string s1 = "DEF"
```

```
string s2 = "123"
```

變數名稱使用時，不需使用雙引號，如

```
s = s1 + " and " + s2 // s = "DEF and 123"
```

// s、s1、s2 都是變數使用，而 "and" 是字串常數

語法除變數名稱使用外，還有常數 (Constant) 的使用，如數值 (Number)、字串 (String) 與布林值 (Boolean)，而其中只有字串常數需要使用雙引號括起來。

變數在 TMflow 中生成時會根據來源增加前綴詞，在調用變數做寫入或讀取時，必須輸入包含前綴字的完整名稱，例如 var_s1 或 g_s2，前綴詞增加規則請參閱該變數相對應的設定頁面說明。

2. 數值

- 支援十進制表示、十進制浮點數、二進制表示、十六進制表示，及科學標記

十進制表示 123

-123

+456

十進制浮點數 34.567

-8.9

二進制表示 0b0000111

OB1110000

十六進制表示 0x123abc

0X00456DEF

科學標記 3.4e5

2.3E-4

- 二進制表示法及十六進制表示法，預設皆為無浮點數表示
- 當為數值表示時，英文大小書寫方式不會影響數值，如
 - 0b0011 等於 0B0011
 - 0xabCD 等於 0XABCD, 0xABCD, 0Xabcd, ... 等等
 - 3.4e5 等於 3.4E5
- 浮點數與 byte 陣列互相轉換時，有時會因浮點精度問題，而有部分數值誤差，如
 - float 5.317302E+030 → float 轉 byte[] {0x72,0x86,0x3A,**0x42**}
 - byte[] {0x72,0x86,0x3A,**0x43**} → byte[] 轉 float 5.317302E+030
- byte 型別僅能表示無正負號數值，0 至 255，因此若對 byte 型別賦值負數，或是做負號運算，仍只保存 8 位元無正負號整數值，如


```
byte b = -100 // b = 156 // -100 若用 16 進制表示為 0xFFFFF9C  
// 因 byte 只能保存 8 位元，即 0x9C (156)，所以 b 內數值為 156
```

3. 字串

字串常數需要使用雙引號括起來，以避免與變數名稱衝突。

雙引號 "Hello World!"

"Hello TM""5" (如果字串需要 " 符號，必須再加 " 符號，兩個雙引號需連著)

- 不支援在雙引號括起來內輸入控制字元，如

"Hello World!\r\n" (會輸出 Hello World!\r\n 字串)

- 當未使用雙引號時，會依下列規則進行

1. 若是數字，則識別為數字
2. 若是數字或字元組合，則優先判斷是否為變數，若變數存在，則識別為變數。
3. 若變數不存在，則識別為字串，並產生警告訊息。

- 變數與字串結合使用

1. 在雙引號字串內，並不會將變數名稱值提取出來，如

s = "TM5" // s = "TM5"

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

2. 標準寫法，若有需要將變數名稱提取出來，得將變數名稱寫在雙引號外面，並使用 + 加號連接，如

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

3. 相容性寫法(不建議使用)，可使用單引號將變數名稱括起來，但會產生警告訊息，如

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

"Hi, 'x' Robot" // s1 = "Hi, 'x' Robot" // 若沒有 x 變數存在，則保持字串內容

4. 單引號 '變數名稱' 內不支援使用陣列索引來取出陣列元素值，需使用標準寫法

```
string[] ss = {"Techman", "Robot"}  
"Hi, 's' 'ss[0]' Robot"           // s1 = "Hi, TM5 'ss[0]' Robot"      // 不支援 'ss[0]' 方式  
"Hi, " + s + " " + ss[0] + " Robot"    // s1 = "Hi, TM5 Techman Robot"
```

5. 單引號不具有 "" 規則，若想打出'變數名稱'，需使用標準寫法，如

```
"Hi, 's' Robot"                  // s1 = "Hi, TM5 Robot"  
// 若想要打出 s1 = "Hi, 's' Robot"，修改如下  
"Hi, " + "s" + "" Robot"       // s1 = "Hi, 's' Robot"
```

- 若需要輸入控制字元(如換行符號)，可配合 Ctrl() 函式，如

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

Hi,

TM5 Robot

- 字串保留字，使用方式類似字串變數，因此不需要使用雙引號括起來。(但不支援單引號寫法)

1. empty 表示空字串，等於 ""
2. newline 或 NewLine 表示換行符號，等於 Ctrl("\r\n") 或 Ctrl(0x0D0A)

4. 布林

布林值為邏輯真假值，可表示為真或為假

真 true

True

假 false

False

書寫方式大小寫有差異，若是非法的真假值表示時，如 TRue，有可能會轉為變數或是字串。

1.3 陣列

- 陣列是同一種型別的資料集合，陣列初始值使用 {} 括起來，各陣列內元素值維持型別特性，並採用 , 分隔，如

```
int[] i = {0,1,2,3}           // 元素保持數值格式  
string[] s = {"ABC", "DEF", "GHI"} // 元素保持字串格式  
bool[] bb = {true, false, true}    // 元素保持布林值格式
```

- 可使用陣列索引(index) 來取得陣列內元素值，索引從 0 開始計數，如

index	0	1	2	3	4	5	6	7
array		共 8 個元素						
	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]

合法可存取範圍為 [0] .. [7]，超過範圍的存取，將會產生錯誤，如 A[8]。

- 只支援一個維度陣列，最大容量為 2048
- 配合 Functions 函式或 Network 節點接收使用時，陣列大小會動態調整，但最大容量維持 2048，超過的部分將會遺失，如

```
string[] ss = {empty, empty, empty}      // 字串陣列初始大小為 3 個元素  
ss = String_Split("A_B_C_D_F_G_H", "_") // 字串切割後，可得到陣列為 7 個元素  
len = Length(ss)                      // len = 7  
ss = String_Split("A,B", ",")          // 字串切割後，可得到陣列為 2 個元素  
len = Length(ss)                      // len = 2
```

1.4 運算子

- 支援下表運算子進行運算或邏輯判斷
- 運算會依運算子(operator) 優先權由大至小處理，當優先權相同時，則依結合性順序處理，如

左結合 left-to-right

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

Diagram illustrating the left-to-right binding of operators in the expression $A = A * B / C \% D$. The expression is grouped into four levels:

- Level 1: $A * B$
- Level 2: $/ C$
- Level 3: $\% D$
- Level 4: The entire expression $A = ((A * B) / C \% D)$

右結合 right-to-left

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

Diagram illustrating the right-to-left binding of operators in the expression $A -= B += 10 \&& !D$. The expression is grouped into five levels:

- Level 1: $!D$
- Level 2: $B +=$
- Level 3: 10
- Level 4: $\&&$
- Level 5: The entire expression $A -= ((B += (10 \&& (!D))))$

- 運算會依運算元(operand) 的型別進行運算

1. 當兩個數值都是整數型別時，會依整數型別進行運算，如

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

依運算子優先順序會先做 / 再做 =

$var_a / var_b = 10 / 3 = 3$ (var_a 與 var_b 都是整數)
 $var_c = 3$ (將整數 3 賦值給 var_c 浮點數)

2. 當兩個數值有一個是浮點數型別時，會依浮點數型別進行運算，如

```
int var_a = 10  
float var_b = 3  
float var_c = var_a / var_b  
 $var\_a / var\_b = 10 / 3 = 3.333333$  (因為 var_b 是浮點數)  
 $var\_c = 3.333333$  (將浮點數 3.333333 賦值給 var_c 浮點數)  
  
 $var\_c = var\_a / 3$   
 $var\_a / 3 = 10 / 3 = 3$  (var_a 與 3 都是整數)  
 $var\_c = 3$   
  
 $var\_c = var\_a / 3.0$   
 $var\_a / 3.0 = 10 / 3.0 = 3.333333$  (因為 3.0 是浮點數)  
 $var\_c = 3.333333$ 
```

優先權	運算子	敘述	範例	使用需求	結合	
17	++	字尾遞增	i++	整數變數	由左至右	
	--	字尾遞減	i--			
	()	函式呼叫	int x = f()			
	[]	陣列存取	array[4] = 2	陣列變數		
16	++	字首遞增	++i	整數變數	由右至左	
	--	字首遞減	--i			
	+	一元正號	int i = +1	數值變數常數		
	-	一元負號	int i = -1			
	!	邏輯非	if (!done) ...	布林變數常數		
	~	按位取反	flag1 = ~flag2	整數變數常數		
	(type)	型別轉換	int i = (int)floatNum			
14	*	乘法	int i = 2 * 4	數值變數常數	由左至右	
	/	除法	float f = 10.0 / 3.0			
	%	模數(取餘)	int rem = 4 % 3			
13	+	加法	int i = 2 + 3	數值變數常數		
	-	減法	int i = 5 - 1			
12	<<	位元左移	int flags = 33 << 1	整數變數常數		
	>>	位元右移	int flags = 33 >> 1			
11	<	小於關係	if (i < 42) ...	數值變數常數	由左至右	
	<=	小於等於關係	if (i <= 42) ...			
	>	大於關係	if (i > 42) ...			
	>=	大於等於關係	if (i >= 42) ...			
10	==	等於關係	if (i == 42) ...			
	!=	不等於關係	if (i != 42) ...			
9	&	位元 AND	flag1 = flag2 & 42	整數變數常數	由左至右	
8	^	位元 XOR	flag1 = flag2 ^ 42			
7		位元 OR	flag1 = flag2 42			
6	&&	邏輯 AND	if (conditionA && conditionB)			
5		邏輯 OR	if (conditionA conditionB)			
4	c ? t : f	三元條件運算	int i = a > b ? a : b			
3	=	直接賦值	int a = b	左側數值變數 右側數值變數常數		
	+=	以和賦值	a += 3			
	-=	以差賦值	b -= 4			
	*=	以乘賦值	a *= 5			
	/=	以除賦值	a /= 2			
	%=	以取餘數賦值	a %= 3		左側整數變數 右側整數變數常數	
	<=>	以位元左移賦值	flags <=> 2			
	>=>	以位元右移賦值	flags >=> 2			
	&=	以位元 AND 賦值	flags &= new_flags			
	^=	以位元 XOR 賦值	flags ^= new_flags			
	=	以位元 OR 賦值	flags = new_flags			

1.5 型別轉換

- 資料型別間可互相轉換，可使用在變數/常數、或陣列型別
- 必須是同格式轉換，如變數/常數轉換、或陣列轉換。不允許變數轉換為陣列，或陣列轉換為變數

原型別	轉換	範例	結果
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 = true (非空值)
	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 (非空值)
	string	string s = (string)1000	s = "1000"
float	byte	byte b = (byte)1.23	b = 1
	int	int i = (int)1.23	i = 1
	double	double d = (double)1.23	d = 1.23
	bool	bool flag = (bool)1.23	flag = true
	string	string s = (string)1.23	s = "1.23"
double	byte	byte b = (byte)1.23	b = 1
	int	int i = (int)1.23	i = 1
	float	float f = (float)1.23	f = 1.23
	bool	bool flag = (bool)1.23	flag = true
	string	string s = (string)1.23	s = "1.23"
bool	byte	byte b = (byte)True	Error
	int	int i = (int)False	Error
	float	float f = (float)true	Error
	double	double d = (double)false	Error
	string	string s = (string)True	s = "True"
string	byte	byte b1 = (byte)"1.23" byte b2 = (byte)"XYZ"	1 Error
	int	int i = (int)"1.23"	1
	float	float f1 = (float)"1.23" float f2 = (float)"XYZ"	1.23 Error
	double	double d = (double)"1.23"	1.23
	bool	bool flag1 = (bool)"1.23" bool flag2 = (bool)""	flag1 = true (非空值) flag2 = false (空值)

- 陣列的轉換方式也依照上表，會針對陣列內的每個元素都進行型別轉換

```
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}
```

- 當轉換出現下列時，將會產生錯誤訊息

- 轉換至數值型別時，無法正確轉換為數值，如布林值(true/false)或非數字的字串值(如, "XYZ")

```
int value = (int)true        // Error  
  
int value = (int)"XYZ"      // Error
```

- 轉換至浮點數值型別時，為無效的浮點數值，如 NaN 或 Infinity

```
string dvalue = "1.79769e+308"  
  
float f = (float)dvalue      // Error 1.79769e+308 為 double 型別的有效值，無法轉入 float 型別
```

1.6 警告訊息

當運算式語法出現下列時，將會產生警告訊息

- 字串常數未使用雙引號括起來

```
string s = Hello          // warning Hello
```

- 在字串常數內有使用單引號變數

```
string s0 = "World"
```

```
string s1 = "Hello 's0'"    // warning 's0'
```

- 浮點數值賦值給整數型變數時，可能會有數值遺失，如

```
int i = 1.234      // warning i = 1
float f = 1.234
i = f            // warning i = 1
```

- 數值型別，較大位元型別賦值給較小位元型別時，可能會有數值遺失，如

```
byte b = 100
int i = 1000
float f = 1.234
double d = 2.345
b = i           // warning b = 232    // 因 byte 只能表示 0 至 255
f = d           // warning f = 2.345
```

- 字串值賦值給數值型變數時，會自動嘗試將字串值轉換為數值，若可以轉換為數值，則僅產生警告訊息，但若無法轉換為數值，將會報錯，如

```
int i = "1234"        // warning i = 1234
int j = "0x89AB"       // warning j = 35243
int k = "0b1010"       // warning k = 10
string s1 = 123         // warning s1 = 123 // 數值型轉入字串型
string s2 = "123"
int x = s2              // warning // 字串型轉入數值型
// 以下在語法檢查時會出現警告，但在專案運行時會報錯
s2 = "XYZ"
x = s2                  // warning // 運行時報錯 // 因 s = "XYZ" 無法正確轉換為數值
s2 = ""
x = s2                  // warning // 運行時報錯 // 因 s = "" 無法正確轉換為數值
```

- 函式使用時(function)，有數值型參數帶入字串參數，如

```
Ctrl(0xA0B0C0D0E)      // warning // 因 0xA0B0C0D0E 不是 int 型別(超過 32bit)  
                           // 但因另有 Ctrl(string) 語法，將會改以 Ctrl(string) 處理
```

警告訊息雖不影響專案運行，但建議必須修正所有警告訊息，以避免產生不確定的結果，導致在專案運行時，因報錯而停止，或是走非預期的路線。

- 如何修正警告訊息

1. 字串常數使用雙引號

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

2. 字串常數與字串變數使用時，使用 + 加號連接

```
string s0 = "World"
```

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

3. 數值型別轉換時，給予明確的型別轉換

```
float f = 1.234
```

```
int i = (int)f           // 使用 (int) 型別轉換，運行時 i=1 // 會將 float 浮點數轉為 int 整數
```

2. 函式

2.1 Byte.ToInt16()

將 byte 陣列，前 2 bytes 轉成整數值，傳回 int 型別

語法 1

```
int ByteToInt16(  
    byte[],  
    int,  
    int  
)
```

參數

byte[] 輸入的 byte 陣列

int 輸入的 byte 陣列是依照 Little Endian 讀取或是依照 Big Endian 讀取

0 Little Endian (預設)

1 Big Endian

int 轉換為 signed int16 (有符號數) 或 unsigned int16 (無符號數)

0 signed int16 (預設)

1 unsigned int16

傳回值

int 將 byte 陣列前 2 bytes 長度，轉換成整數值傳回。

因需 2 bytes 長度，只會取 byte 陣列中的前 [0][1] (不足部分會先補齊) 進行轉換。

說明

```
byte[] bb1 = {0x90, 0x01, 0x05}
```

```
byte[] bb2 = {0x01} // 因 bb2[] 不足 2 bytes 取出時，會先補齊 2 bytes 再進行轉換
```

```
value = ByteToInt16(bb1, 0, 0) // 0x0190 value = 400  
value = ByteToInt16(bb1, 0, 1) // 0x0190 value = 400  
value = ByteToInt16(bb1, 1, 0) // 0x9001 value = -28671  
value = ByteToInt16(bb1, 1, 1) // 0x9001 value = 36865  
value = ByteToInt16(bb2, 0, 0) // 0x0001 value = 1  
value = ByteToInt16(bb2, 0, 1) // 0x0001 value = 1  
value = ByteToInt16(bb2, 1, 0) // 0x0100 value = 256  
value = ByteToInt16(bb2, 1, 1) // 0x0100 value = 256
```

語法 2

```
int ByteToInt16(  
    byte[],  
    int  
)
```

說明

與語法 1 相同，預設將參數 int 填入 0 以 signed int16 轉換

ByteToInt16(bb1, 0) => ByteToInt16(bb1, 0, 0)

語法 3

```
int ByteToInt16(  
    byte[]  
)
```

說明

與語法 1 相同，預設將參數 int 填入 0 以 Little Endian 讀取陣列及以 signed int16 轉換

ByteToInt16(bb1) => ByteToInt16(bb1, 0)

2.2 Byte.ToInt32()

將 byte 陣列，前 4 bytes 轉成整數值，傳回 int 型別

語法 1

```
int Byte.ToInt32(  
    byte[],  
    int  
)
```

參數

byte[] 輸入的 byte 陣列

int 輸入的 byte 陣列是依照 Little Endian 讀取或是依照 Big Endian 讀取

0 Little Endian (預設)

1 Big Endian

傳回值

int 將 byte 陣列前 4 bytes 長度，轉換成整數值傳回。

因需 4 bytes 長度，只會取 byte 陣列中的前 [0][1][2][3] (不足部分會先補齊) 進行轉換。

說明

```
byte[] bb1 = {0x01, 0x02, 0x03, 0x4F, 1}
```

```
byte[] bb2 = {0x01, 0x02, 0x03} // 因 bb2[] 不足 4 bytes 取出時，會先補齊 4 bytes 再進行轉換
```

```
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
```

語法 2

```
int Byte.ToInt32(  
    byte[]  
)
```

說明

與語法 1 相同，預設將參數 int 填入 0 以 Little Endian 讀取陣列。

`Byte.ToInt32(bb1) => Byte.ToInt32(bb1, 0)`

2.3 Byte_ToFloat()

將 byte 陣列，前 4 bytes 轉成浮點數值，傳回 float 型別

語法 1

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

參數

byte[] 輸入的 byte 陣列

int 輸入的 byte 陣列是依照 Little Endian 讀取或是依照 Big Endian 讀取

0 Little Endian (預設)

1 Big Endian

傳回值

float 將 byte 陣列前 4 bytes 長度，轉換成浮點數值傳回。

因需 4 bytes 長度，只會取 byte 陣列中的前 [0][1][2][3] (不足部分會先補齊) 進行轉換。

說明

```
byte[] bb1 = {0x01, 0x02, 0x03, 0x4F, 1}
```

```
byte[] bb2 = {0x01, 0x02, 0x03} // 因 bb2[] 不足 4 bytes 取出時，會先補齊 4 bytes 再進行轉換
```

```
value = Byte_ToFloat(bb1, 0) // 0x4F030201 value = 2.197947E+09
```

```
value = Byte_ToFloat(bb1, 1) // 0x0102034F value = 2.38796E-38
```

```
value = Byte_ToFloat(bb2, 0) // 0x00030201 value = 2.762254E-40
```

```
value = Byte_ToFloat(bb2, 1) // 0x01020300 value = 2.387938E-38
```

語法 2

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

說明

與語法 1 相同，預設將參數 int 填入 0 以 Little Endian 讀取陣列。

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

2.4 Byte_ToDouble()

將 byte 陣列，前 8 bytes 轉成浮點數值，傳回 double 型別

語法 1

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

參數

byte[] 輸入的 byte 陣列

int 輸入的 byte 陣列是依照 Little Endian 讀取或是依照 Big Endian 讀取

0 Little Endian (預設)

1 Big Endian

傳回值

double 將 byte 陣列前 8 bytes 長度，轉換成浮點數值傳回。

因需 8 bytes 長度，只會取 byte 陣列中的前 [0][1][2][3][4][5][6][7] (不足部分會先補齊) 進行轉換。

說明

```
byte[] bb1 = {0x01, 0x02, 0x03, 0x4F, 1} // 因 bb1[] 不足 8 bytes 取出時，會先補齊 8 bytes 再進行轉換  
byte[] bb2 = {0x01, 0x02, 0x03} // 因 bb2[] 不足 8 bytes 取出時，會先補齊 8 bytes 再進行轉換
```

```
value = Byte_ToDouble(bb1, 0) // 0x000000014F030201 value = 2.77692782029764E-314
```

```
value = Byte_ToDouble(bb1, 1) // 0x0102034F01000000 value = 8.20840179153173E-304
```

```
value = Byte_ToDouble(bb2, 0) // 0x00000000000030201 value = 9.73907141738724E-319
```

```
value = Byte_ToDouble(bb2, 1) // 0x0102030000000000 value = 8.20785244926136E-304
```

語法 2

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

說明

與語法 1 相同，預設將參數 int 填入 0 以 Little Endian 讀取陣列。

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

2.5 Byte.ToInt16Array()

將 byte 陣列，每 2 bytes 轉成整數值，傳回 int[] 型別

語法 1

```
int[] Byte.ToInt16Array(  
    byte[],  
    int,  
    int  
)
```

參數

byte[] 輸入的 byte 陣列

int 輸入的 byte 陣列是依照 Little Endian 讀取或是依照 Big Endian 讀取

0 Little Endian (預設)

1 Big Endian

int 轉換為 signed int16 (有符號數) 或 unsigned int16 (無符號數)

0 signed int16 (預設)

1 unsigned int16

傳回值

int[] 將 byte 陣列依每 2 bytes 長度，轉換成整數值陣列傳回。

說明

```
byte[] bb1 = {0x90, 0x01, 0x02, 0x03, 0x04} // 不足 2 bytes 取出時，會先補齊 2 bytes 再進行轉換
```

```
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}
```

語法 2

```
int[] ByteToInt16Array(  
    byte[],  
    int  
)
```

說明

與語法 1 相同，預設將參數 int 填入 0 以 signed int16 轉換

Byte.ToInt16Array(bb1, 0) => Byte.ToInt16Array(bb1, 0, 0)

語法 3

```
int[] ByteToInt16Array(  
    byte[]  
)
```

說明

與語法 1 相同，預設將參數 int 填入 0 以 Little Endian 讀取陣列及以 signed int16 轉換

Byte.ToInt16Array(bb1) => Byte.ToInt16Array(bb1, 0)

2.6 Byte.ToInt32Array()

將 byte 陣列，每 4 bytes 轉成整數值，傳回 int[] 型別

語法 1

```
int[] Byte.ToInt32Array(  
    byte[],  
    int  
)
```

參數

byte[] 輸入的 byte 陣列

int 輸入的 byte 陣列是依照 Little Endian 讀取或是依照 Big Endian 讀取

0 Little Endian (預設)

1 Big Endian

傳回值

int[] 將 byte 陣列依每 4 bytes 長度，轉換成整數值陣列傳回。

說明

```
byte[] bb1 = {0x01, 0x02, 0x03, 0x04, 0x05} // 不足 4 bytes 取出時，會先補齊 4 bytes 再進行轉換
```

```
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}
```

語法 2

```
int[] Byte.ToInt32Array(  
    byte[]  
)
```

說明

與語法 1 相同，預設將參數 int 填入 0 以 Little Endian 讀取陣列。

`Byte.ToInt32Array(bb1) => Byte.ToInt32Array(bb1, 0)`

2.7 Byte_ToFloatArray()

將 byte 陣列，每 4 bytes 轉成浮點數值，傳回 float[] 型別

語法 1

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

參數

byte[] 輸入的 byte 陣列

int 輸入的 byte 陣列是依照 Little Endian 讀取或是依照 Big Endian 讀取

0 Little Endian (預設)

1 Big Endian

傳回值

float[] 將 byte 陣列依每 4 bytes 長度，轉換成浮點數值陣列傳回。

說明

```
byte[] bb1 = {0x01, 0x02, 0x03, 0x04, 0x05} // 不足 4 bytes 取出時，會先補齊 4 bytes 再進行轉換
```

```
byte[] bb2 = {1, 2, 3, 4}
```

```
value = Byte_ToFloatArray(bb1, 0) // {0x04030201, 0x00000005} value = {1.53999E-36, 7.006492E-45}
```

```
value = Byte_ToFloatArray(bb1, 1) // {0x01020304, 0x05000000} value = {2.387939E-38, 6.018531E-36}
```

```
value = Byte_ToFloatArray(bb2, 0) // {0x04030201} value = {1.53999E-36}
```

```
value = Byte_ToFloatArray(bb2, 1) // {0x01020304} value = {2.387939E-38}
```

語法 2

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

說明

與語法 1 相同，預設將參數 int 填入 0 以 Little Endian 讀取陣列。

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

2.8 Byte_ToDoubleArray()

將 byte 陣列，每 8 bytes 轉成浮點數值，傳回 double[] 型別

語法 1

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

參數

byte[] 輸入的 byte 陣列

int 輸入的 byte 陣列是依照 Little Endian 讀取或是依照 Big Endian 讀取

0 Little Endian (預設)

1 Big Endian

傳回值

double[] 將 byte 陣列依每 8 bytes 長度，轉換成浮點數值陣列傳回。

說明

byte[] bb1 = {0x01, 0x02, 0x03, 0x04, 0x05} // 不足 8 bytes 取出時，會先補齊 8 bytes 再進行轉換

byte[] bb2 = {1, 2, 3, 4} // 不足 8 bytes 取出時，會先補齊 8 bytes 再進行轉換

```
value = Byte_ToDoubleArray(bb1, 0) // {0x0000000504030201} value = {1.06432325297744E-313}
```

```
value = Byte_ToDoubleArray(bb1, 1) // {0x0102030405000000} value = {8.20788039849233E-304}
```

```
value = Byte_ToDoubleArray(bb2, 0) // {0x00000000004030201} value = {3.32535749480063E-316}
```

```
value = Byte_ToDoubleArray(bb2, 1) // {0x0102030400000000} value = {8.2078802626846E-304}
```

語法 2

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

說明

與語法 1 相同，預設將參數 int 填入 0 以 Little Endian 讀取陣列。

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

2.9 Byte_ToString()

將 byte 陣列，轉成字串值

語法 1

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

參數

byte[] 輸入的 byte 陣列
int 輸入的 byte 陣列轉換成字串文字編碼方式
0 UTF8 (預設) (0x00 END)
1 HEX BINARY
2 ASCII (0x00 END)

傳回值

string 將 byte 陣列轉換成字串值傳回。轉換依序從 [0][1][2] ... 進行轉換。

說明

```
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 遇到 0x00 結束)  
value = Byte_ToString(bb1, 1) // value = "313233004F01"  
value = Byte_ToString(bb1, 2) // value = "123" (ASCII 遇到 0x00 結束)  
value = Byte_ToString(bb2, 0) // value = "\u01TM5機器人" (UTF8 編碼)  
value = Byte_ToString(bb2, 1) // value = "01544D35E6A99FE599A8E4BABA"  
value = Byte_ToString(bb2, 2) // value = "\u01TM5?????????" (ASCII 編碼)
```

* \u01 為書寫 SOH 控制碼表示，並非字串值

語法 2

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

說明

與語法 1 相同，預設將參數 int 填入 0 以 UTF8 轉換陣列。

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

2.10 Byte_Concat()

合併連接兩個 byte 陣列，或一個 byte 陣列合併連接一個 byte 值，或任意 byte 型別值

語法 1

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

參數

byte[] 輸入的 byte 陣列
byte 輸入的 byte 值合併連接至 byte 陣列之後

傳回值

byte[] 將 byte 陣列參數合併連接後，傳回一個新的 byte 陣列。

說明

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

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

語法 2

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

參數

byte[] 輸入的 byte 陣列1
byte[] 輸入的 byte 陣列2，會合併連接至 byte 陣列1 之後

傳回值

byte[] 將 byte 陣列參數合併連接後，傳回一個新的 byte 陣列。

說明

```
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}
```

語法 3

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

參數

byte[] 輸入的 byte 陣列1
byte[] 輸入的 byte 陣列2，會合併連接至 byte 陣列1 之後
int 陣列2 要合併連接的個數
0..陣列2長度 合法值
<0 非法值，個數會設為陣列2長度
>陣列2長度 非法值，個數會設為陣列2長度

傳回值

byte[] 將 byte 陣列參數合併連接後，傳回一個新的 byte 陣列。

說明

```
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} // 只連接 2 個  
value = Byte_Concat(bb1, bb2, -1) // value = {0x31, 0x32, 0x33, 0x00, 0x4F, 0x01, 0x01, 0x02, 0x03} // -1 非法值  
value = Byte_Concat(bb1, bb2, 10) // value = {0x31, 0x32, 0x33, 0x00, 0x4F, 0x01, 0x01, 0x02, 0x03} // 10 超出長度  
  
// 使用 Length() 可以取得陣列大小  
value = Byte_Concat(bb1, bb2, Length(bb2)) // value = {0x31, 0x32, 0x33, 0x00, 0x4F, 0x01, 0x01, 0x02, 0x03}
```

語法 4

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

參數

byte[] 輸入的 byte 陣列1
int 陣列1 的起始索引
 0..陣列1長度-1 合法值
 <0 起始索引將設為起始0
 >=陣列1長度 起始索引將設為陣列1長度 (因超過陣列1長度，會取到空值)
int 陣列1 的合併連接個數
 0..陣列1長度 合法值
 <0 非法值，個數會設為陣列1長度
 >陣列1長度 非法值，個數會設為陣列1長度
 如果起始加個數超出陣列1長度，則只會從起始取到陣列1結束
byte[] 輸入的 byte 陣列2，會合併連接至 byte 陣列1 之後
int 陣列2 的起始索引
 0..陣列2長度-1 合法值
 <0 起始索引將設為起始0
 >=陣列2長度 起始索引將設為陣列2長度 (因超過陣列2長度，會取到空值)
int 陣列2 的合併連接個數
 0..陣列2長度 合法值
 <0 非法值，個數會設為陣列2長度
 >陣列2長度 非法值，個數會設為陣列2長度
 如果起始加個數超出陣列2長度，則只會從起始取到陣列2結束

傳回值

byte[] 將 byte 陣列參數合併連接後，傳回一個新的 byte 陣列。

說明

```
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}
```

語法 5

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

參數 (動態參數個數)

byte 輸入的 byte 值

byte[] 輸入的 byte 陣列

將依序合併每個參數內容，若參數不為 byte 或 byte 陣列，則忽略該參數，並繼續合併下個參數

傳回值

byte[] 將 byte 參數合併連接後，傳回一個新的 byte 陣列。

說明

```
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, 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}
```

// 參數 "AB" 為 string 型別，忽略

```
value = Byte_Concat(bb1, "AB", bb2, 1000)       // value = {0x31, 0x32, 0x00, 0x4F, 0x01, 0x01, 0x02, 0x03}
```

// 參數 "AB" 為 string 型別，忽略

// 參數 1000 為 int 型別，忽略

2.11 String_ToInteger()

將字串值轉成整數值 (int 型別)

語法 1

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

參數

string 輸入的字串值

int 輸入的字串值是 10 進制、16 進制或 2 進制格式 (優先判斷)

10 10進制 或 依字串數值格式自動判斷 (預設)

16 16進制

2 2進制

字串數值格式

123 表示10進制

0x7F 表示16進制

0b101 表示2進制

傳回值

int 將字串轉換成整數值傳回。若數值格式錯誤，會傳回 0。

說明

```
value = String_ToInteger("1234", 10)      // value = 1234  
value = String_ToInteger("1234", 16)      // value = 4660  
value = String_ToInteger("1234", 2)       // value = 0      // 不是合法的2進制格式  
value = String_ToInteger("1100", 2)        // value = 12  
value = String_ToInteger("0x1234", 10)     // value = 4660  // 自動數值格式判斷為16進制  
value = String_ToInteger("0x1234", 16)     // value = 4660  
value = String_ToInteger("0x1234", 2)       // value = 0      // 不是合法的2進制格式  
value = String_ToInteger("0b1100", 10)      // value = 12      // 自動數值格式判斷為2進制  
value = String_ToInteger("0b1100", 16)      // value = 725248 // 優先判斷16進制合法  
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      // 非法格式  
value = String_ToInteger("-0b1100", 2)        // value = 0      // 非法格式
```

語法 2

```
int String_ToInteger(  
    string  
)
```

說明

與語法 1 相同，預設將參數 int 填入 10 以 10 進制 或 依字串數值格式自動判斷。

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

語法 3

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

參數

`string[]` 輸入的字串陣列
`int` 輸入的字串陣列內的值是 10 進制、16 進制或 2 進制格式 (優先判斷)
 10 10進制 或 依字串數值格式自動判斷 (預設)
 16 16進制
 2 2進制
 字串數值格式
 123 表示10進制
 0x7F 表示16進制
 0b101 表示2進制
 * 陣列內元素的進制格式必須一致

傳回值

`int[]` 將字串陣列轉換成整數陣列傳回。若數值格式錯誤，會傳回 0。

說明

```
ss = {"12", "ab", "cc", "dd", "10"}  
value = String_ToInteger(ss)           // value = {12, 0, 0, 0, 10} // "ab","cc","dd" 不是合法的10進制  
value = String_ToInteger(ss, 2)        // value = {0, 0, 0, 0, 2} // "12","ab","cc","dd" 不是合法的2進制  
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" 不是合法的10進制
```

2.12 String_ToFloat()

將字串值轉成浮點數值 (float 型別)

語法 1

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

參數

string 輸入的字串值

int 輸入的字串值是 10 進制、16 進制或 2 進制格式 (優先判斷)

10 10進制 或 依字串數值格式自動判斷 (預設)

16 16進制

2 2進制

字串數值格式

123 表示10進制

0x7F 表示16進制

0b101 表示2進制

傳回值

float 將字串轉換成浮點數值傳回。若數值格式錯誤，會傳回 0。

說明

```
value = String_ToFloat("12.34", 10)      // value = 12.34  
value = String_ToFloat("12.34", 16)      // value = 0      // 非法的16進制格式  
value = String_ToFloat("12.34", 2)       // value = 0      // 非法的2進制格式  
value = String_ToFloat("11.00", 2)        // value = 0      // 非法的2進制格式  
value = String_ToFloat("0x1234", 10)      // value = 6.530051E-42    // 自動數值格式判斷為16進制  
value = String_ToFloat("0x1234", 16)      // value = 6.530051E-42  
value = String_ToFloat("0x1234", 2)       // value = 0      // 非法的2進制格式  
value = String_ToFloat("0b1100", 10)      // value = 1.681558E-44    // 自動數值格式判斷為2進制  
value = String_ToFloat("0b1100", 16)      // value = 1.016289E-39    // 合法16進制  
value = String_ToFloat("0b1100", 2)       // value = 1.681558E-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      // 非法格式  
value = String_ToFloat("-0b1100", 2)       // value = 0      // 非法格式
```

語法 2

```
float String_ToFloat(  
    string  
)
```

說明

與語法 1 相同，預設將參數 int 填入 10 以 10 進制 或 依字串數值格式自動判斷。

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

語法 3

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

參數

`string[]` 輸入的字串陣列
`int` 輸入的字串陣列內的值是 10 進制、16 進制或 2 進制格式 (優先判斷)
 10 10進制 或 依字串數值格式自動判斷 (預設)
 16 16進制
 2 2進制
 字串數值格式
 123 表示10進制
 0x7F 表示16進制
 0b101 表示2進制
 * 陣列內元素的進制格式必須一致

傳回值

`float[]` 將字串陣列轉換成浮點數陣列傳回。若數值格式錯誤，會傳回 0。

說明

```
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, 2.39622E-43, 2.858649E-43, 3.09687E-43, 0}  
value = String_ToFloat(ss, 10)              // value = {12.345, 0, 0, 0, 10.111}
```

2.13 String_ToDouble()

將字串值轉成浮點數值 (double 型別)

語法 1

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

參數

string 輸入的字串值

int 輸入的字串值是 10 進制、16 進制或 2 進制格式 (優先判斷)

10 10進制 或 依字串數值格式自動判斷 (預設)

16 16進制

2 2進制

字串數值格式

123 表示10進制

0x7F 表示16進制

0b101 表示2進制

傳回值

double 將字串轉換成浮點數值傳回。若數值格式錯誤，會傳回 0。

說明

```
value = String_ToDouble("12.34", 10)      // value = 12.34  
value = String_ToDouble("12.34", 16)      // value = 0      // 非法的16進制格式  
value = String_ToDouble("12.34", 2)       // value = 0      // 非法的2進制格式  
value = String_ToDouble("11.00", 2)       // value = 0      // 非法的2進制格式  
value = String_ToDouble("0x1234", 10)     // value = 2.30234590962021E-320// 自動數值格式判斷為16進制  
value = String_ToDouble("0x1234", 16)     // value = 2.30234590962021E-320  
value = String_ToDouble("0x1234", 2)       // value = 0      // 非法的2進制格式  
value = String_ToDouble("0b1100", 10)     // value = 5.92878775009496E-323// 自動數值格式判斷為2進制  
value = String_ToDouble("0b1100", 16)     // value = 3.58320121515072E-318// 合法16進制  
value = String_ToDouble("0b1100", 2)       // value = 5.92878775009496E-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      // 非法格式  
value = String_ToDouble("-0b1100", 2)     // value = 0      // 非法格式
```

語法 2

```
double String_ToDouble(  
    string  
)
```

說明

與語法 1 相同，預設將參數 int 填入 10 以 10 進制 或 依字串數值格式自動判斷。

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

語法 3

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

參數

`string[]` 輸入的字串陣列
`int` 輸入的字串陣列內的值是 10 進制、16 進制或 2 進制格式 (優先判斷)
 10 10進制 或 依字串數值格式自動判斷 (預設)
 16 16進制
 2 2進制
 字串數值格式
 123 表示10進制
 0x7F 表示16進制
 0b101 表示2進制
 * 陣列內元素的進制格式必須一致

傳回值

`double[]` 將字串陣列轉換成浮點數陣列傳回。若數值格式錯誤，會傳回 0。

說明

```
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, 8.44852254388532E-322, 1.00789391751614E-321, 1.09188507730915E-321, 0}  
value = String_ToDouble(ss, 10)               // value = {12.345, 0, 0, 0, 10.111}
```

2.14 String_ToByte()

將字串依序轉換成 byte 陣列

語法 1

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

參數

string 輸入的字串
int 輸入的字串轉換成 byte 陣列的轉換方式
0 UTF8 (預設)
1 HEX BINARY // 遇到非法 Hex 值即停止
2 ASCII

傳回值

byte[] 字串轉換後的 byte 陣列。

說明

```
value = String_ToByte("12345", 0)          // value = {0x31, 0x32, 0x33, 0x34, 0x35}  
value = String_ToByte("12345", 1)          // value = {0x12, 0x34, 0x50} // 缺少的部分補0  
value = String_ToByte("12345", 2)          // value = {0x31, 0x32, 0x33, 0x34, 0x35}  
value = String_ToByte("0x12345", 0)         // value = {0x30, 0x78, 0x31, 0x32, 0x33, 0x34, 0x35}  
value = String_ToByte("0x12345", 1)         // value = {0x00}      // 只轉 0，遇到 x 為非法的 Hex 值  
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 為非法的 Hex 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 為非法的 Hex 值
```

語法 2

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

說明

與語法 1 相同，預設將參數 int 填入 0 以 UTF8 轉換字串。

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

2.15 String_IndexOf()

搜尋字串第一次出現的位址

語法 1

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

參數

string 輸入的字串值

string 要搜尋的字串值，從輸入的字串值起始 0 開始搜尋第一次出現的位址

傳回值

int	0..字串長度-1	如果有找到字串，則傳回該字串的索引位址
	-1	表示沒有找到
	0	如果要搜尋的字串值為 "" 或 empty

說明

```
value = String_IndexOf("012314", "1")      // value = 1  
value = String_IndexOf("012314", "")        // value = 0  
value = String_IndexOf("012314", empty)     // value = 0  
value = String_IndexOf("012314", "d")       // value = -1  
value = String_IndexOf("", "d")             // value = -1
```

2.16 String_LastIndexOf()

搜尋字串最後一次出現的位址

語法 1

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

參數

string 輸入的字串值

string 要搜尋的字串值，從輸入的字串值起始 0 開始搜尋最後一次出現的位址

傳回值

int	0..字串長度-1	如果有找到字串，則傳回該字串的索引位址
	-1	表示沒有找到
	字串長度-1	如果要搜尋的字串值為 "" 或 empty

說明

```
value = String_LastIndexOf("012314", "1")      // value = 4  
value = String_LastIndexOf("012314", "")        // value = 5  
value = String_LastIndexOf("012314", empty)     // value = 5  
value = String_LastIndexOf("012314", "d")       // value = -1  
value = String_LastIndexOf("", "d")             // value = -1
```

2.17 String_Substring()

取出字串的子字串

語法 1

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

參數

string 輸入的字串值
int 輸入字串的起始位置。(0.. 字串長度-1)
int 要取出的字元數

傳回值

string 傳回取出的字串值
如果起始位置 <0，則傳回空字串
如果起始位置 >= 輸入字串長度，則傳回空字串
如果字元數 <0，則取至最後字元
如果起始位置加字元數大於字串長度，則取至最後字元

說明

```
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"
```

語法 2

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

說明

與語法 1 相同，預設將參數 int 取出字元數填入字串長度，即從起始位置取至最後字元。

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

語法 3

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

參數

string 輸入的字串值
string 輸入要取出的字串，如果有找到，會從第一次出現的字串位置開始取
int 要取出的字元數

傳回值

string 傳回取出的字串值
如果取出字串為空字串，則從字串起始位置 0 開始取出
如果取出字串不在輸入字串中，則傳回空字串
如果字元數 <0，則取至最後字元
如果起始位置加字元數大於字串長度，則取至最後字元

說明

此語法即等於 `String_Substring(str, String_IndexOf(str1), int)`

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

語法 4

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

說明

與語法 3 相同，預設將參數 int 取出字元數填入字串長度，即從起始位置取至最後字元。

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

語法 5

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

參數

string 輸入的字串值
string 要取出的前置字串值
string 要取出的後置字串值
int 符合個數

傳回值

string 傳回取出的字串值
如果前置字串及後置字串為空字串，則傳回輸入字串
如果符合個數 ≤ 0 ，則傳回空字串

說明

```
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" // 取前置符合之後  
value = String_Substring("0x12345122", "1", "", 2)       // value = "122"  
value = String_Substring("0x12345122", "1", "", 4)       // value = ""  
value = String_Substring("0x12345433", "", "4", 1)       // value = "0x123454" // 取後置符合之前  
value = String_Substring("0x12345433", "", "4", 2)       // value = "0x1234"
```

語法 6

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

說明

與語法 5 相同，預設將參數 int 符合個數填入 1 取第 1 次符合的子字串。

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

2.18 String_Split()

依照特定分隔符號字串切割輸入字串至字串陣列

語法 1

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

參數

string 輸入的字串值

string 分隔符號字串值

int 分隔選項

0 依分隔符號切割，保留空字串

1 依分隔符號切割，去除空字串

2 依分隔符號切割，被 "" 包括內的分隔符號不切割，保留空字串

3 依分隔符號切割，被 "" 包括內的分隔符號不切割，去除空字串

傳回值

string[] 傳回切割後的字串陣列

如果輸入字串是空字串，則傳回1個陣列，[0]=空字串

如果分隔字串是空字串，則傳回1個陣列，[0]=輸入字串

說明

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

語法 2

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

說明

與語法 1 相同，預設將參數 int 分隔選項填入 0，即依分隔符號切割，保留空字串。

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

2.19 String_Replace()

依照特定字串取代輸入字串內的字串

語法 1

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

參數

string 輸入的字串值
string 要被取代的字串值
string 取代的字串值

傳回值

string 傳回取代後的字串值
如果要被取代的字串是空字串，則傳回輸入字串

說明

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

2.20 String_Trim()

移除字串的左右空白字元

語法 1

```
string String_Trim(  
    string  
)
```

參數

string 輸入的字串值

傳回值

string 傳回移除左右空白字元後的字串值

說明

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

空白字元

\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					

語法 2

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

參數

string 輸入的字串值

string 要移除的左側字串值

傳回值

string 傳回移除左側字串後的字串值

語法 3

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

參數

string 輸入的字串值

string 要移除的左側字串值

string 要移除的右側字串值

傳回值

string 傳回移除左側字串與右側字串後的字串值

說明

```
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"
```

2.21 String_ToLower()

將字串轉換成小寫

語法 1

```
string String_ToLower(  
    string  
)
```

參數

string 輸入的字串值

傳回值

string 傳回轉換後的字串值。只針對英文字做轉換，非英文字的部分維持原字元傳回。

說明

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

2.22 String_ToUpper()

將字串轉換成大寫

語法 1

```
string String_ToUpper(  
    string  
)
```

參數

string 輸入的字串值

傳回值

string 傳回轉換後的字串值。只針對英文字做轉換，非英文字的部分維持原字元傳回。

說明

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

2.23 Array_Append()

新增元素資料進陣列內(附加在尾端)

語法 1

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

參數

?[] 輸入的陣列型別 1 (可以是 byte, int, float, double, bool, string 的陣列型別)
? or ?[] 輸入的資料型別 2 或陣列型別 2，型別必須與參數 1 相同
* 兩個型別必須相同

傳回值

?[] 傳回將資料 2 附加進參數 1 (附加在尾端) 的新陣列資料

說明

```
?   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, 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"}
```

2.24 Array_Insert()

插入元素資料進陣列內

語法 1

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

參數

?[]	輸入的陣列型別 1 (可以是 byte, int, float, double, bool, string 的陣列型別)
int	插入參數 1 的起始索引位址。
0..陣列1長度-1	合法值
>=陣列1長度	合法值，將插入在陣列 1 尾端
< 0	非法值，專案運行報錯
? or ?[]	輸入的資料型別 2 或陣列型別 2，型別必須與參數 1 相同 * 兩個型別必須相同

傳回值

?[] 傳回將資料 2 插入進參數 1 (從起始索引位址開始插入) 的新陣列資料

說明

```
?  int[] n1 = {100, 200, 30}  
    int[] n2 = {40, 50, 60}  
    n3 = Array_Insert(n1, 0, n2)          // n3 = {40, 50, 60, 100, 200, 30} // 從索引 0 開始插入  
    n4 = Array_Insert(n1, 2, n2)          // n4 = {100, 200, 40, 50, 60, 30} // 從索引 2 開始插入  
    n5 = Array_Insert(n1, -1, n2)         // n5 = {} // 報錯，非法的起始索引值  
  
?  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} // 超過陣列索引範圍，將插入在尾端
```

2.25 Array_Remove()

刪除陣列中的元素資料

語法 1

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

參數

?[]	輸入的陣列型別 1 (可以是 byte, int, float, double, bool, string 的陣列型別)
int	刪除參數 1 的起始索引位址。 0..陣列1長度-1 合法值 >=陣列1長度 非法值，專案運行報錯 <0 非法值，專案運行報錯
int	刪除的元素個數。 >0 從起始索引位址開始刪除元素個數，或至陣列元素最尾端 <0 個數設為 0，不刪除任何元素

傳回值

?[] 傳回刪除指定索引位址後的新陣列資料

語法 2

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

說明

與語法 1 相同，預設將刪除個數填入 1 個

```
? int[] n1 = {100, 200, 30, 40, 50, 60}  
n3 = Array_Remove(n1, -1)      // n3 = {}      // 報錯，非法的起始索引值  
n4 = Array_Remove(n1, 100)      // n4 = {}      // 報錯，非法的起始索引值  
n5 = Array_Remove(n1, 0)      // n5 = {200, 30, 40, 50, 60} // 索引 0 刪除  
n6 = Array_Remove(n1, 1, 2)      // n6 = {100, 40, 50, 60}      // 從索引 1 開始刪除 2 個元素  
n7 = Array_Remove(n1, 1, 100)      // n7 = {100}      // 從索引 1 開始刪除 100 個元素(刪至最尾端)  
n8 = Array_Remove(n1, Length(n1)-1)// n8 = {100, 200, 30, 40, 50}// 索引尾端刪除  
n9 = Array_Remove(n1, Length(n1)) // n9 = {}      // 報錯，非法的起始索引值
```

2.26 Array_Equals()

比較兩個陣列內的元素資料是否一致

語法 1

```
bool Array_Equals(  
    ?,  
    ?  
)
```

參數

- ?[] 輸入的陣列 1 (可以是 byte, int, float, double, bool, string 的陣列型別)
- ?[] 輸入的陣列 2 (可以是 byte, int, float, double, bool, string 的陣列型別)
- * 兩個陣列的型別必須相同

傳回值

- bool 傳回比較後的結果
 - true 兩陣列內的元素資料一致
 - false 兩陣列內的元素資料不一致

語法 2

```
bool Array_Equals(  
    ?,  
    int,  
    ?,  
    int,  
    int  
)
```

參數

- ?[] 輸入的陣列 1 (可以是 byte, int, float, double, bool, string 的陣列型別)
- int 陣列 1 的起始索引 (0 .. 陣列 1 大小-1)
- ?[] 輸入的陣列 2 (可以是 byte, int, float, double, bool, string 的陣列型別)
- int 陣列 2 的起始索引 (0 .. 陣列 2 大小-1)
- int 比較的元素個數 (若是 0 個，傳回 true)
- * 兩個陣列的型別必須相同

傳回值

- bool true 兩陣列的比較元素資料一致
 - false 兩陣列的比較元素資料不一致 (或參數值不正確)

說明

```
? 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 // 因比對4個
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 // 因比對2個
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 // 因比對0個

? 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 // 因起始索引不正確
```

2.27 Array_IndexOf()

搜尋陣列元素內第一次出現的索引值

語法 1

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

參數

?[] 輸入的陣列 (可以是 byte, int, float, double, bool, string 的陣列型別)

? 要搜尋的陣列元素值 (型別需與 ?[] 相同，但非陣列)

傳回值

int 0..陣列大小-1 如果有找到元素，則傳回該元素的索引值
-1 表示沒有找到

說明

```
? byte[] n = {100, 200, 30}  
value = Array_IndexOf(n, 200) // 1  
value = Array_IndexOf(n, 2000) // error // 2000 不是 byte 型別  
  
?  
int[] n = {1000, 2000, 3000}  
value = Array_IndexOf(n, 200) // -1  
  
?  
float[] n = {1.1, 2.2, 3.3}  
value = Array_IndexOf(n, 1.1) // 0  
  
?  
double[] n = {100, 200, 300, 3.3, 2.2, 1.1}  
value = Array_IndexOf(n, 1.1) // 5  
  
?  
bool[] n = {true, false, true, true, true}  
value = Array_IndexOf(n, true) // 0  
  
?  
string[] n = {"123", "ABC", "456", "DEF"}  
value = Array_IndexOf(n, "456") // 2
```

2.28 Array_LastIndexOf()

搜尋陣列元素內最後一次出現的索引值

語法 1

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

參數

?[] 輸入的陣列 (可以是 byte, int, float, double, bool, string 的陣列型別)

? 要搜尋的陣列元素值 (型別需與 ?[] 相同，但非陣列)

傳回值

int 0..陣列大小-1 如果有找到元素，則傳回該元素的索引值
-1 表示沒有找到

說明

```
? byte[] n = {100, 200, 30}  
    value = Array_LastIndexOf(n, 200)      // 1  
    value = Array_LastIndexOf(n, 2000)      // error // 2000 不是 byte 型別  
  
?  
? int[] n = {1000, 2000, 3000}  
    value = Array_LastIndexOf(n, 200)      // -1  
  
?  
? float[] n = {1.1, 2.2, 3.3}  
    value = Array_LastIndexOf(n, 1.1)      // 0  
  
?  
? double[] n = {100, 200, 300, 3.3, 2.2, 1.1}  
    value = Array_LastIndexOf(n, 1.1)      // 5  
  
?  
? bool[] n = {true, false, true, true, true}  
    value = Array_LastIndexOf(n, true)      // 4  
  
?  
? string[] n = {"123", "ABC", "456", "DEF"}  
    value = Array_LastIndexOf(n, "456")      // 2
```

2.29 Array_Reverse()

將陣列內的元素順序反轉

語法 1

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

參數

?[] 輸入的陣列 (可以是 byte, int, float, double, bool, string 的陣列型別)

傳回值

?[] 傳回反轉後的陣列

說明

```
?  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"}
```

語法 2

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

參數

?[] 輸入的陣列 (可以是 byte, int, float, double, bool, string 的陣列型別)

int 在多少個元素為單位內進行反轉

2 以 2 個元素為單位

4 以 4 個元素為單位

8 以 8 個元素為單位

* 指在同單位元素內反轉，而不同單位的陣列元素順序不變

傳回值

?[] 傳回反轉後的陣列

說明

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

n = **Array_Reverse**(n, 2) // n = {200, 100, 30} // 以 2 個元素為單位，所以 {100,200}{30}

n = **Array_Reverse**(n, 4) // n = {30, 200, 100} // 以 4 個元素為單位，所以 {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} // 以 2 個元素為單位，所以 {100,200}{300,400}

n = **Array_Reverse**(n, 4) // n = {400, 300, 200, 100} // 以 4 個元素為單位，所以 {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} // 以 2 個元素為單位，所以 {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} // 以 4 個元素為單位，所以 {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, 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"}
```

2.30 Array_Sort()

將陣列內的元素資料進行排序

語法 1

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

參數

?[] 輸入的陣列 (可以是 byte, int, float, double, bool, string 的陣列型別)
int 排序方向
0 由小至大，遞增排序 (預設)
1 由大至小，遞減排序

傳回值

?[] 傳回排序後的陣列

語法 2

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

說明

與語法 1 相同，預設將參數 int 排序方向填入 0 遞增排序

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"}
```

2.31 Array_SubElements()

取出陣列內的元素資料

語法 1

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

參數

?[] 輸入的陣列 (可以是 byte, int, float, double, bool, string 的陣列型別)
int 要取出的起始索引。(0 .. 陣列大小-1)
int 要取出的元素個數

傳回值

?[] 傳回取出的陣列
如果起始索引 <0，則傳回空陣列
如果起始索引 >= 輸入陣列大小，則傳回空陣列
如果取出元素個數 <0，則從起始索引取至陣列最後元素
如果起始索引加元素個數大於陣列大小，則從起始索引取至陣列最後元素

語法 2

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

說明

與語法 1 相同，預設將參數 int 取出元素個數填入陣列長度，即從起始索引取至最後元素

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

```
? byte[] n = {100, 200, 30}  
n1 = Array_SubElements(n, 0)      // n1 = {100, 200, 30}  
n1 = Array_SubElements(n, -1)     // n1 = {}  
n1 = Array_SubElements(n, 0, 3)    // n1 = {100, 200, 30}  
n1 = Array_SubElements(n, 1, 3)    // n1 = {200, 30}  
n1 = Array_SubElements(n, 2)       // n1 = {30}  
n1 = Array_SubElements(n, 3, 3)    // n1 = {}
```

```

? int[] n = {1000, 2000, 3000}
n1 = Array_SubElements(n, 0)      // n1 = {1000, 2000, 3000}
n1 = Array_SubElements(n, -1)     // n1 = {}
n1 = Array_SubElements(n, 1, 3)   // n1 = {2000, 3000}
n1 = Array_SubElements(n, 2)      // n1 = {3000}

? float[] n = {1.1, 2.2, 3.3}
n1 = Array_SubElements(n, 0)      // n1 = {1.1, 2.2, 3.3}
n1 = Array_SubElements(n, -1)     // n1 = {}
n1 = Array_SubElements(n, 1, 3)   // n1 = {2.2, 3.3}
n1 = Array_SubElements(n, 2)      // n1 = {3.3}

? double[] n = {100, 200, 3.3, 2.2, 1.1}
n1 = Array_SubElements(n, 0)      // n1 = {100, 200, 3.3, 2.2, 1.1}
n1 = Array_SubElements(n, -1)     // n1 = {}
n1 = Array_SubElements(n, 1, 3)   // n1 = {200, 3.3, 2.2}
n1 = Array_SubElements(n, 2)      // n1 = {3.3, 2.2, 1.1}

? bool[] n = {true, false, true, true, true}
n1 = Array_SubElements(n, 0)      // n1 = {true, false, true, true, true}
n1 = Array_SubElements(n, -1)     // n1 = {}
n1 = Array_SubElements(n, 1, 3)   // n1 = {false, true, true}
n1 = Array_SubElements(n, 2)      // n1 = {true, true, true}

? string[] n = {"123", "ABC", "456", "DEF"}
n1 = Array_SubElements(n, 0)      // n1 = {"123", "ABC", "456", "DEF"}
n1 = Array_SubElements(n, -1)     // n1 = {}
n1 = Array_SubElements(n, 1, 3)   // n1 = {"ABC", "456", "DEF"}
n1 = Array_SubElements(n, 2)      // n1 = {"456", "DEF"}

```

2.32 ValueReverse()

對數值 (int 2 bytes or 4 bytes, float 4 bytes, double 8 bytes) 內的 byte 單位反轉；
或對字串值的字元順序反轉

語法 1

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

參數

int 輸入的數值

int 輸入的整數數值是使用 int32 或 int16 型別

0 int32 (預設)

1 int16，自動判斷輸入值是否符合 int16，若不符合則會依 int32

2 int16，強制使用 int16，若輸入值為 int32 將會有數值遺失

傳回值

int 傳回反轉後的數值。int32 為 4 bytes 反轉，int16 為 2 bytes 反轉。

說明

```
int i = 10000  
  
value = ValueReverse(i, 0) // 10000=0x00002710 → 0x10270000 // value = 270991360  
value = ValueReverse(i, 1) // 10000=0x2710 → 0x1027 // value = 4135  
  
i = 100000 // int32 數值  
  
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
```

語法 2

```
int ValueReverse(  
    int  
)
```

參數

int 輸入的數值

說明

與語法 1 相同，預設將參數 int 填入 0 整數為 int32 型別。

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

語法 3

```
float ValueReverse(  
    float  
)
```

參數

float 輸入的數值

傳回值

float 傳回反轉後的數值。float 為 4 bytes 反轉。

說明

float i = 40000

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

語法 4

```
double ValueReverse(  
    double  
)
```

參數

double 輸入的數值

傳回值

double 傳回反轉後的數值。double 為 8 bytes 反轉。

說明

double i = 80000

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

語法 5

```
string ValueReverse(  
    string  
)
```

參數

string 輸入的字串

傳回值

string 傳回反轉後的字串。對字串內的字元順序反轉。

說明

string i = "ABCDEF"

value = ValueReverse(i) // value = "FEDCBA"

語法 6

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

參數

int[] 輸入的數值陣列

int 輸入的整數數值是使用 int32 或 int16 型別

0 int32 (預設)

1 int16，自動判斷輸入值是否符合 int16，若不符合則會依 int32

2 int16，強制使用 int16，若輸入值為 int32 將會有數值遺失

傳回值

int[] 傳回反轉後的數值陣列。將陣列中的每一筆數值，都進行反轉。

說明

```
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}
```

語法 7

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

參數

int[] 輸入的數值陣列

說明

與語法 6 相同，預設將參數 int 填入 0 整數為 int32 型別。

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

語法 8

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

參數

float[] 輸入的數值陣列

傳回值

float[] 傳回反轉後的數值陣列。將陣列中的每一筆數值，都進行反轉。

說明

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

語法 9

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

參數

double[] 輸入的數值陣列

傳回值

double[] 傳回反轉後的數值陣列。將陣列中的每一筆數值，都進行反轉。

說明

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

語法 10

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

參數

string[] 輸入的字串陣列

傳回值

string[] 傳回反轉後的字串陣列。將陣列中的每一筆字串，都進行反轉。

說明

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

2.33 GetBytes()

將任意型別值轉換成 byte 陣列

語法 1

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

參數

- ? 輸入的原始值，可以是整數、浮點數、布林值、字串值或是陣列型別
- int 若輸入值為整數、浮點數時，值是依照 Little Endian 格式取或是依 Big Endian 格式取
 - 0 Little Endian (預設)
 - 1 Big Endian
- 若輸入值為字串陣列時，是否在字串元素值間插入分隔位元 0x00 0x00
 - 0 不插入分隔位元 0x00 0x00 (預設)
 - 1 插入分隔位元 0x00 0x00

傳回值

byte[] 傳回轉換後的 byte 陣列。

語法 2

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

說明

與語法 1 相同，預設將參數 Little Endian 或 Big Endian 填入 0，即依照 Little Endian 傳回

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} // 將陣列中的每筆數值，依 byte 型別取 1 byte 值後，依序轉換
 - 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} // 將陣列中的每筆數值，依 int32 型別取 4 byte 值後，依序轉換
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} // 將陣列中的每筆數值，依 float 型別取 4 byte 值後，依序轉換
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} // 將陣列中的每筆數值，依 double 型別取 8 byte 值後，依序轉換
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           // true 會轉換為 1；false 會轉換為 0
    value = GetBytes(flag)      // value = {1}
    value = GetBytes(flag, 0)    // value = {1}    // bool 為 1 byte，因此 Endian 參數無效
    value = GetBytes(flag, 1)    // value = {1}

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

?   string n = "ABCDEFG"       // string 使用 UTF8 轉換
    value = GetBytes(n)         // value = {0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47}
    value = GetBytes(n, 0)       // value = {0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47} // Endian 參數無效
    value = GetBytes(n, 1)       // value = {0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47}

?   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}

* 若只是將 string[] 轉換成 byte[] 而不插入分隔位元，可保留完整內容，但 byte[] 無法有效轉回 string[]
* 在陣列元素間插入分隔位元(連續兩個0x00位元)，可將 byte[] 有效轉回 string[]，但仍需注意字串值若含有 0x00 0x00 位元，有可能轉換錯誤

```

語法 3

將整數 (int 型別) 轉換成 byte 陣列

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

參數

int	輸入的整數數值 (int 型別)
int	輸入的值是依照 Little Endian 格式或是依 Big Endian 格式
0	Little Endian (預設)
1	Big Endian
int	輸入的整數數值是使用 int32 或 int16 型別
0	int32 (預設)
1	int16，自動判斷輸入值是否符合 int16，若不符合則會依 int32
2	int16，強制使用 int16，若輸入值為 int32 將會有數值遺失

傳回值

byte[] 傳回轉換後的 byte 陣列。

先依 int32 型別取 4 bytes 值或依 int16 型別取 2 bytes 值後，再轉成陣列。

說明

```
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} // 有數值遺失  
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} // 有數值遺失
```

語法 4

將整數陣列 (int[] 型別) 轉換成 byte 陣列

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

參數

int[] 輸入的整數數值陣列 (int[] 型別)

int 輸入的值是依照 Little Endian 格式或是依 Big Endian 格式

0 Little Endian (預設)

1 Big Endian

int 輸入的整數數值是使用 int32 或 int16 型別

0 int32 (預設)

1 int16，自動判斷輸入值是否符合 int16，若不符合則會依 int32

2 int16，強制使用 int16，若輸入值為 int32 將會有數值遺失

傳回值

byte[] 傳回轉換後的 byte 陣列。

將陣列中的每筆數值，依 int32 型別取 4 bytes 值或依 int16 型別取 2 bytes 值後，轉成陣列，依序對輸入的陣列值進行轉換為 byte 陣列。

說明

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} // 有數值遺失

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} // 有數值遺失

2.34 GetString()

將任意型別值轉換成字串值

語法 1

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

參數

- ? 輸入的原始值，可以是整數、浮點數、布林值、字串值或是陣列型別
- int 若輸入值為整數、浮點數時，輸出的字串值是 10 進制、16 進制或 2 進制格式
 - 10 10進制
 - 16 16進制
 - 2 2進制
- 字串數值格式
 - 123 表示10進制
 - 0x7F 表示16進制
 - 0b101 表示2進制
- 若輸入值為字串陣列時，輸出的字串值是否為標準字串格式
 - 0 或 10 自動判斷，若字串值含有雙引號或逗號時，則轉換為標準字串格式
 - 1 強制轉換為標準字串格式
 - 其他 不進行任何轉換
- int 輸出的字串數值格式 (只對 16 進制或 2 進制有效)
 - 0 補滿位數，加前綴 0x 或 0b，如 0x0C 或 0b00001100
 - 1 補滿位數，不加前綴 0x 或 0b，如 0C 或 00001100
 - 2 不補滿位數，加前綴 0x 或 0b，如 0xC 或 0b1100
 - 3 不補滿位數，不加前綴 0x 或 0b，如 C 或 1100

傳回值

- string 將值轉換成字串值傳回，若值無法轉換，則回傳空字串。
若是陣列型別，會將陣列內每筆元素轉換成字串，再依格式 "{,,}" 傳回。

語法 2

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

說明

與語法 1 相同，預設將字串格式參數填入 0，即補滿位數及加前綴

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

語法 3

```
string GetString(  
    ?  
)
```

說明

與語法 1 相同，預設將進制參數填入 10 及字串格式參數填入 0

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

GetString(?) => **GetString(?, 0, 0)** // 若 ? 為字串陣列時

? **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 = "0b0000000000000000000000000000000010011010010"
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,0x00000159,0xFFFFF85,0xFFFFE38}"
value = GetString(n, 2) // value = "{0b00000000000000000000000000000000111011,
                           0b00000000000000000000000000000000101011001,
                           0b1111111111111111111111111111110000101,
                           0b111111111111111111111111111111000111000}""

value = GetString(n, 16, 3) // value = "{7B,159,FFFFF85,FFFFE38}"
value = GetString(n, 2, 2) // value = "{0b1111011,
                           0b101011001,
                           0b111111111111111111111111110000101,
                           0b11111111111111111111111111000111000}""

?

? float n = 12.34

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

?

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

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

?

? double n = 12.34

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

```

```

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

value = GetString(n)      // value = "{123.45,345.67,-123.48,-456.79}"
value = GetString(n, 10)   // value = "{123.45,345.67,-123.48,-456.79}"
value = GetString(n, 16)   // value = "{0x405EDCCCCCCCCC,0x40759AB851EB851F,
                           0xC05EDEB851EB851F,0xC07C8CA3D70A3D71}"
value = GetString(n, 16, 3) // value = "{405EDCCCCCCCCCD,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 會顯示為 -12"3.48
value = GetString(n, 16, 3) // value = "[123.45,345.67,-12""3.48,-45A6.79]" // 使用預設自動判斷

```

語法 4

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

參數

? 輸入的原始值，可以是整數、浮點數、布林值、字串值或是陣列型別

string 輸出陣列型別的字串分隔符號（只對 ? 是陣列型別有效）

int 若輸入值為整數、浮點數時，輸出的字串值是 10 進制、16 進制或 2 進制格式

10 10進制

16 16進制

2 2進制

字串數值格式

123 表示10進制

0x7F 表示16進制

0b101 表示2進制

若輸入值為字串陣列時，輸出的字串值是否為標準字串格式

0 或 10 自動判斷，若字串值含有雙引號或分隔符號時，則轉換為標準字串格式

1 強制轉換為標準字串格式

其他 不進行任何轉換

int 輸出的字串數值格式（只對 16 進制或 2 進制有效）

0 補滿位數，加前綴 0x 或 0b，如 0x0C 或 0b00001100

1 補滿位數，不加前綴 0x 或 0b，如 0C 或 00001100

2 不補滿位數，加前綴 0x 或 0b，如 0xC 或 0b1100

3 不補滿位數，不加前綴 0x 或 0b，如 C 或 1100

傳回值

string 將值轉換成字串值傳回，若值無法轉換，則回傳空字串。

若是陣列型別，會將陣列內每筆元素轉換成字串，依分隔符號隔開，再以字串傳回。
不會有 {} 括起。

語法 5

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

說明

與語法 4 相同，預設將字串格式參數填入 0，即補滿位數及加前綴

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

語法 6

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

說明

與語法 4 相同，預設將進制參數填入 10 及字串格式參數填入 0

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

GetString(?, str) => **GetString(?, str, 0, 0)** // 若 ? 為字串陣列時

```
? 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"
```

* 分隔符號只對陣列型別有效

```
? 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 為書寫換行符號表示，並非字串值
```

```
? 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" // 難識別分隔符號與負號
```

語法 7

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

參數

? 輸入的原始值，可以是整數、浮點數、布林值、字串值或是陣列型別

string 輸出陣列型別的字串元素索引值 (只對 ? 是陣列型別有效)

* 支援數值格式化字串

string 輸出陣列型別的字串分隔符號 (只對 ? 是陣列型別有效)

int 若輸入值為整數、浮點數時，輸出的字串值是 10 進制、16 進制或 2 進制格式

10 10進制

16 16進制

2 2進制

字串數值格式

123 表示10進制

0x7F 表示16進制

0b101 表示2進制

若輸入值為字串陣列時，輸出的字串值是否為標準字串格式

0 或 10 自動判斷，若字串值含有雙引號或分隔符號時，則轉換為標準字串格式

1 強制轉換為標準字串格式

其他 不進行任何轉換

int 輸出的字串數值格式 (只對 16 進制或 2 進制有效)

0 補滿位數，加前綴 0x 或 0b，如 0x0C 或 0b00001100

1 補滿位數，不加前綴 0x 或 0b，如 0C 或 00001100

2 不補滿位數，加前綴 0x 或 0b，如 0xC 或 0b1100

3 不補滿位數，不加前綴 0x 或 0b，如 C 或 1100

傳回值

string 將值轉換成字串值傳回，若值無法轉換，則回傳空字串。

若是陣列型別，會將陣列內每筆元素轉換成字串，前綴加上元素索引值格式化字串，且依分隔符號隔開，再以字串傳回。

不會有 {} 括起。

語法 8

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

說明

與語法 7 相同，預設將字串格式參數填入 0，即補滿位數及加前綴

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

語法 9

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

說明

與語法 7 相同，預設將進制參數填入 10 及字串格式參數填入 0

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"  
  
* 索引值、分隔符號只對陣列型別有效  
  
?  
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]=" 支援數值格式化  
* \u0D0A 為書寫換行符號表示，並非字串值
```

2.35 GetToken()

取出字串的子字串，或是取出 byte[] 陣列內的子元素陣列

語法 1

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

參數

string 輸入的字串值
string 要取出的前置字串
string 要取出的後置字串
int 取出的符合個數
 ≥ 1 取出第幾個符合的字串
 -1 取出最後一個符合的字串
int 取出選項
 0 第 1 個符合不須在輸入字串開頭，且不移除前置字串及後置字串（預設）
 1 第 1 個符合不須在輸入字串開頭，且移除前置字串及後置字串
 2 第 1 個符合須在輸入字串開頭，且不移除前置字串及後置字串
 3 第 1 個符合須在輸入字串開頭，且移除前置字串及後置字串

傳回值

string 傳回取出的字串值
如果前置字串及後置字串為空字串，則傳回輸入字串
如果符合個數 ≤ 0 或大於符合總數，則傳回空字串
如果取出選項為 2 或 3 時，則取出的第 1 個符合必須在輸入值開頭，否則傳回空字串

語法 2

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

說明

與語法 1 相同，預設將取出選項參數填入 0

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

語法 3

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

說明

與語法 1 相同，預設將符合個數參數填入 1 及取出選項參數填入 0

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$"    // 去除前置與後置
value = GetToken(n, Ctrl("\r\n"), "$", 1)         // value = "\u0d0a56\r\n78$"
value = GetToken(n, newline, "$", 2)              // value = ""
value = GetToken(n, NewLine, "$", 1, 1)           // value = "56\r\n78"

* \u0d0a 為書寫換行符號表示，並非字串值

```

```

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 = ""          // 因符合 $ 的字串不在輸入值的開頭
value = GetToken(n, "$", "$", -1, 2)             // value = ""          // 因符合 $ 的字串不在輸入值的開頭
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"

```

語法 4

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

參數

string 輸入的字串值
byte[] 要取出的前置字元，byte[] 陣列格式
byte[] 要取出的後置字元，byte[] 陣列格式
int 取出的符合個數
 ≥ 1 取出第幾個符合的字串
 -1 取出最後一個符合的字串
int 取出選項
 0 第 1 個符合不須在輸入字串開頭，且不移除前置字串及後置字串（預設）
 1 第 1 個符合不須在輸入字串開頭，且移除前置字串及後置字串
 2 第 1 個符合須在輸入字串開頭，且不移除前置字串及後置字串
 3 第 1 個符合須在輸入字串開頭，且移除前置字串及後置字串

傳回值

string 傳回取出的字串值
如果前置字串及後置字串為空字串，則傳回輸入字串
如果符合個數 ≤ 0 或大於符合總數，則傳回空字串
如果取出選項為 2 或 3 時，則取出的第 1 個符合必須在輸入值開頭，否則傳回空字串

語法 5

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

說明

與語法 4 相同，預設將取出選項參數填入 0

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

語法 6

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

說明

與語法 4 相同，預設將符合個數參數填入 1 及取出選項參數填入 0

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

```
string n = "$abcd$1234$ABCD$"  
  
byte[] bb0 = {}, bb1 = {0x24}      // 0x24 is $  
  
value = GetToken(n, bb0, bb0, 0)    // value = "$abcd$1234$ABCD$"  
value = GetToken(n, bb1, bb1)       // value = "$abcd$"  
value = GetToken(n, bb1, bb1, 0)    // value = ""  
value = GetToken(n, bb1, bb1, 1)    // value = "$abcd$"  
value = GetToken(n, bb1, bb1, 2)    // value = "$ABCD$"  
value = GetToken(n, bb1, bb1, 3)    // value = ""  
value = GetToken(n, bb1, bb1, 1, 1) // value = "abcd"  
value = GetToken(n, bb1, bb1, 2, 1) // value = "ABCD"  
value = GetToken(n, bb1, bb0, 1)    // value = "$abcd"  
value = GetToken(n, bb1, bb0, 2)    // value = "$1234"  
value = GetToken(n, bb1, bb0, 3)    // value = "$ABCD"  
value = GetToken(n, bb1, bb0, 4)    // value = "$"  
value = GetToken(n, bb0, bb1, 1)    // value = "$"  
value = GetToken(n, bb0, bb1, 2)    // value = "abcd$"  
value = GetToken(n, bb0, bb1, 3)    // value = "1234$"  
value = GetToken(n, bb0, bb1, 4)    // value = "ABCD$"  
  
string n = "$abcd$1234$ABCD$" + Ctrl("\r\n") + "56\r\n78$"  
  
byte[] bb0 = {0x0D,0x0A}, bb1 = {0x24}      // 0x24 is $ // 0x0D,0x0A is \u0D0A  
  
value = GetToken(n, bb1, bb0, 1)    // value = "$abcd$1234$ABCD$\u0D0A"  
value = GetToken(n, bb1, bb0, 2)    // value = ""  
value = GetToken(n, bb1, bb0, 1, 1) // value = "abcd$1234$ABCD$"    // 去除前置與後置  
value = GetToken(n, bb0, bb1, 1)    // value = "\u0D0A56\r\n78$"  
value = GetToken(n, bb0, bb1, 2)    // value = ""  
value = GetToken(n, bb0, bb1, 1, 1) // value = "56\r\n78"  
  
* \u0D0A 為書寫換行符號表示，並非字串值
```

語法 7

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

參數

byte[] 輸入的 byte[] 陣列
string 要取出的前置字串
string 要取出的後置字串
int 取出的符合個數
 ≥ 1 取出第幾個符合的陣列
 -1 取出最後一個符合的陣列
int 取出選項
 0 第 1 個符合不須在輸入值開頭，且不移除前置字串及後置字串（預設）
 1 第 1 個符合不須在輸入值開頭，且移除前置字串及後置字串
 2 第 1 個符合須在輸入值開頭，且不移除前置字串及後置字串
 3 第 1 個符合須在輸入值開頭，且移除前置字串及後置字串

傳回值

byte[] 傳回取出的 byte[] 陣列
如果前置字串及後置字串為空字串，則傳回輸入陣列
如果符合個數 ≤ 0 或大於符合總數，則傳回空陣列
如果取出選項為 2 或 3 時，則取出的第 1 個符合必須在輸入值開頭，否則傳回空陣列

語法 8

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

說明

與語法 7 相同，預設將取出選項參數填入 0

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

語法 9

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

說明

與語法 7 相同，預設將符合個數參數填入 1 及取出選項參數填入 0

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}      // 去除前置與後置  
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}
```

語法 10

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

參數

byte[] 輸入的 byte[] 陣列
byte[] 要取出的前置字元，byte[] 陣列格式
byte[] 要取出的後置字元，byte[] 陣列格式
int 取出的符合個數
 ≥ 1 取出第幾個符合的陣列
 -1 取出最後一個符合的陣列
int 取出選項
 0 第 1 個符合不須在輸入值開頭，且不移除前置字串及後置字串（預設）
 1 第 1 個符合不須在輸入值開頭，且移除前置字串及後置字串
 2 第 1 個符合須在輸入值開頭，且不移除前置字串及後置字串
 3 第 1 個符合須在輸入值開頭，且移除前置字串及後置字串

傳回值

byte[] 傳回取出的 byte[] 陣列
如果前置字串及後置字串為空字串，則傳回輸入陣列
如果符合個數 ≤ 0 或大於符合總數，則傳回空陣列
如果取出選項為 2 或 3 時，則取出的第 1 個符合必須在輸入值開頭，否則傳回空陣列

語法 11

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

說明

與語法 10 相同，預設將取出選項參數填入 0

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

語法 12

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

說明

與語法 10 相同，預設將符合個數參數填入 1 及取出選項參數填入 0

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

```
string s = "$abcd$1234$ABCD$"  
  
byte[] n = GetBytes(s)  
  
byte[] bb0 = {}, bb1 = {0x24} // 0x24 is $  
  
value = GetToken(n, bb0, bb0, 0) // value = {0x24,0x61,0x62,0x63,0x64,0x24,0x31,0x32,0x33,0x34,0x24,0x41,0x42,0x43,0x44,0x24}  
value = GetToken(n, bb1, bb1) // value = {0x24,0x61,0x62,0x63,0x64,0x24}  
value = GetToken(n, bb1, bb1, 0) // value = {}  
value = GetToken(n, bb1, bb1, 1) // value = {0x24,0x61,0x62,0x63,0x64,0x24}  
value = GetToken(n, bb1, bb1, 2) // value = {0x24,0x41,0x42,0x43,0x44,0x24}  
value = GetToken(n, bb1, bb1, 1, 1) // value = {0x61,0x62,0x63,0x64}  
value = GetToken(n, bb1, bb1, 2, 1) // value = {0x41,0x42,0x43,0x44}  
value = GetToken(n, bb1, bb0, 1) // value = {0x24,0x61,0x62,0x63,0x64}  
value = GetToken(n, bb1, bb0, 2) // value = {0x24,0x31,0x32,0x33,0x34}  
value = GetToken(n, bb1, bb0, 3) // value = {0x24,0x41,0x42,0x43,0x44}  
value = GetToken(n, bb0, bb1, 1) // value = {0x24}  
value = GetToken(n, bb0, bb1, 2) // value = {0x61,0x62,0x63,0x64,0x24}  
value = GetToken(n, bb0, bb1, 3) // value = {0x31,0x32,0x33,0x34,0x24}
```

```
string s = "$abcd$1234$ABCD$" + Ctrl("\r\n") + "56\r\n78$"  
  
byte[] n = GetBytes(s)  
  
byte[] bb0 = {0x0D,0x0A}, bb1 = {0x24}  
  
value = GetToken(n, bb1, bb0, 1)  
// value = {0x24,0x61,0x62,0x63,0x64,0x24,0x31,0x32,0x33,0x34,0x24,0x41,0x42,0x43,0x44,0x24,0x0D,0x0A}  
value = GetToken(n, bb1, bb0, 1, 1)  
// value = {0x61,0x62,0x63,0x64,0x24,0x31,0x32,0x33,0x34,0x24,0x41,0x42,0x43,0x44,0x24} // 去除前置與後置  
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}
```

2.36 GetAllTokens()

取出字串中所有符合條件的子字串

語法 1

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

參數

string 輸入的字串值
string 要取出的前置字串
string 要取出的後置字串
int 取出選項

0	第 1 個符合不須在輸入值開頭，且不移除前置字串及後置字串 (預設)
1	第 1 個符合不須在輸入值開頭，且移除前置字串及後置字串
2	第 1 個符合須在輸入值開頭，且不移除前置字串及後置字串
3	第 1 個符合須在輸入值開頭，且移除前置字串及後置字串

傳回值

string[] 傳回取出的字串陣列

如果前置字串及後置字串為空字串，則傳回輸入字串(字串陣列)

如果取出選項為 2 或 3 時，則取出的第 1 個符合必須在輸入值開頭，否則傳回空陣列

語法 2

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

說明

與語法 1 相同，預設將取出選項參數填入 0

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$"}
value = GetAllTokens(n, "$", "", 0)        // value = {"$1234#ABCD", "$5678#"}
value = GetAllTokens(n, "$", "", 1)        // value = {"1234#ABCD", "5678#"}
value = GetAllTokens(n, "$", "", 2)        // value = {}      // $ 不在輸入值開頭，傳回空陣列
value = GetAllTokens(n, "$", "", 3)        // value = {}      // $ 不在輸入值開頭，傳回空陣列
value = GetAllTokens(n, "$", "$", 0)       // value = {"$1234#ABCD$"}
value = GetAllTokens(n, "$", "$", 1)       // value = {"1234#ABCD"}
value = GetAllTokens(n, "$", "$", 2)       // value = {}      // $ 不在輸入值開頭，傳回空陣列
value = GetAllTokens(n, "$", "$", 3)       // value = {}      // $ 不在輸入值開頭，傳回空陣列
value = GetAllTokens(n, "#", "", 0)        // value = {"#abcd$1234", "#ABCD$5678", "#"}
value = GetAllTokens(n, "#", "", 1)        // value = {"abcd$1234", "ABCD$5678", ""}
value = GetAllTokens(n, "#", "", 2)        // value = {"#abcd$1234", "#ABCD$5678", "#"}
value = GetAllTokens(n, "#", "", 3)        // value = {"abcd$1234", "ABCD$5678", ""}
value = GetAllTokens(n, "#", "$", 0)       // value = {"#abcd$", "#ABCD$"}
value = GetAllTokens(n, "#", "$", 1)       // value = {"abcd", "ABCD"}
value = GetAllTokens(n, "#", "$", 2)       // value = {"#abcd$", "#ABCD$"}
value = GetAllTokens(n, "#", "$", 3)       // value = {"abcd", "ABCD"}
value = GetAllTokens(n, "", "$", 0)        // value = {"#abcd$", "1234#ABCD$"}
value = GetAllTokens(n, "", "$", 1)        // value = {"#abcd", "1234#ABCD"}
value = GetAllTokens(n, "", "$", 2)        // value = {"#abcd$", "1234#ABCD$"}
value = GetAllTokens(n, "", "$", 3)        // value = {"#abcd", "1234#ABCD"}
value = GetAllTokens(n, "", "#", 0)        // value = {"#", "abcd$1234#", "ABCD$5678#"}
value = GetAllTokens(n, "", "#", 1)        // value = {"", "abcd$1234", "ABCD$5678"}
value = GetAllTokens(n, "", "#", 2)        // value = {"#", "abcd$1234#", "ABCD$5678#"}
value = GetAllTokens(n, "", "#", 3)        // value = {"", "abcd$1234", "ABCD$5678"}

```

2.37 GetNow()

取得當前系統時間

語法 1

```
string GetNow(  
    string  
)
```

參數

string 要取得的時間字串格式，下列字為日期和時間格式化字串，會使時間依照對應格式轉換，其他字則維持原字串內容

d	月份天數，從 1 到 31
dd	月份天數，從 01 到 31
ddd	星期幾的縮寫名稱，Sun, Mon, Tue, ...
ddd	星期幾的完整名稱，Sunday, Monday, Tuesday, ...
f	十分之一秒
ff	百分之一秒
fff	千分之一秒
ffff	萬分之一秒
h	12小時制的小時，從 1 到 12
hh	12小時制的小時，從 01 到 12
H	24小時制的小時，從 0 到 23
HH	24小時制的小時，從 00 到 23
m	分鐘，從 0 到 59
mm	分鐘，從 00 到 59
M	月份，從 1 到 12
MM	月份，從 01 到 12
MMM	月份的縮寫名稱，Jun
MMMM	月份的完整名稱，June
s	秒數，從 0 到 59
ss	秒數，從 00 到 59
t	AM/PM 的第一個字元
tt	AM/PM
y	年份，從 0 到 99
yy	年份，從 00 到 99
yyyy	年份，至少四位數
/	日期分隔符號，/ 或 - 或 . (依語系不同)

傳回值

`string` 傳回當前時間，但若輸入格式錯誤，則採用預設時間字串格式"MM/dd/yyyy HH:mm:ss"

說明

```
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 13:40:30.123
value = GetNow("yyyy-MM-dd hh:mm:ss tt")          // value = 2017-08-15 01:40:30 PM
```

語法 2

```
string GetNow()  
)
```

參數

`void` 無輸入值，依預設時間字串格式 "MM/dd/yyyy HH:mm:ss"

傳回值

`string` 傳回當前時間

說明

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

2.38 GetNowStamp()

取得當前專案運行的總毫秒時間或毫秒時間差

語法 1

```
int GetNowStamp()  
)
```

參數

void 無輸入值

傳回值

int 傳回當前專案運行的總毫秒時間，總毫秒時間上限值為 2147483647
< 0 溢位，非法的總毫秒時間

說明

```
value = GetNowStamp() // value = 2147483647  
... others ...  
value = GetNowStamp() // value = -1 // 數值溢位
```

語法 2

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

參數

bool 是否使用 double 型別，以取得當前專案運行的總毫秒時間
true 使用 double 型別，總毫秒時間上限值為 9223372036854775807
false 使用 int32 型別，總毫秒時間上限值為 2147483647

傳回值

double 傳回當前專案運行的總毫秒時間
< 0 溢位，非法的總毫秒時間

說明

```
value = GetNowStamp(false) // value = 2147483647  
... others ...  
value = GetNowStamp(false) // value = -1 // 數值溢位  
value = GetNowStamp(true) // value = 3147483647
```

語法 3

```
int GetNowStamp()  
    int  
)
```

參數

int 輸入前次記錄的總毫秒時間

傳回值

int 傳回當前總毫秒時間與輸入前次記錄的總毫秒時間差，毫秒單位

時間差 = 當前總毫秒 - 輸入的總毫秒

< 0 輸入前次記錄小於零或大於當前總毫秒時間或溢位，表示為非法的毫秒時間差

說明

```
value = GetNowStamp() // value = 2147483546  
... others ... (假設間隔100ms)  
diff = GetNowStamp(value) // diff = 100  
... others ... (假設間隔200ms)  
diff = GetNowStamp(value) // diff = -1 // 數值超過 2147483647
```

語法 4

```
double GetNowStamp()  
    double  
)
```

參數

double 輸入前次記錄的總毫秒時間

傳回值

double 傳回當前總毫秒時間與輸入前次記錄的總毫秒時間差，毫秒單位

時間差 = 當前總毫秒 - 輸入的總毫秒

< 0 輸入前次記錄小於零或大於當前總毫秒時間，表示為非法的毫秒時間差

說明

```
value = GetNowStamp() // value = 2147483546  
... others ... (假設間隔100ms)  
diff = GetNowStamp(value) // diff = 100  
... others ... (假設間隔200ms)  
diff = GetNowStamp(value) // diff = 200
```

語法 5

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

參數

int 輸入前次記錄的總毫秒時間
int 毫秒時間差設定值

傳回值

int 傳回是否大於等於毫秒時間差設定值
true (當前總毫秒 - 輸入的總毫秒) >= 毫秒時間差設定值
或 時間差小於零或是溢位
false (當前總毫秒 - 輸入的總毫秒) < 毫秒時間差設定值

說明

```
value = GetNowStamp()           // value = 41730494  
... others ... (假設間隔60ms)  
flag = GetNowStamp(value, 100)   // diff = 60      // flag = false  
... others ... (假設間隔60ms)  
flag = GetNowStamp(value, 100)   // diff = 120     // flag = true
```

語法 6

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

參數

double 輸入前次記錄的總毫秒時間
double 毫秒時間差設定值

傳回值

int 傳回是否大於等於毫秒時間差設定值
true (當前總毫秒 - 輸入的總毫秒) >= 毫秒時間差設定值
或 時間差小於零或是溢位
false (當前總毫秒 - 輸入的總毫秒) < 毫秒時間差設定值

說明

```
value = GetNowStamp()           // value = 41730494  
... others ... (假設間隔60ms)  
flag = GetNowStamp(value, 100)   // diff = 60      // flag = false  
... others ... (假設間隔60ms)  
flag = GetNowStamp(value, 100)   // diff = 120     // flag = true
```

2.39 GetVarValue()

取出變數值，可利用字串組合出變數名稱，藉以來取出組合後的變數值

語法 1

```
? GetVarValue(  
    string  
)
```

參數

string 變數名稱

傳回值

? 傳回變數值，傳回的型別依變數名稱的型別定義
如果變數不存在，將會報錯

說明

```
string var_s1 = "Hello World"  
string var_s2 = "Hi TM Robot"  
string var_h = "var_s1"  
string var_t = "var_s"  
  
string var_re = var_t          // var_re = "var_s"  
var_re = var_t + "1"          // var_re = "var_s" + "1" = "var_s1"  
var_re = GetVarValue("var_h") // var_re = "var_s1"  
var_re = GetVarValue(var_h)   // var_re = "Hello Wrold" // var_h = "var_s1" // 取出 var_s1 值  
var_re = GetVarValue(var_t + "1") // var_re = "Hello World" // var_b + "1" = "var_s1" // 取出 var_s1 值  
var_re = GetVarValue(var_t + "2") // var_re = "Hi TM Robot" // var_b + "2" = "var_s2" // 取出 var_s2 值  
var_re = GetVarValue(var_t)     // 報錯 // var_t = "var_s" // 取出 var_s 值，但變數不存在
```

2.40 Length()

取得型別長度、字串長度，或是陣列大小(陣列內的元素個數)

語法 1

```
int Length()  
?  
)
```

參數

? 輸入的原始值，可以是整數、浮點數、布林值、字串值或是陣列型別

傳回值

int	傳回長度
若是整數、浮點數、布林值，傳回型別長度	
若是字串，傳回字串長度	
若是陣列，傳回陣列元素個數	

說明

? 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 // 數值常數會先判斷是否可存進較小的型別
? bool n = true	
value = Length(n)	// value = 1
value = Length(false)	// value = 1

```

? string n = "A""BC"
value = Length(n)          // value = 4      // 字串值為 A"BC 兩個雙引號表示為 " 符號
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

```

2.41 Ctrl()

將整數值、字串值轉換成具有控制字元的字串

語法 1

```
string Ctrl(  
    int  
)
```

參數

`int` 輸入的整數值，依 Big Endian 方式轉換，最多轉換 4 個字元，但遇 0x00 不轉換。

傳回值

`string` 傳回轉換後的字串(具有控制字元)

說明

```
b = 0x0D0A  
value = Ctrl(b)           // value = \r\n  
value = Ctrl(0x0D0A)     // value = \r\n  
value = Ctrl(0xD000A09)   // value = \r\n\t      // 0x00 不轉換  
value = Ctrl(0xD300A09)   // value = \r0\n\t      // 0x30 轉換為 0  
value = Ctrl(0x00)         // value = ""        // 為空字串，而不是 NULL 字元，若需要改使用 Ctrl("\0")
```

語法 2

```
string Ctrl(  
    string  
)
```

參數

`string` 輸入的字串值，下列字串會依控制字元方式轉換，其他則維持原字串

\0	0x00 空字元(結束字元)
\a	0x07 警示字元
\b	0x08 倒退字元
\t	0x09 水平TAB字元
\r	0x0D 歸位字元
\v	0x0B 垂直TAB字元
\f	0x0C 換頁字元
\n	0x0A 換行字元

傳回值

`string` 傳回轉換後的字串(具有控制字元)

說明

```
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字元
```

語法 3

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

參數

byte[] 輸入的 byte 陣列，依陣列元素 0 開始向後轉換，直到陣列元素結束 (包含 0x00 也會轉換)

傳回值

string 傳回轉換後的字串(具有控制字元)

說明

```
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 = ""
```

2.42 XOR8()

使用 XOR 8 bits 演算法，計算校驗值。

語法 1

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

參數

byte[] 輸入的 byte 陣列

int 要計算的起始索引
0..陣列長度-1 合法值
<0 非法使用，傳回未計算時的初始值 0
>=陣列長度 非法使用，傳回未計算時的初始值 0

int 要計算的個數
如果個數 <0，則取到陣列結束
如果起始加個數大於陣列長度，則取到陣列結束

傳回值

byte 傳回計算後的校驗值

說明

```
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
```

語法 2

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

說明

與語法 1 相同，預設將要計算的個數取到陣列結束

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

語法 3

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

說明

與語法 1 相同，預設將起始索引設為 0 及要計算的個數取到陣列結束

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, 0x6F}
```

2.43 SUM8()

使用 SUM 8 bits 演算法，計算校驗值。

語法 1

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

參數

`byte[]` 輸入的 byte 陣列

`int` 要計算的起始索引
0..陣列長度-1 合法值
<0 非法使用，傳回未計算時的初始值 0
`>=陣列長度` 非法使用，傳回未計算時的初始值 0

`int` 要計算的個數
如果個數 <0，則取到陣列結束
如果起始加個數大於陣列長度，則取到陣列結束

傳回值

`byte` 傳回計算後的校驗值

說明

```
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
```

語法 2

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

說明

與語法 1 相同，預設將要計算的個數取到陣列結束

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

語法 3

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

說明

與語法 1 相同，預設將起始索引設為 0 及要計算的個數取到陣列結束

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}
```

2.44 SUM16()

使用 SUM 16 bits 演算法，計算校驗值。

語法 1

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

參數

byte[] 輸入的 byte 陣列
int 要計算的起始索引
0..陣列長度-1 合法值
<0 非法使用，傳回未計算時的初始值 0
>=陣列長度 非法使用，傳回未計算時的初始值 0
int 要計算的個數
如果個數 <0，則取到陣列結束
如果起始加個數大於陣列長度，則取到陣列結束

傳回值

byte[] 傳回計算後的校驗值，長度為 16bits 2 bytes (校驗值陣列是依 Big Endian 回傳)

說明

```
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}
```

語法 2

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

說明

與語法 1 相同，預設將要計算的個數取到陣列結束

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

語法 3

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

說明

與語法 1 相同，預設將起始索引設為 0 及要計算的個數取到陣列結束

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, 0x04, 0x6D}
```

2.45 SUM32()

使用 SUM 32 bits 演算法，計算校驗值。

語法 1

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

參數

byte[] 輸入的 byte 陣列
int 要計算的起始索引
0..陣列長度-1 合法值
<0 非法使用，傳回未計算時的初始值 0
>=陣列長度 非法使用，傳回未計算時的初始值 0
int 要計算的個數
如果個數 <0，則取到陣列結束
如果起始加個數大於陣列長度，則取到陣列結束

傳回值

byte[] 傳回計算後的校驗值，長度為 32bits 4 bytes (校驗值陣列是依 Big Endian 回傳)

說明

```
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}
```

語法 2

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

說明

與語法 1 相同，預設將要計算的個數取到陣列結束

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

語法 3

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

說明

與語法 1 相同，預設將起始索引設為 0 及要計算的個數取到陣列結束

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, 0x00, 0x00, 0x04, 0x6D}
```

2.46 CRC16()

使用 CRC 16 bits 演算法，計算校驗值。

語法 1

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

參數

int CRC16 演算法 (參考 <https://www.lammertbies.nl/comm/info/crc-calculation.html>)

0	CRC16	// 初始值 0x0000	// Polynomial 0xA001
1	CRC16 (Modbus)	// 初始值 0xFFFF	// Polynomial 0xA001
2	CRC16 (Sick)	// 初始值 0x0000	// Polynomial 0x8005
3	CRC16-CCITT (0x1D0F)	// 初始值 0x1D0F	// Polynomial 0x1021
4	CRC16-CCITT (0xFFFF)	// 初始值 0xFFFF	// Polynomial 0x1021
5	CRC16-CCITT (XModem)	// 初始值 0x0000	// Polynomial 0x1021
6	CRC16-CCITT (Kermit)	// 初始值 0x0000	// Polynomial 0x8408
7	CRC16 Schunk Gripper	// 初始值 0xFFFF	// Polynomial 0x1021

byte[] 輸入的 byte 陣列

int 要計算的起始索引

0..陣列長度-1 合法值

<0 非法使用，傳回未計算時的初始值

>=陣列長度 非法使用，傳回未計算時的初始值

int 要計算的個數

如果個數 <0，則取到陣列結束

如果起始加個數大於陣列長度，則取到陣列結束

傳回值

byte[] 傳回計算後的校驗值，長度為 16bits 2 bytes (校驗值陣列是依 Big Endian 回傳)

說明

```
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}
```

語法 2

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

說明

與語法 1 相同，預設將要計算的個數取到陣列結束

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

語法 3

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

說明

與語法 1 相同，預設將起始索引設為 0 及要計算的個數取到陣列結束

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, 0x2D, 0xD4}
```

語法 4

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

說明

與語法 1 相同，預設將 CRC16 演算法設為 0 CRC16

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

語法 5

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

說明

與語法 1 相同，預設將 CRC16 演算法設為 0 CRC16 及要計算的個數取到陣列結束

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

語法 6

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

說明

與語法 1 相同，預設將 CRC16 演算法設為 0 CRC16 及起始索引設為 0 及要計算的個數取到陣列結束

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

2.47 CRC32()

使用 CRC 32 bits 演算法，計算校驗值。

語法 1

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

參數

byte[] 輸入的 byte 陣列
int 要計算的起始索引
0..陣列長度-1 合法值
<0 非法使用，傳回未計算時的初始值 0
>=陣列長度 非法使用，傳回未計算時的初始值 0
int 要計算的個數
如果個數 <0，則取到陣列結束
如果起始加個數大於陣列長度，則取到陣列結束

傳回值

byte[] 傳回計算後的校驗值，長度為 32bits 4 bytes (校驗值陣列是依 Big Endian 回傳)

說明

```
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}
```

語法 2

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

說明

與語法 1 相同，預設將要計算的個數取到陣列結束

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

語法 3

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

說明

與語法 1 相同，預設將起始索引設為 0 及要計算的個數取到陣列結束

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}
```

2.48 ListenPacket()

將字串內容封裝成 Listen 節點 (外部 Script 控制模式) 可支援的通訊格式

語法 1

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

參數

string 自定義標頭，若為空字串，則預設 TMSCT
string 資料內容 (即外部控制傳輸協定中的 Data 區段)

傳回值

string 封裝後的內容 (包含標頭，資料長度，校驗碼)

說明

```
string var_data1 = "1,var_i++"  
  
string var_data2 = "Hello World"  
  
value = ListenPacket("TMSCT", var_data1)      // $TMSCT,9,1,var_i++,*06\r\n  
value = ListenPacket("", var_data2)            // $TMSCT,11,Hello World,*51\r\n      // Error for TMSCT  
value = ListenPacket("", "2,Techman Robot")    // $TMSCT,15,2,Techman Robot,*57\r\n  
value = ListenPacket("TMSTA", var_data2)        // $TMSTA,11>Hello World,*53\r\n      // Error for TMSTA  
value = ListenPacket("TMSTA", "00")             // $TMSTA,2,00,*41\r\n
```

語法 2

```
string ListenPacket(  
    string  
)
```

參數

string 資料內容 (即外部控制傳輸協定中的 Data 區段) (預設使用 TMSCT 標頭)

傳回值

string 封裝後的內容 (包含標頭，資料長度，校驗碼)

說明

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

2.49 ListenSend()

發送 Listen 節點通訊格式 TMSTA 給當前連接進 Listen Server 的 client 裝置

語法 1

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

參數

string 目標 IP 過濾，如 "127.0.0.1"，則會發送給從 127.0.0.1 連進來的所有 client 裝置
int TMSTA SubCmd 編號，只提供自定義資料訊息發送 90..99
? 發送值，可以是整數、浮點數、布林值、字串值或是陣列型別
數值將會使用 Little Endian 方式轉換，字串值將會使用 UTF8 方式轉換

傳回值

int 傳回發送結果
0 發送成功
-1 錯誤，Listen Server 未啟動
-2 錯誤，SubCmd 必須 90..99 之間

語法 2

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

參數

int TMSTA SubCmd 編號，只提供自定義資料訊息發送 90..99
? 發送值，可以是整數、浮點數、布林值、字串值或是陣列型別
數值將會使用 Little Endian 方式轉換，字串值將會使用 UTF8 方式轉換

傳回值

int 傳回發送結果
0 發送成功
-1 錯誤，Listen Server 未啟動
-2 錯誤，SubCmd 必須 90..99 之間

說明

沒有目標 IP 過濾，將發送資料訊息給所有連進來的 client 裝置

說明

```
string ip = "127.0.0.1"

byte b = 100

value = ListenSend(ip, 10, b)

    // send 0x64 to ipfilter "127.0.0.1"      // value = -2      // SubCmd 必須在 90..99 之間

value = ListenSend(ip, 90, b)

    // send 0x64 to ipfilter "127.0.0.1"      // value = -1      // 假設 Listen Server 未啟動

value = ListenSend(ip, 90, b)

    // send 0x64 to ipfilter "127.0.0.1"      // value = 0      // 發送成功

    // IP 過濾 127.0.0.1，只會發送由此 IP 連進 Listen Server 的裝置

    // $TMSTA,4,90,d,*06          // 100 值為 0x64

value = ListenSend(ip, 90, 123456)

    // send 0x40 0xE2 0x01 0x00 to ipfilter "127.0.0.1"

    // $TMSTA,7,90,@♦,*C2        // 123456 值為 0x40 0xE2 0x01 0x00 (int, Little Endian)

value = ListenSend(90, "123.456")

    // send 0x31 0x32 0x33 0x2E 0x34 0x35 0x36

    // 沒有 IP 過濾，會發送給所有連進 Listen Server 的裝置

    // $TMSTA,10,90,123.456,*7E   // "123.456" 值為 0x31 0x32 0x33 0x2E 0x34 0x35 0x36 (string, UTF8)

byte[] bb = {100, 200}

value = ListenSend(90, bb)

    // send 0x64 0xC8

    // $TMSTA,5,90,d♦,*CF       // {100, 200} 值為 0x64 0xC8

string[] ss = {"T", "M", "達明機器人" }

value = ListenSend(90, ss)

    // send 0x54 0x4D 0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA

    // $TMSTA,20,90,TM達明機器人,*A1
```

2.50 VarSync()

將 Variables 變數物件，發送給 Techman 手臂管理系統 (TMMManager)

* 使用此函式時，須等待發送成功，或滿足最多發送次數後，才會繼續向下執行

語法 1

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

參數

int 最多發送次數
<= 0 發生錯誤後會持續重送
int 發送錯誤後，等待毫秒時間後再重送 (millisecond)
< 0 非法值，重送等待時間設為預設值 1000ms
? **string** 字串或 **string[]** 字串陣列，表示為要發送的變數物件名稱
可以輸入多個，若有不存在的變數，則該變數不會發送，其餘存在的變數仍會發送
* 須為變數的名稱，而非變數，如變數 var_i 須輸入 "var_i"
* 若填變數，會先取出變數值，再依變數的內容值發送對應的變數物件

傳回值

int 傳回發送次數
> 0 發送成功，傳回值表示第幾次發送，如 1 表示第 1 次發送時成功
0 發送失敗
-1 TMM 功能未開啟
-9 參數錯誤

說明

```
string var_s = "ABC"  
string var_s1 = "var_s"  
string[] var_ss = {"ABC", "var_s", "var_s1"}
```

```
value = VarSync(1, 1000, "var_s") // 發出變數 var_s 物件  
value = VarSync(2, 2000, var_s) // 發出變數 ABC 物件 (因 var_s 為取變數值，會取得"ABC")  
value = VarSync(3, 2000, var_ss) // 發出變數 ABC、var_s、var_s1 物件 (取出 var_ss 字串陣列值)  
value = VarSync(3, 2000, "var_ss") // 發出變數 var_ss 物件  
value = VarSync(4, 2000, "var_ss", "var_s1", "ABC") // 發出變數 var_ss、var_s1、ABC 物件
```

語法 2

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

說明

與語法 1 相同，預設將錯誤等待毫秒後發送設為 1000 ms

VarSync(int, ?) => **VarSync(int, 1000, ?)**

語法 3

```
int VarSync(  
    ?  
)
```

說明

與語法 1 相同，預設將最多發送次數設為 0 及錯誤等待毫秒後發送設為 1000ms

VarSync(?) => **VarSync(0, 1000, ?)**

3. 數學函式

3.1 abs()

傳回指定數字的絕對值

語法 1

```
int abs(  
    int  
)
```

參數

int 輸入數字，int 型別

傳回值

int 傳回輸入數字的絕對值，int 型別

說明

```
int i = 10
```

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

```
i = -10
```

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

語法 2

```
float abs(  
    float  
)
```

參數

float 輸入數字，float 型別

傳回值

float 傳回輸入數字的絕對值，float 型別

說明

```
float f = 10.1
```

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

```
f = -10.1
```

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

語法 3

```
double abs(  
    double  
)
```

參數

double 輸入數字，double 型別

傳回值

double 傳回輸入數字的絕對值，double 型別

說明

```
double d = 10.8
```

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

```
d = -10.8
```

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

3.2 pow()

傳回指定數字的指數運算值

語法 1

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

參數

int 輸入底數，int 型別
double 指數

傳回值

int 傳回底數與指數的指數運算值，int 型別

語法 2

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

參數

float 輸入底數，float 型別
double 指數

傳回值

float 傳回底數與指數的指數運算值，float 型別

語法 3

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

參數

double 輸入底數，double 型別
double 指數

傳回值

double 傳回底數與指數的指數運算值，double 型別

說明

```
? int b = 100  
value = pow(b, 2) // 10000  
value = pow(b, -2) // 0 // 0.0001, but int type  
value = pow(b, 0.1) // 1 // 1.5848931924611136, but int type  
value = pow(b, 2.1) // 15848 // 15848.931924611141, but int type  
value = pow(b, -2.1) // 0 // 0.000063095734448019293, but int type  
  
? float b = -100  
value = pow(b, 2) // 10000  
value = pow(b, -2) // 0.0001  
value = pow(b, 0.2) // Error // NaN  
value = pow(b, 2.2) // Error // NaN  
value = pow(b, -2.2) // Error // NaN  
  
? double b = 100  
value = pow(b, 2) // 10000  
value = pow(b, -2) // 0.0001  
value = pow(b, 0.31) // 4.16869383470335  
value = pow(b, 2.31) // 41686.9383470336  
value = pow(b, -2.31) // 0.0000239883291901949
```

3.3 sqrt()

傳回指定數字的平方根運算值

語法 1

```
float sqrt(  
    float  
)
```

參數

float 輸入數字，float 型別

傳回值

float 傳回輸入數字的平方根運算值，float 型別

語法 2

```
double sqrt(  
    double  
)
```

參數

double 輸入數字，double 型別

傳回值

double 傳回輸入數字的平方根運算值，double 型別

說明

```
value = sqrt(100)      // 10  
value = sqrt(100.1234) // 10.005  
value = sqrt(0.1234)   // 0.3162278  
value = sqrt(-100)     // Error // NaN  
value = sqrt(-100.1234) // Error // NaN  
value = sqrt(-0.1234)   // Error // NaN
```

3.4 ceil()

傳回大於等於指定數字的最小整數

語法 1

```
float ceil(  
    float  
)
```

參數

float 輸入數字，float 型別

傳回值

float 傳回大於等於輸入數字的最小整數，float 型別

語法 2

```
double ceil(  
    double  
)
```

參數

double 輸入數字，double 型別

傳回值

double 傳回大於等於輸入數字的最小整數，double 型別

說明

```
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
```

3.5 ***floor()***

傳回小於等於指定數字的最大整數

語法 1

```
float floor(  
    float  
)
```

參數

float 輸入數字，float 型別

傳回值

float 傳回小於等於輸入數字的最大整數，float 型別

語法 2

```
double floor(  
    double  
)
```

參數

double 輸入數字，double 型別

傳回值

double 傳回小於等於輸入數字的最大整數，double 型別

說明

```
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
```

3.6 round()

傳回四捨五入最接近的整數或是指定的小數位數

語法 1

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

參數

float	輸入數字，float 型別
int	傳回值中，小數點後的個數（預設 0 個，表示取至整數）
0 .. 15	合法值
< 0	非法值，採預設 0 個
> 15	非法值，採預設 0 個

傳回值

float 傳回輸入數字的四捨五入值，float 型別

語法 2

```
float round(  
    float  
)
```

說明

與語法 1 相同，預設將要取得小數點位數為 0 個

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

語法 3

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

參數

double	輸入數字，double 型別
int	傳回值中，小數點後的個數（預設 0 個，表示取至整數）
0 .. 15	合法值
< 0	非法值，採預設 0 個
> 15	非法值，採預設 0 個

傳回值

double 傳回輸入數字的四捨五入值，double 型別

語法 4

```
double round(  
    double  
)
```

說明

與語法 3 相同，預設將要取得小數點位數為 0 個

`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
```

3.7 random()

傳回亂數浮點數或整數

語法 1

```
float random(  
)
```

參數

void 無輸入值

傳回值

float 傳回亂數產生值，float 型別，數值為大於等於 0 且小於 1 之間

說明

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

語法 2

```
int random(  
    int  
)
```

參數

int 指定亂數數字的上限整數

傳回值

int 傳回亂數產生值，int 型別，數值為大於等於 0 且小於上限整數之間

說明

```
value = random(10)    // 8  
value = random(10)    // 1  
value = random(10)    // 5  
value = random(-10)   // 0 // 上限整數必須大於等於 0
```

語法 3

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

參數

- int 指定亂數數字的下限整數
- int 指定亂數數字的上限整數，必須大於等於下限整數，否則傳回下限整數

傳回值

- int 傳回亂數產生值，int 型別，數值為大於等於下限整數且小於上限整數之間

說明

```
value = random(5, 10)    // 8  
value = random(5, 10)    // 8  
value = random(5, 10)    // 6  
value = random(5, -1)    // 5 // 上限整數小於下限整數，傳回下限整數
```

3.8 d2r()

將角度單位轉換至弧度單位 (degree to radian)

語法 1

```
float d2r(  
    float  
)
```

參數

float 輸入角度 degree 單位數字，float 型別

傳回值

float 傳回弧度 radian 單位數字，float 型別

語法 2

```
double d2r(  
    double  
)
```

參數

double 輸入角度 degree 單位數字，double 型別

傳回值

double 傳回弧度 radian 單位數字，double 型別

說明

```
value = d2r(1) // 0.01745329
```

3.9 r2d()

將弧度單位轉換至角度單位 (radian to degree)

語法 1

```
float r2d(  
    float  
)
```

參數

float 輸入弧度 radian 單位數字，float 型別

傳回值

float 傳回角度 degree 單位數字，float 型別

語法 2

```
double r2d(  
    double  
)
```

參數

double 輸入弧度 radian 單位數字，double 型別

傳回值

double 傳回角度 degree 單位數字，double 型別

說明

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

3.10 sin()

傳回指定角的正弦函數 (角度單位)

語法 1

```
float sin(  
    float  
)
```

參數

float 輸入角度 degree 單位數字，float 型別

傳回值

float 傳回輸入角度的正弦函數值，float 型別

語法 2

```
double sin(  
    double  
)
```

參數

double 輸入角度 degree 單位數字，double 型別

傳回值

double 傳回輸入角度的正弦函數值，double 型別

說明

```
value = sin(0)      // 0  
value = sin(15)     // 0.258819  
value = sin(30)     // 0.5  
value = sin(60)     // 0.8660254  
value = sin(90)     // 1
```

3.11 cos()

傳回指定角的餘弦函數 (角度單位)

語法 1

```
float cos(  
    float  
)
```

參數

float 輸入角度 degree 單位數字，float 型別

傳回值

float 傳回輸入角度的餘弦函數值，float 型別

語法 2

```
double cos(  
    double  
)
```

參數

double 輸入角度 degree 單位數字，double 型別

傳回值

double 傳回輸入角度的餘弦函數值，double 型別

說明

value = cos(0)	// 1
value = cos(15)	// 0.9659258
value = cos(30)	// 0.8660254
value = cos(45)	// 0.7071068
value = cos(60)	// 0.5

3.12 tan()

傳回指定角的正切函數 (角度單位)

語法 1

```
float tan(  
    float  
)
```

參數

float 輸入角度 degree 單位數字，float 型別

傳回值

float 傳回輸入角度的正切函數值，float 型別

語法 2

```
double tan(  
    double  
)
```

參數

double 輸入角度 degree 單位數字，double 型別

傳回值

double 傳回輸入角度的正切函數值，double 型別

說明

```
value = tan(0)      // 0  
value = tan(15)     // 0.2679492  
value = tan(30)     // 0.5773503  
value = tan(45)     // 1  
value = tan(60)     // 1.732051
```

3.13 asin()

傳回正弦函數 (Sine) 的角度

語法 1

```
float asin(  
    float  
)
```

參數

float 輸入正弦函數值，float 型別，其值必須大於等於 -1 且小於等於 1

傳回值

float 傳回 degree 角度，float 型別

語法 2

```
double asin(  
    double  
)
```

參數

double 輸入正弦函數值，double 型別，其值必須大於等於 -1 且小於等於 1

傳回值

double 傳回 degree 角度，double 型別

說明

value = asin(0)	// 0
value = asin(0.258819)	// 15
value = asin(0.5)	// 30
value = asin(0.8660254)	// 60
value = asin(1)	// 90

3.14 acos()

傳回餘弦函數 (Cosine) 的角度

語法 1

```
float acos(  
    float  
)
```

參數

float 輸入餘弦函數值，float 型別，其值必須大於等於 -1 且小於等於 1

傳回值

float 傳回 degree 角度，float 型別

語法 2

```
double acos(  
    double  
)
```

參數

double 輸入餘弦函數值，double 型別，其值必須大於等於 -1 且小於等於 1

傳回值

double 傳回 degree 角度，double 型別

說明

```
value = acos(1)          // 0  
value = acos(0.9659258) // 15  
value = acos(0.8660254) // 30  
value = acos(0.7071068) // 45  
value = acos(0.5)        // 60
```

3.15 atan()

傳回正切函數 (Tangent) 的角度

語法 1

```
float atan(  
    float  
)
```

參數

float 輸入正切函數值，float 型別

傳回值

float 傳回 degree 角度，float 型別

語法 2

```
double atan(  
    double  
)
```

參數

double 輸入正切函數值，double 型別

傳回值

double 傳回 degree 角度，double 型別

說明

```
value = atan(0)          // 0  
value = atan(0.2679492) // 15  
value = atan(0.5773503) // 30  
value = atan(1)          // 45  
value = atan(1.732051)  // 60
```

3.16 atan2()

傳回正切函數是兩個指定數字之商數的角度

語法 1

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

參數

float 輸入某個點的 Y 座標，float 型別
float 輸入某個點的 X 座標，float 型別

傳回值

float 傳回 degree 角度，float 型別

語法 2

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

參數

double 輸入某個點的 Y 座標，double 型別
double 輸入某個點的 X 座標，double 型別

傳回值

double 傳回 degree 角度，double 型別

說明

```
value = atan2(2, 1)      // 63.43495  
value = atan2(1, 1)      // 45  
value = atan2(-1, -1)    // -135  
value = atan2(4, -3)     // 126.8699
```

3.17 log()

傳回指定數字的對數值

語法 1

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

參數

float 輸入的數字，float 型別

double 對數的底數

傳回值

float 傳回輸入數字以底數計算的對數值，float 型別

語法 2

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

參數

double 輸入的數字，double 型別

double 對數的底數

傳回值

double 傳回輸入數字以底數計算的對數值，double 型別

說明

```
value = log(16, 2)      // 4  
value = log(16, 8)      // 1.333333  
value = log(16, 10)     // 1.20412  
value = log(16, 16)     // 1
```

語法 3

```
float log(  
    float  
)
```

參數

float 輸入的數字，float 型別

傳回值

float 傳回輸入數字的自然對數值 (底數為 e)，float 型別

語法 4

```
double log(  
    double  
)
```

參數

double 輸入的數字，double 型別

傳回值

double 傳回輸入數字的自然對數值 (底數為 e)，double 型別

說明

```
value = log(16, 2)      // 4  
value = log(16)         // 2.772589  
value = log(2)          // 0.6931472  
value = log(16)/log(2)  // 2.772589 / 0.6931472 = 4.000000288539
```

3.18 log10()

傳回指定數字以 10 為底數的對數值

語法 1

```
float log10(  
    float  
)
```

參數

float 輸入的數字，float 型別

傳回值

float 傳回輸入數字以 10 為底數計算的對數值，float 型別

語法 2

```
double log10(  
    double  
)
```

參數

double 輸入的數字，double 型別

傳回值

double 傳回輸入數字以 10 為底數計算的對數值，float 型別

說明

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

3.19 norm2()

傳回指定向量的第二個範數

語法 1

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

參數

float[] 預備要傳回第二個範數（或稱歐幾里德範數，向量大小）之向量

傳回值

float 向量的第二個範數（或稱歐幾里德範數，向量大小）

說明

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

```
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)      // 12.24745
```

3.20 dist()

傳回兩座標之間距離

語法 1

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

參數

float[]	第一組座標	$\{X_1(mm) \quad Y_1(mm) \quad Z_1(mm) \quad RX_1(^{\circ}) \quad RY_1(^{\circ}) \quad RZ_1(^{\circ})\}$
float[]	第二組座標	$\{X_2(mm) \quad Y_2(mm) \quad Z_2(mm) \quad RX_2(^{\circ}) \quad RY_2(^{\circ}) \quad RZ_2(^{\circ})\}$

傳回值

float 第一組座標與第二組座標之間距離

說明

```
float[] c1 = {100,200,100,30,50,20}
```

```
float[] c2 = {100,100,100,50,50,10}
```

```
value = dist(c1, c2) // 100
```

3.21 trans()

傳回從一特定點位到另一點位的位移與旋轉角度

語法 1

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

參數

float[]	第一點位	$\{X_1(mm) \quad Y_1(mm) \quad Z_1(mm) \quad RX_1(^{\circ}) \quad RY_1(^{\circ}) \quad RZ_1(^{\circ})\}$
float[]	第二點位	$\{X_2(mm) \quad Y_2(mm) \quad Z_2(mm) \quad RX_2(^{\circ}) \quad RY_2(^{\circ}) \quad RZ_2(^{\circ})\}$
bool	參考座標系	
	false	參考機器人座標系(預設)
	true	參考第一點位

傳回值

float[] 從第一點位到第二點位的位移與旋轉角度
 $\{X_{trans} \quad Y_{trans} \quad Z_{trans} \quad RX_{trans} \quad RY_{trans} \quad RZ_{trans}\}$
如果無法計算，則傳回空陣列

語法 2

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

說明

與語法 1 相同，預設將參考座標系填入 false 參考機器人座標系

$$\text{Original Transformation Matrix} = \begin{bmatrix} R & P \\ 0 & 1 \end{bmatrix}$$

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

$$\text{第一點位} = \begin{bmatrix} R_1 & P_1 \\ 0 & 1 \end{bmatrix}$$

$$\text{第二點位} = \begin{bmatrix} R_2 & P_2 \\ 0 & 1 \end{bmatrix}$$

If reference coordinate is `false` (reference coordinate is Robot Base)

$$P_{trans} = P_2 - P_1$$

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

If reference coordinate is `true` (reference coordinate is First Point)

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

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

```
float[] var_P1 = {100, -200, 300, 10, 20, 60}
```

```
float[] var_P2 = {-400, 200, 50, -20, 30, -45}
```

```
float[] var_trans_RB = trans(var_P1, var_P2)
```

```
// {-500, 400, -250, -24.61587, -15.56518, -88.61369}
```

```
float[] var_trans_i = trans(var_P1, var_P2, true)
```

```
// {176.101588.3278,-308.8024,3.745925,23.13792,-92.46916}
```

3.22 `inversetrans()`

傳回與輸入之位移及旋轉角度 $\{x, y, z, rx, ry, rz\}$ 相反的位移與旋轉角度 $\{x, y, z, rx, ry, rz\}$

語法 1

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

參數

<code>float[]</code>	輸入之位移及旋轉角度 $\{X_i \ Y_i \ Z_i \ RX_i \ RY_i \ RZ_i\}$
<code>bool</code>	參考座標系
<code>false</code>	參考機器人座標系(預設)
<code>true</code>	參考輸入之位移及旋轉角度

傳回值

<code>float[]</code>	與輸入之位移及旋轉角度 $\{X_i \ Y_i \ Z_i \ RX_i \ RY_i \ RZ_i\}$ 相反的位移及旋轉角度 $\{X_{inv} \ Y_{inv} \ Z_{inv} \ RX_{inv} \ RY_{inv} \ RZ_{inv}\}$
	如果無法計算，則傳回空陣列

語法 2

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

說明

與語法 1 相同，預設將參考座標系填入 `false` 參考機器人座標系

$$\text{Original Transformation Matrix} = \begin{bmatrix} R & P \\ 0 & 1 \end{bmatrix}$$

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

$$\text{初始點位} = \begin{bmatrix} R_i & P_i \\ 0 & 1 \end{bmatrix}$$

If *reference coordinate* is `false` (reference coordinate is Robot Base)

$$P_{inv} = -P_i$$

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

If *reference coordinate* is `true` (reference coordinate is the input {x, y, z, rx, ry, rz}, which is equivalent to inverse of Transformation Matrix)

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

```
float[] var_P1 = {100, -200, 300, 10, 20, 60}

float[] var_inv_RB = inversetrans(var_P1)
// {-100, 200, -300, 12.48313, -18.59011, -60.28337}
float[] var_inv_i = inversetrans(var_P1, true)
// {218.381, 142.1322, -268.5297, 12.48313, -18.59011, -60.28337}
```

3.23 applytrans()

傳回套用位移及旋轉角度計算特定點位後的終點位

語法 1

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

參數

float[]	初始點位	$\{X_i \ Y_i \ Z_i \ RX_i \ RY_i \ RZ_i\}$
float[]	位移及旋轉角度	$\{X_{trans} \ Y_{trans} \ Z_{trans} \ RX_{trans} \ RY_{trans} \ RZ_{trans}\}$
bool	參考座標系	
	false	參考機器人座標系(預設)
	true	參考初始點位

傳回值

float[] 套用位移及旋轉角度計算後的終點位 $\{X_t \ Y_t \ Z_t \ RX_t \ RY_t \ RZ_t\}$
如果無法計算，則傳回空陣列

語法 2

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

說明

與語法 1 相同，預設將參考座標系填入 false 參考機器人座標系

$$\text{Original Transformation Matrix} = \begin{bmatrix} R & P \\ 0 & 1 \end{bmatrix}$$

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

$$\text{初始點位} = \begin{bmatrix} R_i & P_i \\ 0 & 1 \end{bmatrix}$$

If reference coordinate is false (reference coordinate is Robot Base)

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

$$R_t = R_{trans} * R_i$$

If *reference coordinate* is `true` (reference coordinate is Initial point)

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

$$R_t = R_i * R_{trans}$$

```
float[] var_P1 = {100, -200, 300, 10, 20, 60}
```

```
float[] var_P2 = {-400, 200, 50, -20, 30, -45}
```

```
float[] var_trans_RB = trans(var_P1, var_P2)
```

```
// {-500, 400, -250, -24.61587, -15.56518, -88.61369}
```

```
float[] var_trans_i = trans(var_P1, var_P2, true)
```

```
// {176.101588.3278,-308.8024,3.745925,23.13792,-92.46916}
```

```
float[] var_apply_RB = applytrans(var_P1, var_trans_RB)
```

```
// {-400, 200, 50, -20, 30, -45.00001}
```

```
float[] var_apply_i = applytrans(var_P1, var_trans_i, true)
```

```
// {-400, 200.0001, 49.99998, -20, 30, -45}
```

```
float[] var_inv_RB = inversetrans(var_P1)
```

```
// {-100, 200, -300, 12.48313, -18.59011, -60.28337}
```

```
float[] var_inv_i = inversetrans(var_P1, true)
```

```
// {218.381, 142.1322, -268.5297, 12.48313, -18.59011, -60.28337}
```

```
float[] var_apply_1 = applytrans(var_P1, var_inv_RB)
```

```
// {0, 0, 0, 1.488326E-06, 4.389056E-06, -4.243126E-06}
```

```
float[] var_apply_2 = applytrans(var_P1, var_inv_i, true)
```

```
// {-1.029038E-05, 7.456403E-05, 2.154642E-05, -4.266358E-06, 2.728826E-06, -3.719508E-06}
```

3.24 *interpoint()*

根據指定點位及比例傳回兩點位之間的插值點

語法 1

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

參數

float[]	第一點位	{X _{1(mm)} Y _{1(mm)} Z _{1(mm)} RX _{1(°)} RY _{1(°)} RZ _{1(°)} }
float[]	第二點位	{X _{2(mm)} Y _{2(mm)} Z _{2(mm)} RX _{2(°)} RY _{2(°)} RZ _{2(°)} }
float	比例	

傳回值

float[] 按照比例在第一點位與第二點位之間的線性插值點{X_i Y_i Z_i RX_i RY_i RZ_i}
如果無法計算，則傳回空陣列

說明

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

```
float[] var_P1 = {-388.3831,-199.8061,367.0702,177.4319,1.717448,-46.02005}  
float[] var_P2 = {-436.9584,115.7343,371.4378,179.4419,-42.86601,-96.91867}  
float[] var_interp = interpoint(var_P1, var_P2, 0.5)  
// {-412.6707,-42.0359,369.254,172.919,-20.6906,-69.3384}
```

3.25 changeref()

傳回由原點位座標值經座標系轉換後，以新座標系描述的新座標值。在座標轉換的過程中，被計算點位在世界座標中的物理位置並不會改變，僅改變描述此點位的參考座標以及其相對應的座標值。

語法 1

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

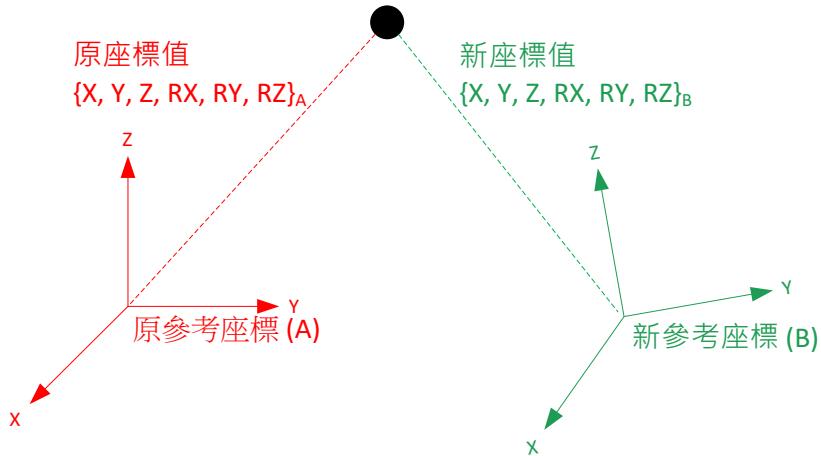
參數

float[] 原點位座標值 $\{X_o \ Y_o \ Z_o \ RX_o \ RY_o \ RZ_o\}_A$
float[] 原參考座標系值 $\{X_{oa} \ Y_{oa} \ Z_{oa} \ RX_{oa} \ RY_{oa} \ RZ_{oa}\}_A$
float[] 新參考座標系值 $\{X_n \ Y_n \ Z_n \ RX_n \ RY_n \ RZ_n\}_B$

傳回值

float[] 新點位座標值 $\{X_{nb} \ Y_{nb} \ Z_{nb} \ RX_{nb} \ RY_{nb} \ RZ_{nb}\}_B$
如果無法計算，則傳回空陣列

說明



```
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 = {0.002052, 0.000020, -0.002272, 113.9423, 14.9346, -123.1989}  
// 將 "P1" 點位座標系"RobotBase"值轉換至新點位座標系"base1"值
```

語法 2

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

參數

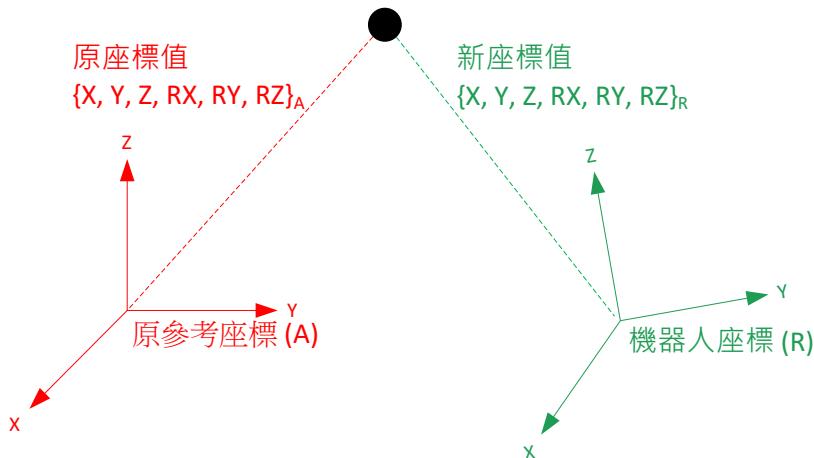
float[] 原點位座標值 $\{X_o \ Y_o \ Z_o \ RX_o \ RY_o \ RZ_o\}_A$
float[] 原參考座標系值 $\{X_{oa} \ Y_{oa} \ Z_{oa} \ RX_{oa} \ RY_{oa} \ RZ_{oa}\}_A$

傳回值

float[] 新點位座標值 $\{X_{nr} \ Y_{nr} \ Z_{nr} \ RX_{nr} \ RY_{nr} \ RZ_{nr}\}_R$
如果無法計算，則傳回空陣列

說明

與語法 1 相同，但新參考座標系值使用預設值 - 機器人座標系 $\{0 \ 0 \ 0 \ 0 \ 0 \ 0\}_R$ 。



```
base1 = {-431.93, -140.61, 368.73, -57.70, -44.98, 33.62}  
f0 = {0.002052, 0.000020, -0.002272, 113.9423, 14.9346, -123.1989}  
float[] f1 = changeref(f0, Base["base1"].Value)  
// f1 = {-431.927, -140.6103, 368.7306, -179.288, -0.6893424, -105.8449}
```

3.26 points2coord()

根據輸入的點位，計算位於輸入點位上的座標平面，傳回由平面上三點所轉換成的平面參數值

語法 1

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

參數

float[] 平面原點 $X_{1(mm)}$ $Y_{1(mm)}$ $Z_{1(mm)}$ $RX_{1(^{\circ})}$ $RY_{1(^{\circ})}$ $RZ_{1(^{\circ})}$
float[] 平面X軸上任一點 $X_{2(mm)}$ $Y_{2(mm)}$ $Z_{2(mm)}$ $RX_{2(^{\circ})}$ $RY_{2(^{\circ})}$ $RZ_{2(^{\circ})}$
float[] +X, +Y平面上任一點 $X_{3(mm)}$ $Y_{3(mm)}$ $Z_{3(mm)}$ $RX_{3(^{\circ})}$ $RY_{3(^{\circ})}$ $RZ_{3(^{\circ})}$

*以上輸入點位需要描述在相同座標系。

傳回值

float[] 由平面上三點所構成的平面參數值
 $X_{p(mm)}$ $Y_{p(mm)}$ $Z_{p(mm)}$ $RX_{p(^{\circ})}$ $RY_{p(^{\circ})}$ $RZ_{p(^{\circ})}$
，此平面原點會與
 $X_{1(mm)}$ $Y_{1(mm)}$ $Z_{1(mm)}$
重合，轉角則由
 $X_{2(mm)}$ $Y_{2(mm)}$ $Z_{2(mm)}$ $RX_{2(^{\circ})}$ $RY_{2(^{\circ})}$ $RZ_{2(^{\circ})}$
與
 $X_{3(mm)}$ $Y_{3(mm)}$ $Z_{3(mm)}$ $RX_{3(^{\circ})}$ $RY_{3(^{\circ})}$ $RZ_{3(^{\circ})}$
計算而來。

說明

假設有 P1、P2、P3 三個點

```
Point["P1"].Value = {389.9641,-4.797323,416.2175,177.3384,0.9190881,91.07789}
```

```
Point["P2"].Value = {365.0222,137.3036,423.2249,177.4598,0.9549707,112.4033}
```

```
Point["P3"].Value = {546.7307,94.02614,385.1812,176.3468,0.5975906,95.82078}
```

```
Base["points2coord_b1"].Value = points2coord(Point["P1"].Value, Point["P2"].Value, Point["P3"].Value)  
// {389.9641,-4.7973,416.2175,-168.6551,-2.7807,99.9553}  
Point["P4"].Value = {0,0,0,0,0,0}  
ChangeBase("points2coord_b1")  
PTP("CPP",Point["P4"].Value,10,200,0,false)
```

語法 2

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

參數

float, float, float 欲計算之座標平面的原點座標 $X_{1(mm)}$ $Y_{1(mm)}$ $Z_{1(mm)}$
float, float, float 欲計算之座標平面X軸上任意一點的座標 $X_{2(mm)}$ $Y_{2(mm)}$ $Z_{2(mm)}$
float, float, float 欲計算之座標平面上任意一點的座標 $X_{3(mm)}$ $Y_{3(mm)}$ $Z_{3(mm)}$

*以上輸入點位需要描述在相同座標系。

傳回值

float[] 欲計算之座標平面的定義參數 $X_p(mm)$ $Y_p(mm)$ $Z_p(mm)$ $RX_{p(^{\circ})}$ $RY_{p(^{\circ})}$ $RZ_{p(^{\circ})}$

說明

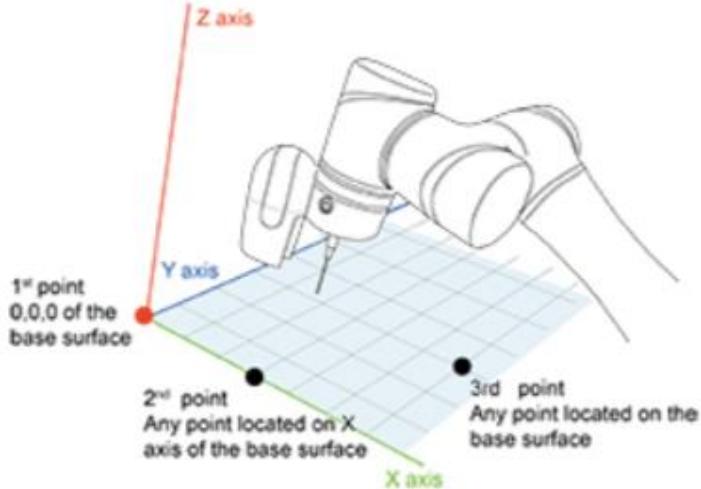
```
Base["points2coord_2"].Value = points2coord(260,0,360,260,100,360,100,0,360)
```

```
// {260,0,360,0,0,90}
```

```
Point["P1"].Value = {0,0,0,180,0,0}
```

```
ChangeBase("points2coord_2")
```

```
PTP("CPP",Point["P1"].Value,10,200,0,false)
```



3.27 intercoord()

將兩輸入平面合成轉換為新平面

語法 1

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

參數

float[] 第一平面 $X_{1(mm)}$ $Y_{1(mm)}$ $Z_{1(mm)}$ $RX_{1(^{\circ})}$ $RY_{1(^{\circ})}$ $RZ_{1(^{\circ})}$

float[] 第二平面 $X_{2(mm)}$ $Y_{2(mm)}$ $Z_{2(mm)}$ $RX_{2(^{\circ})}$ $RY_{2(^{\circ})}$ $RZ_{2(^{\circ})}$

*以上輸入點位需要描述在相同座標系。

傳回值

float[] 由兩輸入平面構成的新平面 $X_{3(mm)}$ $Y_{3(mm)}$ $Z_{3(mm)}$ $RX_{3(^{\circ})}$ $RY_{3(^{\circ})}$ $RZ_{3(^{\circ})}$

說明

$$X_3 = (X_1 + X_2)/2$$

$$Y_3 = (Y_1 + Y_2)/2$$

$$Z_3 = (Z_1 + Z_2)/2$$

$$\hat{t}_3 = (X_2 - X_1, Y_2 - Y_1, Z_2 - Z_1)$$

$$\hat{j}_3 = (\hat{k}_1 \times \hat{t}_3)$$

$$\hat{k}_3 = (\hat{t}_3 \times \hat{j}_3)$$

$$x_3 = (x_1 + x_2)/2$$

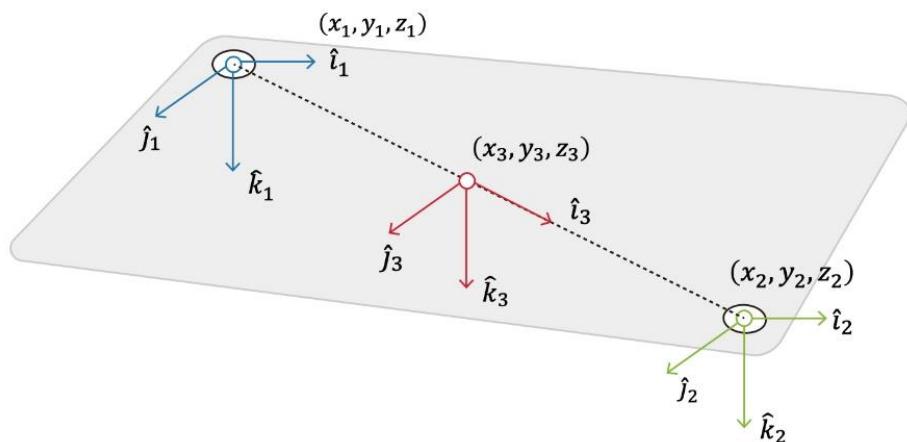
$$y_3 = (y_1 + y_2)/2$$

$$z_3 = (z_1 + z_2)/2$$

$$\hat{t}_3 = (x_2 - x_1, y_2 - y_1, z_2 - z_1)$$

$$\hat{j}_3 = \hat{k}_1 \times \hat{t}_3$$

$$\hat{k}_3 = \hat{t}_3 \times \hat{j}_3$$



假設有視覺 Landmark 任務 vb_1 及 vb_2

Expression Editor and Listen Node 軟體版本 : TMflow 1.82 文件版本 : 1.00

```
vb_1 輸出 Base["vision_vb_1"].Value = {-69.73,380.02,141.79,-176.11,1.13,-121.27}
vb_2 輸出 Base["vision_vb_2"].Value = {58.81,613.03,140.7,171.62,-0.89,0.8}

Base["mix_base"].Value = intercoord(Base["vision_vb_1"].Value, Base["vision_vb_2"].Value)
// {-5.46,496.525,141.245,176.0655,0.2347,61.1167}
```

4. 檔案函式

- 檔案函式可用來進行檔案相關的讀寫或查詢操作
- 檔案路徑

1. 本機路徑，只支援在 TextFiles 或 XmlFiles 目錄下使用

FileName.txt	預設目錄為 .\TextFiles (檔案路徑與 .\TextFiles\FileName.txt 相同)
.\TextFiles\FileName.txt	本機 TextFiles 目錄下的檔案
.\XmlFiles\FileName.xml	本機 XmlFiles 目錄下的檔案
.\XmlFiles\folder\file	本機 XmlFiles 目錄下的子目錄
..\folder	無法使用
.\TextFiles\..\..\folder	無法使用

不支援絕對路徑

C:\file1	無法使用
D:\folder\file2	無法使用
\TextFiles\FileName.txt	無法使用

2. 外部裝置路徑，只支援使用 TMROBOT 標籤的 USB 或 SSD

\USB\TMROBOT	外接 USB 根目錄
--------------	------------

3. 遠端路徑，可配合 TMflow 中的 Network Service 功能

\127.0.0.1\shared	網路芳鄰
ftp://127.0.0.1	FTP

- 路徑不區分大小寫，如以下檔案路徑都是指向同一個檔案

.\TextFiles\FileName.txt
.\textfiles\fileName.txt
.\Textfiles\Filename.TXT

- 路徑可支援子目錄使用，如

subfolder\file
.\TextFiles\subfolder1\subfolder2\file
.\XmlFiles\subfolder\file
\USB\TMROBOT\subfolder\file
\127.0.0.1\shared\subfolder\file

- 檔案大小，限制最大容量為 2MB (2097152 Bytes)

- 讀取或寫入的陣列型別，仍需依陣列所定義

4.1 File_ReadBytes()

讀取檔案內容，傳回 byte[] 型別

語法 1

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

參數

string 檔案路徑

傳回值

byte[] 傳回檔案內容，byte[] 型別

說明

```
. \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") // 報錯，不支援絕對路徑  
byte[] var_bb4 = File_ReadBytes("\SampleFile1.txt") // 報錯，路徑不在 TextFiles 或 XmlFiles 目錄下  
byte[] var_bb5 = File_ReadBytes("SampleFileXX.txt") // 報錯，檔案不存在
```

4.2 File_ReadText()

讀取檔案內容，傳回 string 型別

語法 1

```
string File_ReadText(  
    string  
)
```

參數

string 檔案路徑

傳回值

string 傳回檔案內容，字串型別

說明

```
.\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 為書寫換行符號表示，並非字串值
```

```
string var_s3 = File_ReadText("C:\SampleFile1.txt") // 報錯，不支援絕對路徑  
string var_s4 = File_ReadText(".\SampleFile1.txt")  // 報錯，路徑不在 TextFiles 或 XmlFiles 目錄下
```

4.3 File_ReadLines()

讀取檔案內容，並依照換行符號切割，傳回 string[] 陣列型別

語法 1

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

參數

string 檔案路徑

傳回值

string[] 傳回檔案內容，並依照換行符號切割成行列字串陣列

語法 2

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

參數

string 檔案路徑

int 要取出的起始行 (1 .. N)

int 要取出的行數

傳回值

string[] 傳回檔案內容，並依照換行符號切割成行列字串陣列
如果起始行 <= 0，則傳回空陣列
如果起始行 > 總行數，則傳回空陣列
如果取出的行數 <= 0，則從起始行取至最後行
如果起始行加取出的行數大於總行數，則從起始行取至最後行

語法 3

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

說明

與語法 2 相同，預設將要取出的行數參數填入 0，取至最後行

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

說明

```
. \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!"} // 大於總行數，取至最後行

var_ss = File_ReadLines("SampleFile2.txt", 0)    // {}      // 空陣列

var_ss = File_ReadLines("SampleFile2.txt", 4)    // {}      // 空陣列

int var_len = Length(var_ss)                  // 0

var_ss = File_ReadLines("SampleFile2.txt", 3, 0) // {"2Hi TM Robot!"} // 從第 3 行取至最後行

.

.\TextFiles\SampleFile3.txt
1| 

var_ss = File_ReadLines("SampleFile3.txt")      // {""}      // var_ss[0] = ""

var_len = Length(var_ss)                      // 1
```

4.4 File_NextLine()

記錄最後讀取的檔案路徑，並接續讀取檔案內容的下一行，或是重新開檔讀取

語法 1

```
string File_NextLine(  
    string  
)
```

參數

string 檔案路徑

傳回值

string 如果與最後讀取路徑相同，則傳回檔案內容的下一行字串
如果與最後讀取路徑不同，則重新開檔讀取，並傳回檔案內容的第一行字串
若讀取至檔案結尾，則傳回空字串

語法 2

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

參數

string 檔案路徑

bool 是否重新開檔讀取

false 判斷路徑，若相同則接續讀取下一行；若不同則重新開檔讀取
true 重新開檔，並讀取第一行

傳回值

string 是否重新開檔讀取 false

如果與最後讀取路徑相同，則傳回檔案內容的下一行字串
如果與最後讀取路徑不同，則重新開檔讀取，並傳回檔案內容的第一行字串
是否重新開檔讀取 true
重新開檔讀取，並傳回檔案內容的第一行字串
若讀取至檔案結尾，則傳回空字串

語法 3

```
string File_NextLine(  
)
```

參數

void 無輸入值

傳回值

`string` 傳回最後記錄的檔案內容的下一行字串，若未讀取則傳回空字串

說明

```
.\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() // "" // 未開檔讀取  
  
var_s = File_NextLine("SampleFile4.txt") // "4Hello World!"  
  
var_s = File_NextLine("SampleFile4.txt") // "" // 接續讀取下一行  
  
var_s = File_NextLine("SampleFile5.txt") // "5Hello World!" // 路徑不同，重新開檔讀取  
  
var_s = File_NextLine("SampleFile4.txt") // "4Hello World!" // 路徑不同，重新開檔讀取  
  
var_s = File_NextLine("SampleFile4.txt") // "" // 接續讀取下一行  
  
var_s = File_NextLine("SampleFile4.txt") // "4Hello TM Robot!"  
  
var_s = File_NextLine("SampleFile4.txt") // "" // 接續讀取下一行 (讀完)  
  
var_s = File_NextLine("SampleFile4.txt", true) // "4Hello World!" // 重新開檔讀取第 1 行  
  
var_s = File_NextLine("SampleFile4.txt", true) // "4Hello World!" // 重新開檔讀取第 1 行  
  
var_s = File_NextLine("SampleFile4.txt", false) // "" // 接續讀取下一行  
  
var_s = File_NextLine() // "4Hello TM Robot!"
```

- * 若需要判斷讀取到空白行或是讀取至檔案結尾，需使用語法 4，由 `string[]` 的大小來判斷
- * 或是配合 `File_NextEOF()` 判斷是否讀取至檔案結尾

語法 4

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

參數

string	檔案路徑
int	讀取參數
0	判斷路徑，若相同則接續讀取下一行；若不同則重新開檔讀取
1	重新開檔，並讀取第 1 行
2	重新開檔，但不讀取，傳回空陣列

傳回值

string[]	傳回檔案內容的下一行字串陣列
	若陣列大小為 1，表示有讀取到下一行字串
	若陣列大小為 0，表示已讀取到檔案結尾

語法 5

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

參數

int	讀取參數
0	判斷路徑，若相同則接續讀取下一行；若不同則重新開檔讀取
1	重新開檔，並讀取第 1 行
2	重新開檔，但不讀取，傳回空陣列

傳回值

string[]	傳回最後記錄的檔案內容的下一行字串陣列，若未讀取則傳回空字串陣列
	若陣列大小為 1，表示有讀取到下一行字串
	若陣列大小為 0，表示已讀取到檔案結尾

說明

```
. \TextFiles\SampleFile4.txt          . \TextFiles\SampleFile5.txt
1| 4Hello World!                   1| 5Hello World!
2|                               2| 5Hello TM Robot!
3| 4Hello TM Robot!                3| 5Hi TM Robot!
```



```
string[] var_ss = {""
}
var_ss = File_NextLine(0)           // {}           // 未開檔讀取
var_ss = File_NextLine("SampleFile4.txt", 0) // {"4Hello World!"}
var_ss = File_NextLine("SampleFile4.txt", 0) // {""
}           // 接續讀取下一行
var_ss = File_NextLine("SampleFile5.txt", 0) // {"5Hello World!"} // 路徑不同，重新開檔讀取
var_ss = File_NextLine("SampleFile4.txt", 0) // {"4Hello World!"} // 路徑不同，重新開檔讀取
var_ss = File_NextLine("SampleFile4.txt", 0) // {""
}           // 接續讀取下一行
int var_len = Length(var_ss)        // 1
var_ss = File_NextLine("SampleFile4.txt", 0) // {"4Hello TM Robot!"}
var_ss = File_NextLine("SampleFile4.txt", 0) // {}           // 接續讀取下一行 (讀完)
var_len = Length(var_ss)            // 0
var_ss = File_NextLine("SampleFile4.txt", 1) // {"4Hello World!"} // 重新開檔讀取第 1 行
var_ss = File_NextLine("SampleFile4.txt", 2) // {}           // 重新開檔，但不讀取
var_len = Length(var_ss)            // 0
var_ss = File_NextLine("SampleFile4.txt") // {"4Hello World!"} // 接續讀取下一行
var_ss = File_NextLine(0)           // {""
}           // 接續讀取下一行
var_ss = File_NextLine(0)           // {"4Hello TM Robot!"}
var_ss = File_NextLine(0)           // {}
```

4.5 File_NextEOF()

判斷最後讀取的檔案路徑，是否已讀取至檔案結尾

語法 1

```
bool File_NextEOF(  
)
```

參數

void 無輸入值，但需配合 File_NextLine() 使用

傳回值

bool	若未讀取則傳回 true
false	未讀取至檔案結尾
true	未開檔，或已讀取至檔案結尾

說明

```
. \TextFiles\SampleFile4.txt  
1| 4Hello World!  
2|  
3| 4Hello TM Robot!  
  
bool var_eof = File_NextEOF()           // true    // 未開檔讀取  
  
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)      // 重新開檔，但不讀取  
var_eof = File_NextEOF()               // false
```

4.6 File_WriteBytes()

將資料內容轉為 byte[] 陣列後寫入檔案

語法 1

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

參數

string	檔案路徑
?	寫入資料值，可以是整數、浮點數、布林值、字串值或是陣列型別
int	數值將會使用 Little Endian 方式轉換，字串值將會使用 UTF8 方式轉換
int	覆寫檔案或附加檔案
0	覆寫檔案，若檔案不存在則建立新檔案
1	附加檔案，若檔案不存在則建立新檔案
int	寫入資料值的起始索引 (對字串值或陣列型別有效)
0.. 長度-1	合法值
< 0	非法值，起始索引將設為 0
=> 長度	非法值，起始索引將設為 0
int	寫入資料值的長度 (對字串值或陣列型別有效)
<= 0	從起始索引，寫至資料結束
> 0	從起始索引，寫入指定個數長度，最多寫至資料結束

傳回值

bool	true	寫入成功
	false	寫入失敗

語法 2

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

說明

與語法 1 相同，將寫入資料值的長度設為 0 寫至資料結束

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

語法 3

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

說明

與語法 1 相同，將寫入資料值的起始索引設為 0，且寫入資料值的長度設為 0 寫至資料結束

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

語法 4

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

說明

與語法 1 相同，使用覆寫檔案，將寫入資料值的起始索引設為 0，且寫入資料值的長度設為 0 寫至資料結束

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 var_flag = false  
  
var_flag = File_WriteBytes("writebytes.txt", var_bb1) // 將 var_bb1 覆寫進檔案  
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) // 將 var_bb2 覆寫進檔案  
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) // 將 var_bb3 覆寫進檔案  
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) // 將 var_bb1 附加進檔案
```

```
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) // 將 var_bb2 附加進檔案
```

```
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) // 將 var_bb1 起始索引 3 至結束，附加進檔案
```

```
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) // 將 var_bb2 起始索引 3 取 2 個，附加進檔案
```

```
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 非法值 // 將 var_bb1 起始索引 0 至結束，附加進檔案
```

```
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)
```

```
// 將 var_bb2 起始索引 0 取 100 個(或結束)，附加進檔案
```

```
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          // 數值採用 Little Endian 轉換

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} // 將陣列中的每筆數值，依序採用 Little Endian 轉換

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 00 00 00 00 00 00 00 00 00
00000010 38 01 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 00 10 27 00 00 00 20 4E 00 00
00000010 38 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 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 00 10 27 00 00 00 20 4E 00 00
00000010 38 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

? 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 00 10 27 00 00 00 20 4E 00 00
00000010 38 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

```

?    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
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

?    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           // true 會轉換為 1 ; false 會轉換為 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"    // string 使用 UTF8 轉換

File_WriteBytes("writebytes3.txt", var_n11, 1)

writebytes3.txt
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 8D 97 6E 12 83 C0 F3 BF AE 47 E1 7A 14 AE F3 3F
00000010 8F C2 F5 28 5C 8F 1F C0 01 01 00 01 00 00 01 01
00000020 41 42 43 44 45 46 47

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

File_WriteBytes("writebytes3.txt", var_n12, 1)

writebytes3.txt
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 8D 97 6E 12 83 C0 F3 BF AE 47 E1 7A 14 AE F3 3F
00000010 8F C2 F5 28 5C 8F 1F C0 01 01 00 01 00 00 01 01
00000020 41 42 43 44 45 46 47 41 42 43 44 45 46 E9 81 94
00000030 E6 98 8E E6 A9 9F E5 99 A8 E4 BA BA

```

4.7 File_WriteText()

將資料內容轉為 string 字串後寫入檔案

語法 1

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

參數

string	檔案路徑
?	寫入資料值，可以是整數、浮點數、布林值、字串值或是陣列型別
int	覆寫檔案或附加檔案
0	覆寫檔案，若檔案不存在則建立新檔案
1	附加檔案，若檔案不存在則建立新檔案
int	寫入資料值的起始索引 (對字串值或陣列型別有效)
0 .. 長度-1	合法值
< 0	非法值，起始索引將設為 0
>= 長度	非法值，起始索引將設為 0
int	寫入資料值的長度 (對字串值或陣列型別有效)
<= 0	從起始索引，寫至資料結束
> 0	從起始索引，寫入指定個數長度，最多寫至資料結束

傳回值

bool	true	寫入成功
	false	寫入失敗

語法 2

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

說明

與語法 1 相同，將寫入資料值的長度設為 0 寫至資料結束

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

語法 3

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

說明

與語法 1 相同，將寫入資料值的起始索引設為 0，且寫入資料值的長度設為 0 寫至資料結束

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

語法 4

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

說明

與語法 1 相同，使用覆寫檔案，將寫入資料值的起始索引設為 0，且寫入資料值的長度設為 0 寫至資料結束

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_WriteText("writetext.txt", var_s1) // 將 "Hi TM Robot" 覆寫進檔案
```

```
writetext.txt  
1| Hi TM Robot
```

```
var_flag = File_WriteText("writetext.txt", var_s2) // 將 "達明機器人" 覆寫進檔案
```

```
writetext.txt  
1| 達明機器人
```

```
var_flag = File_WriteText("writetext.txt", var_s1, 1) // 將 "Hi TM Robot" 附加進檔案
```

```
writetext.txt  
1| 達明機器人 Hi TM Robot
```

```
var_flag = File_WriteText("writetext.txt", var_s2, 1, 2, 3) // 起始索引 2 取 3 個字"機器人"附加進檔案
```

```
writetext.txt  
1| 達明機器人 Hi TM Robot 機器人
```

```

?   byte var_n1 = 100           // 數值採用 10 進制轉換成字串

File_WriteText("writetext2.txt", var_n1, 1)
    writetext2.txt
    1| 100

?   int[] var_n4 = {10000, 20000, 80000} // 陣列採用 "{,}" 格式

File_WriteText("writetext2.txt", var_n4, 1)
    writetext2.txt
    1| 100{10000,20000,80000}
// 若需要其他格式，可先用 GetString() 轉換成字串，再寫入字串

?   float var_n5 = 1.234

File_WriteText("writetext2.txt", var_n5, 1)
    writetext2.txt
    1| 100{10000,20000,80000}1.234

?   double[] var_n8 = {1.23, -7.89}

File_WriteText("writetext2.txt", var_n8, 1)
    writetext2.txt
    1| 100{10000,20000,80000}1.234{1.23,-7.89}

?   bool var_n9 = true

File_WriteText("writetext2.txt", var_n9, 1)
    writetext2.txt
    1| 100{10000,20000,80000}1.234{1.23,-7.89}true

?   string var_n11 = "ABCDEFG"

File_WriteText("writetext2.txt", var_n11, 1)
    writetext2.txt
    1| 100{10000,20000,80000}1.234{1.23,-7.89}trueABCDEFG

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

File_WriteText("writetext2.txt", var_n12, 1)
    writetext2.txt
    1| 100{10000,20000,80000}1.234{1.23,-7.89}trueABCDEFG{ABC,DEF,達明機器人}

```

4.8 File_WriteLine()

將資料內容轉為 string 字串，且在資料尾端加上換行符號 (0x0D 0x0A) 後，再寫入檔案

語法 1

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

參數

string	檔案路徑
?	寫入資料值，可以是整數、浮點數、布林值、字串值或是陣列型別
int	覆寫檔案或附加檔案
0	覆寫檔案，若檔案不存在則建立新檔案
1	附加檔案，若檔案不存在則建立新檔案
int	寫入資料值的起始索引 (對字串值或陣列型別有效)
0.. 長度-1	合法值
<0	非法值，起始索引將設為 0
>= 長度	非法值，起始索引將設為 0
int	寫入資料值的長度 (對字串值或陣列型別有效)
<=0	從起始索引，寫至資料結束
>0	從起始索引，寫入指定個數長度，最多寫至資料結束

傳回值

bool	true	寫入成功
	false	寫入失敗

語法 2

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

說明

與語法 1 相同，將寫入資料值的長度設為 0 寫至資料結束

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

語法 3

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

說明

與語法 1 相同，將寫入資料值的起始索引設為 0，且寫入資料值的長度設為 0 寫至資料結束

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

語法 4

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

說明

與語法 1 相同，使用覆寫檔案，將寫入資料值的起始索引設為 0，且寫入資料值的長度設為 0 寫

至資料結束 `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)      // 將 "達明機器人\u0DOA" 覆寫進檔案  
    writeline.txt  
    1| 達明機器人  
    2|  
  
var_flag = File_WriteLine("writeline.txt", var_s1)      // 將 "Hi TM Robot\u0DOA" 覆寫進檔案  
    writeline.txt  
    1| Hi TM Robot  
    2|  
  
var_flag = File_WriteLine("writeline.txt", var_s2, 1)  // 將 "達明機器人\u0DOA" 附加進檔案  
    writeline.txt  
    1| Hi TM Robot  
    2| 達明機器人  
    3|  
  
var_flag = File_WriteLine("writeline.txt", var_s1, 1, 3) // 起始索引 3 至結束"TM Robot\u0DOA"附加進檔案  
    writeline.txt  
    1| Hi TM Robot  
    2| 達明機器人  
    3| TM Robot  
    4|
```

```

? byte[] var_n2 = {100, 200}      // 陣列採用 "{,}" 格式

File_WriteLine("writeln2.txt", var_n2, 1)

writeln2.txt
1| {100,200}
2|
// 若需要其他格式，可先用 GetString() 轉換成字串，再寫入字串

? int var_n3 = 10000           // 數值採用 10 進制轉換成字串

File_WriteLine("writeln2.txt", var_n3, 1)

writeln2.txt
1| {100,200}
2| 10000
3|
? float[] var_n6 = {1.23, 4.56, -7.89}

File_WriteLine("writeln2.txt", var_n6, 1)

writeln2.txt
1| {100,200}
2| 10000
3| {1.23,4.56,-7.89}
4|
? bool var_n9 = true

File_WriteLine("writeln2.txt", var_n9, 1)

writeln2.txt
1| {100,200}
2| 10000
3| {1.23,4.56,-7.89}
4| true
5|
? string var_n11 = "ABCDEFG"

File_WriteLine("writeln2.txt", var_n11, 1)

writeln2.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("writeln2.txt", var_n12, 1)

writeln2.txt
1| {100,200}
2| 10000
3| {1.23,4.56,-7.89}
4| true
5| ABCDEFG
6| {ABC,DEF,達明機器人}
7|

```

4.9 File_WriteLines()

將資料內容轉為 string[] 字串，且在每筆陣列元素字串值的資料尾端加上換行符號 (0x0D 0x0A) 後，再寫入檔案

語法 1

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

參數

string	檔案路徑
?	寫入資料值，可以是整數、浮點數、布林值、字串值或是陣列型別
int	覆寫檔案或附加檔案
0	覆寫檔案，若檔案不存在則建立新檔案
1	附加檔案，若檔案不存在則建立新檔案
int	寫入資料值的起始索引 (對字串值或陣列型別有效)
0.. 長度-1	合法值
<0	非法值，起始索引將設為 0
>= 長度	非法值，起始索引將設為 0
int	寫入資料值的長度 (對字串值或陣列型別有效)
<=0	從起始索引，寫至資料結束
>0	從起始索引，寫入指定個數長度，最多寫至資料結束

傳回值

bool	true	寫入成功
	false	寫入失敗

語法 2

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

說明

與語法 1 相同，將寫入資料值的長度設為 0 寫至資料結束

File_WriteLines(string,?,int,int) => File_WriteLines(string,?,int,int,0)

語法 3

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

說明

與語法 1 相同，將寫入資料值的起始索引設為 0，且寫入資料值的長度設為 0 寫至資料結束

File_WriteLines(string,?,int) => File_WriteLines(string,?,int,0,0)

語法 4

```
bool File_WriteLines(  
    string,  
    ?  
)
```

說明

與語法 1 相同，使用覆寫檔案，將寫入資料值的起始索引設為 0，且寫入資料值的長度設為 0 寫至資料結束

File_WriteLines(string,?) => File_WriteLines(string,?,0,0,0)

- * **File_WriteText()** 是寫入資料值轉為字串值，不會在字串值尾端加上換行符號
- File_WriteLine()** 是寫入資料值轉為字串值，會在字串值尾端加上換行符號
- File_WriteLines()** 是寫入資料值轉為字串陣列，會在每筆陣列元素字串值尾端加上換行符號

```
string var_ss1 = {"Hi TM Robot", "達明機器人"}  
bool var_flag = false  
  
var_flag = File_WriteLines("writelines.txt", var_ss1)           // 將 ss1 覆寫進檔案  
  
writelines.txt  
1| Hi TM Robot  
2| 達明機器人  
3|  
  
var_flag = File_WriteLines("writelines.txt", var_ss1, 1, 1)   // 將 ss1 起始索引 1 至結束，附加進檔案  
  
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|

```

4.10 File_Exists()

檢查檔案路徑是否存在

語法 1

```
bool File_Exists(  
    string  
)
```

參數

string 檔案路徑

傳回值

bool	true	檔案路徑存在
	false	檔案路徑不存在

* 若是不支援的檔案路徑，會傳回 false 檔案路徑不存在，不會報錯

說明

```
bool var_exists = false  
  
var_exists = File_Exists("sampleFile1.txt")      // true  
  
var_exists = File_Exists("sampleFileX.txt")      // false    // 檔案不存在  
  
var_exists = File_Exists("C:\SampleFile1.txt")    // false    // 不支援絕對路徑
```

4.11 File_Length()

取得檔案大小

語法 1

```
int File_Length(  
    string  
)
```

參數

string 檔案路徑

傳回值

int	使用 int32 型別，檔案大小上限值為 2147483647 bytes
-1	檔案路徑不存在
-2	檔案大小超過上限

* 若是不支援的檔案路徑，會傳回 -1 檔案路徑不存在，不會報錯

說明

```
Int var_len = 0  
  
var_len = File_Length("sampleFile1.txt")      // 31  
  
var_len = File_Length("sampleFileX.txt")      // -1 // 檔案不存在  
  
var_len = File_Length("C:\SampleFile1.txt")    // -1 // 不支援絕對路徑
```

4.12 File_Delete()

刪除檔案

語法 1

```
bool File_Delete(  
    string  
    ...  
)
```

參數

string 檔案路徑
... 可以帶入多個 string

傳回值

bool true 刪除成功 (包含檔案不存在或不支援的檔案路徑)
false 刪除失敗 (因為檔案佔用，而無法正確刪除檔案)

語法 2

```
bool File_Delete(  
    string[]  
)
```

參數

string[] 檔案路徑

傳回值

bool true 刪除成功 (包含檔案不存在或不支援的檔案路徑)
false 刪除失敗 (因為檔案佔用，而無法正確刪除檔案)

說明

```
bool var_flag = false  
  
var_flag = File_Delete("sampleFile1.txt")      // true  
  
var_flag = File_Delete("sampleFileX.txt")      // true      // 檔案不存在  
  
var_flag = File_Delete("C:\SampleFile1.txt")    // true      // 不支援絕對路徑  
  
var_flag = File_Delete("sampleFile1.txt", "sampleFileX.txt") // 可帶入多個檔案路徑  
  
var_flag = File_Delete("sampleFile1.txt", "sampleFileX.txt", "C:\SampleFile1.txt") // 可帶入多個檔案路徑  
  
string[] var_ss = {"sampleFile1.txt", "sampleFileX.txt", "C:\SampleFile1.txt"}  
  
var_flag = File_Delete(var_ss)  
  
var_flag = File_Delete(var_ss, "sampleFile2.txt") // 報錯，語法不支援
```

4.13 File_Copy()

複製檔案

語法 1

```
bool File_Copy(  
    string,  
    string,  
    string  
)
```

參數

string	來源檔案路徑
string	目的目錄路徑
string	目的檔案名稱，若名稱為空字串，預設與來源檔案路徑相同檔名

傳回值

bool	true	複製成功
	false	複製失敗

* 若目的路徑檔案已存在，將會覆寫目的檔案

語法 2

```
bool File_Copy(  
    string,  
    string  
)
```

說明

與語法 1 相同，將目標檔案名稱設為空字串，與來源檔案路徑相同檔名

File_Copy(string,string) => File_Copy(string,string,"")

```
File_Copy("sampleFile1.txt", ".\TextFiles", "s1.txt") // 複製 .\TextFiles\sampleFile1.txt 到 .\TextFiles\s1.txt  
File_Copy("sampleFile1.txt", ".\XmlFiles", "") // 複製 .\TextFiles\sampleFile1.txt 到 .\XmlFiles\sampleFile1.txt  
File_Copy("sampleFile1.txt", "\USB\TMROBOT", "s1.txt") // 複製 .\TextFiles\sampleFile1.txt 到 USB\s1.txt  
File_Copy("sampleFile1.txt", "\USB\TMROBOT") // 複製 .\TextFiles\sampleFile1.txt 到 USB\sampleFile1.txt  
bool var_flag = false  
var_flag = File_Copy("sampleFile1.txt", "C:\folder") // 報錯，不支援絕對路徑  
var_flag = File_Copy("sampleFile1.txt", ".") // 報錯，路徑不在 TextFiles 或 XmlFiles 目錄下
```

4.14 File_CopyImage()

複製視覺存圖檔案

語法 1

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

參數

string	來源視覺存圖檔案路徑
string	目的目錄路徑 只允許複製存圖至外部路徑，如外部裝置路徑目錄或遠端路徑目錄 不允許複製存圖至本機路徑，將會報錯，如 ".\TextFiles" 或 ".\XmlFiles"
string	目的檔案名稱，若名稱為空字串，預設與來源檔案路徑相同檔名
int	複製選項 0 複製失敗時不報錯，不留來源圖檔相對目錄 (預設) 1 複製失敗時不報錯，保留來源圖檔相對目錄 2 複製失敗時報錯，不留來源圖檔相對目錄 3 複製失敗時報錯，保留來源圖檔相對目錄

傳回值

bool	true	複製成功
	false	複製失敗
* 若目的路徑檔案已存在，將會覆寫目的檔案		

語法 2

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

說明

與語法 1 相同，將複製選項設為 0 複製失敗不報錯，不留來源圖檔相對目錄

`File_CopyImage(string,string,string) => File_CopyImage(string,string,string,0)`

語法 3

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

說明

與語法 1 相同，將目標檔案名稱設為空字串，與來源檔案路徑相同檔名；複製選項設為 0 複製失敗不報錯，不保留來源圖檔相對目錄

File_CopyImage(string,string) => File_CopyImage(string,string,"",0)

語法 4

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

說明

與語法 1 相同，將目標檔案名稱設為空字串，與來源檔案路徑相同檔名

File_CopyImage(string,string,int) => File_CopyImage(string,string,"",int)

```
bool var_flag = false  
  
var_flag = File_CopyImage(Job1ImagePath_TM, ".\TextFiles", "1.png") // false // 不支援複製至本地目錄  
  
var_flag = File_CopyImage(Job1ImagePath_TM, ".\XmlFiles", "1.png") // false // 不支援複製至本地目錄  
  
var_flag = File_CopyImage(Job1ImagePath_TM, ".\XmlFiles", "1.png", 2) // 報錯 // 不支援複製至本地目錄  
  
var_flag = File_CopyImage(Job1ImagePath_TM, "\USB\TMROBOT", "1.png")  
  
// true // 複製 Job1ImagePath_TM (視覺 AOI-only 變數) 到 USB\1.png  
  
var_flag = File_CopyImage(Job1ImagePath_TM, "\USB\TMROBOT", "1.png", 3)  
  
// true // 複製 Job1ImagePath_TM (視覺 AOI-only 變數) 到 USB\ProjectName\Job1\Date\source\1.png // 保留存圖目錄  
  
var_flag = File_CopyImage(Job1ImagePath_TM, "\USB\TMROBOT")  
  
// true // 複製 Job1ImagePath_TM (視覺 AOI-only 變數) 到 USB\15-16-12_423.png // 保留存圖檔名  
  
var_flag = File_CopyImage(Job1ImagePath_TM, "\USB\TMROBOT", 3)  
  
// true // 複製 Job1ImagePath_TM (視覺 AOI-only 變數) 到 USB\ProjectName\Job1\Date\source\15-16-12_423.png  
  
// 保留存圖目錄及檔名
```

4.15 File_Replace()

依照特定字串取代檔案內的字串，並覆寫回檔案

語法 1

```
bool File_Replace(  
    string,  
    string,  
    string  
)
```

參數

string	檔案路徑
string	要被取代的字串值
string	取代的字串值

傳回值

bool	true	成功 1. 要被取代的字串為空字串 2. 找不到要被取代的字串值 3. 找到要被取代的字串值，且覆寫回檔案
	false	失敗

說明

```
.\TextFiles\SampleFile6.txt  
1| 6Hello World!  
2| 6Hello TM Robot!  
3| 6Hi TM Robot!  
  
bool var_flag = false  
var_flag = File_Replace("SampleFile6.txt", "Hello", "Hi")  
SampleFile6.txt  
1| 6HI World!  
2| 6HI TM Robot!  
3| 6Hi TM Robot!  
  
var_flag = File_Replace("SampleFile6.txt", "TM", "Techman")  
SampleFile6.txt  
1| 6HI World!  
2| 6HI Techman Robot!  
3| 6Hi Techman Robot!  
  
var_flag = File_Replace("SampleFile6.txt", "6", "")  
SampleFile6.txt  
1| HI World!  
2| HI Techman Robot!  
3| Hi Techman Robot!
```

4.16 File_GetToken()

依 string 字串格式讀取檔案，並從檔案內容中，取出字串的子字串

語法 1

```
string File_GetToken(  
    string,  
    string,  
    string,  
    int,  
    int  
)
```

參數

string	檔案路徑
string	要取出的前置字串
string	要取出的後置字串
int	取出的符合個數
	>=1 取出第幾個符合的字串
	-1 取出最後一個符合的字串
int	取出選項
0	第 1 個符合不須在檔案開頭，且不移除前置字串及後置字串（預設）
1	第 1 個符合不須在檔案開頭，且移除前置字串及後置字串
2	第 1 個符合須在檔案開頭，且不移除前置字串及後置字串
3	第 1 個符合須在檔案開頭，且移除前置字串及後置字串

傳回值

string	傳回取出的字串值
	如果前置字串及後置字串為空字串，則傳回檔案字串內容
	如果符合個數<=0 或大於符合總數，則傳回空字串
	如果取出選項為 2 或 3 時，則取出的第 1 個符合必須在檔案開頭，否則傳回空字串

語法 2

```
string File_GetToken(  
    string,  
    string,  
    string,  
    int  
)
```

說明

與語法 1 相同，預設將取出選項填入 0

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

語法 3

```
string File_GetToken(  
    string,  
    string,  
    string  
)
```

說明

與語法 1 相同，預設將符合個數填入 1 及取出選項填入 0

File_GetToken(string,string,int) => **File_GetToken(string,string,int,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!"      // 去除前置與後置
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 為書寫換行符號表示，並非字串值

```

```

.\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) // ""      // 因 $ 不在檔案開頭，傳回空字串
var_s = File_GetToken(var_n, "$", "", 1, 2)       // ""      // 因 $ 不在檔案開頭，傳回空字串
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)        // ""      // 超過符合個數
var_s = File_GetToken(var_n, "", "#", 1, 2)          // "#"

```

語法 4

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

參數

string	檔案路徑
byte[]	要取出的前置字元，byte[] 陣列格式
byte[]	要取出的後置字元，byte[] 陣列格式
int	取出的符合個數
	>=1 取出第幾個符合的字串
	-1 取出最後一個符合的字串
int	取出選項
0	第 1 個符合不須在檔案開頭，且不移除前置字串及後置字串 (預設)
1	第 1 個符合不須在檔案開頭，且移除前置字串及後置字串
2	第 1 個符合須在檔案開頭，且不移除前置字串及後置字串
3	第 1 個符合須在檔案開頭，且移除前置字串及後置字串

傳回值

string	傳回取出的字串值
	如果前置字串及後置字串為空陣列，則傳回檔案字串內容
	如果符合個數<=0 或大於符合總數，則傳回空字串
	如果取出選項為 2 或 3 時，則取出的第 1 個符合必須在檔案開頭，否則傳回空字串

語法 5

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

說明

與語法 4 相同，預設將取出選項填入 0

File_GetToken(string,byte[],byte[],int) => File_GetToken(string,byte[],byte[],int,0)

語法 6

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

說明

與語法 4 相同，預設將符合個數填入 1 及取出選項填入 0

```
File_GetToken(string,byte[],byte[]) => File_GetToken(string,byte[],byte[],1,0)
```

```
. \TextFiles\SampleFile8.txt  
1| $Hello World!  
2| Hello$ TM Robot!  
3| Hi$ TM Robot!$  
  
string var_n = "SampleFile8.txt", var_s = ""  
  
byte[] var_bb0 = {}, var_bb1 = {0x24}, var_bb2 = {0x0D, 0x0A} // 0x24 is $ and 0x0D 0x0A is \u0D0A  
var_s = File_GetToken(var_n, bb0, bb0, 0) // "$Hello World\u0D0AHello$ TM Robot!\u0D0AHi$ TM Robot!"  
var_s = File_GetToken(var_n, bb1, bb1) // "$Hello World\u0D0AHello$"  
var_s = File_GetToken(var_n, bb1, bb1, 0) // ""  
var_s = File_GetToken(var_n, bb1, bb1, 1) // "$Hello World\u0D0AHello$"  
var_s = File_GetToken(var_n, bb1, bb1, 2) // "$ TM Robot!$"  
var_s = File_GetToken(var_n, bb1, bb1, 3) // ""  
var_s = File_GetToken(var_n, bb1, bb1, 1, 1) // "Hello World\u0D0AHello"  
var_s = File_GetToken(var_n, bb1, bb1, 2, 1) // " TM Robot!"  
var_s = File_GetToken(var_n, bb1, bb0, 1) // "$Hello World\u0D0AHello"  
var_s = File_GetToken(var_n, bb1, bb0, 2) // "$ TM Robot!\u0D0AHi"  
var_s = File_GetToken(var_n, bb1, bb0, 3) // "$ TM Robot!"  
var_s = File_GetToken(var_n, bb1, bb0, 4) // "$"  
var_s = File_GetToken(var_n, bb0, bb1, 1) // "$"  
var_s = File_GetToken(var_n, bb0, bb1, 2) // "Hello World\u0D0AHello$"  
var_s = File_GetToken(var_n, bb0, bb1, 3) // " TM Robot!\u0D0AHi$"  
var_s = File_GetToken(var_n, bb0, bb1, 4) // " TM Robot!$"  
var_s = File_GetToken(var_n, bb1, bb2, 1) // "$Hello World\u0D0A"  
var_s = File_GetToken(var_n, bb1, bb2, 2) // "$ TM Robot!\u0D0A"  
var_s = File_GetToken(var_n, bb1, bb2, 1, 1) // "Hello World" // 去除前置與後置  
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 為書寫換行符號表示，並非字串值
```

4.17 File_GetAllTokens()

依 string 字串格式讀取檔案，並從檔案內容中，取出所有符合條件的子字串

語法 1

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

參數

string	檔案路徑
string	要取出的前置字串
string	要取出的後置字串
int	取出選項
0	第 1 個符合不須在檔案開頭，且不移除前置字串及後置字串（預設）
1	第 1 個符合不須在檔案開頭，且移除前置字串及後置字串
2	第 1 個符合須在檔案開頭，且不移除前置字串及後置字串
3	第 1 個符合須在檔案開頭，且移除前置字串及後置字串

傳回值

string[] 傳回取出的字串陣列

如果前置字串及後置字串為空字串，則傳回檔案字串內容(字串陣列)

如果取出選項為 2 或 3 時，則取出的第 1 個符合必須在檔案開頭，否則傳回空陣列

語法 2

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

說明

與語法 1 相同，預設將取出選項填入 0

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!"}

```

5. Serial Port 函式

當專案開始運行時(進入 Start 節點)，將會開啟 Serial Port 連接埠，並持續接收 Serial Port 上的資料，且將接收到的資料加入接收緩衝區(ReceivedBuffer)內，可利用 com_read 相關函式讀出接收緩衝區內的資料。當專案停止運行時，將會關閉已開啟的 Serial Port 連接埠，且清空接收緩衝區。

接收緩衝區的最大容量為 2MB bytes，當有資料要進入接收緩衝區時，若緩衝區容量不足，將會自動刪除最舊資料，並將最新資料加入資料緩衝區內。

5.1 com_read()

Serial Port 讀取接收緩衝區內的資料，並傳回 byte[] 陣列

語法 1

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

參數

string Serial Port 的裝置名稱 (在 Serial Port Device 中設定)

傳回值

byte[] 傳回接收緩衝區內的所有資料，若資料內容為空，則傳回 byte[0] 個

說明

```
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 = {}
```

* 此函式可讀取接收緩衝區內的所有資料，且可清空接收緩衝區

語法 2

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

參數

string Serial Port 的裝置名稱 (在 Serial Port Device 中設定)
int 固定取出多少個位元數 (依 byte[] 長度取)
 ≤ 0 取出全部
 > 0 取出指定個數 (需取滿指定個數，才會有資料)
int 讀取時間(millisecond)
 ≤ 0 只讀取一次
 > 0 讀取多次，直到有資料，或是時間滿足

傳回值

byte[] 傳回取出指定位元數的 byte[] 陣列值，若資料個數不足，將傳回 byte[0] 個

語法 3

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

說明

與語法 2 相同，預設將讀取時間填入 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[] = {}          // 因緩衝區內的資料個數未滿 100 個，個數不足，將傳回 byte[0] 個  
    // ReceivedBuffer = {0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}  
value = com_read("spd", 0)  
    // value byte[] = {0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}      // 取出全部資料  
    // ReceivedBuffer = {}
```

```

value = com_read("spd", 4, 100)
    // value byte[] = {} // 因緩衝區內的資料個數未滿 4 個，個數不足，將傳回 byte[0] 個
        // 但設定讀取時間 100 ms，所以仍停留在函式內，等到有資料或是讀取時間滿足才離開
    // ReceivedBuffer = {0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38} // 假設在 50ms 後收到資料
    // value byte[] = {0x31,0x32,0x33,0x34}                                // 取出 4 個，並離開函式
    // ReceivedBuffer = {0x35,0x36,0x37,0x38}

```

語法 4

```

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

```

參數

string Serial Port 的裝置名稱 (在 Serial Port Device 中設定)

byte[] or string
要讀取的前置條件，若輸入 byte[0] 或 "" 空字串，表示不限定前置條件

byte[] or string
要讀取的後置條件，若輸入 byte[0] 或 "" 空字串，表示不限定後置條件

int 取出選項

- 0 第 1 個符合不須在開頭，且不移除前置及後置 (預設)
- 1 第 1 個符合不須在開頭，且移除前置及後置
- 2 第 1 個符合須在開頭，且不移除前置及後置
- 3 第 1 個符合須在開頭，且移除前置及後置

int 讀取時間(millisecond)

- <= 0 只讀取一次
- > 0 讀取多次，直到有資料，或是時間滿足

傳回值

byte[] 傳回第一筆符合前置條件及後置條件的 byte[] 陣列
只會取出接收緩衝區內符合條件的第一筆資料，後面的資料仍保留在接收緩衝區內
如果前置條件及後置條件為 byte[0] 或空字串，則取出接收緩衝區內的所有資料
如果找不到符合條件的資料，將傳回 byte[0] 個

語法 5

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

說明

與語法 4 相同，預設將讀取時間填入 0

語法 6

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

說明

與語法 4 相同，預設將取出選項填入 0，及讀取時間填入 0

```
ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}  
value = com_read("spd", "", "") // 前置字空，後置字空，取出全部資料  
// value 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}  
value = com_read("spd", "He", newline) // 前置字 "He" 後置字 \u0D0A  
// value byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A} // Hello,\u0D0A  
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A} // 只取出第一筆符合，後面資料保留  
value = com_read("spd", "", newline, 1) // 前置字 "" 後置字 \u0D0A 移除前置及後置  
// value byte[] = {0x57,0x6F,0x72,0x6C,0x64} // World  
// ReceivedBuffer = {}  
value = com_read("spd", "", newline, 1, 100) // 前置字 "" 後置字 \u0D0A 移除前置及後置，100ms  
// value byte[] = {} // 因讀取條件未滿足，讀出 byte[0]，在 100ms 內等待  
// ReceivedBuffer = {}  
// ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A} // 假設在 50ms 後收到資料  
// value 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}

value = com_read("spd", "lo", newline)           // 前置字 "lo" 後置字 \u0D0A
// value byte[] = {0x6C,0x6F,0x2C,0x0D,0x0A}      // lo,\u0D0A
// 在第一筆符合條件之前的資料 {0x48,0x65,0x6C} 將會被移除
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}

byte[] bb = {}

value = com_read("spd", bb, newline)             // 前置字 byte[0] 後置字 \u0D0A
// value byte[] = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}      // World\u0D0A
// ReceivedBuffer = {}

value = com_read("spd", bb, newline, 0, 100)    // 前置字 byte[0] 後置字 \u0D0A, 100ms
// value byte[] = {}                                // 因讀取條件未滿足，讀出 byte[0]，在 100ms 內等待
// ReceivedBuffer = {}
// ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A} // 假設在 50ms 後收到資料
// 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)         // 前置字 "#" 後置字 \u0D0A
// value byte[] = {}                                // 因 # 不在開頭，傳回空陣列
// 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", "$", newline, 2)         // 前置字 "$" 後置字 \u0D0A
// value byte[] = {0x24,0x48,0x65,0x6C,0x6C,0x6F,0x0D,0x0A}      // 取出第 1 筆符合須在開頭
// 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)         // 前置字 "#" 後置字 \u0D0A
// value byte[] = {0x23,0x48,0x69,0x0D,0x0A}          // 取出第 1 筆符合須在開頭
// ReceivedBuffer = {0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

value = com_read("spd", "$", newline, 3)         // 前置字 "$" 後置字 \u0D0A
// value byte[] = {0x54,0x4D}
// ReceivedBuffer = {0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

value = com_read("spd", "#", newline, 3)         // 前置字 "#" 後置字 \u0D0A
// value byte[] = {0x52,0x6F,0x62,0x6F,0x74}
// ReceivedBuffer = {}

```

語法 7

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

參數

string Serial Port 的裝置名稱 (在 Serial Port Device 中設定)
byte[] or string
要讀取的後置條件，若輸入 byte[0] 或 "" 空字串，表示不限定後置條件
int 取出選項
0 第 1 個符合不須在開頭，且不移除前置及後置 (預設)
1 第 1 個符合不須在開頭，且移除前置及後置
2 第 1 個符合須在開頭，且不移除前置及後置
3 第 1 個符合須在開頭，且移除前置及後置
int 讀取時間(millisecond)
<=0 只讀取一次
>0 讀取多次，直到有資料，或是時間滿足

傳回值

byte[] 傳回第一筆符合後置條件的 byte[] 陣列 (不限定前置條件)
只會取出接收緩衝區內符合條件的第一筆資料，後面的資料仍保留在接收緩衝區內
如果後置條件為 byte[0] 或空字串，則取出接收緩衝區內的所有資料
如果找不到符合條件的資料，將傳回 byte[0] 個

語法 8

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

說明

與語法 7 相同，預設將讀取時間填入 0

語法 9

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

說明

與語法 7 相同，預設將取出選項填入 0，及讀取時間填入 0

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

```
value = com_read("spd", "") // 後置字空，取出全部資料
```

```
// value 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}
```

```
value = com_read("spd", newline) // 後置字 \u0D0A
```

```
// value byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A} // Hello,\u0D0A
```

```
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A} // 只取出第一筆符合，後面資料保留
```

```
value = com_read("spd", newline) // 後置字 \u0D0A
```

```
// value 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
```

```
value = com_read("spd", newline, 0) // 後置字 \u0D0A
```

```
// value 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}
```

```
value = com_read("spd", newline, 1) // 後置字 \u0D0A
```

```
// value byte[] = {0x23,0x48,0x69} // #Hi
```

```
// ReceivedBuffer = {0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}
```

```
value = com_read("spd", newline, 2) // 後置字 \u0D0A
```

```
// value byte[] = {0x24,0x54,0x4D,0x0D,0x0A} // $TM\u0D0A
```

```
// ReceivedBuffer = {0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}
```

```
value = com_read("spd", newline, 3) // 後置字 \u0D0A
```

```
// value byte[] = {0x23,0x52,0x6F,0x62,0x6F,0x74} // #Robot
```

```
// ReceivedBuffer = {}
```

5.2 com_read_string()

Serial Port 讀取接收緩衝區內的資料，並轉換 byte[] 陣列為 UTF8 編碼的字串

語法 1

```
string com_read_string(  
    string  
)
```

參數

string Serial Port 的裝置名稱 (在 Serial Port Device 中設定)

傳回值

string 傳回接收緩衝區內的所有資料，若資料內容為空，則傳回空字串

語法 2

```
string com_read_string(  
    string,  
    int,  
    int  
)
```

參數

string Serial Port 的裝置名稱 (在 Serial Port Device 中設定)

int 固定取出多少個字串數 (依字串長度取)

<= 0 取出全部

> 0 取出指定個數 (需取滿指定個數，才會有資料)

int 讀取時間(millisecond)

<= 0 只讀取一次

> 0 讀取多次，直到有資料，或是時間滿足

傳回值

string 傳回取出指定字串長度的字串值，若資料個數不足，將傳回空字串

語法 3

```
string com_read_string(  
    string,  
    int  
)
```

說明

與語法 2 相同，預設將讀取時間填入 0

語法 4

```
string com_read_string(  
    string,  
    byte[] or string,  
    byte[] or string,  
    int,  
    int  
)
```

參數

string Serial Port 的裝置名稱 (在 Serial Port Device 中設定)
byte[] or string
要讀取的前置條件，若輸入 byte[0] 或 "" 空字串，表示不限定前置條件
byte[] or string
要讀取的後置條件，若輸入 byte[0] 或 "" 空字串，表示不限定後置條件
int 取出選項
0 第 1 個符合不須在開頭，且不移除前置及後置 (預設)
1 第 1 個符合不須在開頭，且移除前置及後置
2 第 1 個符合須在開頭，且不移除前置及後置
3 第 1 個符合須在開頭，且移除前置及後置
int 讀取時間(millisecond)
<= 0 只讀取一次
> 0 讀取多次，直到有資料，或是時間滿足

傳回值

string 傳回第一筆符合前置條件及後置條件的字串
只會取出接收緩衝區內符合條件的第一筆資料，後面的資料仍保留在接收緩衝區內
如果前置條件及後置條件為 byte[0] 或空字串，則取出接收緩衝區內的所有資料
如果找不到符合條件的資料，將傳回空字串

語法 5

```
string com_read_string(  
    string,  
    byte[] or string,  
    byte[] or string,  
    int  
)
```

說明

與語法 4 相同，預設將讀取時間填入 0

語法 6

```
string com_read_string(  
    string,  
    byte[] or string,  
    byte[] or string  
)
```

說明

與語法 4 相同，預設將取出選項填入 0，及讀取時間填入 0

語法 7

```
string com_read_string(  
    string,  
    byte[] or string,  
    int,  
    int  
)
```

參數

string Serial Port 的裝置名稱 (在 Serial Port Device 中設定)

byte[] or string

要讀取的後置條件，若輸入 byte[0] 或 "" 空字串，表示不限定後置條件

int 取出選項

0 第 1 個符合不須在開頭，且不移除前置及後置 (預設)

1 第 1 個符合不須在開頭，且移除前置及後置

2 第 1 個符合須在開頭，且不移除前置及後置

3 第 1 個符合須在開頭，且移除前置及後置

int 讀取時間(millisecond)

<= 0 只讀取一次

> 0 讀取多次，直到有資料，或是時間滿足

傳回值

string 傳回第一筆符合後置條件的字串 (不限定前置條件)

只會取出接收緩衝區內符合條件的第一筆資料，後面的資料仍保留在接收緩衝區內

如果後置條件為 byte[0] 或空字串，則取出接收緩衝區內的所有資料

如果找不到符合條件的資料，將傳回空字串

語法 8

```
string com_read_string(  
    string,  
    byte[] or string,  
    int  
)
```

說明

與語法 7 相同，預設將讀取時間填入 0

語法 9

```
string com_read_string(  
    string,  
    byte[] or string  
)
```

說明

與語法 7 相同，預設將取出選項填入 0，及讀取時間填入 0

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} // 依字串長度取 4 個  
// ReceivedBuffer = {0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA}  
value = com_read_string("spd", 5, 100)  
// value string = "" // 因緩衝區內的資料個數未滿 5 個 (依字串長度取)，在 100ms 內等待  
// 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", "", "")  
// value string = "Hello,\u0D0AWorld\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) // 前置字 "He" 後置字 \u0D0A
// value string = "Hello,\u0D0A"
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}

value = com_read_string("spd", "", newline, 1) // 前置字 "" 後置字 \u0D0A 移除前置及後置
// value string = "World"
// ReceivedBuffer = {}

value = com_read_string("spd", "", newline, 1, 100)
// value string = "" // 因讀取條件未滿足，讀出空字串，在 100ms 內等待
// 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) // 前置字 "lo" 後置字 \u0D0A
// value string = "lo,\u0D0A"
// 在第一筆符合條件之前的資料 "Hel" 將會被移除
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}

value = com_read_string("spd", newline, 1) // 後置字 \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) // 前置字 "#" 後置字 \u0D0A
// value string = "" // 因 # 不在開頭，傳回空字串
// 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}

value = com_read_string("spd", "$", newline, 2) // 前置字 "$" 後置字 \u0D0A
// value string = "$Hello\u0D0A" // 取出第 1 筆符合須在開頭
// 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) // 前置字 "#" 後置字 \u0D0A
// value string = "#Hi\u0D0A" // 取出第 1 筆符合須在開頭
// ReceivedBuffer = {0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

```

```

value = com_read_string("spd", "$", newline, 3)      // 前置字 "$" 後置字 \u0D0A
// value string = "TM"
// ReceivedBuffer = {0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

value = com_read_string("spd", "#", newline, 3)      // 前置字 "#" 後置字 \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)      // 後置字 \u0D0A
// value string = "Hello,\u0D0A"
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64}

value = com_read_string("spd", newline, 0)      // 後置字 \u0D0A
// value string = ""                      // 因讀取條件未滿足，讀出空字串
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64}

value = com_read_string("spd", newline, 1, 100)// 後置字 \u0D0A
// value string = ""                      // 因讀取條件未滿足，讀出空字串，在 100ms 內等待
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64}

// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A,0x31,0x32,0x33,0x0D,0x0A}
// 假設資料進來，且使讀取條件滿足

// value string = "World"
// ReceivedBuffer = {0x31,0x32,0x33,0x0D,0x0A}

value = com_read_string("spd", newline, 2)      // 後置字 \u0D0A
// value string = "123\u0D0A"
// ReceivedBuffer = {}

```

5.3 com_write()

Serial Port 寫入資料至連接埠

語法 1

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

參數

string Serial Port 的裝置名稱 (在 Serial Port Device 中設定)
? 寫入資料值，可以是整數、浮點數、布林值、字串值或是陣列型別
數值將會使用 Little Endian 方式轉換，字串值將會使用 UTF8 方式轉換
int 寫入資料值的起始索引 (對字串值或陣列型別有效)
0.. 長度-1 合法值
<0 非法使用，起始索引將設為 0
≥ 長度 非法使用，起始索引將設為 0
int 寫入資料值的長度 (對字串值或陣列型別有效)
≤ 0 從起始索引，寫至資料結束
≥ 0 從起始索引，寫入指定個數長度，最多寫至資料結束

傳回值

bool True 寫入成功
False 寫入失敗 1. 如果 ? 寫入資料值為空字串或空陣列
2. 無法正確發送

語法 2

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

說明

與語法 1 相同，預設將寫入資料值的長度填入 0

語法 3

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

說明

與語法 1 相同，預設將寫入資料值的起始索引填入 0 及寫入資料值的長度填入 0

```
flag = com_write("spd", 100)           // write 0x64  
flag = com_write("spd", 1000)          // write 0xE8 0x03 0x00 0x00 (int, Little Endian)  
flag = com_write("spd", (float)1.234) // write 0xB6 0xF3 0x9D 0x3F (float, Little Endian)  
flag = com_write("spd", (double)123.456)  
                                         // write 0x77 0xBE 0x9F 0x1A 0x2F 0xDD 0x5E 0x40 (double, Little Endian)  
flag = com_write("spd", "Hello, World"+newline)  
                                         // write 0x48 0x65 0x6C 0x6C 0x6F 0x2C 0x20 0x57 0x6F 0x72 0x6C 0x64 0x0D 0x0A (string, UTF8)  
flag = com_write("spd", 1000, 1, 2)      // 數值，起始位址與長度無效  
                                         // write 0xE8 0x03 0x00 0x00 (int, Little Endian)  
  
byte[] bb = {100, 200}  
flag = com_write("spd", bb)            // write 0x64 0xC8  
flag = com_write("spd", bb, 1, 1)      // write 0xC8    // 陣列，從位址 1 起取 1 個，[1]=200  
flag = com_write("spd", bb, -1, 1)     // write 0x64    // 陣列，從位址 0 起取 1 個，[0]=100  
flag = com_write("spd", "達明機器人", 2) // 字串，從位址 2 起取至結束，"機器人"  
                                         // write 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA (string, UTF8)  
string[] ss = {"TM", "", "達明機器人"}  
flag = com_write("spd", ss)  
                                         // write 0x54 0x4D 0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA  
flag = com_write("spd", Byte_Concat(GetBytes(ss), GetBytes(newline)))  
                                         // write 0x54 0x4D 0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA 0x0D 0x0A  
flag = com_write("spd", ss, 2, 100)       // 陣列，從位址 2 起取 100 個(結束)，[2]=達明機器人  
                                         // write 0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA
```

5.4 com_writeline()

Serial Port 寫入資料至連接埠，會自動在寫入的資料尾端加上換行符號 0x0D 0x0A

語法 1

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

參數

string Serial Port 的裝置名稱 (在 Serial Port Device 中設定)
? 寫入資料值，可以是整數、浮點數、布林值、字串值或是陣列型別
數值將會使用 Little Endian 方式轉換，字串值將會使用 UTF8 方式轉換
int 寫入資料值的起始索引 (對字串值或陣列型別有效)
0.. 長度-1 合法值
<0 非法使用，起始索引將設為 0
≥ 長度 非法使用，起始索引將設為 0
int 寫入資料值的長度 (對字串值或陣列型別有效)
≤ 0 從起始索引，寫至資料結束
≥ 0 從起始索引，寫入指定個數長度，最多寫至資料結束

傳回值

bool True 寫入成功
False 寫入失敗 1. 如果 ? 寫入資料值為空字串或空陣列
2. 無法正確發送

語法 2

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

說明

與語法 1 相同，預設將寫入資料值的長度填入 0

語法 3

```
bool com_writeline(  
    string,  
    ?  
)
```

說明

與語法 1 相同，預設將寫入資料值的起始索引填入 0 及寫入資料值的長度填入 0

```
flag = com_writeline("spd", 100)           // write 0x64 0x0D 0x0A  
flag = com_writeline("spd", 1000)          // write 0xE8 0x03 0x00 0x00 0x0D 0x0A (int, Little Endian)  
flag = com_writeline("spd", (float)1.234)    // write 0xB6 0xF3 0x9D 0x3F 0x0D 0x0A (float, Little Endian)  
flag = com_writeline("spd", (double)123.456)  
    // write 0x77 0xBE 0x9F 0x1A 0x2F 0xDD 0x5E 0x40 0x0D 0x0A (double, Little Endian)  
flag = com_write("spd", "Hello, World"+newline)  
    // write 0x48 0x65 0x6C 0x6C 0x6F 0x2C 0x20 0x57 0x6F 0x72 0x6C 0x64 0x0D 0x0A (string, UTF8)  
flag = com_writeline("spd", "Hello, World")  
    // write 0x48 0x65 0x6C 0x6C 0x6F 0x2C 0x20 0x57 0x6F 0x72 0x6C 0x64 0x0D 0x0A (string, UTF8)  
flag = com_writeline("spd", 1000, 1, 2)        // 數值，起始位址與長度無效  
    // write 0xE8 0x03 0x00 0x00 0x0D 0x0A (int, Little Endian)  
byte[] bb = {100, 200}  
flag = com_writeline("spd", bb)      // write 0x64 0xC8 0x0D 0x0A  
flag = com_writeline("spd", bb, 1, 1) // write 0xC8 0x0D 0x0A    // 陣列，從位址 1 起取 1 個，[1]=200  
flag = com_writeline("spd", bb, -1, 1) // write 0x64 0x0D 0x0A    // 陣列，從位址 0 起取 1 個，[0]=100  
flag = com_writeline("spd", "達明機器人", 2)        // 字串，從位址 2 起取至結束，"機器人"  
    // write 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA 0x0D 0x0A (string, UTF8)  
string[] ss = {"TM", "", "達明機器人"}  
flag = com_writeline("spd", ss)  
    // write 0x54 0x4D 0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA 0x0D 0x0A  
flag = com_write("spd", Byte_Concat(GetBytes(ss), GetBytes(newline)))  
    // write 0x54 0x4D 0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA 0x0D 0x0A  
flag = com_writeline("spd", ss)  
    // write 0x54 0x4D 0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA 0x0D 0x0A  
flag = com_writeline("spd", ss, 2, 100)        // 陣列，從位址 2 起取 100 個(結束)，[2]=達明機器人  
    // write 0xE9 0x81 0x94 0xE6 0x98 0x8E 0xE6 0xA9 0x9F 0xE5 0x99 0xA8 0xE4 0xBA 0xBA 0x0D 0x0A
```

6. Socket 函式

當專案開始運行時(進入 Start 節點)，將會開啟 TCP/IP Socket *Client* 連線至指定的 IP 及 Port，並持續接收連線上的資料，且將接收到的資料加入接收緩衝區(ReceivedBuffer)內，可利用 `socket_read` 相關函式讀出接收緩衝區內的資料。當專案停止運行時，將會關閉已存在的 TCP/IP Socket 連線，且清空接收緩衝區。

接收緩衝區具有容量限制，當有資料要進入接收緩衝區時，若緩衝區容量不足，將會自動刪除最舊資料，並將最新資料加入資料緩衝區內。

6.1 `socket_read()`

讀取接收緩衝區內的資料，並傳回 `byte[]` 陣列

語法 1

```
byte[] socket_read(  
    string  
)
```

參數

`string` Network 的裝置名稱 (在 Network Device 中設定)

傳回值

`byte[]` 傳回接收緩衝區內的所有資料，若資料內容為空，則傳回 `byte[0]` 個

說明

```
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 = {}
```

* 此函式可讀取接收緩衝區內的所有資料，且可清空接收緩衝區

語法 2

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

參數

string Network 的裝置名稱 (在 Network Device 中設定)
int 固定取出多少個位元數 (依 byte[] 長度取)
 ≤ 0 取出全部
 > 0 取出指定個數 (需取滿指定個數，才會有資料)
int 讀取時間(millisecond)
 ≤ 0 只讀取一次
 > 0 讀取多次，直到有資料，或是時間滿足

傳回值

byte[] 傳回取出指定位元數的 byte[] 陣列值，若資料個數不足，將傳回 byte[0] 個

語法 3

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

說明

與語法 2 相同，預設將讀取時間填入 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[] = {}      // 因緩衝區內的資料個數未滿 100 個，個數不足，將傳回 byte[0] 個  
    // 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}      // 取出全部資料  
    // ReceivedBuffer = {}
```

```

var_value = socket_read("ntd_a", 4, 100)
    // byte[] = {} // 因緩衝區內的資料個數未滿 4 個，個數不足，將傳回 byte[0] 個
        // 但設定讀取時間 100 ms，所以仍停留在函式內，等到有資料或是讀取時間滿足才離開
    // ReceivedBuffer = {0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38} // 假設在 50ms 後收到資料
    // byte[] = {0x31,0x32,0x33,0x34}                                // 取出 4 個，並離開函式
    // ReceivedBuffer = {0x35,0x36,0x37,0x38}

```

語法 4

```

byte[] socket_read(
    string,
    byte[] or string,
    byte[] or string,
    int,
    int
)

```

參數

string Network 的裝置名稱 (在 Network Device 中設定)

byte[] or string
要讀取的前置條件，若輸入 byte[0] 或 "" 空字串，表示不限定前置條件

byte[] or string
要讀取的後置條件，若輸入 byte[0] 或 "" 空字串，表示不限定後置條件

int 取出選項

- 0 第 1 個符合不須在開頭，且不移除前置及後置 (預設)
- 1 第 1 個符合不須在開頭，且移除前置及後置
- 2 第 1 個符合須在開頭，且不移除前置及後置
- 3 第 1 個符合須在開頭，且移除前置及後置

int 讀取時間(millisecond)

- <= 0 只讀取一次
- > 0 讀取多次，直到有資料，或是時間滿足

傳回值

byte[] 傳回第一筆符合前置條件及後置條件的 byte[] 陣列
只會取出接收緩衝區內符合條件的第一筆資料，後面的資料仍保留在接收緩衝區內
如果前置條件及後置條件為 byte[0] 或空字串，則取出接收緩衝區內的所有資料
如果找不到符合條件的資料，將傳回 byte[0] 個

語法 5

```
byte[] socket_read(  
    string,  
    byte[] or string,  
    byte[] or string,  
    int  
)
```

說明

與語法 4 相同，預設將讀取時間填入 0

語法 6

```
byte[] socket_read(  
    string,  
    byte[] or string,  
    byte[] or string  
)
```

說明

與語法 4 相同，預設將取出選項填入 0，及讀取時間填入 0

```
ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}  
byte[] var_value = socket_read("ntd_a", "", "") // 前置字空，後置字空，取出全部資料  
// 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) // 前置字 "He" 後置字 \u0D0A  
// byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A} // Hello,\u0D0A  
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A} // 只取出第一筆符合，後面資料保留  
var_value = socket_read("ntd_a", "", newline, 1) // 前置字 "" 後置字 \u0D0A 移除前置及後置  
// byte[] = {0x57,0x6F,0x72,0x6C,0x64} // World  
// ReceivedBuffer = {}  
var_value = socket_read("ntd_a", "", newline, 1, 100) // 前置字 "" 後置字 \u0D0A 移除前置及後置，100ms  
// byte[] = {} // 因讀取條件未滿足，讀出 byte[0]，在 100ms 內等待  
// ReceivedBuffer = {}  
// ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A} // 假設在 50ms 後收到資料  
// 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", "Io", newline)      // 前置字 "Io" 後置字 \u0D0A
// byte[] = {0x6C,0x6F,0x2C,0x0D,0x0A}
// 在第一筆符合條件之前的資料 {0x48,0x65,0x6C} 將會被移除
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}

byte[] var_bb = {}

var_value = socket_read("ntd_a", var_bb, newline)          // 前置字 byte[0] 後置字 \u0D0A
// byte[] = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}          // World\u0D0A
// ReceivedBuffer = {}

var_value = socket_read("ntd_a", var_bb, newline, 0, 100) // 前置字 byte[0] 後置字 \u0D0A, 100ms
// byte[] = {}                                              // 因讀取條件未滿足，讀出 byte[0]，在 100ms 內等待
// ReceivedBuffer = {}
// ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A} // 假設在 50ms 後收到資料
// 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)    // 前置字 "#" 後置字 \u0D0A
// byte[] = {}                                              // 因 # 不在開頭，傳回空陣列
// ReceivedBuffer = {0x24,0x48,0x65,0x6C,0x6C,0x6F,0x0D,0x0A,
                  0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

var_value = socket_read("ntd_a", "$", newline, 2)           // 前置字 "$" 後置字 \u0D0A
// byte[] = {0x24,0x48,0x65,0x6C,0x6C,0x6F,0x0D,0x0A}      // 取出第 1 筆符合須在開頭
// 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)           // 前置字 "#" 後置字 \u0D0A
// byte[] = {0x23,0x48,0x69,0x0D,0x0A}                      // 取出第 1 筆符合須在開頭
// ReceivedBuffer = {0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

var_value = socket_read("ntd_a", "$", newline, 3)           // 前置字 "$" 後置字 \u0D0A
// byte[] = {0x54,0x4D}
// ReceivedBuffer = {0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

var_value = socket_read("ntd_a", "#", newline, 3)           // 前置字 "#" 後置字 \u0D0A
// byte[] = {0x52,0x6F,0x62,0x6F,0x74}
// ReceivedBuffer = {}

```

語法 7

```
byte[] socket_read(  
    string,  
    byte[] or string,  
    int,  
    int  
)
```

參數

string Network 的裝置名稱 (在 Network Device 中設定)
byte[] or string
要讀取的後置條件，若輸入 byte[0] 或 "" 空字串，表示不限定後置條件
int 取出選項
0 第 1 個符合不須在開頭，且不移除前置及後置 (預設)
1 第 1 個符合不須在開頭，且移除前置及後置
2 第 1 個符合須在開頭，且不移除前置及後置
3 第 1 個符合須在開頭，且移除前置及後置
int 讀取時間(millisecond)
<=0 只讀取一次
>0 讀取多次，直到有資料，或是時間滿足

傳回值

byte[] 傳回第一筆符合後置條件的 byte[] 陣列 (不限定前置條件)
只會取出接收緩衝區內符合條件的第一筆資料，後面的資料仍保留在接收緩衝區內
如果後置條件為 byte[0] 或空字串，則取出接收緩衝區內的所有資料
如果找不到符合條件的資料，將傳回 byte[0] 個

語法 8

```
byte[] socket_read(  
    string,  
    byte[] or string,  
    int  
)
```

說明

與語法 7 相同，預設將讀取時間填入 0

語法 9

```
byte[] socket_read(  
    string,  
    byte[] or string  
)
```

說明

與語法 7 相同，預設將取出選項填入 0，及讀取時間填入 0

```
ReceivedBuffer = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}  
byte[] var_value = socket_read("ntd_a", "") // 後置字空，取出全部資料  
// 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) // 後置字 \u0D0A  
// byte[] = {0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x0D,0x0A} // Hello,\u0D0A  
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A} // 只取出第一筆符合，後面資料保留  
var_value = socket_read("ntd_a", newline) // 後置字 \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) // 後置字 \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) // 後置字 \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) // 後置字 \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) // 後置字 \u0D0A  
// byte[] = {0x23,0x52,0x6F,0x62,0x6F,0x74} // #Robot  
// ReceivedBuffer = {}
```

6.2 socket_read_string()

讀取接收緩衝區內的資料，並轉換 byte[] 陣列為 UTF8 編碼的字串

語法 1

```
string socket_read_string(  
    string  
)
```

參數

string Network 的裝置名稱 (在 Network Device 中設定)

傳回值

string 傳回接收緩衝區內的所有資料，若資料內容為空，則傳回空字串

語法 2

```
string socket_read_string(  
    string,  
    int,  
    int  
)
```

參數

string Network 的裝置名稱 (在 Network Device 中設定)

int 固定取出多少個字串數 (依字串長度取)

<= 0 取出全部

> 0 取出指定個數 (需取滿指定個數，才會有資料)

int 讀取時間(millisecond)

<= 0 只讀取一次

> 0 讀取多次，直到有資料，或是時間滿足

傳回值

string 傳回取出指定字串長度的字串值，若資料個數不足，將傳回空字串

語法 3

```
string socket_read_string(  
    string,  
    int  
)
```

說明

與語法 2 相同，預設將讀取時間填入 0

語法 4

```
string socket_read_string(  
    string,  
    byte[] or string,  
    byte[] or string,  
    int,  
    int  
)
```

參數

string Network 的裝置名稱 (在 Network Device 中設定)
byte[] or string
要讀取的前置條件，若輸入 byte[0] 或 "" 空字串，表示不限定前置條件
byte[] or string
要讀取的後置條件，若輸入 byte[0] 或 "" 空字串，表示不限定後置條件
int 取出選項
0 第 1 個符合不須在開頭，且不移除前置及後置 (預設)
1 第 1 個符合不須在開頭，且移除前置及後置
2 第 1 個符合須在開頭，且不移除前置及後置
3 第 1 個符合須在開頭，且移除前置及後置
int 讀取時間(millisecond)
<= 0 只讀取一次
> 0 讀取多次，直到有資料，或是時間滿足

傳回值

string 傳回第一筆符合前置條件及後置條件的字串
只會取出接收緩衝區內符合條件的第一筆資料，後面的資料仍保留在接收緩衝區內
如果前置條件及後置條件為 byte[0] 或空字串，則取出接收緩衝區內的所有資料
如果找不到符合條件的資料，將傳回空字串

語法 5

```
string socket_read_string(  
    string,  
    byte[] or string,  
    byte[] or string,  
    int  
)
```

說明

與語法 4 相同，預設將讀取時間填入 0

語法 6

```
string socket_read_string(  
    string,  
    byte[] or string,  
    byte[] or string  
)
```

說明

與語法 4 相同，預設將取出選項填入 0，及讀取時間填入 0

語法 7

```
string socket_read_string(  
    string,  
    byte[] or string,  
    int,  
    int  
)
```

參數

string Network 的裝置名稱 (在 Network Device 中設定)

byte[] or string

要讀取的後置條件，若輸入 byte[0] 或 "" 空字串，表示不限定後置條件

int 取出選項

0 第 1 個符合不須在開頭，且不移除前置及後置 (預設)

1 第 1 個符合不須在開頭，且移除前置及後置

2 第 1 個符合須在開頭，且不移除前置及後置

3 第 1 個符合須在開頭，且移除前置及後置

int 讀取時間(millisecond)

<= 0 只讀取一次

> 0 讀取多次，直到有資料，或是時間滿足

傳回值

string 傳回第一筆符合後置條件的字串 (不限定前置條件)

只會取出接收緩衝區內符合條件的第一筆資料，後面的資料仍保留在接收緩衝區內

如果後置條件為 byte[0] 或空字串，則取出接收緩衝區內的所有資料

如果找不到符合條件的資料，將傳回空字串

語法 8

```
string socket_read_string(  
    string,  
    byte[] or string,  
    int  
)
```

說明

與語法 7 相同，預設將讀取時間填入 0

語法 9

```
string socket_read_string(  
    string,  
    byte[] or string  
)
```

說明

與語法 7 相同，預設將取出選項填入 0，及讀取時間填入 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}      // 依字串長度取 4 個
```

```
    // ReceivedBuffer = {0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA}
```

```
var_value = socket_read_string("ntd_a", 5, 100)  
    // string = ""          // 因緩衝區內的資料個數未滿 5 個 (依字串長度取)，在 100ms 內等待
```

```
    // 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 var_value = socket_read_string("ntd_a", "He", newline) // 前置字 "He" 後置字 \u0D0A
// string = "Hello,\u0D0A"
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}

var_value = socket_read_string("ntd_a", "", newline, 1) // 前置字 "" 後置字 \u0D0A 移除前置及後置
// string = "World"
// ReceivedBuffer = {}

var_value = socket_read_string("ntd_a", "", newline, 1, 100)
// string = "" // 因讀取條件未滿足，讀出空字串，在 100ms 內等待
// 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) // 前置字 "lo" 後置字 \u0D0A
// string = "lo,\u0D0A"
// 在第一筆符合條件之前的資料 "Hel" 將會被移除
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A}

var_value = socket_read_string("ntd_a", newline, 1) // 後置字 \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) // 前置字 "#" 後置字 \u0D0A
// string = "" // 因 # 不在開頭，傳回空字串
// 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", "$", newline, 2) // 前置字 "$" 後置字 \u0D0A
// string = "$Hello\u0D0A" // 取出第 1 筆符合須在開頭
// 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) // 前置字 "#" 後置字 \u0D0A
// string = "#Hi\u0D0A" // 取出第 1 筆符合須在開頭
// ReceivedBuffer = {0x24,0x54,0x4D,0x0D,0x0A,0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

```

```

var_value = socket_read_string("ntd_a", "$", newline, 3) // 前置字 "$" 後置字 \u0D0A
// string = "TM"
// ReceivedBuffer = {0x23,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A}

var_value = socket_read_string("ntd_a", "#", newline, 3) // 前置字 "#" 後置字 \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) // 後置字 \u0D0A
// string = "Hello,\u0D0A"
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64}

var_value = socket_read_string("ntd_a", newline, 0) // 後置字 \u0D0A
// string = "" // 因讀取條件未滿足，讀出空字串
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64}

var_value = socket_read_string("ntd_a", newline, 1, 100) // 後置字 \u0D0A
// string = "" // 因讀取條件未滿足，讀出空字串，在 100ms 內等待
// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64}

// ReceivedBuffer = {0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A,0x31,0x32,0x33,0x0D,0x0A}
// 假設資料進來，且使讀取條件滿足

// string = "World"
// ReceivedBuffer = {0x31,0x32,0x33,0x0D,0x0A}

var_value = socket_read_string("ntd_a", newline, 2) // 後置字 \u0D0A
// string = "123\u0D0A"
// ReceivedBuffer = {}

```

6.3 socket_send()

發送資料值至遠端裝置

語法 1

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

參數

string	Network 的裝置名稱 (在 Network Device 中設定)
?	發送資料值，可以是整數、浮點數、布林值、字串值或是陣列型別 數值將會使用 Little Endian 方式轉換，字串值將會使用 UTF8 方式轉換
int	發送資料值的起始索引 (對字串值或陣列型別有效) 0.. 長度-1 合法值 <0 非法使用，起始索引將設為 0 ≥ 長度 非法使用，起始索引將設為 0
int	發送資料值的長度 (對字串值或陣列型別有效) ≤ 0 從起始索引，寫至資料結束 > 0 從起始索引，寫入指定個數長度，最多寫至資料結束

傳回值

int	發送結果
1	發送成功
0	無法發送資料值為空字串或空陣列
-1	發送時發生 socket 異常
-2	無法連線至遠端裝置
-3	裝置名稱不存在或是 IP 或 Port 不正確

語法 2

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

說明

與語法 1 相同，預設將發送資料值的長度填入 0

語法 3

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

說明

與語法 1 相同，預設將發送資料的起始索引填入 0 及發送資料值的長度填入 0

```
int var_re = socket_send("ntd_a", 100)           // send 0x64  
var_re = socket_send("ntd_a", 1000)              // send 0xE8,0x03,0x00,0x00 (int, Little Endian)  
var_re = socket_send("ntd_a", (float)1.234)       // send 0xB6,0xF3,0x9D,0x3F (float, Little Endian)  
var_re = socket_send("ntd_a", (double)123.456)  
    // send 0x77,0xBE,0x9F,0x1A,0x2F,0xDD,0x5E,0x40 (double, Little Endian)  
var_re = socket_send("ntd_a", "Hello, World"+newline)  
    // send 0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A (string, UTF8)  
int[] var_ii = {100, 200, 300, 400}  
var_re = socket_send("ntd_a", var_ii)  
    // send 0x64,0x00,0x00,0x00,0xC8,0x00,0x00,0x00,0x2C,0x01,0x00,0x00,0x90,0x01,0x00,0x00 (int[], Little Endian)  
string[] var_ss = {"TM", "", "Robot"}  
var_re = socket_send("ntd_a", var_ss)  
    // send 0x54,0x4D,0x52,0x6F,0x62,0x6F,0x74 (string[], UTF8) // var_ss[1] 為空字串，轉換值仍為空  
  
var_re = socket_send("ntd_a", 1000, 1, 2)        // 數值，起始位址與長度無效  
    // send 0xE8,0x03,0x00,0x00 (int, Little Endian)  
var_re = socket_send("ntd_a", "Hello, World"+newline, 0, 7)  
    // send 0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x20 (string, UTF8)  
byte[] var_bb = {100, 200}  
var_re = socket_send("ntd_a", var_bb)            // send 0x64,0xC8  
var_re = socket_send("ntd_a", var_bb, 1, 1)       // send 0xC8      // 陣列，從位址 1 起取 1 個，[1]=200  
var_re = socket_send("ntd_a", var_bb, -1, 1)       // send 0x64      // 陣列，從位址 0 起取 1 個，[0]=100  
var_re = socket_send("ntd_a", "達明機器人", 2)     // 字串，從位址 2 起取至結束，"機器人"  
    // send 0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA (string, UTF8)  
var_ss = {"TM", "", "達明機器人"}  
var_re = socket_send("ntd_a", var_ss)  
    // send 0x54,0x4D,0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA  
var_re = socket_send("ntd_a", Byte_Concat(GetBytes(var_ss), GetBytes(newline)))  
    // send 0x54,0x4D,0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA,0x0D,0x0A  
var_re = socket_send("ntd_a", var_ss, 2, 100)      // 陣列，從位址 2 起取 100 個(至結束)，[2]=達明機器人  
    // send 0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA
```

6.4 `socket_sendline()`

發送資料值至遠端裝置，且會自動在發送的資料尾端加上換行符號 `0x0D 0x0A`

語法 1

```
int socket_sendline(  
    string,  
    ?,  
    int,  
    int  
)
```

參數

<code>string</code>	Network 的裝置名稱 (在 Network Device 中設定)
<code>?</code>	發送資料值，可以是整數、浮點數、布林值、字串值或是陣列型別 數值將會使用 Little Endian 方式轉換，字串值將會使用 UTF8 方式轉換
<code>int</code>	發送資料值的起始索引 (對字串值或陣列型別有效) <code>0.. 長度-1</code> 合法值 <code><0</code> 非法使用，起始索引將設為 0 <code>>= 長度</code> 非法使用，起始索引將設為 0
<code>int</code>	發送資料值的長度 (對字串值或陣列型別有效) <code><= 0</code> 從起始索引，寫至資料結束 <code>>0</code> 從起始索引，寫入指定個數長度，最多寫至資料結束

傳回值

<code>int</code>	發送結果
<code>1</code>	發送成功
<code>0</code>	無法發送資料值為空字串或空陣列
<code>-1</code>	發送時發生 socket 異常
<code>-2</code>	無法連線至遠端裝置
<code>-3</code>	裝置名稱不存在或是 IP 或 Port 不正確

語法 2

```
int socket_sendline(  
    string,  
    ?,  
    int  
)
```

說明

與語法 1 相同，預設將發送資料值的長度填入 0

語法 3

```
int socket_sendline(  
    string,  
    ?  
)
```

說明

與語法 1 相同，預設將發送資料的起始索引填入 0 及發送資料值的長度填入 0

```
int var_re = socket_sendline("ntd_a", 200)      // send 0xC8,0x0D,0x0A  
var_re = socket_sendline("ntd_a", 2000)         // send 0xD0,0x07,0x00,0x00,0x0D,0x0A (int, Little Endian)  
var_re = socket_sendline("ntd_a", (float)0.234)  // send 0xB2,0x9D,0x6F,0x3E,0x0D,0x0A (float, Little Endian)  
var_re = socket_sendline("ntd_a", (double)0.234)  
    // send 0xC1,0xCA,0xA1,0x45,0xB6,0xF3,0xCD,0x3F,0x0D,0x0A (double, Little Endian)  
var_re = socket_sendline("ntd_a", "Hello, World"+newline)  
    // send 0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x20,0x57,0x6F,0x72,0x6C,0x64,0x0D,0x0A,0x0D,0x0A (string, UTF8)  
int[] var_ii = {100, 200, 300}  
var_re = socket_sendline("ntd_a", var_ii)  
    // send 0x64,0x00,0x00,0x00,0xC8,0x00,0x00,0x00,0x2C,0x01,0x00,0x00,0x0D,0x0A (int[], Little Endian)  
string[] var_ss = {"TM", "", "Robot"}  
var_re = socket_sendline("ntd_a", var_ss)  
    // send 0x54,0x4D,0x52,0x6F,0x62,0x6F,0x74,0x0D,0x0A (string[], UTF8) // var_ss[1] 為空字串，轉換值仍為空  
  
var_re = socket_sendline("ntd_a", 1000, 1, 2)      // 數值，起始位址與長度無效  
    // send 0xE8,0x03,0x00,0x00,0x0D,0x0A (int, Little Endian)  
var_re = socket_sendline("ntd_a", "Hello, World"+newline, 0, 7)  
    // send 0x48,0x65,0x6C,0x6C,0x6F,0x2C,0x20,0x0D,0x0A (string, UTF8)  
byte[] var_bb = {123, 234}  
var_re = socket_sendline("ntd_a", var_bb)        // send 0x7B,0xEA,0x0D,0x0A  
var_re = socket_sendline("ntd_a", var_bb, 1, 1)   // send 0xEA,0x0D,0x0A      // 陣列，從位址 1 起取 1 個  
var_re = socket_sendline("ntd_a", var_bb, -1, 1)  // send 0x7B,0x0D,0x0A      // 陣列，從位址 0 起取 1 個  
var_re = socket_sendline("ntd_a", "達明機器人", 2) // 字串，從位址 2 起取至結束，"機器人"  
    // send 0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA,0x0D,0x0A (string, UTF8)  
var_ss = {"TM", "", "達明機器人"}  
var_re = socket_sendline("ntd_a", var_ss)  
    // send 0x54,0x4D,0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA,0x0D,0x0A  
var_re = socket_sendline("ntd_a", Byte_Concat(GetBytes(var_ss), GetBytes(newline)))  
    // send 0x54,0x4D,0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA,0x0D,0x0A,0x0D,0x0A  
var_re = socket_sendline("ntd_a", var_ss, 2, -1)    // 陣列，從位址 2 起取至結束，[2]=達明機器人  
    // send 0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,0xE4,0xBA,0xBA,0x0D,0x0A
```

7. 參數化物件

參數化物件在使用上與使用者定義變數相同，唯可不須宣告直接可使用，並能在專案運行中，透過語法方式取得點位資料或修改點位資料，使手臂運行更具彈性。其表示法大致由三個部分組成(項目、索引、屬性)，其語法格式如下

參數化項目[索引].屬性

其中參數化項目支援

- | | |
|----------|--------|
| 1. 點位 | Point |
| 2. 座標系 | Base |
| 3. 工具中心點 | TCP |
| 4. 視覺初始點 | VPoint |
| 5. 輸出入 | IO |
| 6. 手臂狀態 | Robot |
| 7. 力規狀態 | FT |

索引及屬性，則再依參數化項目分別定義。

以讀寫 Point(項目) "P1"(索引) 的 點位座標(屬性) 為例，其中索引定義為點位名稱，屬性點位座標則定義為 float[] 型別 (與陣列的使用方式相同)，且具有 R/W (可讀可寫) 模式

Value float[] R/W 點位座標 {X,Y,Z,RX,RY,RZ}

讀值

```
float[] f = Point["P1"].Value                          // 點位項目內，索引定義為"點位名稱"，屬於 string 型別  
float f1 = Point["P1"].Value[0]                        // 或可單獨取出點位 "P1" 的 X 值
```

寫值

```
Point["P1"].Value = {0, 0, 90, 0, 90, 0}            // 將點位 "P1" 的座標修改至 {0,0,90,0,90,0}  
Point["P1"].Value[2] = 120                              // 或可單獨修改點位 "P1" 的 Z 值至 120
```

7.1 Point

語法

點位

Point[string].屬性

項目

Point 點位

索引

string 點位名稱 (在點位管理員內的點位名稱)

屬性

名稱	型別	讀寫	內容說明	內容格式
Value	float[]	R/W	點位座標	{X, Y, Z, RX, RY, RZ}, Size = 6
Pose	int[]	R/W	手臂姿態	{Config1, Config2, Config3}, Size = 3
Flange	float[]	R	法蘭中心座標	{X, Y, Z, RX, RY, RZ}, Size = 6
BaseName	string	R	座標系名稱	"Base Name"
TCPName	string	R	工具中心點名稱	"TCP Name"
TeachValue	float[]	R	點位座標(原教導點座標)	{X, Y, Z, RX, RY, RZ}, Size = 6
TeachPose	int[]	R	手臂姿態(原教導點姿態)	{Config1, Config2, Config3}, Size = 3

說明

```
// 讀值  
float[] f = Point["P1"].Value          // 取得點位 "P1" 的點位座標 {X, Y, Z, RX, RY, RZ}  
float f1 = Point["P1"].Value[0]         // 或單獨取出點位 "P1" 的 X 值  
float f1 = Point["P1"].Value[6]         // 報錯，超過陣列存取範圍  
string s = Point["P1"].BaseName        // s = "RobotBase"  
  
// 寫值  
Point["P1"].Value = {0, 0, 90, 0, 90, 0} // 將點位 "P1" 的點位座標修改至 {0,0,90,0,90,0}  
Point["P1"].Value[2] = 120               // 或單獨修改點位 "P1" 的 Z 值至 120  
Point["P1"].Flange = {0, 0, 90, 0, 90, 0} // 唯讀，無效操作  
Point["P1"].Value = {0, 0, 90, 0, 90}    // 報錯，寫入的陣列元素未符合 6 個 (寫 5 個)  
Point["P1"].Pose = {1, 2, 4, 0}          // 報錯，寫入的陣列元素未符合 3 個 (寫 4 個)
```

7.2 Base

語法

座標系

Base[string].屬性 或

Base[string, int].屬性

項目

Base 座標系

索引

string 座標系名稱 (在座標系管理員內的座標系名稱)

*系統座標系名稱，其所有屬性只具讀取模式，不具寫入模式

"RobotBase"

int 座標系索引，可指定由視覺 one shot get all 所建立出的多個座標系，值為 0..N，預設值為 0

屬性

名稱	型別	讀寫	內容說明	內容格式
Value	float[]	R/W	座標系值	{X, Y, Z, RX, RY, RZ}, Size = 6
Type	string	R	座標系種類	"R": Robot Base "V": Vision Base "C": Custom Base
TeachValue	float[]	R	座標系值(原教導座標系值)	{X, Y, Z, RX, RY, RZ}, Size = 6

說明

```
// 讀值

float[] f = Base["RobotBase"].Value                    // 取得座標系 "RobotBase" 的座標系值 {0,0,0,0,0,0}
float f1 = Base["base1"].Value[0]                    // 或單獨取出座標系 "base1" 的 X 值
string s = Base["base1"].Type                        // s = "C"
s = Base[Point["P1"].BaseName].Type                // s = "R"            // 假設 "P1" 座標系是採用 "RobotBase"
float[] f = Base["vision_osga",1].Value            // 取得座標系 "vision_osga" 中的第 2 個座標系值

// 寫值

Base["RobotBase"].Value = {0, 0, 90, 0, 90, 0}    // 唯讀，無效操作，因 "RobotBase" 為系統座標系
Base["base1"].Value = {0, 90, 0, 0, 90, 0}            // 將座標系 "base1" 修改為 {0,90,0,0,90,0}
Base["base1"].Value[4] = 120                          // 或單獨修改座標系 "base1" 的 RY 值至 120
Base["base1"].Value[6] = 120                          // 報錯，超過陣列存取範圍
Base["base1"].Type = "C"                            // 唯讀，無效操作
Base["base1"].Value = {0, 0, 90, 0, 90}            // 報錯，寫入的陣列元素未符合 6 個 (5 個)
Base["base1"].Value = {0, 0, 90, 0, 90, 0, 100}    // 報錯，寫入的陣列元素未符合 6 個 (7 個)
```

7.3 TCP

語法

工具中心點

TCP[string].屬性

項目

TCP 工具中心點

索引

string 工具中心點名稱 (在工具中心點列表內的工具中心點名稱)
*系統工具中心點名稱，其所有屬性只具讀取模式，不具寫入模式
 "NOTOOL"
 "HandCamera"

屬性

名稱	型別	讀寫	內容說明	內容格式
Value	float[]	R/W	工具中心點值	{X, Y, Z, RX, RY, RZ}, Size = 6
Mass	float	R/W	質量	Mass in kg
MOI	float[]	R/W	主轉動慣量(Principal Moments of Inertia)	{Ix, Iy, Iz}, Size = 3
MCF	float[]	R/W	質心主軸座標系相對於工具座標系的方位(Mass center frame with principle axes w.r.t tool frame)	{X, Y, Z, RX, RY, RZ}, Size = 6
TeachValue	float[]	R	工具中心點值(原 TCP 設定值)	{X, Y, Z, RX, RY, RZ}, Size = 6
TeachMass	float	R	質量(原 TCP 設定值)	Mass in kg
TeachMOI	float[]	R	主轉動慣量(Principal Moments of Inertia)(原 TCP 設定值)	{Ix, Iy, Iz}, Size = 3
TeachMCF	float[]	R	質心主軸座標系相對於工具座標系的方位(Mass center frame with principle axes w.r.t tool frame)(原 TCP 設定值)	{X, Y, Z, RX, RY, RZ}, Size = 6

說明

```
// 讀值  
float[] f = TCP["NOTOOL"].Value          // 取得工具名稱 "NOTOOL" 的工具中心點值 {0,0,0,0,0,0}  
float f1 = TCP["NOTOOL"].Value[0]           // 或單獨取出工具名稱 "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}  
  
// 寫值  
TCP["NOTOOL"].Value = {0, -10, 0, 0, 0, 0} // 唯讀，無效操作，因 "NOTOOL" 為系統工具中心點  
TCP["T1"].Value = {0, -10, 0, 0, 0, 0}       // 將工具中心點名稱 "T1" 修改為 {0,-10,0,0,0,0}  
TCP["T1"].Value[0] = 10                      // 或單獨修改 "T1" 的 X 值至 10  
TCP["T1"].Mass = 2.4                         // 將工具中心點名稱 "T1" 質量修改為 2.4 kg  
TCP["T1"].MOI = {0, 0, 0, 1, 2}               // 報錯，寫入的陣列元素未符合 3 個 (5 個)  
TCP["T1"].MCF = {0, -20, 0, 0, 0, 0}         // 報錯，寫入的陣列元素未符合 6 個 (7 個)
```

7.4 VPoint

語法

視覺起始點

VPoint[string].屬性

項目

VPoint 視覺起始點

索引

string 視覺任務名稱

屬性

名稱	型別	讀寫	內容說明	內容格式
Value	float[]	R/W	視覺初始點位座標	{X, Y, Z, RX, RY, RZ}, Size = 6
BaseName	string	R	座標系名稱	"Base Name"
TeachValue	float[]	R	視覺初始點位座標(原 Job 初 始點位座標)	{X, Y, Z, RX, RY, RZ}, Size = 6

說明

```
// 讀值
float[] f = VPoint["Job1"].Value           // 取得視覺任務 "Job1" 的初始點位座標 {X, Y, Z, RX, RY, RZ}
float f1 = VPoint["Job1"].Value[0]           // 或單獨取出視覺任務 "Job1" 的 X 值
float f1 = VPoint["Job1"].Value[6]           // 報錯，超過陣列存取範圍
string s = VPoint["Job1"].BaseName // s = "RobotBase"

// 寫值
VPoint["Job1"].Value = {0, 0, 90, 0, 90, 0} // 將視覺任務 "Job1" 的初始點座標修改至 {0,0,90,0,90,0}
VPoint["Job1"].Value[2] = 120                 // 或單獨修改視覺任務 "Job1" 的 Z 值至 120
VPoint["Job1"].BaseName = "base1"             // 唯讀，無效操作
VPoint["Job1"].Value = {0, 0, 90, 0, 90}       // 報錯，寫入的陣列元素未符合 6 個 (5 個)
VPoint["Job1"].Value = {0, 0, 90, 0, 90, 100} // 報錯，寫入的陣列元素未符合 6 個 (7 個)
```

7.5 IO

語法

輸出入

IO[string].屬性

項目

IO 輸出入

索引

string	控制模組名稱
ControlBox	電控箱
EndModule	末端模組
ExtModuleN	外部模組 (N = 0 .. n)
Safety	安全模組

屬性

ControlBox / EndModule / ExtModuleN

名稱	型別	讀寫	內容說明	內容格式
DI	byte[]	R	數位輸入	[0] = DI0 0: Low, 1: High [1] = DI1 [n] = DI _n
DO	byte[]	R/W	數位輸出	[0] = DO0 0: Low, 1: High [1] = DO1 [n] = DO _n
AI	float[]	R	類比輸入	-10.24V .. +10.24V (Voltage) [0] = AI0 [1] = AI1 [n] = AI _n
AO	float[]	R/W	類比輸出	-10.00V .. + 10.00V (Voltage) [0] = AO0 [1] = AO1 [n] = AO _n
InstantDO	byte[]	R/W	數位輸出(立即指令)	[0] = DO0 0: Low, 1: High [1] = DO1 [n] = DO _n
InstantAO	float[]	R/W	類比輸出(立即指令)	-10.00V .. + 10.00V (Voltage) [0] = AO0 [1] = AO1 [n] = AO _n

* DI[n]/DO[n]/AI[n]/AO[n] 的組數，會依實際的硬體裝置識別

Safety

名稱	型別	讀寫	內容說明	內容格式
SI	byte[]	R	安全功能輸入	0: Low, 1: High SI[0] = SF1 User Connected ESTOP input SI[1] = SF3 User Connected External Safeguard Input SI[2] = SF9 User Connected External Safeguard Input for Human-Machine Safety Setting SI[3] = SF15 User Connected Enabling Device Input SI[4] = SF16 User Connected ESTOP Input without Robot ESTOP Output
SO	byte[]	R	安全功能輸出	0: Low, 1: High SO[0] = SF10 Robot ESTOP Output SO[1] = SF11 User Connected External Safeguard Output SO[2] = SF12 User Connected External Safeguard Output for Human-Machine Safety Settings SO[3] = SF13 Robot Internal Protective Stop Output SO[4] = SF14 Robot Encoder Standstill Output

DO 與 InstantDO 差異

DO 在專案主流程中為預約排隊指令，若 DO 在機器人運動指令之後，如設有軌跡混合的 Point 節點之後，則 DO 會在 Point 節點完成之後，才會執行。若使用 InstantDO 指令，則會在執行 Point 節點時，就會執行 DO 指令，不會等待 Point 節點完成，其使用效果會如同在執行緒頁面中使用 DO 指令。

說明

```
// 讀值
byte[] di = IO["ControlBox"].DI          // 取得 "ControlBox" 的數位輸入狀態
int dilen = Length(di)                  // 配合陣列大小，可取得數位輸入的 PIN 數
byte di0 = IO["ControlBox"].DI[0]        // 取得 "ControlBox" DI[0] 的狀態
byte di32 = IO["ControlBox"].DI[32]       // 報錯，超過陣列存取範圍 (假設 DI 16 組，索引 0..15)
float[] ai = IO["ControlBox"].AI         // 取得 "ControlBox" 的類比輸入狀態
float[] ao = IO["ControlBox"].AO         // 取得 "ControlBox" 的類比輸出狀態
byte si0 = IO["Safety"].SI[0]            // 取得 "Safety" SI[0] 的安全功能輸入狀態
byte so4 = IO["Safety"].SO[4]             // 取得 "Safety" SO[4] 的安全功能輸出狀態
byte si1 = IO["ControlBox"].SI[1]         // 報錯，"ControlBox" 不支援 SI 屬性
byte di2 = IO["Safety"].DI[2]             // 報錯，"Safety" 不支援 DI 屬性
// 寫值
```

```
IO["ControlBox"].DI = {1,1,0,0}          // 唯讀，無效操作
IO["ControlBox"].DI[0] = 0                // 唯讀，無效操作
IO["ControlBox"].DO[2] = 1                // 設 DO 2 為 High
IO["ControlBox"].AO[0] = 3.3              // 設定 AO0 為 3.3V
IO["ControlBox"].DO = {1,1,0,0}          // 報錯，寫入的陣列元素未符合 (假設 DO 16 組，陣列元素需 16
個)
IO["ControlBox"].InstantDO[0] = 1        // 設 DO 0 為 High (立即執行)
IO["Safety"].SI[0] = 0                  // 唯讀，無效操作
IO["Safety"].SO[4] = 1                  // 唯讀，無效操作
IO["ControlBox"].SO[1] = 1                // 報錯，"ControlBox" 不支援 SO 屬性
IO["Safety"].DO[2] = 1                  // 報錯，"Safety" 不支援 DO 屬性
```

7.6 Robot

語法

手臂狀態

Robot[int].屬性

項目

Robot 手臂狀態

索引

int 手臂索引號，固定為 0 值

屬性

名稱	型別	讀寫	內容說明	內容格式
CoordRobot	float[]	R	手臂末端點 TCP 位置座標，相對於機械手臂座標系(RobotBase)	{X, Y, Z, RX, RY, RZ}, Size = 6
CoordBase	float[]	R	手臂末端點 TCP 位置座標，相對於機械手臂當前座標系	{X, Y, Z, RX, RY, RZ}, Size = 6
Joint	float[]	R	當前手臂關節角度	{J1, J2, J3, J4, J5, J6}, Size = 6
BaseName	string	R	當前座標系名稱	"Base Name"
TCPName	string	R	當前工具中心點名稱	"TCP Name"
CameraLight	byte	R/W	手臂相機採光	0: Low (Off), 1: High (On)
TCPForce3D	float	R	當前 TCP 合力 (RobotBase X,Y,Z 的合成力)	N
TCPSpeed3D	float	R	當前 TCP 速度 (RobotBase X,Y,Z 的合成速度)	mm/s

說明

```
// 讀值
float[] rtool = Robot[0].CoordRobot           // 取得當前位置座標 (手臂座標系)
float[] ftool = Robot[0].CoordBase            // 取得當前位置座標 (當前座標系)
float f = Robot[0].CoordBase[0]                // 或單獨取得當前位置座標的 X 值
f = Robot[0].CoordBase[6]                      // 報錯，超過陣列存取範圍
float[] joint = Robot[0].Joint                 // 取得當前手臂關節角度
float j = Robot[0].Joint[0]                    // 或單獨取得當前手臂第 1 軸關節角度值
string b = Robot[0].BaseName                  // b = "RobotBase"
string t = Robot[0].TCPName                   // t = "NOTOOL"
byte light = Robot[0].CameraLight             // light = 0 (OFF)
float tf3d = Robot[0].TCPForce3D              // tf3d = 1.234
float ts3d = Robot[0].TCPSpeed3D              // ts3d = 1.234
```

```
// 寫值  
Robot[0].CoordRobot = {0, 90, 0, 0, 0, 0} // 唯讀，無效操作  
Robot[0].CoordBase = {0, 0, 90, 0, 90, 0} // 唯讀，無效操作  
Robot[0].BaseName = "Base1" // 唯讀，無效操作  
Robot[0].TCPName = "Tool1" // 唯讀，無效操作  
Robot[0].CameraLight = 1 // 開啟手臂相機採光  
Robot[0].CameraLight = 0 // 關閉手臂相機採光  
Robot[0].TCPSpeed3D = 1.234 // 唯讀，無效操作
```

7.7 FT

語法

力規狀態

FT[string].屬性

項目

FT 力規狀態

索引

string 力規裝置名稱 (在力規裝置管理內的裝置名稱)

屬性

名稱	型別	讀寫	內容說明	內容格式
X	float	R	X 軸力量數值	
Y	float	R	Y 軸力量數值	
Z	float	R	Z 軸力量數值	
TX	float	R	X 軸力矩數值	
TY	float	R	Y 軸力矩數值	
TZ	float	R	Z 軸力矩數值	
F3D	float	R	XYZ 合力	
T3D	float	R	XYZ 合力矩	
ForceValue	float[]	R	XYZ 力量值陣列	{X, Y, Z}, Size = 3
TorqueValue	float[]	R	XYZ 力矩值陣列	{TX, TY, TZ}, Size = 3
RefCoordX	float	R	根據節點中設定的參考座標系，所量測得到的 X 軸力量數值	
RefCoordY	float	R	根據節點中設定的參考座標系，所量測得到的 Y 軸力量數值	
RefCoordZ	float	R	根據節點中設定的參考座標系，所量測得到的 Z 軸力量數值	
RefCoordTX	float	R	根據節點中設定的參考座標系，所量測得到的 X 軸力矩數值	
RefCoordTY	float	R	根據節點中設定的參考座標系，所量測得到的 Y 軸力矩數值	
RefCoordTZ	float	R	根據節點中設定的參考座標系，所量測得到的 Z 軸力矩數值	
RefCoordF3D	float	R	根據節點中設定的參考座標系，所量測得到的 XYZ 合力	
RefCoordT3D	float	R	根據節點中設定的參考座標系，所量測得到的 XYZ 合力矩	

RefCoordForceValue	float[]	R	根據節點中設定的參考座標系，所量測得到的 XYZ 力量值矩陣	{RefCoordX, RefCoordY, RefCoordZ}, Size = 3
RefCoordTorqueValue	float[]	R	根據節點中設定的參考座標系，所量測得到的 XYZ 力矩值矩陣	{RefCoordTX, RefCoordTY, RefCoordTZ}, Size = 3
Model	string	R	力規裝置型號	
Zero	byte	R/W	開啟/關閉力規偏移值	0: Zero OFF, 1: Zero ON

說明

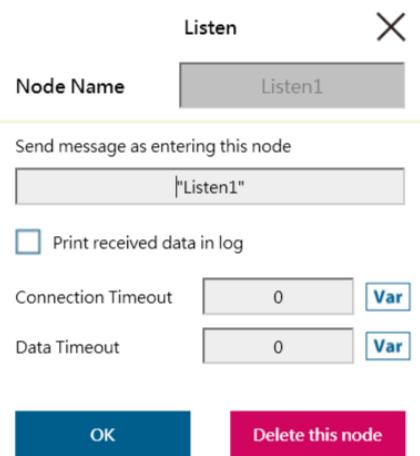
```
// 讀值
float x = FT["fts1"].X           // 取得 "fts1" 力規當前 X 軸力量值
float tx = FT["fts1"].TX          // 取得 "fts1" 力規當前 X 軸力矩值
float f3d = FT["fts1"].F3D         // 取得 "fts1" 力規當前 XYZ 合力值
float[] force = FT["fts1"].ForceValue // 取得 "fts1" 力規當前 {X,Y,Z} 值
string mode = FT["fts1"].Model      // 取得 "fts1" 力規型號

// 寫值
FT["fts1"].Y = 3.14             // 唯讀，無效操作
FT["fts1"].TY = 1.34              // 唯讀，無效操作
FT["fts1"].T3D = 4.13              // 唯讀，無效操作
FT["fts1"].TorqueValue = {1.1, 2.2, 3.3} // 唯讀，無效操作
FT["fts1"].Zero = 1                // 記錄當前力規偏移值
```

8. 外部命令

8.1 Listen 節點

在 Listen 節點內，可建立 Socket TCPListener (Server site) 與外部裝置連接後，依照封包格式進行通訊，則可從這份文件所提列的所有功能，藉由此通訊來執行。



1. Send Message: 當進入此節點時，會主動發出訊息
2. Print Log: 啟用通訊 Log (顯示於右側)
3. Connection Timeout: 當進入此節點後，超過多少時間(毫秒)
未連線時，將會逾時
若 $<= 0$ 沒有逾時
4. Data Timeout: 當連接後，超過多少時間(毫秒)
沒有通訊封包時，將會逾時
若 $<= 0$ 沒有逾時

Socket TCPListener 是跟著專案運行後建立，並於專案停止後關閉，當成功建立起 TCP Listener 時，其所建立的 IP 及 Port 會顯示於右側的 Notice Log 視窗。

IP HMI → System → Network → IP Address

Port 5890

當流程進入 Listen 節點時，將進入外部 Script 控制模式，且會停留在 Listen 節點內，直到觸發下列出口條件，才依條件而離開。

Pass: 執行 `ScriptExit()` 或專案停止

- Fail:**
1. 發生 Connection Timeout
 2. 發生 Data Timeout
 3. 當 TCP Listener 尚未建立成功，流程即走進 Listen 節點

藉由 Listen 節點接收的指令會被檢查後依照內容執行。若指令不合法會回傳錯誤訊息，並標示出錯誤行數；若指令合法便會依據指令內容執行。

指令可以分為兩類，第一類為可立即完成的指令，如變數計算；第二類為必須依序消化執行的指令，包含運動指令與 DO/AO 設置。第二類指令會以預約的形式被執行，預約的指令會累積在序列中並依照順序被依序執行。

8.2 ScriptExit()

退出外部 Script 控制模式

語法 1

```
bool ScriptExit()  
)
```

參數

void 無輸入值

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

退出外部 Script 控制模式，等待命令執行完後，離開 Listen 節點並走 Pass 路線

* 透過 TMSCT 通訊封包執行

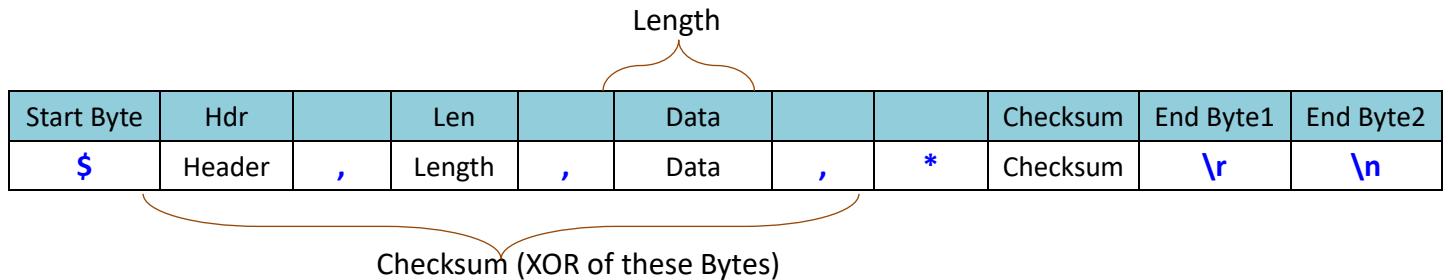
* 在 ScriptExit() 之後的語法，不會執行，如

```
< $TMSCT,78,2,ChangeBase("RobotBase")\r\n  
    ChangeTCP("NOTOOL")\r\n  
    ScriptExit()\r\n                // 退出外部 Script 控制模式  
    ChangeLoad(10.1),*6C\r\n                // ChangeLoad 將不會執行
```

* 退出 Script 模式後，需再等所有的語法命令執行完之後(包含手臂運動指令到位)，才會離開

Listen 節點並走 Pass 路線，在等待離開 Listen 節點期間，屬於未進入外部 Script 控制模式，因此將不再接受任何外部命令，且回應 CPERR 錯誤封包

8.3 通訊協定



Name	Size	ASCII	HEX	Description
Start Byte	1	\$	0x24	Start Byte for Communication
Header	X			Header for Communication
Separator	1	,	0x2C	Separator between Header and Length
Length	Y			Length of Data
Separator	1	,	0x2C	Separator between Length and Data
Data	Z			Communication Data
Separator	1	,	0x2C	Separator between Data and Checksum
Sign	1	*	0x2A	Begin Sign of Checksum
Checksum	2			Checksum of Communication
End Byte 1	1	\r	0x0D	
End Byte 2	1	\n	0x0A	End Byte of Communication

1. Header

定義通訊封包的用途，不同的 Header 對於通訊封包及 Data 的定義都會有所不同。

- **TMSCT** 定義外部 Script 功能
- **TMSTA** 定義狀態或屬性值取得
- **CPERR** 定義通訊封包錯誤 (如 Packet 錯誤、Checksum 錯誤、Header 錯誤 ...等)

2. Length

長度指出 Data 所佔用的 UTF8 bytes 長度，可以使用十進制、十六進制、或二進制的書寫方式，最大表示長度為 int 32 bits。

如 \$TMSCT,**100**,Data,*CS\r\n // 十進制 100，指出 Data 長度為 100 bytes
 \$TMSCT,**0x100**,Data,*CS\r\n // 十六進制 0x100，指出 Data 長度為 256 bytes
 \$TMSCT,**0b100**,Data,*CS\r\n // 二進制 0b100，指出 Data 長度為 4 bytes
 \$TMSCT,**8,1,達明**,*58\r\n // 指出 Data 1,達明 長度為 8 bytes (UTF8)

3. Data

通訊封包內容，可以支援任意字元（包含 0x00 .. 0xFF 並採用 UTF8 編碼），資料長度可依據 Length 來決定。而 Data 內所定義用途與說明，則須再依據 Header 定義。

4. Checksum

通訊封包的校驗碼，計算方式採用 XOR (exclusive OR)，計算範圍為 \$ 與 * 之間的所有 Bytes (不包含 \$ 與 *)，如下表示

\$**TMSCT,100,Data**,*CS\r\n

Checksum = Byte[1] ^ Byte[2] ... ^ Byte[N-6]

其中 Checksum 的書寫方式，固定為 2 bytes，且採用十六進制方式書寫（但不書寫 0x）。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)

8.4 TMSCT

Start Byte	Hdr		Len		Data			Checksum	End Byte1	End Byte2
\$	TMSCT	,	Length	,	Data	,	*	Checksum	\r	\n
ID				SCRIPT						
Script ID				,				Script Language		

TMSCT 定義為 External Script Language 使用，其中對於封包的 Data 區段再分成兩個部分，一是為 ID、另一是為 SCRIPT，兩個部分依分隔符號逗號隔開，說明如下

- ID 定義 Script ID，可用任意的英數字表示（若遇到非英數字的 byte 時，將會回報 CPERR 04錯誤），做為通訊封包回應時，一個識別回應哪組 SCRIPT 的識別碼。
- ,
- 分隔符號
- SCRIPT 任意 Script Language 內容，且一次通訊封包內，可以帶有多行 Script 內容，行與行之間則可透過換行符號 (0x0D 0x0A) 區分。

說明

TMSCT 需進入外部 Script 控制模式時才可使用，否則將回應 CPERR 錯誤封包

發送 (手臂→外部)

- 進入 Listen 節點時，手臂會主動對當前連接的外部裝置發送訊息，其 ID 固定為 0

\$TMSCT,9,0,Listen1,*4C\r\n

9 指出 0,Listen1 的長度共 9 bytes

0 指出 Script ID 為 0

,

分隔符號

Listen1 發送的訊息內容

- 依 Script 內容的正確性，會回應 OK 或 ERROR，若後面還帶有 ;N 表示為警告行數或是錯誤行數。手臂在接收訊息後，若 Script 內容語法正確(或警告)，會開始執行 Script 內容，且執行完後，才回應外部裝置。若 Script 內容語法錯誤，則不會執行 Script 內容，會立即回應錯誤。

\$TMSCT,4,1,OK,*5C\r\n // 回應 ID 1

// OK 表示 Script 正確

\$TMSCT,8,2,OK;2;3,*52\r\n // 回應 ID 2

// OK;2;3 表示 Script 正確，但有警告行數第 2 行與第 3 行

\$TMSCT,13,3,ERROR;1;2;3,*3F\r\n // 回應 ID 3

接收 (手臂←外部)

- 當手臂進入 Listen 節點時，可接收外部發送 Script 內容，並檢查與執行 Script 內容。若手臂尚未進入 Listen 節點(未進入外部 Script 控制模式)即接收到 Script 內容，都將會丟棄不處理且回應 CPERR 錯誤封包。
- 外部發送訊息時，需定義 Script ID，使手臂在回應時，可依據所接收到的 Script ID 進行回應。

```
< $TMSCT,25,1,ChangeBase("RobotBase"),*08\r\n      // 定義 ID 1
> $TMSCT,4,1,OK,*5C\r\n                         // 回應 ID 1
```

- 在一次的通訊封包內，可以帶有多行 Script 內容，行與行之間透過換行符號分開。

```
< $TMSCT,64,2,ChangeBase("RobotBase")\r\n
    ChangeTCP("NOTOOL")\r\n
    ChangeLoad(10.1),*68\r\n      // 一次發送三行 Script 內容(行與行之間用\r\n 分開)
> $TMSCT,4,2,OK,*5F\r\n
```

- 在 Listen 節點內，可支援區域變數使用，且在離開 Listen 節點之前都仍保持有效性。

```
< $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      // 因為 int var_i 已經定義過，所以報錯
```

- 在 Listen 節點內，可取得專案內的變數使用(讀取或修改)，但無法建立專案內的新變數，因其所建立的變數是在 Listen 節點使用的區域變數。

8.5 TMSTA

Start Byte	Hdr		Len		Data			Checksum	End Byte1	End Byte2
\$	TMSTA	,	Length	,	Data	,	*	Checksum	\r	\n
SubCmd										
SubCmd					

TMSTA 定義為取得狀態或屬性值使用，其中對於封包的 Data 區段會以不同的 SubCmd 做為子命令，再依照每個不同的 SubCmd 定義而有不同的封包格式，定義如下，

SubCmd

- 00 是否進入外部 Script 控制模式
- 01 設定的 QueueTag 編號是否完成
- 90..99 資料訊息發送(資料格式可自定義)

說明

TMSTA 不需進入外部 Script 控制模式即可使用

SubCmd 00 是否進入外部 Script 控制模式

格式

發送 (手臂→外部)

SubCmd		Entry		Message
00	,	false	,	
00	,	true	,	message

接收 (手臂←外部)

SubCmd
00

發送 (手臂→外部)

- 若未進入外部 Script 控制模式，則會回應 false

\$TMSTA,9,00,false,*37\r\n

9 指出 00,false, 的長度共 9 bytes
 00 指出 SubCmd 00
 , 分隔符號
 false 未進入外部 Script 控制模式
 , 分隔符號
 空字串 (因未進入外部控制模式)

- 若進入外部 Script 控制模式，則會回應 true

\$TMSTA,15,00,true,Listen1,*79\r\n

15 指出 00,true,Listen1 的長度共 15 bytes
 00 指出 SubCmd 00
 , 分隔符號
 true 進入外部 Script 控制模式
 , 分隔符號
 Listen1 進入 Listen 節點時會發送的內容 (表示目前處於此 Listen1 內)

接收 (手臂←外部)

- 外部裝置發送給手臂

\$TMSTA,2,00,*41\r\n

2 指出 00 的長度共 2 bytes
 00 指出 SubCmd 00 是否進入外部 Script 控制模式

SubCmd 01 設定的 QueueTag 編號是否完成

格式

發送 (手臂→外部)

SubCmd		Tag Number		Status
01	,	01 .. 15	,	true/false/none

接收 (手臂←外部)

SubCmd		Tag Number
01	,	01 .. 15

說明

使用 TMSTA 01 詢問時，可查詢最後四個 Tag Number 的狀態

發送 (手臂→外部)

1. 手臂發送給外部裝置，在完成 QueueTag 編號時，會主動發送

\$TMSTA,10,01,08,true,*6D\r\n

10 指出 01,00,true 的長度共 10 bytes
 01 指出 SubCmd 01 發送 Tag Number 狀態
 , 分隔符號
 08 Tag Number 08
 , 分隔符號
 true true 表示 Tag Number 已完成
 false 表示 Tag Number 未完成
 none 表示 Tag Number 不存在

接收 (手臂←外部)

1. 外部裝置發送給手臂，可查詢最後四個 Tag Number 狀態

\$TMSTA,5,01,15,*6F\r\n

5 指出 01,88 的長度共 5 bytes
 01 指出 SubCmd 01 詢問 Tag Number 狀態
 , 分隔符號
 15 Tag Number 15

> \$TMSTA,10,01,15,none,*7D\r\n // TagNumber 15 不存在

2. Tag Number 使用值為 1 .. 15 的整數值，若數值為非法值，則傳回 none 不存在

\$TMSTA,5,01,88,*6B\r\n

> \$TMSTA,10,01,88,none,*79\r\n // TagNumber 88 不存在

SubCmd 90 .. 99 資料訊息發送

格式

發送 (手臂→外部)

SubCmd		Data
90 .. 99	,	...

接收 (手臂←外部)

None

說明

1. 使用 TMSTA 90 .. 99 發送時，可自定義 Data 的發送格式
2. 自定義是指由專案流程與外部裝置雙方自行定義 Data 格式
3. 為增強使用彈性，可利用 90 .. 99 不同的 SubCmd 去定義不同的發送格式，如

SubCmd 90 定義為 string；

SubCmd 91 定義為 float[]；

SubCmd 92 定義為 byte[]

...

等，使外部裝置在進行接收解析時，可依照 SubCmd 進行不同的解析方式

發送 (手臂→外部)

1. 手臂發送給外部裝置，當外部 Script 中執行 ListenSend() 命令時，將會發送資料

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

```
float[] var_f = {1,2,3,4}
```

```
byte[] var_b = {0x10, 0x11, 0x12, 0x13}
```

```
ListenSend(90, var_s)
```

```
// 通訊內容 $TMSTA,14,90,Hello World,*73\r\n
```

```
// 0x39,0x30,0x2C,0x48,0x65,0x6C,0x6C,0xF,0x20,0x57,0x6F,0x72,0x6C,0x64
```

```
ListenSend(91, var_f)
```

```
// 通訊內容 $TMSTA,19,91,...,*60\r\n
```

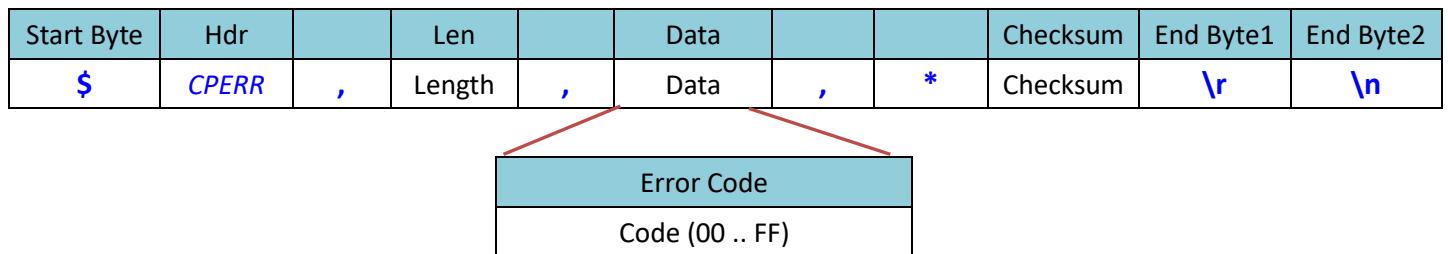
```
// 0x39,0x31,0x2C,0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40,0x00,0x00,0x80,0x40
```

```
ListenSend(92, var_b)
```

```
// 通訊內容 $TMSTA,7,92,...,*63\r\n
```

```
// 0x39,0x32,0x2C,0x10,0x11,0x12,0x13
```

8.6 CPERR



CPERR 定義為 Communication Protocol Error 使用，其中對於封包的 Data 區段則定義為 Error Code。

Error Code 定義錯誤碼，固定為 2 bytes，且採用十六進制方式書寫 (但不書寫0x)

- 00 封包正確，無錯誤。(通常會回應相應的封包內容，而不會回應封包無錯誤)
- 01 Packet Error 封包格式錯誤。
- 02 Checksum Error 封包校驗碼錯誤。
- 03 Header Error 封包標頭錯誤。
- 04 Packet Data Error 封包資料錯誤。
- F1 未進入外部 Script 控制模式。

說明

常使用於手臂回應外部裝置

發送 (手臂→外部)

01 Packet Error

```
< $TMSCT,-100,1,ChangeBase("RobotBase"),*13\r\n // Length 不該是負值  
> $CPERR,2,01,*49\r\n // CPERR Error Code 01
```

02 Checksum Error

```
< $TMSCT,25,1,ChangeBase("RobotBase"),*09\r\n // 09 不是正確的 Checksum  
> $CPERR,2,02,*4A\r\n // CPERR Error Code 02
```

03 Header Error

```
< $TMsc,25,1,ChangeBase("RobotBase"),*28\r\n // TMsc 不是正確的 Header  
> $CPERR,2,03,*4B\r\n // CPERR Error Code 03
```

04 Packet Data Error

```
< $TMSCT,23,ChangeBase("RobotBase"),*13\r\n // TMSCT 缺少 Script ID  
> $CPERR,2,04,*4C\r\n // CPERR Error Code 04
```

04 Packet Data Error

```
< $TMSTA,4,XXXX,*47\r\n // TMSTA 沒有 XXXX 這個 SubCmd  
> $CPERR,2,04,*4C\r\n // CPERR Error Code 04
```

F1 No External Script Mode

```
< $TMSCT,25,1,ChangeBase("RobotBase"),*0D\r\n // 假設當前未進入外部 Script 控制模式  
> $CPERR,2,F1,*3F\r\n // CPERR Error Code F1
```

8.7 優先命令

因所有的 Expression 語法，都是依序執行，若有使用到等待語法，例如 QueueTag(1, 1) 或 WaitQueueTag(1) 等待標籤編號到達的語法，程序將會停留直到條件成立才會繼續執行，因此若在等待的過程中，收到外部發送的 ScriptExit() 語法，將無法退出外部 Script 控制模式（因為程序還停留在等待條件成立）

當程序進入 Listen 節點（外部 Script 控制模式）時，除可使用依序執行的 Expression 語法外，尚多增加可使用優先命令，其執行優先權較高於 Listen 節點中的語法執行，因此命令會被立即執行，定義如下

1. **ScriptExit(0)**

立即停止手臂運動並清除緩衝區內的手臂運動指令，且退出外部 Script 控制模式，在離開 Listen 節點之後走 Fail 路線

2. **ScriptExit(1)**

立即停止手臂運動並清除緩衝區內的手臂運動指令，且退出外部 Script 控制模式，在離開 Listen 節點之後走 Pass 路線

3. **StopAndClearBuffer(0)**

立即停止手臂運動並清除緩衝區內的手臂運動指令

4. **StopAndClearBuffer(1)**

立即停止手臂運動並清除緩衝區內的手臂運動指令，退出當前正在執行的 Script 程序，且接續執行下一個 Script 程序

5. **StopAndClearBuffer(2)**

立即停止手臂運動並清除緩衝區內的手臂運動指令，退出當前正在執行的 Script 程序，且清除已接收 Script 緩衝區內的所有 Script 程序

- 優先命令僅支援完整使用命令定義，不支援配合變數或函式使用，如

```
< $TMSCT,42,1,int var_st=2\r\n
    StopAndClearBuffer(var_st),*3A\r\n
> $TMSCT,9,1,ERROR,2,*04\r\n          // 錯誤的語法 StopAndClearBuffer(var_st)
```

- 優先命令若與 Expression 語法混合使用時，則只會執行優先命令，不會執行語法

```
< $TMSCT,94,2,float[] targetP1= {0,0,90,0,90,0}\r\n      // 不會執行
    PTP("JPP",targetP1,10,200,0,false)\r\n                  // 不會執行
    StopAndClearBuffer(0),*75\r\n                            // 優先執行 StopAndClearBuffer(0)
> $TMSCT,4,2,OK,*5F\r\n
```

- 在一個外部 Script 封包下，只會執行一個優先命令，若同時有多個優先命令，則 ScriptExit(0/1) 優於 StopAndClearBuffer(0/1/2) 先處理，若 ScriptExit 或 StopAndClearBuffer 分別有多個，則只執行第 1 個。

```
< $TMSCT,46,3,StopAndClearBuffer(2)\r\n // 會執行 // 有多個 StopAndClearBuffer 將執行第 1 個
    StopAndClearBuffer(1),*68\r\n           // 不會執行
> $TMSCT,4,3,OK,*5E\r\n

< $TMSCT,61,4,StopAndClearBuffer(2)\r\n // 不會執行
    StopAndClearBuffer(1)\r\n            // 不會執行
    ScriptExit(1),*52\r\n              // 會執行 // 因 ScriptExit 優於 StopAndClearBuffer 先處理
> $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，程序會停留在此指令，直到標籤 1 完成
< $TMSCT,15,2,ScriptExit(0),*55\r\n    // 若標籤 1 未完成時，收到 ScriptExit(0) 命令
> $TMSCT,4,1,OK,*5C\r\n    // 回應 1 OK
                           // 但因運動指令及 QueueTag() 被清除，因此不會傳出 TMSTA 標籤編號完成
> $TMSCT,4,2,OK,*5F\r\n    // 回應 2 OK

```

將會退出外部 Script 控制模式，離開 Listen 節點後，並走 Fail 路線

```

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，程序會停留在此指令，直到標籤 1 完成
< $TMSCT,23,2,StopAndClearBuffer(0),*55\r\n    // 若標籤 1 未完成時，收到 StopAndClearBuffer(0) 命令
> $TMSCT,4,1,OK,*5C\r\n    // 回應 1 OK
                           // 但因運動指令及 QueueTag() 被清除，因此不會傳出 TMSTA 標籤編號完成
> $TMSCT,4,2,OK,*5F\r\n    // 回應 2 OK

```

未退出外部 Script 模式，仍在 Listen 節點內

```

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          // 編號 0 將做等待逾時 60 秒
   PTP("JPP",targetP1,40,200,0,false),*64\r\n // 需等待 60 秒之後才會執行
< $TMSCT,23,2,StopAndClearBuffer(0),*55\r\n
   // 若在 60 秒內收到 StopAndClearBuffer(0) 命令，將會清除第 1 筆 PTP() 函式
> $TMSCT,4,2,OK,*5F\r\n    // 回應 2 OK

```

當等待逾時 60 秒過後，才會執行第 2 筆 PTP() 函式，且回應 1 OK

> \$TMSCT,4,1,OK,*5C\r\n // 回應 1 OK

* StopAndClearBuffer(0) 只清除運動指令，若是函式語法或邏輯運算則不會清除，而 WaitQueueTag(0) 設定編號 0 時，將只做等待逾時，因此無法透過清除運動指令退出，需使用 StopAndClearBuffer(1/2) 命令退出當前執行的 Script 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，程序會停留在此指令，直到標籤 1 完成
< $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          // 前一個封包仍在等待，這個封包尚未執行
< $TMSCT,23,3,StopAndClearBuffer(1),*55\r\n    // 若標籤 1 未完成時，收到 StopAndClearBuffer(1) 命令
> $TMSCT,4,1,OK,*5C\r\n    // 回應 1 OK
                           // 但因運動指令及 QueueTag() 被清除，因此不會傳出 TMSTA 標籤編號完成
> $TMSCT,4,3,OK,*5E\r\n    // 回應 3 OK
清除並退出當前正在執行的 Script 1，接續執行下一個 Script 2。
> $TMSCT,4,2,OK,*5F\r\n    // 回應 2 OK
> $TMSTA,10,01,02,true,*67\r\n // 標籤編號 2 完成

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，程序會停留在此指令，直到標籤 1 完成
< $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          // 前一個封包仍在等待，這個封包尚未執行
< $TMSCT,23,3,StopAndClearBuffer(2),*56\r\n    // 若標籤 1 未完成時，收到 StopAndClearBuffer(2) 命令
> $TMSCT,4,1,OK,*5C\r\n    // 回應 1 OK
                           // 但因運動指令及 QueueTag() 被清除，因此不會傳出 TMSTA 標籤編號完成
> $TMSCT,4,3,OK,*5E\r\n    // 回應 3 OK
清除並退出當前正在執行的 Script 1，且清除已接收 Script 緩衝區內的所有 Script 程序，因此 Script 2 被清除，也不會回應 Script 2

```

9. 機器人運動及視覺任務函式

機器人運動函式及視覺任務函式，限制必須使用於外部命令 (External Script) 下，即專案流程必須進入 Listen 節點，且進入外部 Script 控制模式時，才可使用機器人運動函式 (Robot Motion) 或視覺任務函式 (Vision Do Job)。

所有的機器人運動函式或視覺任務函式皆會暫存於序列之中並依序被執行。

9.1 QueueTag()

設定機器人運動指令的執行標籤編號 (Queue Tag Number)，可利用來判斷目前的機器人運動指令執行到哪一個，並配合 TMSTA SubCmd 01 封包做為判斷

語法 1

```
bool QueueTag(  
    int,  
    int  
)
```

參數

int	標籤編號，合法值為 1..15 的整數值
int	是否等待標籤號完成，再向下執行
0	不等待 (預設)
1	等待

當使用等待時，則程序會停留在此指令內，等待標籤完成後，才繼續向下執行

傳回值

bool	設定成功回傳 True，設定失敗回傳 False
* 標籤編號可重複使用	

語法 2

```
bool QueueTag(  
    int  
)
```

說明

與語法 1 相同，預設不等待標籤編號完成，使程序繼續向下執行。

QueueTag(int, int) => QueueTag(int, 0)

9.2 WaitQueueTag()

等待機器人運動指令的執行標籤編號 (Queue Tag Number) 完成

語法 1

```
int WaitQueueTag(  
    int,  
    int  
)
```

參數

int	等待的標籤編號
1..15	合法的標籤編號
0	無效的標籤編號，但會等待逾時 (若等待逾時設為無限期等待，則不會等待逾時)
<0	不存在的標籤編號，不會等待逾時
>15	不存在的標籤編號，不會等待逾時
int	設定等待逾時
< 0	無限期等待，當等待標籤為合法 1..15 時有效 (預設)
= 0	等待檢查一次
> 0	等待多少時間後逾時 (毫秒)

當使用等待時，則程序會停留在此指令內，直到標籤完成或不存在，或是等待逾時，才會繼續向下執行

傳回值

int	傳回等待結果
1	標籤編號完成
0	標籤編號未完成，或等待逾時
-1	標籤編號不存在

* 標籤編號可重複使用

* 標籤編號會保留最後四筆的標籤狀態，若等待的標籤編號未發生，或是已超過最後四筆之前，則會傳回不存在

語法 2

```
int WaitQueueTag(  
    int  
)
```

說明

與語法 1 相同，預設無限期等待，需等待標籤編號完成(或不存在)

WaitQueueTag(int, int) => **WaitQueueTag(int, -1)**

● 運動指令標籤編號

運動指令標籤編號，主要用來與機器人運動指令配合使用。由於所有的運動指令皆會暫存於序列之中並依序被執行，配合標籤編號使用，可以得知當前的運動指令執行至哪一個。

```
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，程序會再向下執行
   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，程序會再向下執行
   ,*49\r\n
```

當執行 Script 內容後，因為 QueueTag() 並未等待，程序執行完後傳回

```
> $TMSCT,4,2,OK,*5F\r\n
```

當機器人運動執行完 PTP() targetP1 後，因設定 QueueTag(1) 將傳回

```
> $TMSTA,10,01,01,true,*64\r\n // TMSTA SubCmd 01, TagNumber 01, 已完成
```

當機器人運動執行完 PTP() targetP2 後，因設定 QueueTag(2) 將傳回

```
> $TMSTA,10,01,02,true,*67\r\n // TMSTA SubCmd 01, TagNumber 02, 已完成
```

```
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，程序會停留在此指令，直到標籤 3 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，程序會再向下執行
   ,*56\r\n
```

當執行 Script 內容後，因為 QueueTag(3,1) 設定等待，程序將會停留，直到標籤 3 完成後傳回

```
> $TMSTA,10,01,03,true,*66\r\n // TMSTA SubCmd 01, TagNumber 03, 已完成
```

當 QueueTag(3) 完成後，程序繼續向下執行，因 QueueTag(4) 設定不等待，程序執行完後傳回

```
> $TMSCT,4,2,OK,*5F\r\n
```

當機器人運動執行完 PTP() targetP2 後，因設定 QueueTag(4) 將傳回

```
> $TMSTA,10,01,04,true,*61\r\n // TMSTA SubCmd 01, TagNumber 04, 已完成
```

* \$TMSCT,4,2,OK 是在程序執行完該筆 Script 之後，才會傳回，因此若使用 QueueTag 等待，或用 WaitQueueTag 等待，則也會在等待結束後才傳回。

9.3 StopAndClearBuffer()

停止手臂運動並清除存在緩衝區內的手臂運動指令

語法 1

```
bool StopAndClearBuffer()  
)
```

參數

void 無輸入值

傳回值

bool 設定成功回傳**True**，設定失敗回傳**False**

說明

StopAndClearBuffer()

9.4 Pause()

暫停專案與手臂運動 (除了 NonPaused Thread 與外部 Script 通訊行為不暫停)，可以使用 Resume() 或是按 Stick Play 鍵恢復

語法 1

```
bool Pause()  
)
```

參數

void 無輸入值

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

Pause()

9.5 *Resume()*

恢復專案運行與手臂運動

語法 1

```
bool Resume(  
)
```

參數

void 無輸入值

傳回值

bool 設定成功回傳**True**，設定失敗回傳**False**

說明

Resume()

9.6 PTP()

設定 PTP 絕對位置運動參數並下達運動指令。

語法 1

```
bool PTP(  
    string,  
    float[],  
    int,  
    int,  
    int,  
    bool  
)
```

參數

string 定義運動參數格式，由三個字母組成

第一字母：運動目標表示方式：

"J" 以角度表示

"C" 以座標表示

第二字母：速度表示方式：

"P" 以百分比表示

第三字母：軌跡混合表示方式

"P" 以百分比表示

float[] 運動目標。若以角度表示，為手臂關節的六個元素：關節1(°)，關節2(°)，關節3(°)，關節

4(°)，關節5(°)，關節6(°)；若以座標表示，為手臂末端工具中心點座標的六個元素：

X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)

int 手臂末端移動速度百分比(%)

int 運動加速到最高速所花費時間(millisecond)

int 軌跡混合百分比(%)

bool 取消精準到位

true 取消精準到位

false 精準到位

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

運動參數格式表示方式有：(1) "JPP" 與(2) "CPP"

```
float[] targetP1= {0,0,90,0,90,0} //宣告浮點數陣列儲存移動目標座標  
PTP("JPP",targetP1,10,200,0,false) //以PTP、速度10%、加速時間200ms運動至目標
```

語法 2

```
bool PTP(  
    string,  
    float[],  
    int,  
    int,  
    int,  
    bool,  
    int[]  
)
```

參數

`string` 定義運動參數格式，由三個字母組成

第一字母：運動目標表示方式：

"C" 以座標表示

第二字母：速度表示方式：

"P" 以百分比表示

第三字母：軌跡混合表示方式：

"P" 以百分比表示

`float[]` 運動目標。以座標表示，為手臂末端工具中心點座標的六個元素：X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)

`int` 手臂末端移動速度百分比(%)

`int` 運動加速到最高速所花費時間(millisecond)

`int` 軌跡混合百分比(%)

`bool` 取消精準到位

`true` 取消精準到位

`false` 精準到位

`int[]` 手臂姿態定義：Config1, Config2, Config3，手臂姿態定義詳見附註

傳回值

`bool` 設定成功回傳 `True`，設定失敗回傳 `False`

說明

運動參數格式表示方式有：(1) "CPP"

```
float[] targetP1 = {417.50,-122.30,343.90,180.00,0.00,90.00}      //宣告浮點數陣列儲存移動目標座標  
int[] pose = {0,2,4}                                              //宣告整數陣列儲存移動目標姿態  
PTP("CPP",targetP1,50,200,0,false,pose)                          //以PTP、速度50%、加速時間200ms運動至目標
```

語法 3

```
bool PTP(
    string,
    float, float, float, float, float, float,
    int,
    int,
    int,
    bool
)
```

參數

string 定義運動參數格式，由三個字母組成

第一字母：運動目標表示方式：

"J" 以角度表示

"C" 以座標表示

第二字母：速度表示方式：

"P" 以百分比表示

第三字母：軌跡混合表示方式：

"P" 以百分比表示

float, float, float, float, float, float

運動目標。若以角度表示，為手臂關節的六個元素：關節1(°)，關節2(°)，關節3(°)，關節4(°)，關節5(°)，關節6(°)；若以座標表示，為手臂末端工具中心點座標的六個元素：

X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)

int 手臂末端移動速度百分比(%)

int 運動加速到最高速所花費時間(millisecond)

int 軌跡混合百分比(%)

bool 取消精準到位

true 取消精準到位

false 精準到位

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

運動參數格式表示方式有：(1) "JPP" 與(2) "CPP"

```
PTP("JPP",0,0,90,0,90,0,35,200,0,false) //以PTP、速度35%、加速時間200ms運動至目標角度  
0,0,90,0,90,0
```

語法 4

```
bool PTP(  
    string,  
    float, float, float, float, float, float,  
    int,  
    int,  
    int,  
    bool,  
    int, int, int  
)
```

參數

string 定義運動參數格式，由三個字母組成

第一字母：運動目標表示方式：

"C" 以座標表示

第二字母：速度表示方式：

"P" 以百分比表示

第三字母：軌跡混合表示方式：

"P" 以百分比表示

float, float, float, float, float, float

運動目標。以座標表示，為手臂末端工具中心點座標的六個元素：X(mm), Y(mm), Z(mm), RX(mm), RY(mm), RZ(mm)

int 手臂末端移動速度百分比(%)

int 運動加速到最高速所花費時間(millisecond)

int 軌跡混合百分比(%)

bool 取消精準到位

true 取消精準到位

false 精準到位

int, int, int

手臂姿態定義：Config1, Config2, Config3，手臂姿態定義詳見附註

傳回值

bool 設定成功回傳**True**，設定失敗回傳**False**

說明

運動參數格式表示方式有：(1) "CPP"

```
PTP("CPP",417.50,-122.30,343.90,180.00,0.00,90.00,10,200,0,false,0,2,4) //以PTP、速度10%、加速時間  
200ms運動至目標座標 417.50,-  
122.30,343.90,180.00,0.00,90.00  
、姿態定義為024
```

9.7 Line()

設定 Line 絶對位置運動參數並下達運動指令。

語法 1

```
bool Line(  
    string,  
    float[],  
    int,  
    int,  
    int,  
    bool  
)
```

參數

string 定義運動參數格式，由三個字母組成

第一字母：運動目標表示方式：

"C" 以座標表示

第二字母：速度表示方式：

"P" 以百分比表示

"A" 以絕對速度表示

第三字母：軌跡混合表示方式：

"P" 以百分比表示

"R" 以半徑表示

float[] 運動目標，為手臂末端工具中心點座標的六個元素：X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)

int 手臂末端移動速度百分比(%)或絕對速度(mm/s)

int 運動加速到最高速所花費時間(millisecond)

int 軌跡混合參數百分比(%)或半徑(mm)

bool 取消精準到位

true 取消精準到位

false 精準到位

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

運動參數格式表示方式有：(1) "CPP"、(2) "CPR"、(3) "CAP" 與(4) "CAR"

```
float[] Point1 = {417.50,-122.30,343.90,180.00,0.00,90.00} //宣告浮點數陣列儲存移動目標座標
```

```
Line("CAR",Point1,100,200,50,false) //以Line、速度100mm/s、加速時間200ms、軌  
跡混合半徑50mm運動至目標
```

語法2

```
bool Line(  
    string,  
    float, float, float, float, float, float,  
    int,  
    int,  
    int,  
    bool  
)
```

參數

`string` 定義運動參數格式，由三個字母組成

第一字母：運動目標表示方式：

"C" 以座標表示

第二字母：速度表示方式：

"P" 以百分比表示

"A" 以絕對速度表示

第三字母：軌跡混合表示方式：

"P" 以百分比表示

"R" 以半徑表示

`float, float, float, float, float, float`

圓弧末點，為手臂末端工具中心點座標的六個元素：X(mm), Y(mm), Z(mm), RX(°), RY(°),

RZ(°)

`int` 手臂末端移動速度百分比(%)或絕對速度(mm/s)

`int` 運動加速到最高速所花費時間(millisecond)

`int` 軌跡混合參數百分比(%)或半徑(mm)

`bool` 取消精準到位

`true` 取消精準到位

`false` 精準到位

傳回值

`bool` 設定成功回傳True，設定失敗回傳False

說明

運動參數格式表示方式有：(1) "CPP"、(2) "CPR"、(3) "CAP" 與(4) "CAR"

`Line("CAR", 417.50,-122.30,343.90,180.00,0.00,90.00,100,200,50,false)`

//以Line、速度100mm/s、加速時間200ms、

軌跡混合半徑50mm運動至417.50,-122.30,343.90,180.00,0.00,90.00

9.8 Circle()

設定 Circle 絶對位置運動參數並下達運動指令。

語法 1

```
bool Circle(  
    string,  
    float[],  
    float[],  
    int,  
    int,  
    int,  
    int,  
    bool  
)
```

參數

string 定義運動參數格式，由三個字母組成

第一字母：運動目標表示方式：

"C" 以座標表示

第二字母：速度表示方式：

"P" 以百分比表示

"A" 以絕對速度表示

第三字母：軌跡混合表示方式：

"P" 以百分比表示

float[] 圓弧上任意點，為手臂末端工具中心點座標的六個元素：X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)

float[] 圓弧末點，為手臂末端工具中心點座標的六個元素：X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)

int 手臂末端移動速度百分比(%)或絕對速度(mm/s)

int 運動加速到最高速所花費時間(millisecond)

int 軌跡混合參數百分比(%)

int 圓運動角度(°)，若輸入非0(°)之數值會依照指定之運動角度移動，並保持初始之末端運動轉角，若輸入0(°)則代表運動走至圓弧末點時即結束，同時末端運動轉角會做線性補差。

bool 取消精準到位

true 取消精準到位

false 精準到位

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

運動參數格式表示方式有:(1) "CPP" 與(2) "CAP"

```
float[] PassP = {417.50,-122.30,343.90,180.00,0.00,90.00}  
float[] EndP = {381.70,208.74,343.90,180.00,0.00,135.00}  
Circle("CAP",PassP,EndP,100,200,50,270,false)
```

```
//宣告浮點數陣列儲存圓弧上任意點座標  
//宣告浮點數陣列儲存圓弧末點座標  
//以Circle、速度100mm/s、加速時間  
200ms、軌跡混合半徑50%走270°的圓弧
```

語法 2

```
bool Circle(  
    string,  
    float, float, float, float, float, float,  
    float, float, float, float, float, float,  
    int,  
    int,  
    int,  
    int,  
    bool  
)
```

參數

string 定義運動參數格式，由三個字母組成

第一字母：運動目標表示方式：

"C" 以座標表示

第二字母：速度表示方式：

"P" 以百分比表示

"A" 以絕對速度表示

第三字母：軌跡混合表示方式：

"P" 以百分比表示

float, float, float, float, float, float

圓弧上任意點，為手臂末端工具中心點座標的六個元素：X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)

float, float, float, float, float, float

圓弧末點，為手臂末端工具中心點座標的六個元素：X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)

int 手臂末端移動速度百分比(%)或絕對速度(mm/s)

int 運動加速到最高速所花費時間(millisecond)
int 軌跡混合參數百分比(%)
int 圓運動角度($^{\circ}$)，若輸入非 $0(^{\circ})$ 之數值會依照指定之運動角度移動，並保持初始之末端運動轉角，若輸入 $0(^{\circ})$ 則代表運動走至圓弧末點時即結束，同時末端運動轉角會做線性補差。
bool 取消精準到位
 true 取消精準到位
 false 精準到位

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

運動參數格式表示方式有:(1) "CPP"與(2) "CAP"

```
Circle("CAP", 417.50,-122.30,343.90,180.00,0.00,90.00,  
      381.70,208.74,343.90,180.00,0.00,135.00,100,200,50,270,false)  
      //以Circle、速度100mm/s、加速時間200ms、軌跡混合半徑50%於圓弧上運動 $270^{\circ}$ ，圓弧上一點為  
      417.50,-122.30,343.90,180.00,0.00,90.00、圓弧末點為 381.70,208.74,343.90,180.00,0.00,135.00
```

9.9 PLine()

設定 PLine 絕對位置運動參數並下達運動指令。

語法 1

```
bool PLine(  
    string,  
    float[],  
    int,  
    int,  
    int  
)
```

參數

string 定義運動參數格式，由三個字母組成

第一字母：運動目標表示方式：

"J": 以角度表示

"C": 以座標表示

第二字母：速度表示方式：

"A" 以絕對速度表示

第三字母：軌跡混合表示方式：

"P" 以百分比表示

float[] 運動目標。若以角度表示，為手臂關節的六個元素：關節1(°), 關節2(°), 關節3(°), 關節4(°), 關節5(°), 關節6(°)；若以座標表示，為手臂末端工具中心點座標的六個元素：

X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)

int 手臂末端移動的絕對速度(mm/s)

int 運動加速到最高速所花費時間(millisecond)

int 軌跡混合百分比(%)

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

運動參數格式表示方式有：(1) "JAP" 與(2) "CAP"

```
float[] targetP1 = {417.50,-122.30,343.90,180.00,0.00,90.00} //宣告浮點數陣列儲存移動目標座標
```

```
PLine("CAP",targetP1,100,200,50) //以PLine、速度100mm/s、加速時間200ms、軌跡混合半徑50%運動至目標
```

語法2

```
bool PLine(  
    string,  
    float, float, float, float, float, float,  
    int,  
    int,  
    int  
)
```

參數

`string` 定義運動參數格式，由三個字母組成

第一字母：運動目標表示方式：

"J": 以角度表示

"C": 以座標表示

第二字母：速度表示方式：

"A" 以絕對速度表示

第三字母：軌跡混合表示方式：

"P" 以百分比表示

`float, float, float, float, float, float,`

運動目標。若以角度表示，為手臂關節的六個元素：關節1(°)，關節2(°)，關節3(°)，關節4(°)，關節5(°)，關節6(°)；若以座標表示，為手臂末端工具中心點座標的六個元素：

X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)

`int` 手臂末端移動的絕對速度(mm/s)

`int` 運動加速到最高速所花費時間(millisecond)

`int` 軌跡混合百分比(%)

傳回值

`bool` 設定成功回傳True，設定失敗回傳False

說明

運動參數格式表示方式有：(1) "JAP" 與(2) "CAP"

```
PLine("CAP", 417.50,-122.30,343.90,180.00,0.00,90.00,100,200,50)
```

//以PLine、速度100mm/s、加速時間200ms、

軌跡混合半徑50%運動至417.50,-122.30,343.90,180.00,0.00,90.00

9.10 Move_PTP()

設定 PTP 相對運動參數並下達運動指令。

語法 1

```
bool Move_PTP(  
    string,  
    float[],  
    int,  
    int,  
    int,  
    bool  
)
```

參數

string 定義運動參數格式，由三個字母組成

第一字母：相對運動表示方式：

"C": 以機器人當前座標表示

"T": 以工具座標表示

"J": 以角度表示

第二字母：速度表示方式：

"P": 以百分比表示

第三字母：軌跡混合表示方式：

"P": 以百分比表示

float[] 相對運動參數，若以座標(機器人當前座標或工具座標)表示，為手臂末端工具中心點相對移動的六個元素：X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)，其參考坐標可為機器人當前座標或是工具座標；若以角度表示，為手臂關節的六個元素：關節1(°), 關節2(°), 關節3(°), 關節4(°), 關節5(°), 關節6(°)

int 手臂末端移動速度百分比(%)

int 運動加速到最高速所花費時間(millisecond)

int 軌跡混合百分比(%)

bool 取消精準到位

true 取消精準到位

false 精準到位

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

運動參數格式表示方式有：(1) "CPP"、(2) "TPP"、(3) "JPP"

float[] relmove = {0,0,10,45,0,0} //宣告浮點數陣列儲存相對移動值

Move_PTP("TPP",relmove,10,200,0,false) //以PTP、速度10%、加速時間200ms進行相對運動

語法2

```
bool Move_PTP(  
    string,  
    float, float, float, float, float, float,  
    int,  
    int,  
    int,  
    bool  
)
```

參數

string 定義運動參數格式，由三個字母組成

第一字母：相對運動表示方式：

"C": 以機器人當前座標表示

"T": 以工具座標表示

"J": 以角度表示

第二字母：速度表示方式：

"P": 以百分比表示

第三字母：軌跡混合表示方式：

"P": 以百分比表示

float, float, float, float, float, float

相對運動參數，若以座標(機器人當前座標或工具座標)表示，為手臂末端工具中心點相對移動的六個元素：X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)，其參考坐標可為機器人當前座標或是工具座標；若以角度表示，為手臂關節的六個元素：關節1(°), 關節2(°), 關節3(°), 關節4(°), 關節5(°), 關節6(°)

int 手臂末端移動速度百分比(%)

int 運動加速到最高速所花費時間(millisecond)

int 軌跡混合百分比(%)

bool 取消精準到位

true 取消精準到位

false 精準到位

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

運動參數格式表示方式有：(1) "CPP"、(2) "TPP"、(3) "JPP"

**Move_PTP("TPP",0,0,10,45,0,0,10,200,0,false) //以PTP、速度10%、加速時間200ms相對移動
0,0,10,45,0,0、**

移動參考坐標為機器人工具座標

9.11 Move_Line()

設定 Line 相對運動參數並下達運動指令。

語法 1

```
bool Move_Line(  
    string,  
    float[],  
    int,  
    int,  
    int,  
    bool  
)
```

參數

string 定義運動參數格式，由三個字母組成

第一字母：相對運動表示方式：

"C": 以機器人當前座標表示

"T": 以工具座標表示

第二字母：速度表示方式：

"P": 以百分比表示

"A": 以絕對速度表示

第三字母：軌跡混合表示方式：

"P": 以百分比表示

"R": 以半徑表示

float[] 相對運動參數，以座標表示，為手臂末端工具中心點相對移動的六個元素：X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)。

int 手臂末端移動速度百分比(%)或絕對速度(mm/s)

int 運動加速到最高速所花費時間(millisecond)

int 軌跡混合參數百分比(%)或半徑(mm)

bool 取消精準到位

true 取消精準到位

false 精準到位

傳回值

bool 設定成功回傳**True**，設定失敗回傳**False**

說明

運動參數格式表示方式有：(1) "CPP"、(2) "CPR"、(3) "CAP"、(4) "CAR"、(5) "TPP"、(6) "TPR"、(7) "TAP"、(8) "TAR"

float[] relmove = {0,0,10,45,0,0} //宣告浮點數陣列儲存相對移動值

Move_Line("TAP",relmove,125,200,0,false) //以Line、速度125mm/s、加速時間200ms相對移動至目標

語法2

```
bool Move_Line(  
    string,  
    float, float, float, float, float, float,  
    int,  
    int,  
    int,  
    bool  
)
```

參數

`string` 定義運動參數格式，由三個字母組成

第一字母：相對運動表示方式：

"C": 以機器人當前座標表示

"T": 以工具座標表示

第二字母：速度表示方式：

"P": 以百分比表示

"A": 以絕對速度表示

第三字母：軌跡混合表示方式：

"P": 以百分比表示

"R": 以半徑表示

`float, float, float, float, float, float`

相對運動參數，以座標表示，為手臂末端工具中心點相對移動的六個元素：`X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)`。

`int` 手臂末端移動速度百分比(%)或絕對速度(mm/s)

`int` 運動加速到最高速所花費時間(millisecond)

`int` 軌跡混合參數百分比(%)或半徑(mm)

`bool` 取消精準到位

`true` 取消精準到位

`false` 精準到位

傳回值

`bool` 設定成功回傳`True`，設定失敗回傳`False`

說明

運動參數格式表示方式有：(1) "CPP"、(2) "CPR"、(3) "CAP"、(4) "CAR"、(5) "TPP"、(6) "TPR"、(7) "TAP"、(8) "TAR"

`Move_Line("TAP", 0,0,10,45,0,0,125,200,0,false)`

//以Line、速度125mm/s、加速時間200ms相對
移動 0,0,10,45,0,0 至目標

9.12 Move_PLine()

設定 PLine 相對運動參數並下達運動指令。

語法 1

```
bool Move_PLine(  
    string,  
    float[],  
    int,  
    int,  
    int  
)
```

參數

string 定義運動參數格式，由三個字母組成

第一字母：相對運動表示方式：

"C"：以機器人座標表示

"T"：以工具座標表示

"J"：以角度表示

第二字母：速度表示方式：

"A"：以絕對速度表示

第三字母：軌跡混合表示方式：

"P"：以百分比表示

float[] 相對運動參數，以座標表示，為手臂末端工具中心點相對移動的六個元素：X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)。

int 手臂末端移動的絕對速度(mm/s)

int 運動加速到最高速所花費時間(millisecond)

int 軌跡混合參數百分比(%)

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

運動參數格式表示方式有：(1) "CAP"、(2) "TAP"、(3) "JAP"

```
float[] target = {0,0,10,45,0,0}          //宣告浮點數陣列儲存相對移動值  
Move_PLine("CAP",target,125,200,0)      //以PLine、速度125mm/s、加速時間200ms相對移動至目標
```

語法 2

```
bool Move_PLine(  
    string,  
    float, float, float, float, float, float,  
    int,  
    int,  
    int,  
)
```

參數

string 定義運動參數格式，由三個字母組成

第一字母：相對運動表示方式：

"C": 以機器人座標表示

"T": 以工具座標表示

"J": 以角度表示

第二字母：速度表示方式：

"A" 以絕對速度表示

第三字母：軌跡混合表示方式：

"P" 以百分比表示

float, float, float, float, float, float

相對運動參數，以座標表示，為手臂末端工具中心點相對移動的六個元素: X(mm), Y(mm), Z(mm), RX(°), RY(°), RZ(°)。

int 手臂末端移動的絕對速度(mm/s)

int 運動加速到最高速所花費時間(millisecond)

int 軌跡混合參數百分比(%)

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

運動參數格式表示方式有：(1) "CAP"、(2) "TAP"、(3) "JAP"

Move_PLine("CAP",0,0,10,45,0,0,125,200,0)

//以PLine、速度125mm/s、加速時間200ms相對移動
至0,0,10,45,0,0

9.13 ChangeBase()

設定接續運動的參考坐標系。

語法 1

```
bool ChangeBase(  
    string  
)
```

參數

string 坐標系名稱

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

ChangeBase("RobotBase") //改變運動參考座標至 TMflow 座標系列表中的"RobotBase"

語法 2

```
bool ChangeBase(  
    float[]  
)
```

參數

float[] 座標系參數值，分別為 X, Y, Z, RX, RY, RZ

傳回值

設定成功回傳True，設定失敗回傳False

說明

float[] Base1 = {20,30,10,0,0,90} //宣告浮點數陣列儲存座標系值

ChangeBase(Base1) //改變運動參考座標至設定座標系值

語法 3

```
bool ChangeBase(  
    float, float, float, float, float, float  
)
```

參數

float, float, float, float, float, float
座標系參數值，分別為 X, Y, Z, RX, RY, RZ

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

ChangeBase(20,30,10,0,0,90) //改變運動參考座標至{20,30,10,0,0,90}

9.14 ChangeTCP()

設定接續運動的末端工具參數。

語法 1

```
bool ChangeTCP(  
    string  
)
```

參數

string TCP名稱

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

```
ChangeTCP("NOTOOL") //改變末端工具至 TMflow 末端工具列表中的"NOTOOL"
```

語法 2

```
bool ChangeTCP(  
    float[]  
)
```

參數

float[] TCP參數值，分別為: X, Y, Z, RX, RY, RZ

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

```
float[] Tool1 = {0,0,150,0,0,90} //宣告浮點數陣列儲存末端工具值
```

```
ChangeTCP(Tool1) //改變末端工具值至設定末端工具值
```

語法 3

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

參數

float[] TCP參數值，分別為: X, Y, Z, RX, RY, RZ

float TCP重量

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

```
float[] Tool1 = {0,0,150,0,0,90}           //宣告浮點數陣列儲存末端工具值  
ChangeTCP(Tool1,2)                      //變末端工具值至設定末端工具值，末端工具重量 2kg
```

語法 4

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

參數

float[] TCP參數值，分別為: X, Y, Z, RX, RY, RZ
float TCP 重量
float[] TCP轉動慣量，共9個值，分別為: (1)lxx, (2)lyy, (3)lzz 與 質量中心座標相對於工具座標的
(4)X, (5)Y, (6)Z, (7)RX, (8)RY, (9)RZ

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

```
float[] Tool1 = {0,0,150,0,0,90}           //宣告浮點數陣列儲存末端工具值  
float[] COM1 = {2,0.5,0.5,0,0,-80,0,0,0}   //宣告浮點數陣列儲存轉動慣量值及其相對於工具座標的  
                                              參考座標  
ChangeTCP(Tool1,2,COM1)                   //改變末端工具值至設定末端工具值，末端工具重量 2kg，  
                                              並套用轉動慣量值及其參考座標
```

語法 5

```
bool ChangeTCP(  
    float, float, float, float, float, float  
)
```

參數

float, float, float, float, float, float
TCP參數值，分別為: X, Y, Z, RX, RY, RZ

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

```
ChangeTCP(0,0,150,0,0,90)                //改變末端工具值至{0,0,150,0,0,90}
```

語法 6

```
bool ChangeTCP(  
    float, float, float, float, float, float,  
    float  
)
```

參數

float, float, float, float, float, float
TCP參數值，分別為: X, Y, Z, RX, RY, RZ
float TCP重量

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

ChangeTCP(0,0,150,0,0,90,2) //改變末端工具值至{0,0,150,0,0,90}，末端工具重量 2kg

語法 7

```
bool ChangeTCP(  
    float, float, float, float, float, float,  
    float,  
    float, float, float, float, float, float, float, float  
)
```

參數

float, float, float, float, float, float
TCP參數值，分別為: X, Y, Z, RX, RY, RZ
float TCP重量
float, float, float, float, float, float, float, float, float
TCP轉動慣量，共9個值，分別為: (1)lxx, (2)lyy, (3)lzz 與 質量中心座標相對於工具座標的
(4)X, (5)Y, (6)Z, (7)RX, (8)RY, (9)RZ

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

ChangeTCP(0,0,150,0,0,90,2, 2,0.5,0.5,0,0,-80,0,0,0) //改變末端工具值至{0,0,150,0,0,90}，末端工具重量 2kg，
轉動慣量值{2,0.5,0.5}及其參考座標為{0,0,-80,0,0,0}

9.15 ChangeLoad()

設定需補償之附載重量

語法 1

```
bool ChangeLoad(  
    float  
)
```

參數

float 附載重量，單位公斤

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

```
ChangeLoad(5.3) //設定附載重量為 5.3 公斤
```

9.16 PVTEnter()

以關節/笛卡兒指令設定 PVT 模式開始

語法 1

```
bool PVTEnter(  
    int  
)
```

參數

int	PVT Mode
0	Joint
1	Cartesian

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

語法 2

```
bool PVTEnter(  
)
```

參數

void 無輸入值 (預設採用 PVT with Joint Command 模式)

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

PVTEnter(1)

9.17 PVTExit()

設定 PVT 模式出口

語法 1

```
bool PVTExit(  
)
```

參數

void 無輸入值

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

PVTExit()

9.18 PVTPoint()

設定 PVT 模式位置、速度和時間等運動參數

語法 1

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

參數

float[]	目標位置
	{J1, J2, J3, J4, J5, J6} in PVT mode with Joint
	{X, Y, Z, RX, RY, RZ} in PVT mode with Cartesian
float[]	目標速度
	{J1, J2, J3, J4, J5, J6}' in PVT mode with Joint
	{X, Y, Z, RX, RY, RZ}' in PVT mode with Cartesian
float	時間(second)

傳回值

bool 設定成功回傳True，設定失敗回傳False

說明

語法 2

```
bool PVTPoint(  
    float, float, float, float, float, float,  
    float, float, float, float, float,  
    float  
)
```

參數

float, float, float, float, float, float,	
	Target Position.
	{J1, J2, J3, J4, J5, J6} in PVT mode with Joint
	{X, Y, Z, RX, RY, RZ} in PVT mode with Cartesian
float, float, float, float, float, float,	
	Target Velocity.
	{J1, J2, J3, J4, J5, J6}' in PVT mode with Joint
	{X, Y, Z, RX, RY, RZ}' in PVT mode with Cartesian
float	Duration(second)

傳回值

`bool` 設定成功回傳`True`，設定失敗回傳`False`

說明

```
PVTEEnter(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)  
PVTEExit()
```

9.19 PVTPause()

設定 PVT 模式暫停運動

語法 1

```
bool PVTPause(  
)
```

參數

void 無輸入值

傳回值

bool 設定成功回傳**True**，設定失敗回傳**False**

說明

PVTPause()

9.20 PVTResume()

設定 PVT 模式恢復運動

語法 1

```
bool PVTResume()  
)
```

參數

void 無輸入值

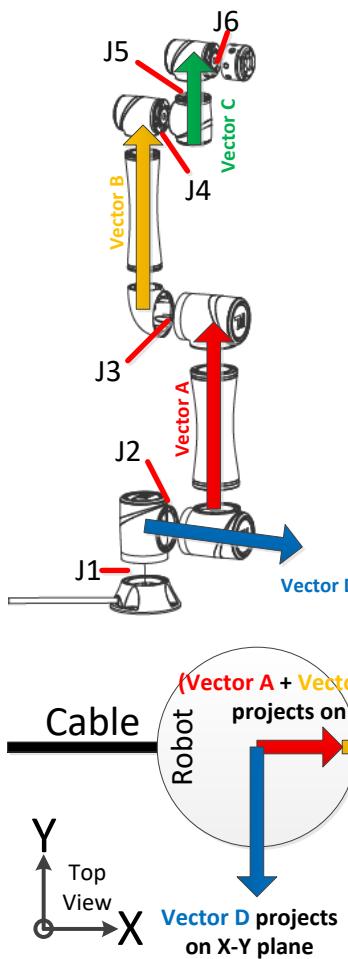
傳回值

bool 設定成功回傳**True**，設定失敗回傳**False**

說明

PVTResume()

手臂姿態定義 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 (config1=0 and J3 is positive) or (config1=1 and J3 is negative)

config2=3:

if (config1=0 and J3 is negative) or (config1=1 and J3 is positive)

config3=4:

if (config1=0 and J5 is positive) or (config1=1 and J5 is negative)

config3=5:

if (config1=0 and J5 is negative) or (config1=1 and J5 is positive)

9.21 Vision_DoJob()

設定執行專案中已經存在的視覺任務，與運動指令一同排序

語法 1

```
bool Vision_DoJob(  
    string  
)
```

參數

string 視覺任務名稱，不可設定有初始點的視覺任務。

傳回值

bool	True	視覺任務完成且結果為 Pass
	False	視覺任務失敗
		1. 運行結果為 Fail
		2. 運行失敗
		3. 視覺任務設有初始點

說明

被呼叫的視覺任務需在 TMflow 頁面中先行建立，在視覺任務的 Initial 中有"開始在初始點"的勾選框，欲使用此功能，需反勾選此選項。

在視覺任務執行後，相對應的視覺坐標系或變數會一同被更新。

```
bool var_result = Vision_DoJob("job1")
```

9.22 Vision_DoJob_PTP()

執行專案中已經存在的視覺任務，以 PTP 運動至初始點，與運動指令一同排序

語法 1

```
bool Vision_DoJob_PTP(  
    string,  
    int,  
    int,  
    bool  
)
```

參數

string	視覺任務名稱
int	手臂末端移動速度百分比(%)
int	運動加速到最高速所花費時間(millisecond)
bool	是否使用智慧選擇手臂恣態模式
true	由系統決定點位手臂恣態
false	使用教學視覺任務時紀錄的點位手臂恣態

傳回值

bool	True	視覺任務完成且結果為 Pass
	False	視覺任務失敗 1. 運行結果為 Fail 2. 運行失敗

說明

被呼叫的視覺任務需在 TMflow 頁面中先行建立，在視覺任務的 Initial 中有"開始在初始點"的勾選框，欲使用此功能，需勾選此選項；若不勾選，則不會運動到初始點。

在視覺任務執行後，相對應的視覺坐標系或變數會一同被更新。

```
bool var_result = Vision_DoJob_PTP("job1", 100, 250, true)
```

9.23 Vision_DoJob_Line()

執行專案中已經存在的視覺任務，以 Line 運動至初始點，與運動指令一同排序

語法 1

```
bool Vision_DoJob_Line(  
    string,  
    int  
)
```

參數

string	視覺任務名稱
int	手臂末端移動速度百分比(%)

傳回值

bool	True	視覺任務完成且結果為 Pass
	False	視覺任務失敗
		1. 運行結果為 Fail
		2. 運行失敗

說明

被呼叫的視覺任務需在 TMflow 頁面中先行建立，在視覺任務的 Initial 中有"開始在初始點"的勾選框，欲使用此功能，需勾選此選項；若不勾選，則不會運動到初始點。

在視覺任務執行後，相對應的視覺坐標系或變數會一同被更新。

```
bool var_result = Vision_DoJob_Line("job1", 100)
```

語法 2

```
bool Vision_DoJob_Line(  
    string,  
    int,  
    int  
)
```

參數

string	視覺任務名稱
int	手臂末端移動絕對速度(mm/s)
int	運動加速到最高速所花費時間(millisecond)

傳回值

bool	True	視覺任務完成且結果為 Pass
	False	視覺任務失敗 1. 運行結果為 Fail 2. 運行失敗

說明

被呼叫的視覺任務需在 TMflow 頁面中先行建立，在視覺任務的 Initial 中有"開始在初始點"的勾選框，欲使用此功能，需勾選此選項；若不勾選，則不會運動到初始點。

在視覺任務執行後，相對應的視覺坐標系或變數會一同被更新。

```
bool var_result = Vision_DoJob_Line("job1", 150, 250)
```

9.24 Vision_IsJobAvailable()

檢查專案中的視覺任務是否存在及是否合法

語法 1

```
bool Vision_IsJobAvailable(  
    string  
)
```

參數

string 視覺任務名稱

傳回值

bool	True	視覺任務存在且合法
	False	視覺任務不存在或不合法

說明

在呼叫的視覺任務前可使用此功能檢查視覺任務是否存在及合法，可避免在呼叫視覺任務執行時因任務不存在或不合法而觸發錯誤。

```
bool var_result = Vision_IsJobAvailable("job1")
```

10. Modbus 函式

10.1 modbus_read()

Modbus TCP/RTU 讀取

語法 1 (TCP/RTU)

```
? modbus_read(  
    string,  
    string  
)
```

參數

string TCP/RTU 的裝置名稱 (在 Modbus Device 中設定)

string TCP/RTU 裝置名稱下預先設置的讀寫參數名稱 (在 Modbus Device 中設定)

傳回值

? 依照預先設置的讀寫參數，決定傳回型別

Signal Type	Function Code	Type	Num Of Addr	傳回型別
Digital Output	01	byte	1	byte (H: 1)(L: 0)
		bool	1	bool (H: true)(L: false)
Digital Input	02	byte	1	byte (H: 1)(L: 0)
		bool	1	bool (H: true)(L: false)
Register Output	03	byte	1	byte
		int16	1	int
Register Input	04	int32	2	int
		float	2	float
Register Input	04	double	4	double
		string	?	string
Register Input	04	bool	1	bool
		byte	1	byte
Register Input	04	int16	1	int
		int32	2	int
Register Input	04	float	2	float
		double	4	double
Register Input	04	string	?	string
		bool	1	bool

* int32, float, double 會依使用者選擇 Little Endian (CD AB) 或 Big Endian (AB CD) 格式轉換

* string 會依 UTF8 格式轉換 (遇到 0x00 結束)

說明

Modbus Address data size

Digital 1 address = 1 bit size

Register 1 address = 2 bytes size

假設 Preset Setting 使用預先設置中定義

preset_800	DO	800	byte	
preset_7202	DI	7202	bool	
preset_9000	RO	9000	string	4
preset_7001	RI	7001	float	Big-Endian (AB CD)

```
value = modbus_read("TCP_1", "preset_800") // value = 1           // DO 1 address = 1 bit
value = modbus_read("TCP_1", "preset_7202") // value = true        // DI 1 address = 1 bit
value = modbus_read("TCP_1", "preset_9000") // value = ab1234cd    // RO 4 address = 8 bytes size
value = modbus_read("TCP_1", "preset_7001") // value = -314.1593   // RI 2 address = 4 bytes size (float)
```

語法 2 (TCP/RTU)

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

參數

string TCP/RTU 的裝置名稱 (在 Modbus Device 中設定)

byte Slave ID

string 讀取類型

 DO Digital Output (Function Code : 01)

 DI Digital Input (Function Code : 02)

 RO Register Output (Function Code : 03)

 RI Register Input (Function Code : 04)

int 起始位址

int 讀取位址個數

傳回值

`byte[]` 傳回讀取後的 modbus 資料 byte 陣列

*自定義 `modbus_read` 僅會依 [Big-Endian \(AB CD\)](#) 格式讀出 `byte[]`

說明

Modbus Address data size

Digital 1 address = 1 bit size

Register 1 address = 2 bytes size

假設 User Setting 使用自定義

TCP device	0	DO	800	4
TCP device	0	DI	7202	3
TCP device	0	RO	9000	6
TCP device	0	RI	7001	12
TCP device	0	RI	7301	6

```
value = modbus_read("TCP_1", 0, "DO", 800, 4)
// value = {0,0,0,0} // DO 4 address = 4 bit to byte array

value = modbus_read("TCP_1", 0, "DI", 7202, 3)
// value = {1,0,0} // DI 3 address = 3 bit to byte array

value = modbus_read("TCP_1", 0, "RO", 9000, 6)
// value = {0x54,0x65,0x63,0x68,0x6D,0x61,0x6E,0xE9,0x81,0x94,0xE6,0x98} // RO 6 address = 12 bytes size

value = modbus_read("TCP_1", 0, "RI", 7001, 12)
// value = {0x29,0x30,0x9F,0x4C,0xC3,0x7C,0x99,0x9A,0x44,0x5E,0xEC,0xCD,0x42,0xB4,0x00,0x00,
// 0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00} // RI 12 address = 24 bytes size

value = modbus_read("TCP_1", 0, "RI", 7301, 6)
// value = {0x07,0xE2,0x00,0x05,0x00,0x12,0x00,0xF,0x00,0xA,0x00,0x39} // RI 6 address = 12 bytes size
```

10.2 modbus_read_int16()

Modbus TCP/RTU 讀取並轉換 modbus 位址資料陣列為 16 位元整數型別陣列

語法 1 (TCP/RTU)

```
int[] modbus_read_int16(  
    string,  
    byte,  
    string,  
    int,  
    int,  
    int  
)
```

參數

string TCP/RTU 的裝置名稱 (在 Modbus Device 中設定)
byte Slave ID
string 讀取類型
 DO Digital Output (Function Code : 01)
 DI Digital Input (Function Code : 02)
 RO Register Output (Function Code : 03)
 RI Register Input (Function Code : 04)
int 起始位址
int 讀取位址個數
int 讀出後的 Address 資料是依照 Little Endian (CD AB) 或依照 Big Endian (AB CD) 轉成 int16 陣列。*此參數無效，僅支援 int32, float, double。
 0 Little Endian
 1 Big Endian (預設)

傳回值

int[] 傳回讀取後的 modbus 資料 int 陣列

語法 2 (TCP/RTU)

```
int[] modbus_read_int16(  
    string,  
    byte,  
    string,  
    int,  
    int  
)
```

說明

與語法 1 相同，預設將參數 int 填入 1 以 Big Endian (AB CD) 轉換陣列。

modbus_read_int16("TCP_1", 0, "DI", 7200, 2) => **modbus_read_int16("TCP_1", 0, "DI", 7200, 2, 1)**

Modbus Address data size

Digital 1 address = 1 bit size

Register 1 address = 2 bytes size

假設 User Setting 使用自定義

TCP device	0	DO	800	4
TCP device	0	DI	7202	3
TCP device	0	RO	9000	6
TCP device	0	RI	7001	12
TCP device	0	RI	7301	6

```
value = modbus_read_int16("TCP_1", 0, "DO", 800, 4)
    // byte[] = {0,0,0,0}      to int16[]  value = {0,0}      // byte[0][1] , byte[2][3]

value = modbus_read_int16("TCP_1", 0, "DI", 7202, 3)
    // byte[] = {1,0,0}      to int16[]  value = {256,0}  // byte[0][1] , byte[2][3] // 不足 [3] 自動補齊

value = modbus_read_int16("TCP_1", 0, "RO", 9000, 6)
    // byte[] = {0x54,0x65,0x63,0x68,0x6D,0x61,0x6E,0xE9,0x81,0x94,0xE6,0x98}
    // to int16[]      value = {21605,25448,28001,28393,-32364,-6504}

value = modbus_read_int16("TCP_1", 0, "RI", 7001, 12)
    // byte[] = {0x29,0x30,0x9F,0x4C,0xC3,0x7C,0x99,0x9A,0x44,0x5E,0xEC,0xCD,0x42,0xB4,0x00,0x00,
    //           0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
    // to int16[]      value = {10544,-24756,-15492,-26214,17502,-4915,17076,0,-32768,0,0,0}

value = modbus_read_int16("TCP_1", 0, "RI", 7301, 6)
    // byte[] = {0x07,0xE2,0x00,0x05,0x00,0x12,0x00,0x0F,0x00,0x31,0x00,0x23}
    // to int16[]      value = {2018,5,18,15,49,35}

value = modbus_read_int16("TCP_1", 0, "RI", 7301, 6, 0)
    // byte[] = {0x07,0xE2,0x00,0x05,0x00,0x12,0x00,0x0F,0x00,0x31,0x00,0x23}
    // to int16[]      value = {2018,5,18,15,49,35}
```

10.3 modbus_read_int32()

Modbus TCP/RTU 讀取並轉換 modbus 位址資料陣列為 32 位元整數型別陣列

語法 1 (TCP/RTU)

```
int[] modbus_read_int32(  
    string,  
    byte,  
    string,  
    int,  
    int,  
    int  
)
```

參數

string TCP/RTU 的裝置名稱 (在 Modbus Device 中設定)
byte Slave ID
string 讀取類型
 DO Digital Output (Function Code : 01)
 DI Digital Input (Function Code : 02)
 RO Register Output (Function Code : 03)
 RI Register Input (Function Code : 04)
int 起始位址
int 讀取位址個數
int 讀出後的 Address 資料是依照 Little Endian (CD AB) 或依照 Big Endian (AB CD) 轉成 int32 陣列。
 0 Little Endian
 1 Big Endian (預設)

傳回值

int[] 傳回讀取後的 modbus 資料 int 陣列

語法 2 (TCP/RTU)

```
int[] modbus_read_int32(  
    string,  
    byte,  
    string,  
    int,  
    int  
)
```

說明

與語法 1 相同，預設將參數 int 填入 1 以 Big Endian (AB CD) 轉換陣列。

modbus_read_int32("TCP_1", 0, "DI", 7200, 2) => modbus_read_int32("TCP_1", 0, "DI", 7200, 2, 1)

Modbus Address data size

Digital 1 address = 1 bit size

Register 1 address = 2 bytes size

假設 User Setting 使用自定義

TCP device	0	DO	800	4
TCP device	0	DI	7202	3
TCP device	0	RO	9000	6
TCP device	0	RI	7001	12
TCP device	0	RI	7301	6

```
value = modbus_read_int32("TCP_1", 0, "DO", 800, 4)
    // byte[] = {0,0,0,0}      to int32[]  value = {0} // byte[0][1][2][3]

value = modbus_read_int32("TCP_1", 0, "DI", 7202, 3)
    // byte[] = {1,0,0}      to int32[]  value = {16777216}    // byte[0][1][2][3] // 不足 [3] 自動補齊

value = modbus_read_int32("TCP_1", 0, "RO", 9000, 6)
    // byte[] = {0x54,0x65,0x63,0x68,0x6D,0x61,0x6E,0xE9,0x81,0x94,0xE6,0x98}
    // to int32[]      value = {1415930728,1835101929,-2120948072}

value = modbus_read_int32("TCP_1", 0, "RI", 7001, 12)
    // byte[] = {0x29,0x30,0x9F,0x4C,0xC3,0x7C,0x99,0x9A,0x44,0x5E,0xEC,0xCD,0x42,0xB4,0x00,0x00,
    //           0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
    // to int32[]      value = {691052364,-1015244390,1147071693,1119092736,-2147483648,0}

value = modbus_read_int32("TCP_1", 0, "RI", 7301, 6)
    // byte[] = {0x07,0xE2,0x00,0x05,0x00,0x12,0x00,0x0F,0x00,0x31,0x00,0x23}
    // to int32[]      value = {132251653,1179663,3211299}

value = modbus_read_int32("TCP_1", 0, "RI", 7301, 6, 0) // byte[2][3][0][1]
    // byte[] = {0x07,0xE2,0x00,0x05,0x00,0x12,0x00,0x0F,0x00,0x31,0x00,0x23}
    // to int32[]      value = {0x000507E2,0x000F0012,0x00230031} = {329698,983058,2293809}
```

10.4 modbus_read_float()

Modbus TCP/RTU 讀取並轉換 modbus 位址資料陣列為 32 位元浮點數型別陣列

語法 1 (TCP/RTU)

```
float[] modbus_read_float(  
    string,  
    byte,  
    string,  
    int,  
    int,  
    int  
)
```

參數

string TCP/RTU 的裝置名稱 (在 Modbus Device 中設定)
byte Slave ID
string 讀取類型
 DO Digital Output (Function Code : 01)
 DI Digital Input (Function Code : 02)
 RO Register Output (Function Code : 03)
 RI Register Input (Function Code : 04)
int 起始位址
int 讀取位址個數
int 讀出後的 Address 資料是依照 Little Endian (CD AB) 或依照 Big Endian (AB CD) 轉成 float 陣列。
 0 Little Endian
 1 Big Endian (預設)

傳回值

float[] 傳回讀取後的 modbus 資料 float 陣列

語法 2 (TCP/RTU)

```
float[] modbus_read_float(  
    string,  
    byte,  
    string,  
    int,  
    int  
)
```

說明

與語法 1 相同，預設將參數 int 填入 1 以 Big Endian (AB CD) 轉換陣列。

modbus_read_float("TCP_1", 0, "DI", 7200, 2) => modbus_read_float("TCP_1", 0, "DI", 7200, 2, 1)

Modbus Address data size

Digital 1 address = 1 bit size

Register 1 address = 2 bytes size

假設 User Setting 使用自定義

TCP device	0	DO	800	4
TCP device	0	DI	7202	3
TCP device	0	RO	9000	6
TCP device	0	RI	7001	12
TCP device	0	RI	7301	6

```
value = modbus_read_float("TCP_1", 0, "DO", 800, 4)
    // byte[] = {0,0,0,0}      to float[]   value = {0} // byte[0][1][2][3]
value = modbus_read_float("TCP_1", 0, "DI", 7202, 3)
    // byte[] = {1,0,0}      to float[]   value = {2.350989E-38} // byte[0][1][2][3] // 不足 [3] 自動補齊
value = modbus_read_float("TCP_1", 0, "RO", 9000, 6)
    // byte[] = {0x54,0x65,0x63,0x68,0x6D,0x61,0x6E,0xE9,0x81,0x94,0xE6,0x98}
    // to float[]      value = {3.940861E+12,4.360513E+27,-5.46975E-38}
value = modbus_read_float("TCP_1", 0, "RI", 7001, 12)
    // byte[] = {0x29,0x30,0x9F,0x4C,0xC3,0x7C,0x99,0x9A,0x44,0x5E,0xEC,0xCD,0x42,0xB4,0x00,0x00,
    //           0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
    // to float[]      value = {3.921802E-14,-252.6,891.7,90,0,0}
value = modbus_read_float("TCP_1", 0, "RI", 7001, 12, 0) // byte[2][3][0][1]
    // to float[]      value = {0x9F4C2930,0x999AC37C,0xECCD445E,0x000042B4,0x00008000,0x00000000}
    //           = {-4.323275E-20,-1.600218E-23,-1.985221E+27,2.392857E-41,4.591775E-41,0}
value = modbus_read_float("TCP_1", 0, "RI", 7301, 6)
    // byte[] = {0x07,0xE2,0x00,0x05,0x00,0x12,0x00,0x0F,0x00,0x3A,0x00,0x26}
    // to float[]      value = {3.400471E-34,1.65306E-39,5.326512E-39}
```

10.5 modbus_read_double()

Modbus TCP/RTU 讀取並轉換 modbus 位址資料陣列為 64 位元浮點數型別陣列

語法 1 (TCP/RTU)

```
double[] modbus_read_double(  
    string,  
    byte,  
    string,  
    int,  
    int,  
    int  
)
```

參數

string TCP/RTU 的裝置名稱 (在 Modbus Device 中設定)
byte Slave ID
string 讀取類型
 DO Digital Output (Function Code : 01)
 DI Digital Input (Function Code : 02)
 RO Register Output (Function Code : 03)
 RI Register Input (Function Code : 04)
int 起始位址
int 讀取位址個數
int 讀出後的 Address 資料是依照 Little Endian (CD AB) 或依照 Big Endian (AB CD) 轉成 double 陣列。
 0 Little Endian
 1 Big Endian (預設)

傳回值

double[] 傳回讀取後的 modbus 資料 double 陣列

語法 2 (TCP/RTU)

```
double[] modbus_read_double(  
    string,  
    byte,  
    string,  
    int,  
    int  
)
```

說明

與語法 1 相同，預設將參數 int 填入 1 以 Big Endian (AB CD) 轉換陣列。

modbus_read_double("TCP_1", 0, "DI", 7200, 2) => **modbus_read_double("TCP_1", 0, "DI", 7200, 2, 1)**

Modbus Address data size

Digital 1 address = 1 bit size

Register 1 address = 2 bytes size

假設 User Setting 使用自定義

TCP device	0	DO	800	4
TCP device	0	DI	7202	3
TCP device	0	RO	9000	6
TCP device	0	RI	7001	12
TCP device	0	RI	7301	6

```
value = modbus_read_double("TCP_1", 0, "DO", 800, 4)
    // byte[] = {0,0,0,0}      to double[]      value = {0} // byte[0][1][2][3][4][5][6][7]

value = modbus_read_double("TCP_1", 0, "DI", 7202, 3)
    // byte[] = {1,0,0}      to double[]      value = {7.2911220195564E-304} // byte[0][1][2][3][4][5][6][7]

value = modbus_read_double("TCP_1", 0, "RO", 9000, 6)
    // byte[] = {0x54,0x65,0x63,0x68,0x6D,0x61,0x6E,0xE9,0x81,0x94,0xE6,0x98}
    // to double[]      value = {3.65481260356117E+98,-4.87647898854073E-301}

value = modbus_read_double("TCP_1", 0, "RI", 7001, 12)
    // byte[] = {0x29,0x30,0x9F,0x4C,0xC3,0x7C,0x99,0x9A,0x44,0x5E,0xEC,0xCD,0x42,0xB4,0x00,0x00,
    //           0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
    // to double[]      value = {2.76472410615396E-110,2.2818627604613E+21,0}

value = modbus_read_double("TCP_1", 0, "RI", 7001, 12, 0)      // byte[6][7][4][5][2][3][0][1]
    // to double[]      value = {0x999AC37C9F4C2930,0x000042B4ECCD445E,0x000000000000008000}
    // = {-2.4604103205376E-185,3.62371629877526E-310,1.6189543082926E-319}

value = modbus_read_double("TCP_1", 0, "RI", 7301, 6)
    // byte[] = {0x07,0xE2,0x00,0x05,0x00,0x12,0x00,0x10,0x00,0x0B,0x00,0x29}
    // to double[]      value = {1.06475148078395E-270,1.52982527955113E-308}
```

10.6 modbus_read_string()

Modbus TCP/RTU 讀取並轉換 modbus 位址資料陣列為 UTF8 編碼的字串文本

語法 1 (TCP/RTU)

```
string modbus_read_string(  
    string,  
    byte,  
    string,  
    int,  
    int,  
    int  
)
```

參數

string TCP/RTU 的裝置名稱 (在 Modbus Device 中設定)
byte Slave ID
string 讀取類型
 DO Digital Output (Function Code : 01)
 DI Digital Input (Function Code : 02)
 RO Register Output (Function Code : 03)
 RI Register Input (Function Code : 04)
int 起始位址
int 讀取位址個數
int 讀出後的 Address 資料是依照 Little Endian (CD AB) 或依照 Big Endian (AB CD) 轉成字串。
*此參數無效，僅支援 int32, float, double，字串採 UTF8 編碼方式依位址序轉為字串。
0 Little Endian
1 Big Endian (預設)

傳回值

string 傳回讀取後的 modbus 資料 string UTF8 字串格式 (遇到 0x00 結束)

語法 2 (TCP/RTU)

```
string modbus_read_string(  
    string,  
    byte,  
    string,  
    int,  
    int)
```

)

說明

與語法 1 相同，預設將參數 int 填入 1 以 Big Endian (AB CD) 轉換陣列。

modbus_read_string("TCP_1", 0, "RO", 9000, 2) => modbus_read_string("TCP_1", 0, "RO", 9000, 2, 1)

Modbus Address data size

Digital 1 address = 1 bit size

Register 1 address = 2 bytes size

假設 User Setting 使用自定義

TCP device 0 RO 9000 12

modbus_write("TCP_1", 0, "RO", 9000) = "1234 達明機器手臂"

// 未定義寫入位址個數，預設位址個數 0 則寫入完整資料長度，共 22 bytes

// 寫入 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 address = 6 bytes size

// to string = 1234◆

value = **modbus_read_string("TCP_1", 0, "RO", 9000, 6)**

// byte[] = {0x31,0x32,0x33,0x34,0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0x9C}

// to string = 1234 達明◆

value = **modbus_read_string("TCP_1", 0, "RO", 9000, 12)**

// byte[] = {0x31,0x32,0x33,0x34,0xE9,0x81,0x94,0xE6,0x98,0x8E,

0xE6,0x9C,0xBA,0xE5,0x99,0xA8,0xE6,0x89,0x8B,0xE8,0x87,0x82, 0x41,0x42}

// to string = 1234 達明機器手臂 AB // UTF8 轉碼

// 寫入時不會帶入 0x00 結尾，當讀取 12 個位址資料時(24 bytes)，會讀出超過預期的值

modbus_write("TCP_1", 0, "RO", 9000) = "1234"+Ctrl("\0")

// 寫入 byte[] = {0x31,0x32,0x33,0x34,0x00} // 需要寫入 3 個 Register 位址

value = **modbus_read_string("TCP_1", 0, "RO", 9000, 5)**

// byte[] = {0x31,0x32,0x33,0x34,0x00,0x00, 0x94,0xE6,0x98,0x8E} // 後面是原本位址內的內容

// to string = 1234 // 轉 UTF8 遇 0x00 結束

10.7 modbus_write()

Modbus TCP/RTU 寫入

語法 1 (TCP/RTU)

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

參數

string TCP/RTU 的裝置名稱 (在 Modbus Device 中設定)

string TCP/RTU 裝置名稱下預先設置的讀寫參數名稱 (在 Modbus Device 中設定)

? 寫入資料，依照預先設置的讀寫參數，支援如下表

Signal Type	Function Code	Type	寫入 ? 型別	輸入值
Digital Output	05	byte	byte	(H: 1)(L: 0)
		bool	bool	(H: true)(L: false)
Register Output	06	byte	byte	
		bool	bool	
		int16	int	
Register Output	16	int32	int	
		float	float	
		double	double	
		string	string	

* int32, float, double 會依使用者選擇 Little Endian (CD AB) 或 Big Endian (AB CD) 轉換後寫入

* string 會依 UTF8 格式轉換後寫入

* 預先設置不支援陣列值寫入，若需寫入陣列值，得採使用自定義方式 (語法 3/4)

int 最大寫入位址個數，只對 string 型別有效

>0 合法位址長度，依位址個數寫入

<=0 非法位址長度，依寫入資料的完整長度寫入

當省略此參數(即語法2)，表示使用預先設置內的長度設定。

傳回值

bool True 寫入成功

False 寫入失敗 1. 如果 ? 寫入資料值為空字串或空陣列

2. 無法正確發送或接收 Modbus 通訊

語法 2 (TCP/RTU)

```
bool modbus_write(  
    string,  
    string,  
    ?,  
)
```

說明

與語法 1 相同，但省略參數 int 最大寫入位址個數，則優先依預先設置內的位址長度設定值寫入，但若預先設置內的位址長度設定為 $<= 0$ ，則會改依寫入資料的完整長度寫入。

Modbus Address data size

Digital	1 address = 1 bit size
Register	1 address = 2 bytes size

假設 Preset Setting 使用預先設置中定義

preset_800	DO	800	bool
preset_9000	RO	9000	string 4

```
modbus_write("TCP_1", "preset_800", 1)      // write 1 (true)  
modbus_write("TCP_1", "preset_800", 0)      // write 0 (false)  
bool flag = true  
modbus_write("TCP_1", "preset_800", flag)    // write 1 (true)  
modbus_write("TCP_1", "preset_800", false)   // write 0 (false)  
  
string ss = "ABCDEFGHIJKLMNPQRSTUVWXYZ"  
// 未帶位址個數，以預先設置內的位址長度 4，所以 4 RO = 8 bytes size 可以寫入  
modbus_write("TCP_1", "preset_9000", ss)      // write ABCDEFGH // 超過無法寫入  
// 未帶位址個數，以預先設置內的位址長度 4，所以 4 RO = 8 bytes size 可以寫入  
modbus_write("TCP_1", "preset_9000", "1234567") // write 1234567\0 // 補寫 0x00 滿足 4 個位址  
// 帶位址個數 0，依寫入資料長度寫入，"09AB123" 需要 4 個位址才足夠  
modbus_write("TCP_1", "preset_9000", "09AB123", 0) // write 09AB123\0 // 補寫 0x00 滿足 4 個位址  
// 帶位址個數 5，依位址個數寫入，所以 5 RO = 10 bytes size 可以寫入  
modbus_write("TCP_1", "preset_9000", "09AB1234", 5) // write 09AB1234 // 但資料只需 4 個位址
```

語法 3 (TCP/RTU)

```
bool modbus_write(  
    string,  
    byte,  
    string,  
    int,  
    ?,  
    int  
)
```

參數

string TCP/RTU 的裝置名稱 (在 Modbus Device 中設定)

byte Slave ID

string 寫入類型

DO Digital Output (Function Code : 05/15)

RO Register Output (Function Code : 06/16)

int 起始位址

? 寫入資料

Signal Type	Function Code	寫入 ? 型別	輸入值
Digital Output	05	byte	(H: 1)(L: 0)
		bool	(H: true)(L: false)
Digital Output	15	byte[]	(H: 1)(L: 0)
		bool[]	(H: true)(L: false)
Register Output	06	byte	
		bool	
Register Output	16	int	
		float	
		double	
		string	
		byte[]	
		int[]	
		float[]	
		double[]	
		string[]	
		bool[]	

* 自定義 modbus_write 僅會依照 Big-Endian (AB CD) 格式寫入

* int 為 int32，若需要寫入 int16 得先使用 GetBytes() 轉成 byte[] 後再寫入

int 最大寫入位址個數，只對 string 型別與陣列型別有效

>0 合法位址長度，依位址個數寫入

<=0 非法位址長度，依寫入資料的完整長度寫入

傳回值

bool	True	寫入成功
	False	寫入失敗
1. 如果 ? 寫入資料值為空字串或空陣列		
2. 無法正確發送或接收 Modbus 通訊		

語法 4 (TCP/RTU)

```
bool modbus_write(  
    string,  
    byte,  
    string,  
    int,  
    ?  
)
```

說明

與語法 3 相同，預設將參數 int 最大寫入位址個數設為 0，即依寫入資料的完整長度寫入。

`modbus_write("TCP_1", 0, "RO", 9000, bb) => modbus_write("TCP_1", 0, "RO", 9000, bb, 0)`

Modbus Address data size

Digital	1 address = 1 bit size
Register	1 address = 2 bytes size

假設 User Setting 使用自定義

TCP device	0	DO	800	4
TCP device	0	RO	9000	12

```
byte[] bb = {10, 20, 30}  
  
modbus_write("TCP_1", 0, "DO", 800, bb)      // write 1,1,1    // 零值寫入 0，非零值寫入 1  
modbus_write("TCP_1", 0, "DO", 800, bb, 2)    // write 1,1      // 帶位址個數 2 只會寫入 2 個位址  
modbus_write("TCP_1", 0, "DO", 800, true)     // write 1  
  
int i = 10000  
  
modbus_write("TCP_1", 0, "RO", 9000, i) // write 0x00,0x00,0x27,0x10// 預設 int32 BigEndian (AB CD) 寫  
bb = GetBytes(i, 0, 1)                  // bb = {0x10,0x27}          // 轉 int16 LittleEndian (CD AB)  
  
modbus_write("TCP_1", 0, "RO", 9000, bb)// write 0x10,0x27  
  
string[] n = {"ABC", "12", "34"}  
  
modbus_write("TCP_1", 0, "RO", 9000, n, 2) // write ABC1           // 僅 2 個位址可以寫入，超過無法寫入  
modbus_write("TCP_1", 0, "RO", 9000, n, 5) // write ABC12340        // 需 4 個位址 (0xAB 0xC1 0x23 0x40)
```

11. TM Ethernet Slave

Ethernet Slave 功能是由 Socket TCP 所建立的 client/server 連線架構，當功能設定啟動時，手臂將建立 Socket TCP Listener Server，且定時（但不保證實時性）週期性地將手臂狀態與資料，發送給所有連線進來的 client 端，或是可以接收 client 端所發進來的內容，並執行對應的指令，或是更新對應的資料。

當 TMflow 程式啟動後，Ethernet Slave 會依照設定檔自動啟動或不啟動，並在 TMflow 程式關閉時（或設定關閉時），才關閉 Ethernet Slave，其所建立的 IP 及 Port 會顯示於右側的 Notice Log 內。

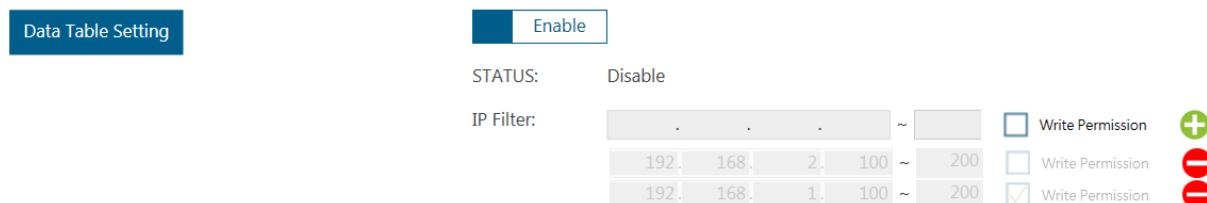
IP TMflow → System → Network → IP Address

Port 5891

發送週期為 100Hz (不保證實時性)

11.1 界面設定

Modbus Slave ✓ Ethernet Slave



啟用/禁用 開啟或關閉 Ethernet Slave

IP 過濾器 IP 白名單

當有設置 IP 過濾器時，表示只有這些 IP 範圍內可以連線，其他 IP 則會拒絕連線
當沒有設置 IP 過濾器時，則表示接受任何 IP 連線

寫入權限 勾選時，表示 IP 範圍內具有寫入權限，未勾選時，表示 IP 範圍內不具有寫入權限

例如設置 IP 過濾器，

1. 192.168.2.100 ~ 200，表示 IP 192.168.2.100, 192.168.2.101, ..., 192.168.2.200 可接受連線
2. 192.168.1.100 ~ 200，表示 IP 192.168.2.100, 192.168.1.101, ..., 192.168.1.200 可接受連線
若 client 端的 IP 不在這兩組範圍內的 IP，則將拒絕該 client 連線。

第 2 組 192.168.1.100 ~ 200 具有寫入權限，由這組 IP 連線進來的 client，當對 Ethernet Slave 發送內容時，會使 Ethernet Slave 做資料寫入的動作。第 1 組 192.168.2.100 ~ 200 因不具有寫入權限，當對 Ethernet Slave 發送內容時，將不會做資料寫入的動作，且會回應 WritePermission 的錯誤碼。

11.2 svr_read()

讀取本機 Ethernet Slave 通訊資料表的項目值

語法 1

```
? svr_read(  
    string  
)
```

參數

string 項目名稱

傳回值

? 依設定的型別傳回值

說明

如勾選 TCP_Value float[]、Ctrl_D00 byte、Ctrl_D01 byte、g_ss string[] 做為通訊資料表

Predefined		User defined		Global Variable				
	Item	Description		Data Type	Data Length	Accessibility	Write Restriction	Note
<input checked="" type="checkbox"/>	TCP_Value	TCP Value		float	6	R		Unit: mm
<input checked="" type="checkbox"/>	Ctrl_D00	Digital Output 0		byte	1	R/W		High: 1 Low: 0
<input checked="" type="checkbox"/>	Ctrl_D01	Digital Output 1		byte	1	R/W		High: 1 Low: 0
<input checked="" type="checkbox"/>	g_ss			string[]	0	R/W		

```
float[] fval = svr_read("TCP_Value") // fval = {1, 1, 1, 0.1, 0.2, 0.1}  
byte b0 = svr_read("Ctrl_D00") // b0 = 0  
byte b1 = svr_read("Ctrl_D01") // b1 = 1  
string[] ss = svr_read("g_ss") // ss = {"Hi", "TM", "Robot"} // 全域變數也可直接用 g_ss 變數名稱  
byte st = svr_read("Robot_Link") // st = 0 (手臂未連線) 1 (手臂已連線)  
  
float[] fval = svr_read("TCP_Value") // 報錯，假設 未啟動Ethernet Slave  
float[] fval = svr_read("TCP_Value1") // 報錯，項目名稱 TCP_Value1 不存在  
  
float[] fval = svr_read("Coord_Base_Flange") // 報錯，項目名稱 Coord_Base_Flange 不存在  
// 項目名稱 Coord_Base_Flange 存在於系統定義區內，但因未勾選為通訊資料，而不存在
```

11.3 svr_write()

寫入本機 Ethernet Slave 通訊資料表的項目值

語法 1

```
bool svr_write(  
    string,  
    ?  
)
```

參數

string 項目名稱
? 項目值

傳回值

bool	True	寫入成功
	False	寫入失敗
		1. 項目名稱不存在
		2. 項目名稱唯讀，無法寫入
		3. 如果 ? 寫入資料值與設定的型別不符合

說明

如勾選 TCP_Value float[]、Ctrl_D00 byte、Ctrl_D01 byte、g_ss string[] 做為通訊資料表

Predefined		User defined	Global Variable				
	Item	Description	Data Type	Data Length	Accessibility	Write Restriction	Note
<input checked="" type="checkbox"/>	TCP_Value	TCP Value	float	6	R		Unit: mm
<input checked="" type="checkbox"/>	Ctrl_D00	Digital Output 0	byte	1	R/W		High: 1 Low: 0
<input checked="" type="checkbox"/>	Ctrl_D01	Digital Output 1	byte	1	R/W		High: 1 Low: 0
<input checked="" type="checkbox"/>	g_ss		string[]	0	R/W		

```
float[] tvalue = {1,2,3,0.1,0.2,0.3}  
flag = svr_write("TCP_Value", tvalue)           // flag = false 唯讀，無效操作 (不報錯)  
flag = svr_write("Ctrl_D00", 1)                 // flag = true , Ctrl_D00 = 1  
flag = svr_write("Ctrl_D01", 0)                 // flag = true , Ctrl_D01 = 0  
  
flag = svr_write("TCP_Value", tvalue)           // 報錯，假設 未啟動Ethernet Slave  
flag = svr_write("TCP_Value1", tvalue)           // 報錯，項目名稱 TCP_Value1 不存在  
flag = svr_write("Ctrl_D00", "True")            // 報錯，項目名稱 Ctrl_D00 寫入字串值 (型別定義為 byte)
```

11.4 資料表

Ethernet Slave 與 client 端間的發送內容，可由資料表中的項目列表，自定義選擇所需的發送/接收內容，以及可設定通訊協定中 Data 區段資料格式，之後可儲存成通訊檔案，當 Ethernet Slave 啟動時，會依照所選通訊檔案內的項目列表，建立對應項目的資料內容，並依固定週期(不保證實時性)向已連接的 client 端發送內容，其 Data 區段資料格式，會以通訊檔案中所設定的格式定義，而 client 端則可使用任一支援的資料格式向 server 端發送。

通訊協定中 Data 區段資料格式，可支援

BINARY 二進制格式，轉為 Byte 陣列 (Little Endian / UTF8)

STRING 字串格式，類似外部命令格式

JSON JSON 字串格式

Receive/Send Data Table Setting

Setting User Defined File Name:	開啟	Setting Transmit File Name:	開啟	Communicate Mode :	BINARY																																																																												
<table border="1"><thead><tr><th>Predefined</th><th>User defined</th><th colspan="4">Global Variable</th></tr><tr><th>Item</th><th>Description</th><th>Data Type</th><th>Data Length</th><th>Accessibility</th><th>WritableMode</th><th>Note</th></tr></thead><tbody><tr><td>Robot_Error</td><td>Error or Not</td><td>bool</td><td>1</td><td>R</td><td></td><td>Yes:1 No: 0</td></tr><tr><td>Project_Run</td><td>Project Running or Not</td><td>bool</td><td>1</td><td>R</td><td></td><td>Yes:1 No: 0</td></tr><tr><td>Project>Edit</td><td>Project Editing or Not</td><td>bool</td><td>1</td><td>R</td><td></td><td>Yes:1 No: 0</td></tr><tr><td>Project_Pause</td><td>Project Pause or Not</td><td>bool</td><td>1</td><td>R</td><td></td><td>Yes:1 No: 0</td></tr><tr><td>Get_Control</td><td>Get Control or Not</td><td>bool</td><td>1</td><td>R</td><td></td><td>Yes:1 No: 0</td></tr><tr><td>Camera_Light</td><td>Light</td><td>byte</td><td>1</td><td>R/W</td><td></td><td>Enable: 1 Disable: 0</td></tr><tr><td>Safeguard_A</td><td>Safety IO (Safeguard Port A trigger)</td><td>bool</td><td>1</td><td>R</td><td></td><td>Triggered: 1 Restored: 0</td></tr><tr><td>ESTOP</td><td>Emergency Stop</td><td>bool</td><td>1</td><td>R</td><td></td><td>Triggered: 1 Restored: 0</td></tr><tr><td>Coord_Base_F</td><td>Cartesian coordinate w.r.t. current Base</td><td>float</td><td>6</td><td>R</td><td></td><td>Unit: mm</td></tr></tbody></table>						Predefined	User defined	Global Variable				Item	Description	Data Type	Data Length	Accessibility	WritableMode	Note	Robot_Error	Error or Not	bool	1	R		Yes:1 No: 0	Project_Run	Project Running or Not	bool	1	R		Yes:1 No: 0	Project>Edit	Project Editing or Not	bool	1	R		Yes:1 No: 0	Project_Pause	Project Pause or Not	bool	1	R		Yes:1 No: 0	Get_Control	Get Control or Not	bool	1	R		Yes:1 No: 0	Camera_Light	Light	byte	1	R/W		Enable: 1 Disable: 0	Safeguard_A	Safety IO (Safeguard Port A trigger)	bool	1	R		Triggered: 1 Restored: 0	ESTOP	Emergency Stop	bool	1	R		Triggered: 1 Restored: 0	Coord_Base_F	Cartesian coordinate w.r.t. current Base	float	6	R		Unit: mm
Predefined	User defined	Global Variable																																																																															
Item	Description	Data Type	Data Length	Accessibility	WritableMode	Note																																																																											
Robot_Error	Error or Not	bool	1	R		Yes:1 No: 0																																																																											
Project_Run	Project Running or Not	bool	1	R		Yes:1 No: 0																																																																											
Project>Edit	Project Editing or Not	bool	1	R		Yes:1 No: 0																																																																											
Project_Pause	Project Pause or Not	bool	1	R		Yes:1 No: 0																																																																											
Get_Control	Get Control or Not	bool	1	R		Yes:1 No: 0																																																																											
Camera_Light	Light	byte	1	R/W		Enable: 1 Disable: 0																																																																											
Safeguard_A	Safety IO (Safeguard Port A trigger)	bool	1	R		Triggered: 1 Restored: 0																																																																											
ESTOP	Emergency Stop	bool	1	R		Triggered: 1 Restored: 0																																																																											
Coord_Base_F	Cartesian coordinate w.r.t. current Base	float	6	R		Unit: mm																																																																											
<div style="text-align: right;"><button>清除</button> <button>儲存</button></div>																																																																																	

項目名稱 **Robot_Link** 為 Ethernet Slave 內置項目資料，型別為 byte 唯讀屬性，說明是否與手臂連線

1. **Predefined** 系統定義區

由 TMflow 系統所定義的項目與設定，並由 TMflow 更新項目的資料內容。定義的項目一般是與手臂相關的狀態，如手臂座標、專案狀態、電控箱狀態等等；或是與 IO 輸入/輸出相關狀態，如數位輸入/數位輸出、類比輸入/類比輸出等等

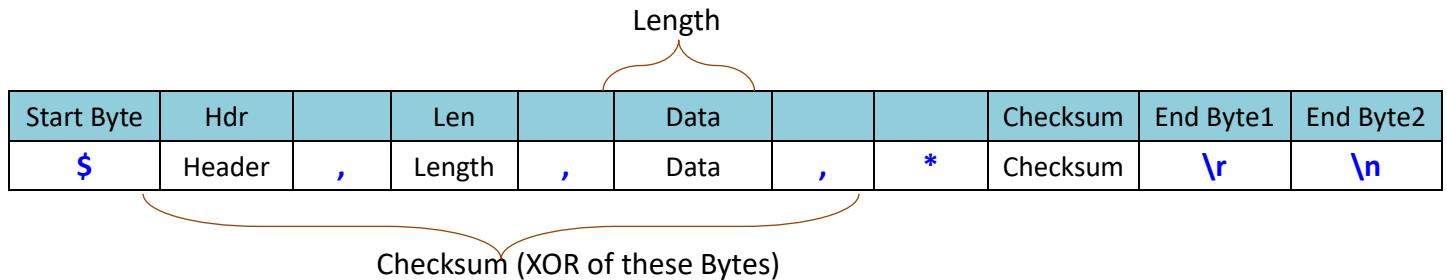
2. **Userdefined** 使用者定義區

由 TMflow 使用者所定義的項目與設定，並由專案編程來讀/寫項目資料，或是由外部使用者透過通訊協定來讀/寫項目資料。藉由使用者定義區，可使專案編程與外部通訊裝置，做為一種資料交握協定。使用者定義區中的項目列表，可以儲存成使用者定義檔，未來可再做編輯或資料交換使用。

3. *GlobalVariable* 全域變數定義區

由 TMflow 使用者在全域變數功能下，所建立的變數列表，在專案編程中可直接使用變數名稱進行讀/寫操作，而外部通訊裝置，則可使用通訊協定的方式讀/寫全域變數。

11.5 通訊協定



Name	Size	ASCII	HEX	Description
Start Byte	1	\$	0x24	Start Byte for Communication
Header	X			Header for Communication
Separator	1	,	0x2C	Separator between Header and Length
Length	Y			Length of Data
Separator	1	,	0x2C	Separator between Length and Data
Data	Z			Communication Data
Separator	1	,	0x2C	Separator between Data and Checksum
Sign	1	*	0x2A	Begin Sign of Checksum
Checksum	2			Checksum of Communication
End Byte 1	1	\r	0x0D	
End Byte 2	1	\n	0x0A	End Byte of Communication

* 與"外部命令" 使用相同的通訊協定格式

1. Header

定義通訊封包的用途，不同的 Header 對於通訊封包及 Data 的定義都會有所不同。

- **TMSVR** 定義 TM Ethernet Slave 功能
 - **CPERR** 定義通訊封包錯誤 (如 Packet 錯誤、Checksum 錯誤、Header 錯誤 ...等)
- * 與"外部命令" 使用相同的定義內容

2. Length

長度指出 Data 所佔用的 UTF8 bytes 長度，可以使用十進制、十六進制、或二進制的書寫方式，最大表示長度為 int 32 bits。

如 \$TMSVR,100,Data,*CS\r\n // 十進制 100，指出 Data 長度為 100 bytes
 \$TMSVR,0x100,Data,*CS\r\n // 十六進制 0x100，指出 Data 長度為 256 bytes
 \$TMSVR,0b100,Data,*CS\r\n // 二進制 0b100，指出 Data 長度為 4 bytes
 \$TMSVR,8,1,達明,*CS\r\n // 指出 Data 1,達明 長度為 8 bytes (UTF8)

3. Data

通訊封包內容，可以支援任意字元（包含 0x00 .. 0xFF 並採用 UTF8 編碼），資料長度可依據 Length 來決定。而 Data 內所定義用途與說明，則須再依據 Header 定義。

4. Checksum

通訊封包的校驗碼，計算方式採用 XOR (exclusive OR)，計算範圍為 \$ 與 * 之間的所有 Bytes (不包含 \$ 與 *)，如下表示

\$TMSVR,100,Data,*CS\r\n

Checksum = Byte[1] ^ Byte[2] ... ^ Byte[N-6]

其中 Checksum 的書寫方式，固定為 2 bytes，且採用十六進制方式書寫（但不書寫 0x）。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)

11.6 TMSVR

Start Byte	Hdr		Len		Data			Checksum	End Byte1	End Byte2
\$	TMSVR	,	Length	,	Data	,	*	Checksum	\r	\n
ID		Mode	Content		Item and Value					
Transaction ID		,	0/1/2/3/ 11/12/13	,						

TMSVR 定義為 TM Ethernet Slave 通訊協定使用，其中對於封包的 Data 區段再分成三個部分，分別為 ID (Transaction ID)、Mode (Content Mode)、Content (Item and Value)，三個部分依分隔符號逗號隔開，說明如下

ID 傳輸編號，可用任意的英數字表示 (若遇到非英數字的 byte 時，將會回報 CPERR 04 錯誤)，做為通訊封包回應時，一個識別回應哪組命令的傳輸編號。

,

Mode 資料內容的格式與模式，用以指出 Content 的資料格式

0 表示 Server 回應 Client 的命令，格式為字串

1 表示 Content 的資料格式為 BINARY，採二進制格式

2 表示 Content 的資料格式為 STRING，採字串格式

3 表示 Content 的資料格式為 JSON，採 JSON 字串格式

11 表示 Content 的資料格式為 BINARY，採二進制格式 (Request read)

12 表示 Content 的資料格式為 STRING，採字串格式 (Request read)

13 表示 Content 的資料格式為 JSON，採 JSON 字串格式 (Request read)

- 1/2/3 為 client write to server 及 client read from server 功能，其中 client read from server 方式，為 server 端週期性向已連接的 client 端發送內容

- 11/12/13 為 Request read 式的 client read from server 功能，係由 client 端發出項目讀取請求，再由 server 端做項目值回應給 client 端

,

Content 資料內容，格式依照 Mode 定義。

說明

TMSVR 命令是 client/server 通訊雙向使用，在常態情況下，會由 server 端依資料表設定的項目列表，將資料內容週期性發送給已連接的 client 端，當 server 發送給 client 時表示是資料發送(client read from server)，則 client 不需要回應 server；當 client 發送給 server 時是資料接收(client write to server)，則 server 需要回應 client。

傳輸編號，在 server 做週期性的資料發送時，會由編號 0 每次加 1 直至編號 9 之後，再回到編號 0 使用。若在 client 發送資料寫入給 server 時，傳輸編號可由 client 端自定義任意的英數字，在檢查通訊封包格式正確後，server 會依據封包內的傳輸編號，回應 client 端命令處理狀態。

在 client 發送資料寫入給 server 時，server 會檢查所有寫入條件是否正確，才會執行資料寫入，當有一個資料項目不正確時，則不會執行資料寫入。其檢查的寫入條件有

1. 傳輸的資料格式 (Mode)
2. 連線 IP 是否有寫入權限 (IP Filter with Write Permission)
3. 資料內容是否符合資料格式 (Mode and Content)
4. 寫入項目是否存在
5. 寫入項目是否可寫入
6. 寫入 M/A 模式是否正確
7. 寫入的值是否與設定型別相符

而當 Mode 為 11/12/13 時，則採用 Request read 的方式，由 client 端發出項目請求，再由 server 端回應項目值，其請求的項目可在 Predefined 區中所有列入的項目，不限於需勾選週期性發送，但 Userdefined 區及 GlobalVariable 區，因為是由使用者定義，仍需勾選週期性發送，才能取得請求。

由 client 發出項目請求時，不限於只取得一個項目，可以一次取得多個項目，當請求項目成功時，server 端會依照 Mode 11/12/13 對應的格式發送項目值給 client 端。但若請求項目失敗時，例如項目名稱不存在，server 端則會依照 Mode 0 的格式發送命令處理狀態。

0. Mode=0 由 Server 回應 Client 命令處理狀態

當手臂 server 接收到 client 發送的資料寫入時，在 server 處理通訊封包後，會回應 client 封包處理狀態。回應的封包格式為字串格式，其內容為

Data

ID		Mode		Error Code		Error Description
Transaction ID	,	0	,	00 .. 07	,	

Transaction ID 由 client 發送命令時，所定義的 Transaction ID，做為 server 回應使用

Mode 0 由 server 回應 client

Error Code 定義錯誤碼，固定為 2 bytes，且採用十六進制方式書寫 (但不書寫0x)

00 寫入正確，無錯誤。

01 傳輸的資料格式或模式不支援。(如: Mode=99)

02 連線的 client 不具有寫入權限。(IP filter without write permission)

03 傳輸的資料格式與 content 的資料格式不符合。(如: Mode=3但卻不是JSON資料)

04 寫入(或讀取)的項目不存在。

05 無法寫入唯讀項目。

06 寫入時的 M/A 模式不正確。

07 寫入的值與設定的型別或大小不符合。

Error Description 錯誤碼描述，接續 Error Code 後的描述

00 OK

01 NotSupport

02 WritePermission

03 InvalidData

04NotExist;XXX // ;XXX 表示哪個資料項目錯誤

05 ReadOnly;XXX

06 ModeError;XXX

07 ValueError;XXX

```

< $TMSVR,15,S0,2,Ctrl_D00=1,*76\r\n      // 編號 S0、字串格式、設 Ctrl_D00=1
> $TMSVR,10,S0,0,00,OK,*18\r\n          // 回應編號 S0、回應模式 0、錯誤碼 00、寫入正確

< $TMSVR,16,S1,99,Ctrl_D00=1,*46\r\n      // 編號 S1、模式 99
> $TMSVR,18,S1,0,01,NotSupport,*0E\r\n // 回應編號 S1、回應模式 0、錯誤碼 01、模式不支援

< $TMSVR,15,S2,2,Ctrl_D00=1,*74\r\n      // 編號 S2、字串格式、設 Ctrl_D00=1
> $TMSVR,23,S2,0,02,WritePermission,*6A\r\n
    // 回應編號 S2、模式 0、錯誤碼 02、連線的 client 端不具有寫入權限

< $TMSVR,15,S3,3,Ctrl_D00=1,*74\r\n      // 編號 S3、JSON 格式、設 Ctrl_D00=1
> $TMSVR,19,S3,0,03,InvalidData,*74\r\n
    // 回應編號 S3、模式 0、錯誤碼 03、資料格式(JSON) 與 content 資料格式(STRING) 不符合

< $TMSVR,16,S4,2,Ctrl_DO32=1,*40\r\n      // 編號 S4、字串格式、設 Ctrl_DO32=1
> $TMSVR,26,S4,0,04,NotExist;Ctrl_DO32,*58\r\n
    // 回應編號 S4、模式 0、錯誤碼 04、Ctrl_DO32 項目不存在

< $TMSVR,17,S5,2,Robot_Link=1,*07\r\n      // 編號 S5、字串格式、設 Robot_Link=1
> $TMSVR,27,S5,0,05,ReadOnly;Robot_Link,*1E\r\n
    // 回應編號 S5、模式 0、錯誤碼 05、Robot_Link 項目唯讀

假設 User Defined 下建立 Item: adata, Type: int, Size: 4, Write: Auto

< $TMSVR,20,S6,2,adata={1,2,3,4},*55\r\n // 編號 S6、字串格式、設 adata={1,2,3,4}
> $TMSVR,23,S6,0,06,ModeError;adata,*2D\r\n
    // 回應編號 S6、模式 0、錯誤碼 06、寫入時的 M/A 模式不符合 (假設寫入時是 Manual Mode)

< $TMSVR,18,S7,2,adata={1,2,3},*47\r\n      // 編號 S7、字串格式、設 adata={1,2,3}
> $TMSVR,24,S7,0,07,ValueError;adata,*42\r\n
    // 回應編號 S7、模式 0、錯誤碼 07、寫入值與設定型別或大小不符合 (定義 Size:4 個，但卻只寫 3 個)

```

1. Mode = 1 BINARY

資料內容使用二進制方式傳輸，將資料項目名稱與值，依照數值 Little Endian、字串值 UTF8 方式轉換為 byte 陣列，其格式為

Data

ID		Mode		Content
Transaction ID	,	1	,	Item and Value

Length of Item	Item	Length of Value	Value	...
2 bytes Little Endian	UTF8	2 bytes Little Endian	Little Endian / UTF8	...

Length of Item 項目名稱長度，2 bytes (Little Endian)，數值 0 .. 65535，指出接續的項目名稱長度

Item 項目名稱

Length of Value 資料值長度，2 bytes (Little Endian)，數值 0 .. 65535，指出接續的資料值長度

Value 資料值

如勾選 TCP_Value float[] 及 Ctrl_DOO byte 做為通訊資料，依 BINARY 模式傳輸

Predefined	User defined	Global Variable				
Item	Description	Data Type	Data Length	Accessibility	Write Restriction	Note
<input checked="" type="checkbox"/> TCP_Value	TCP Value	float	6	R		Unit: mm
<input checked="" type="checkbox"/> Ctrl_DOO	Digital Output 0	byte	1	R/W		High: 1 Low: 0

```

> 24 54 4D 53 56 52 2C // $TMSVR, // Header
    36 39 2C // 69, // Length
    30 2C 31 2C // 0,1, // 編號 0 模式 1 二進制
    0A 00 52 6F 62 6F 74 5F 4C 69 6E 6B 01 00 00 // Robot_Link=0 // 名稱佔用 10 bytes，值佔用 1 byte
    09 00 54 43 50 5F 56 61 6C 75 65 18 00 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} // 名稱佔用 9 bytes，值佔用 24 bytes
    08 00 43 74 72 6C 5F 44 4F 30 01 00 00 // Ctrl_DOO=0 // 名稱佔用 8 bytes，值佔用 1 byte
    2C 2A 39 36 0D 0A // ,*96\r\n // Checksum

< 24 54 4D 53 56 52 2C // $TMSVR, // Header
    31 38 2C // 18, // Length
    54 31 2C 31 2C // T1,1, // 編號 T1 模式 1 二進制
    08 00 43 74 72 6C 5F 44 4F 30 01 00 01 // Ctrl_DOO=1 // 名稱佔用 8 bytes，值佔用 1 byte
    2C 2A 37 41 0D 0A // ,*7A\r\n // Checksum
    > $TMSVR,10,T1,0,00,OK,*1E\r\n // server 回應編號 T1、回應模式 0、錯誤碼 00、寫入正確

```

當發送資料項目型別為 string[] 時，在字串陣列元素之間，插入 0x00 0x00 兩個 bytes 作為字串元素的分隔符號。

```
> 24 54 4D 53 56 52 2C      // $TMSVR,      // Header  
    39 30 2C                  // 90,        // Length  
    30 2C 31 2C                // 0,1        // 編號 0 模式 1 二進制  
    0A 00 52 6F 62 6F 74 5F 4C 69 6E 6B 01 00 00  // Robot_Link=0 // 名稱佔用 10 bytes, 值佔用 1 byte  
    09 00 54 43 50 5F 56 61 6C 75 65 18 00 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} // 名稱佔用 9 bytes, 值佔用 24 bytes  
    08 00 43 74 72 6C 5F 44 4F 30 01 00 01  // Ctrl_D00=1 // 名稱佔用 8 bytes, 值佔用 1 byte  
    04 00 67 5F 73 73 0D 00 48 69 00 00 54 4D 00 00 52 6F 62 6F 74  
                                // g_ss={"Hi","TM","Robot"} // 名稱佔用 4 bytes, 值佔用 13 byte  
    2C 2A 44 43 0D 0A          // ,*DC\r\n      // Checksum
```

同樣的，當接收資料項目型別為 string[] 時，在字串陣列元素之間，也需要插入 0x00 0x00 兩個 bytes 作為字串元素的分隔符號。

```
< 24 54 4D 53 56 52 2C      // $TMSVR,      // Header  
    32 35 2C                  // 25,        // Length  
    54 32 2C 31 2C              // T2,1,       // 編號 T2 模式 1 二進制  
    04 00 67 5F 73 73 0C 00 48 65 6C 6F 00 00 57 6F 72 6C 64  
                                // g_ss={"Hello", "World"} // 名稱佔用 4 bytes, 值佔用 12 byte  
    2C 2A 30 32 0D 0A          // ,*02\r\n      // Checksum  
> $TMSVR,10,T2,0,00,OK,*1D\r\n      // server 回應編號 T2、回應模式 0、錯誤碼 00、寫入正確
```

2. Mode = 2 STRING

資料內容使用字串方式傳輸，將資料項目名稱與值，採用外部命令的 Script 字串，其格式為

Data

ID		Mode		Content
Transaction ID	,	2	,	Item and Value

Item	=	Value	\r\n	...
------	---	-------	------	-----

Item 項目名稱

= 等於

Value 資料值

\r\n 換行符號，若有下筆項目資料才需要，做為與下筆項目資料的分隔符號

如勾選 TCP_Value float[]、Ctrl_DO0 byte、Ctrl_DO1 byte、g_ss string[] 做為通訊資料，依 STRING 模式傳輸

Predefined		User defined	Global Variable				
	Item	Description	Data Type	Data Length	Accessibility	Write Restriction	Note
<input checked="" type="checkbox"/>	TCP_Value	TCP Value	float	6	R	Unit: mm	
<input checked="" type="checkbox"/>	Ctrl_DO0	Digital Output 0	byte	1	R/W	High: 1 Low: 0	
<input checked="" type="checkbox"/>	Ctrl_DO1	Digital Output 1	byte	1	R/W	High: 1 Low: 0	
<input checked="" type="checkbox"/>	g_ss		string[]	0	R/W		

```
> $TMSVR,97,2,Robot_Link=0\r\n // 編號 9 模式 2 字串
    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 // 設 Ctrl_DO0=0 // 編號 T2 模式 2 字串
    Ctrl_DO1=1,*34\r\n // 設 Ctrl_DO1=1
    > $TMSVR,10,T2,0,00,OK,*1D\r\n // 回應編號 T2、回應模式 0、錯誤碼 00、寫入正確
```

3. Mode = 3 JSON

資料內容使用 JSON 字串方式傳輸，將資料項目名稱與值，依 JSON 格式進行序列化(Serialize)，其格式為

Data

ID		Mode		Content
Transaction ID	,	3	,	Item and Value

Item 項目名稱

Value 資料值

*當多個項目時，採用 [] 陣列

```
public class TMSVRJsonData
{
    public string Item;
    public object Value;
}
```

如勾選 TCP_Value float[]、Ctrl_DO0 byte、Ctrl_DO1 byte、g_ss string[] 做為通訊資料，依 JSON 模式傳輸

Predefined	User defined	Global Variable						
	Item	Description	Data Type	Data Length	Accessibility	Write Restriction	Note	
<input checked="" type="checkbox"/>	TCP_Value	TCP Value	float	6	R		Unit: mm	
<input checked="" type="checkbox"/>	Ctrl_DO0	Digital Output 0	byte	1	R/W		High: 1 Low: 0	
<input checked="" type="checkbox"/>	Ctrl_DO1	Digital Output 1	byte	1	R/W		High: 1 Low: 0	
<input checked="" type="checkbox"/>	g_ss		string[]	0	R/W			

```
> $TMSVR,196,5,3,[{"Item":"Robot_Link","Value":0},           // Robot_Link=0 // 編號 5 模式 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      // 回應編號 T9、回應模式 0、錯誤碼 00、寫入正確
```

11. Mode = 11 BINARY (Request read)

資料內容使用二進制方式傳輸，將讀取請求的資料項目名稱，依照數值 Little Endian、字串值 UTF8 方式轉換為 byte 陣列，其格式為

Data (client to server)

ID		Mode		Content
Transaction ID	,	11	,	Item

Length of Item 2 bytes Little Endian	Item UTF8	...
--	---------------------	-----

讀取請求，與 Mode = 1 的差異，
在於不需要 Value 值

Length of Item 項目名稱長度，2 bytes (Little Endian)，數值 0 .. 65535，指出接續的項目名稱長度
Item 項目名稱

如勾選 TCP_Value float[] 及 Ctrl_DOO byte 做為通訊資料，依 BINARY 模式傳輸

Predefined	User defined	Global Variable
<input type="checkbox"/> Item	Description	Data Type Data Length Accessibility Write Restriction Note
<input checked="" type="checkbox"/> TCP_Value	TCP Value	float 6 R Unit: mm
<input checked="" type="checkbox"/> Ctrl_DOO	Digital Output 0	byte 1 R/W High: 1 Low: 0

server 週期發送

```
> 24 54 4D 53 56 52 2C // $TMSVR, // Header
    36 39 2C // 69, // Length
    30 2C 31 2C // 0,1, // 編號 0 模式 1 二進制
    0A 00 52 6F 62 6F 74 5F 4C 69 6E 6B 01 00 00 // Robot_Link=0 // 名稱佔用 10 bytes，值佔用 1 byte
    09 00 54 43 50 5F 56 61 6C 75 65 18 00 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} // 名稱佔用 9 bytes，值佔用 24 bytes
    08 00 43 74 72 6C 5F 44 4F 30 01 00 00 // Ctrl_DOO=0 // 名稱佔用 8 bytes，值佔用 1 byte
    2C 2A 39 36 0D 0A // ,*96\r\n // Checksum
```

client 讀取請求

```
< 24 54 4D 53 56 52 2C // $TMSVR, // Header  
    32 36 2C // 26, // Length  
    51 31 2C 31 31 2C // Q1,11, // 編號 Q1 模式 11 二進制 (Request read)  
    08 00 43 74 72 6C 5F 44 4F 30 // Ctrl_D00 // 名稱佔用 8 bytes  
    08 00 54 43 50 5F 4D 61 73 73 // TCP_Mass // 名稱佔用 8 bytes  
    2C 2A 37 46 0D 0A // ,*7F\r\n // Checksum
```

server 回應項目值

```
> 24 54 4D 53 56 52 2C // $TMSVR, // Header  
    33 35 2C // 35, // Length  
    51 31 2C 31 31 2C // Q1,11, // 回應編號 Q1 模式 11 二進制  
    08 00 43 74 72 6C 5F 44 4F 30 01 00 00 // Ctrl_D00=0 // 名稱佔用 8 bytes, 值佔用 1 byte  
    08 00 54 43 50 5F 4D 61 73 73 04 00 00 00 00 00 // TCP_Mass=0 // 名稱佔用 8 bytes, 值佔用 4 bytes  
    2C 2A 37 38 0D 0A // ,*78\r\n // Checksum
```

* server 回應的內容格式與 Mode = 1 BINARY 相同。

client 讀取請求

```
< 24 54 4D 53 56 52 2C // $TMSVR, // Header  
    32 36 2C // 26, // Length  
    51 32 2C 31 31 2C // Q2,11, // 編號 Q1 模式 11 二進制 (Request read)  
    08 00 43 74 72 6C 5F 44 4F 30 // Ctrl_D00 // 名稱佔用 8 bytes  
    08 00 54 43 50 5F 4D 61 58 58 // TCP_MaXX // 名稱佔用 8 bytes  
    2C 2A 37 43 0D 0A // ,*7C\r\n // Checksum
```

server 回應項目值

```
> $TMSVR,25,Q2,0,04,NotExist;TCP_MaXX,*17\r\n  
// server 回應編號 Q2、模式 0、錯誤碼 04、項目不存在
```

12. Mode = 12 STRING (Request read)

資料內容使用字串方式傳輸，將讀取請求的資料項目名稱，採用外部命令的 Script 字串，其格式為

Data (client to server)

ID		Mode		Content
Transaction ID	,	12	,	Item and Value

Item	\r\n	...	不需要 Value 值
-------------	------	-----	-------------

Item 項目名稱

\r\n 换行符號，若有下筆項目資料才需要，做為與下筆項目資料的分隔符號

如勾選 TCP_Value float[]、Ctrl_DO0 byte、Ctrl_DO1 byte、g_ss string[] 做為通訊資料，依 STRING 模式傳輸

Predefined		User defined	Global Variable				
	Item	Description	Data Type	Data Length	Accessibility	Write Restriction	Note
<input checked="" type="checkbox"/>	TCP_Value	TCP Value	float	6	R	Unit: mm	
<input checked="" type="checkbox"/>	Ctrl_DO0	Digital Output 0	byte	1	R/W	High: 1 Low: 0	
<input checked="" type="checkbox"/>	Ctrl_DO1	Digital Output 1	byte	1	R/W	High: 1 Low: 0	
<input checked="" type="checkbox"/>	g_ss		string[]	0	R/W		

server 週期發送

```
> $TMSVR,97,2,Robot_Link=0\r\n      // Robot_Link=0      // 編號 9 模式 2 字串
    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"}
```

client 讀取請求

```
< $TMSVR,28,Q2,12,Robot_Link\r\n // 項目 Robot_Link // 編號 Q2 模式 12 字串 (Request read)
    TCP_Mass,*0E\r\n           // 項目 TCP_Mass
```

server 回應項目值

```
> $TMSVR,30,Q2,12,Robot_Link=0\r\n // 回應編號 Q2、回應模式 12
    TCP_Mass=0,*09\r\n
```

* server 回應的內容格式與 Mode = 2 STRING 相同。

13. Mode = 13 JSON (Request read)

資料內容使用 JSON 字串方式傳輸，將讀取請求的資料項目名稱，依 JSON 格式進行序列化(Serialize)，其格式為

Data (client to server)

ID		Mode		Content
Transaction ID	,	13	,	Item and Value

Item 項目名稱
Value 資料值
* 當多個項目時，採用 [] 陣列

```
public class TMSVRJsonData
{
    public string Item;
    public object Value;
}
```

* 與 Mode = 3 JSON 共用，採用相同的 class 進行序列化/反序列化，但其中 Value 屬性可不存在

如勾選 TCP_Value float[]、Ctrl_D00 byte、Ctrl_D01 byte、g_ss string[] 做為通訊資料，依 JSON 模式傳輸

Predefined	User defined	Global Variable					
	Item	Description	Data Type	Data Length	Accessibility	Write Restriction	Note
<input checked="" type="checkbox"/>	TCP_Value	TCP Value	float	6	R		Unit: mm
<input checked="" type="checkbox"/>	Ctrl_D00	Digital Output 0	byte	1	R/W		High: 1 Low: 0
<input checked="" type="checkbox"/>	Ctrl_D01	Digital Output 1	byte	1	R/W		High: 1 Low: 0
<input checked="" type="checkbox"/>	g_ss		string[]	0	R/W		

server 週期發送

```
> $TMSVR,196,5,3,[{"Item":"Robot_Link","Value":0},           // Robot_Link=0 // 編號 5 模式 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_D00","Value":0},                                // Ctrl_D00=0
   {"Item":"Ctrl_D01","Value":0},                                // Ctrl_D01=0
   {"Item":"g_ss","Value":["Hi","TM","Robot"]}],*3A\r\n          // g_ss={"Hi","TM","Robot"}
```

client 讀取請求

```
< $TMSVR,27,Q3,13,[{"Item":"TCP_Mass"}],*3C\r\n          // 編號 Q3 模式 13 JSON (Request read)
```

server 回應項目值

```
> $TMSVR,39,Q3,13,[{"Item":"TCP_Mass","Value":0.0}],*40\r\n // 回應編號 Q3、回應模式 13
```

* server 回應的內容格式與 Mode = 3 JSON 相同。

12. Profinet 函式

機器人與外部控制器以Profinet通信協定進行通訊。在Profinet通信協定的機制下，機器人做為Profinet IO裝置，提供外部設備讀寫機器人資料；同時TMflow可透過Profinet函式監控機器人「自外部設備接收的資料表」與「傳送給外部設備的資料表」，亦可改變「傳送給外部設備的資料表」中的使用者自定義區域。

通訊資料表：

資料表類型分為輸入/輸出兩式，輸入資料表是自外部設備發送至機器人，輸出資料表是自機器人發送至外部設備；此二資料表內容皆包含「系統定義區」和「使用者定義區」兩大資料區。

1. 系統定義區：由機器人系統所定義的項目與設定，並由機器人系統或外部裝置更新資料內容。定義的項目與機器人狀態相關，如手臂座標、專案狀態、電控箱狀態等等；或是與 IO 輸入/輸出相關狀態，如數位輸入/數位輸出、類比輸入/類比輸出等等。使用者可透過Profinet函式讀取「輸入資料表」與「輸出資料表」中的「系統定義區」。
2. 使用者定義區：由使用者所定義的項目與設定，並由使用者或外部裝置更新資料內容。使用者可在專案編程中透過Profinet函式寫入/讀取「輸出資料表中」的「使用者定義區」，或是讀取「輸入資料表中」的「使用者定義區」。藉由使用者定義區，可使專案編程與外部通訊裝置，做為資料交換的寄存器。

通訊資料表 (機器人觀點)	資料區	TMflow Profinet函式權限	外部裝置權限
輸入資料表	系統定義區	讀取	寫入
	使用者定義區	讀取	寫入
輸出資料表	系統定義區	讀取	讀取
	使用者定義區	讀取/寫入	讀取

12.1 profinet_read_input()

讀取輸入表內容

語法 1

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

參數

int 起始位址
int 讀取位址個數

傳回值

byte[] 傳回資料 byte 陣列

說明

```
byte[] var_ba = profinet_read_input(148,16)  
// {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
```

語法 2

```
byte profinet_read_input(  
    int,  
)
```

參數

int 起始位址

傳回值

byte 傳回資料 byte

說明

```
byte var_b = profinet_read_input(148)  
// 0x30
```

語法 3

```
? profinet_read_input(  
    string,  
    int,  
    int  
)
```

參數

string 項目名稱
int 項目的起始偏移位址
int 讀取位址個數

傳回值

? 依據通訊資料表所定義的項目決定回傳資料型別。
* 支援型別包含 byte,byte[],int,int[],float,float[],string

語法 4

```
? profinet_read_input(  
    string,  
    int,  
)
```

參數

string 項目名稱
int 項目的起始偏移位址

傳回值

? 依據通訊資料表所定義的項目決定回傳資料型別。
* 支援型別包含 byte,byte[],int,int[],float,float[],string

說明

*與語法 3 相同，預設讀取到項目結尾。
*依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 寫入資料。

語法 5

```
? profinet_read_input(  
    string  
)
```

參數

string 項目名稱

傳回值

? 依據通訊資料表所定義的項目決定回傳資料型別和資料長度。

* 支援型別包含 byte,byte[],int,int[],float,float[],string

說明

* 與語法 3 相同，預設將項目的起始偏移位址填入 0；預設讀取到項目結尾。

* 依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 寫入資料。

```
byte var_b = profinet_read_input("StickStatus",0,1)
```

// 0x02

```
var_b = profinet_read_input("StickStatus",0)
```

// 0x02

```
var_b = profinet_read_input("StickStatus")
```

// 0x02

```
byte[] var_ba = profinet_read_input("CtrlBox_DO",0,2)
```

// {0x00,0x04}

```
var_ba = profinet_read_input("CtrlBox_DO")
```

// {0x00,0x04}

```
int[] var_ia = profinet_read_input("Register_Int",0,12)
```

// byte[] = {0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x02,0x00,0x00,0x00,0x03} (Little Endian) to int[]

// int[] = {0x00000001,0x00000002,0x00000003} (Little Endian)

// int[] = {1,2,3}

```
var_ia = profinet_read_input("Register_Int",12)
```

// byte[] = {0x00,0x00,0x00,0x04,0x00,0x00,0x00,0x05,0x00,0x00,0x00,0x06,0x00,0x00,0x00,0x07,

0x00,0x00,0x00,0x08,0x00,0x00,0x00,0x09,0x00,0x00,0x00,0xA,0x00,0x00,0x00,0xB,

0x00,0x00,0x00,0xC,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

```

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)    to int[]

// int[] = {0x00000004,0x00000005,0x00000006,0x00000007,0x00000008,0x00000009,
0x0000000A,0x0000000B,0x0000000C,0x00000000,0x00000000,0x00000000,
0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,
0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,
0x00000000,0x00000000} (Little Endian)

// int[] = {4,5,6,7,8,9,10,11,12,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}

var _ia = profinet_read_input("Register_Int")

// byte[] = {0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x00,0x02,0x00,0x00,0x00,0x03,0x00,0x00,0x00,0x04,
0x00,0x00,0x00,0x05,0x00,0x00,0x00,0x00,0x06,0x00,0x00,0x00,0x07,0x00,0x00,0x00,0x08,
0x00,0x00,0x00,0x09,0x00,0x00,0x00,0x00,0xA,0x00,0x00,0x00,0xB,0x00,0x00,0x00,0xC,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00} (Little Endian)      to int[]

// int[] = {0x00000001,0x00000002,0x00000003,0x00000004,0x00000005,0x00000006,
0x00000007,0x00000008,0x00000009,0x0000000A,0x0000000B,0x0000000C,
0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,
0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,
0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000} (Little Endian)

// int[] = {1,2,3,4,5,6,7,8,9,10,11,12,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}

float[] var_fa = profinet_read_input("Register_Float",4,12)

// byte[] = {0x3F,0x99,0x99,0x9A,0x3F,0xA6,0x66,0x66,0x40,0x06,0x66,0x66} (Big Endian)      to float[]
// float[] = {0x3F9999A,0x3FA66666,0x40066666} (Big Endian)
// float[] = {1.2,1.3,2.1}

```


12.2 profinet_read_input_int()

讀取輸入表內容並轉換資料為 32 位元整數型別

語法 1

```
int[] profinet_read_input_int(  
    int,  
    int,  
    int  
)
```

參數

int	起始位址
int	讀取位址個數
int	讀出後的資料是依照 Little-Endian (DCBA) 或 Big-Endian (ABCD) 轉成 int 陣列。 0 Little-Endian 1 Big-Endian 2 依照設定檔

傳回值

int[] 傳回資料 int 陣列

語法 2

```
int[] profinet_read_input_int(  
    int,  
    int  
)
```

參數

int	起始位址
int	讀取位址個數

說明

與語法 1 相同，預設將最後一個參數 int 填入 2。

* 讀出的資料是依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 轉成 int 陣列。

```

int[] var_ia = profinet_read_input_int(164,12,0)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)      to int[]
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)
    // int[] = {32767,99999,-32768}

var_ia = profinet_read_input_int(164,11,0)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF} (Little Endian)      to int[]
    // int[] = {0x00007FFF,0x0001869F,0x00FF8000} (Little Endian)
    // int[] = {32767,99999,16744448}

var_ia = profinet_read_input_int(164,10,0)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80} (Little Endian)      to int[]
    // int[] = {0x00007FFF,0x0001869F,0x00008000} (Little Endian)
    // int[] = {32767,99999,32768}

var_ia = profinet_read_input_int(164,12,1)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Big Endian) to int[]
    // int[] = {0xFF7F0000,0x9F860100,0x0080FFFF} (Big Endian)
    // int[] = {-8454144,-1618607872,8454143}

var_ia = profinet_read_input_int(164,12,2)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)      to int[]
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)
    // int[] = {32767,99999,-32768}

var_ia = profinet_read_input_int(164,12)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)      to int[]
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)
    // int[] = {32767,99999,-32768}

```

語法 3

```
int profinet_read_input_int(  
    int  
)
```

參數

int 起始位址

* 讀出的資料是依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 轉成 int。

傳回值

int 傳回資料 int

說明

```
int var_i = profinet_read_input_int(164)  
  
    // byte[] = {0xE4,0x07,0x00,0x00} (Little Endian)      to int  
  
    // int = 0x000007E4 (Little Endian)  
  
    // int = 2020  
  
var_i = profinet_read_input_int(164)  
  
    // byte[] = {0x00,0x00,0x07,0xE4} (Big Endian)      to int  
  
    // int = 0x000007E4 (Big Endian)  
  
    // int = 2020
```

12.3 profinet_read_input_float()

讀取輸入表內容並轉換資料為 32 位元浮點數型別

語法 1

```
float[] profinet_read_input_float(  
    int,  
    int,  
    int  
)
```

參數

int	起始位址
int	讀取位址個數
int	讀出後的資料是依照 Little-Endian (DCBA) 或 Big-Endian (ABCD) 轉成 float 陣列。 0 Little-Endian 1 Big-Endian 2 依照設定檔

傳回值

float[] 傳回資料 float 陣列

語法 2

```
float[] profinet_read_input_float(  
    int,  
    int  
)
```

參數

int	起始位址
int	讀取位址個數

說明

與語法 1 相同，預設將最後一個參數 int 填入 2。

* 讀出的資料是依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 轉成 float 陣列。

```

float[] var_fa = profinet_read_input_float(284,12,0)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)    to float[]
// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)
// float[] = {1.0,2.0,3.0}

var_fa = profinet_read_input_float(284,11,0)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40} (Little Endian)    to float[]
// float[] = {0x3F800000,0x40000000,0x00400000} (Little Endian)
// float[] = {1.0,2.0,5.877472E-39}

var_fa = profinet_read_input_float(284,10,0)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00} (Little Endian)    to float[]
// float[] = {0x3F800000,0x40000000,0x00000000} (Little Endian)
// float[] = {1.0,2.0,0.0}

var_fa = profinet_read_input_float(284,12,1)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)    to float[]
// float[] = {0x0000803F,0x00000040,0x00004040} (Big Endian)
// float[] = {4.600603E-41,8.96831E-44,2.304856E-41}

var_fa = profinet_read_input_float(284,12,2)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)    to float[]
// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)
// float[] = {1.0,2.0,3.0}

var_fa = profinet_read_input_float(284,12)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)    to float[]
// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)
// float[] = {1.0,2.0,3.0}

```

語法 3

```
float profinet_read_input_float(  
    int  
)
```

參數

int 起始位址

* 讀出的資料是依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 轉成 float。

傳回值

float 傳回資料 float

說明

```
float var_f = profinet_read_input_float(284)  
  
// byte[] = {0x00,0x00,0x80,0x3F} (Little Endian) to float  
  
// float = 0x3F800000 (Little Endian)  
  
// float = 1.0  
  
var_f = profinet_read_input_float(284)  
  
// byte[] = {0x3F,0x80,0x00,0x00} (Big Endian) to float  
  
// float = 0x3F800000 (Big Endian)  
  
// float = 1.0
```

12.4 profinet_read_input_string()

讀取輸入表內容並轉換資料陣列為 UTF8 編碼的字串文本

語法 1

```
string profinet_read_input_string(  
    int,  
    int  
)
```

參數

int 起始位址
int 讀取位址個數

傳回值

string 傳回資料 string UTF8 字串格式 (遇到 0x00 結束)

說明

```
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[] = {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  
// 0x54,0x4D,0x35,0x2D,0x37,0x30,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}  
  
// 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,0x31,0x32,0x33,0x34,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}  
  
// 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◆"
```

12.5 profinet_read_input_bit()

讀取輸入表內容，取得 byte 資料中 bit 的數值

語法 1

```
byte profinet_read_input_bit(  
    int,  
    int  
)
```

參數

int 起始位址
int byte 資料中的第 n 個 bit 數值

傳回值

byte 傳回資料 byte 型別。
bit 數值 == 1，回傳 1
bit 數值 == 0，回傳 0
*bit 資料將會以 byte 型式回傳。

說明

```
byte var_b = profinet_read_input_bit(148,0)  
    // 0x30    get bit: "0"  
    // 0  
  
var_b = profinet_read_input_bit(148,5)  
    // 0x30    get bit: "5"  
    // 1
```

語法 2

```
byte profinet_read_input_bit(  
    string,  
    int  
)
```

參數

string 項目名稱
int 第 n 個 bit 數值

傳回值

`byte` 傳回資料 byte 型別。
bit 數值 == 1，回傳 1
bit 數值 == 0，回傳 0
`*bit` 資料將會以 `byte` 型式回傳。

說明

```
byte var_b = profinet_read_input_bit("Register_Bit",0)  
// byte[] = {1,0,0,1,1,1,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total  
// 1 get bit: "0"  
  
var_b = profinet_read_input_bit("Register_Bit",17)  
// byte[] = {1,0,0,1,1,1,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total  
// 0 get bit: "17"
```

語法 3

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

參數

`int` 起始位址
`int` 起始 bit
`int` 讀取 bit 個數

傳回值

`byte[]` 傳回資料 `byte[]` 型別。
bit 數值 == 1，回傳 1
bit 數值 == 0，回傳 0
`*bit` 資料將會以 `byte` 型式回傳，即 `bit[0]` 讀為 `byte[0]`，`bit[1]` 讀為 `byte[1]`。

說明

```
byte[] var_ba = profinet_read_input_bit(148,0,20)  
// byte[] = {1,0,0,1,1,1,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total  
// byte[] = {1,0,0,1,1,1,0,0,0,0,1,1,1,0,1,0,0,1,1}  
  
var_ba = profinet_read_input_bit(148,12,8)  
// byte[] = {1,0,0,1,1,1,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total  
// byte[] = {1,1,0,1,0,0,1,1}
```

語法 4

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

參數

string 項目名稱
int 起始 bit
int 讀取 bit 個數

傳回值

byte[] 傳回資料 byte[] 型別。
bit 數值 == 1，回傳 1
bit 數值 == 0，回傳 0
*bit 資料將會以 byte 型式回傳，即 bit[0] 讀為 byte[0]， bit[1] 讀為 byte[1]。

說明

```
byte[] var_ba = profinet_read_input_bit("Register_Bit",0,20)  
// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total  
// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1}  
  
var_ba = profinet_read_input_bit("Register_Bit",12,8)  
// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total  
// byte[] = {1,1,0,1,0,0,1,1}
```

12.6 profinet_read_output()

讀取輸出表內容

語法 1

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

參數

int 起始位址
int 讀取位址個數

傳回值

byte[] 傳回資料 byte 陣列

說明

```
byte[] var_ba = profinet_read_output(540,16)  
// {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
```

語法 2

```
byte profinet_read_output(  
    int  
)
```

參數

int 起始位址

傳回值

byte 傳回資料 byte

說明

與語法 1 相同，預設將最後一個參數 int 填入 1。

```
byte var_b = profinet_read_output(540)  
// 0x30
```

語法 3

```
? profinet_read_output(  
    string,  
    int,  
    int  
)
```

參數

string 項目名稱
int 項目的起始偏移位址
int 讀取位址個數

傳回值

? 依據通訊資料表所定義的項目決定回傳資料型別。
* 支援型別包含 byte,byte[],int,int[],float,float[],string

語法 4

```
? profinet_read_output(  
    string,  
    int,  
)
```

參數

string 項目名稱
int 項目的起始偏移位址

傳回值

? 依據通訊資料表所定義的項目決定回傳資料型別。
* 支援型別包含 byte,byte[],int,int[],float,float[],string

說明

*與語法 3 相同，預設讀取到項目結尾。
*依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 寫入資料。

語法 5

```
? profinet_read_output(  
    string  
)
```

參數

string 項目名稱

傳回值

? 依據通訊資料表所定義的項目決定回傳資料型別和資料長度。

* 支援型別包含 byte,byte[],int,int[],float,float[],string

說明

* 與語法 3 相同，預設將項目的起始偏移位址填入 0；預設讀取到項目結尾。

* 依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 寫入資料。

```
byte var_b = profinet_read_output("ManualAuto",0,1)
```

// 0x02

```
var_b = profinet_read_output("ManualAuto",0)
```

// 0x02

```
var_b = profinet_read_output("ManualAuto")
```

// 0x02

```
byte[] var_ba = profinet_read_output("Error_Code",0,4)
```

// {0x00,0x04,0x80,0x0C}

```
var_ba = profinet_read_output("Error_Code",0,2)
```

// {0x00,0x04}

```
var_ba = profinet_read_output("Error_Code")
```

// {0x00,0x04,0x80,0x0C}

```
int var_i = profinet_read_output("Current_Time_YY")
```

// byte[] = {0x00,0x00,0x07,0xE4} (Little Endian) to int

// int = 0x000007E4 (Little Endian)

// int = 2020


```

// int[] = {1,2,3,4,5,6,7,8,9,10,11,12,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}

float var_f = profinet_read_output("Current_TCP_Mass")
    // byte[] = {0x40,0x40,0x00,0x00} (Big Endian)      to float
    // float = 0x40400000 (Big Endian)
    // float = 3.0

var_fa = profinet_read_output("Current_TCP_Value",4,12)
    // byte[] = {0x3F,0x99,0x99,0x9A,0x3F,0xA6,0x66,0x66,0x40,0x06,0x66,0x66} (Big Endian)      to float[]
    // float[] = {0x3F99999A,0x3FA66666,0x40066666} (Big Endian)
    // float[] = {1.2,1.3,2.1}

var_fa = profinet_read_output("Current_TCP_Value",12)
    // byte[] = { 0x40,0x06,0x66,0x66,0x40,0x0C,0xCC,0xCD,0x40,0x13,0x33,0x33} (Big Endian)      to float[]
    // float[] = {0x40066666,0x400CCCCD,0x40133333} (Big Endian)
    // float[] = {2.1,2.2,2.3}

var_fa = profinet_read_output("Current_TCP_Value")
    // byte[] = {0x3F,0x8C,0xCC,0xCD,0x3F,0x99,0x99,0x9A,0x3F,0xA6,0x66,0x66,0x40,0x06,0x66,0x66,
    //           0x40,0x0C,0xCC,0xCD,0x40,0x13,0x33,0x33} (Big Endian)      to float[]
    // float[] = {0x3F8CCCCD,0x3F99999A,0x3FA66666,0x40066666,0x400CCCCD,0x40133333} (Big Endian)
    // float[] = { 1.1,1.2,1.3,2.1,2.2,2.3}

string var_s = profinet_read_output ("RobotModel",0,16)
    // byte[] = {0x54,0x4D,0x35,0x2D,0x37,0x30,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
    // 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"

```

12.7 profinet_read_output_int()

讀取輸出表內容並轉換資料為 32 位元整數型別

語法 1

```
int[] profinet_read_output_int(  
    int,  
    int,  
    int  
)
```

參數

int	起始位址
int	讀取位址個數
int	讀出後的資料是依照 Little-Endian (DCBA) 或 Big-Endian (ABCD) 轉成 int 陣列。
0	Little-Endian
1	Big-Endian
2	依照設定檔

傳回值

int[] 傳回資料 int 陣列

語法 2

```
int[] profinet_read_output_int(  
    int,  
    int  
)
```

參數

int	起始位址
int	讀取位址個數

說明

與語法 1 相同，預設將最後一個參數 int 填入 2。

* 讀出的資料是依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 轉成 int 陣列。

```

int[] var_ia = profinet_read_output_int(556,12,0)
// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]
// int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)
// int[] = {32767,99999,-32768}

var_ia = profinet_read_output_int(556,11,0)
// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF} (Little Endian)    to int[]
// int[] = {0x00007FFF,0x0001869F,0x00FF8000} (Little Endian)
// int[] = {32767,99999,16744448}

var_ia = profinet_read_output_int(556,10,0)
// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80} (Little Endian)    to int[]
// int[] = {0x00007FFF,0x0001869F,0x00008000} (Little Endian)
// int[] = {32767,99999,32768}

var_ia = profinet_read_output_int(556,12,1)
// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]
// int[] = {0xFF7F0000,0x9F860100,0x0080FFFF} (Big Endian)
// int[] = {-8454144,-1618607872,8454143}

var_ia = profinet_read_output_int(556,12,2)
// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]
// int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)
// int[] = {32767,99999,-32768}

var_ia = profinet_read_output_int(556,12)
// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]
// int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)
// int[] = {32767,99999,-32768}

```

語法 3

```
int profinet_read_output_int(  
    int  
)
```

參數

int 起始位址

* 讀出的資料是依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 轉成 int。

傳回值

int 傳回資料 int

說明

```
int var_i = profinet_read_output_int(556)  
  
    // byte[] = {0xE4,0x07,0x00,0x00} (Little Endian)      to int  
  
    // int = 0x000007E4 (Little Endian)  
  
    // int = 2020  
  
var_i = profinet_read_output_int(556)  
  
    // byte[] = {0x00,0x00,0x07,0xE4} (Big Endian)      to int  
  
    // int = 0x000007E4 (Big Endian)  
  
    // int = 2020
```

12.8 profinet_read_output_float()

讀取輸出表內容並轉換資料為 32 位元浮點數型別

語法 1

```
float[] profinet_read_output_float(  
    int,  
    int,  
    int  
)
```

參數

int	起始位址
int	讀取位址個數
int	讀出後的資料是依照 Little-Endian (DCBA) 或 Big-Endian (ABCD) 轉成 float 陣列。 0 Little-Endian 1 Big-Endian 2 依照設定檔

傳回值

float[] 傳回資料 float 陣列

語法 2

```
float[] profinet_read_output_float(  
    int,  
    int  
)
```

參數

int	起始位址
int	讀取位址個數

說明

與語法 1 相同，預設將最後一個參數 int 填入 2。

* 讀出的資料是依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 轉成 float 陣列。

```

float[] var_fa = profinet_read_output_float(676,12,0)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)    to float[]
// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)
// float[] = {1.0,2.0,3.0}

var_fa = profinet_read_output_float(676,11,0)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40} (Little Endian)    to float[]
// float[] = {0x3F800000,0x40000000,0x00400000} (Little Endian)
// float[] = {1.0,2.0,5.877472E-39}

var_fa = profinet_read_output_float(676,10,0)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00} (Little Endian)    to float[]
// float[] = {0x3F800000,0x40000000,0x00000000} (Little Endian)
// float[] = {1.0,2.0,0.0}

var_fa = profinet_read_output_float(676,12,1)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)    to float[]
// float[] = {0x0000803F,0x00000040,0x00004040} (Big Endian)
// float[] = {4.600603E-41,8.96831E-44,2.304856E-41}

var_fa = profinet_read_output_float(676,12,2)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)    to float[]
// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)
// float[] = {1.0,2.0,3.0}

var_fa = profinet_read_output_float(676,12)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)    to float[]
// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)
// float[] = {1.0,2.0,3.0}

```

語法 3

```
float profinet_read_output_float(  
    int  
)
```

參數

int 起始位址

* 讀出的資料是依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 轉成 float。

傳回值

float 傳回資料 float

說明

```
float var_f = profinet_read_output_float(676)  
  
// byte[] = {0x00,0x00,0x80,0x3F} (Little Endian) to float  
  
// float = 0x3F800000 (Little Endian)  
  
// float = 1.0  
  
var_f = profinet_read_output_float(676)  
  
// byte[] = {0x3F,0x80,0x00,0x00} (Big Endian) to float  
  
// float = 0x3F800000 (Big Endian)  
  
// float = 1.0
```

12.9 profinet_read_output_string()

讀取輸出表內容並轉換資料陣列為 UTF8 編碼的字串文本

語法 1

```
string profinet_read_output_string(  
    int,  
    int  
)
```

參數

int 起始位址
int 讀取位址個數

傳回值

string 傳回資料 string UTF8 字串格式 (遇到 0x00 結束)

說明

```
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[] = {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  
//             0x54,0x4D,0x35,0x2D,0x37,0x30,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}  
  
// 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,0x31,0x32,0x33,0x34,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}  
  
// 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◆"
```

12.10 profinet_read_output_bit()

讀取輸出表內容，取得 byte 資料中 bit 的數值

語法 1

```
byte profinet_read_output_bit(  
    int,  
    int  
)
```

參數

int 起始位址
int byte 資料中的第 n 個 bit 數值

傳回值

byte 傳回資料 byte 型別。
bit 數值 == 1，回傳 1
bit 數值 == 0，回傳 0
*bit 資料將會以 byte 型式回傳。

說明

```
byte var_b = profinet_read_output_bit(540,0)  
    // 0x30    get bit: "0"  
    // 0  
  
var_b = profinet_read_output_bit(540,5)  
    // 0x30    get bit: "5"  
    // 1
```

語法 2

```
byte profinet_read_output_bit(  
    string,  
    int  
)
```

參數

string 項目名稱
int 第 n 個 bit 數值

傳回值

byte 傳回資料 byte 型別。
bit 數值 == 1，回傳 1

bit 數值 == 0，回傳 0
*bit 資料將會以 byte 型式回傳。

說明

```
byte[] var_data = {57,184,12}  
profinet_write_output(540,var_data,3)  
// {00111001,10111000,00001100} (binary)  
byte var_b = profinet_read_output_bit("Register_Bit",0)  
// 1  
var_b = profinet_read_output_bit("Register_Bit",17)  
// 0
```

語法 3

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

參數

int	起始位址
int	起始 bit
int	讀取 bit 個數

傳回值

byte[] 傳回資料 byte[] 型別。
bit 數值 == 1，回傳 1
bit 數值 == 0，回傳 0
*bit 資料將會以 byte 型式回傳，即 bit[0] 讀為 byte[0]， bit[1] 讀為 byte[1]。

說明

```
byte[] var_data = {57,184,12}  
profinet_write_output(540,var_data,3)  
// {00111001,10111000,00001100} (binary)  
byte[] var_ba = profinet_read_output_bit(540,0,20)  
// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1}  
var_ba = profinet_read_output_bit(540,12,8)  
// byte[] = {1,1,0,1,0,0,1,1}
```

語法 4

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

參數

string 項目名稱
int 起始 bit
int 讀取 bit 個數

傳回值

byte[] 傳回資料 byte[] 型別。
bit 數值 == 1，回傳 1
bit 數值 == 0，回傳 0
*bit 資料將會以 byte 型式回傳，即 bit[0] 讀為 byte[0]， bit[1] 讀為 byte[1]。

說明

```
byte[] var_data = {57,184,12}  
  
profinet_write_output(540,var_data,3)  
  
// {00111001,10111000,00001100} (binary)  
  
byte[] var_ba = profinet_read_output_bit("Register_Bit",0,20)  
  
// byte[] = {1,0,0,1,1,0,0,0,0,1,1,1,0,1,0,0,1,1}  
  
var_ba = profinet_read_output_bit("Register_Bit",12,8)  
  
// byte[] = {1,1,0,1,0,0,1,1}
```

12.11 profinet_write_output()

寫入輸出表內容

語法 1

```
bool profinet_write_output(  
    int,  
    ?,  
    int  
)
```

參數

int	起始位址
?	寫入資料
	* 支援型別包含 byte,byte[],int,int[],float,float[],string
int	最大寫入位址個數
>0	合法位址長度，依位址個數寫入
<=0	非法位址長度，依寫入資料的完整長度寫入

傳回值

bool	True	寫入成功
	False	寫入失敗

1. 如果 ? 寫入資料值為空字串或空陣列
2. 無法正確發送或接收通訊

說明

* 依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 寫入資料。

語法 2

```
bool profinet_write_output(  
    int,  
    ?  
)
```

參數

int	起始位址
?	寫入資料
	* 支援型別包含 byte,byte[],int,int[],float,float[],string

傳回值

bool	True	寫入成功
	False	寫入失敗

1. 如果 ? 寫入資料值為空字串或空陣列
2. 無法正確發送或接收通訊

說明

與語法 1 相同，預設依寫入資料的完整長度寫入。

*依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 寫入資料。

語法 3

```
bool profinet_write_output(  
    int,  
    ?,  
    int,  
    int  
)
```

參數

int	起始位址
?	寫入資料
*支援型別包含 byte,byte[],int,int[],float,float[],string	
int	寫入資料的起始位址
int	寫入資料的位址個數

傳回值

bool	True	寫入成功
	False	寫入失敗
		1. 如果 ? 寫入資料值為空字串或空陣列
		2. 無法正確發送或接收通訊

說明

*依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 寫入資料。

```
byte var_data = 255  
profinet_write_output(540,var_data,1)  
byte var_b = profinet_read_output(540)  
// 0xFF
```

```
byte[] var_data = {1,127,255}  
profinet_write_output(540,var_data,3)  
byte[] var_ba = profinet_read_output(540,3)  
// {0x01,0x7F,0xFF}  
profinet_write_output(540,var_data,2)  
var_ba = profinet_read_output(540,3)
```

```

// {0x01,0x7F,0x00}

profinet_write_output(540,var_data,-1)

var_ba = profinet_read_output(540,3)

// {0x00,0x7F,0xFF}

int var_data = 32767

profinet_write_output(556,var_data,4)

int var_i = profinet_read_output_int(556)

// byte[] = {0xFF,0x7F,0x00,0x00} (Little Endian)      to int

// int = 0x00007FFF (Little Endian)

// int = 32767

profinet_write_output(556,var_data,1)

var_i = profinet_read_output_int(556)

// byte[] = {0xFF,0x00,0x00,0x00} (Little Endian)      to int

// int = 0x000000FF (Little Endian)

// int = 255

int[] var_data = {32767,99999,-32768}

profinet_write_output(556,var_data,12)

int[] var_ia = profinet_read_output_int(556,12)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)      to int[]

// int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)

// int[] = {32767,99999,-32768}

profinet_write_output(556,var_data,3)

var_ia = profinet_read_output_int(556,12)

// byte[] = {0xFF,0x7F,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)      to int[]

// int[] = {0x00007FFF,0x00000000,0x00000000} (Little Endian)

// int[] = {32767,0,0}

profinet_write_output(556,var_data,11)

var_ia = profinet_read_output_int(556,12)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0x00} (Little Endian)      to int[]

// int[] = {0x00007FFF,0x0001869F,0x00FF8000} (Little Endian)

```

```

// int[] = {32767,99999,16744448}

profinet_write_output(556,var_data,4,4)

var_ia = profinet_read_output_int(556,12)

// byte[] = {0x9F,0x86,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)    to int[]

// int[] = {0x0001869F,0x00000000,0x00000000} (Little Endian)

// int[] = {99999,0,0}

float var_data = -10.0

profinet_write_output(676,var_data,4)

float var_f = profinet_read_output_float(676)

// byte[] = {0x00,0x00,0x20,0xC1} (Little Endian)      to float

// float = 0xC1200000 (Little Endian)

// float = -10.0

profinet_write_output(676,var_data,1)

var_f = profinet_read_output_float(676)

// byte[] = {0x00,0x00,0x00,0x00} (Little Endian)      to float

// float = 0x00000000 (Little Endian)

// float = 0

float[] var_data = {-10.0,3.3,123.45}

profinet_write_output(676,var_data,12)

float[] var_fa = profinet_read_output_float(676,12)

// byte[] = {0x00,0x00,0x20,0xC1,0x33,0x33,0x53,0x40,0x66,0xE6,0xF6,0x42} (Little Endian)    to float[]

// float[] = {0xC1200000,0x40533333,0x42F6E666} (Little Endian)

// float[] = {-10,3.3,123.45}

profinet_write_output(676,var_data,3)

var_fa = profinet_read_output_float(676,12)

// byte[] = {0x00,0x00,0x20,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)    to float[]

// float[] = {0x00200000,0x00000000,0x00000000} (Little Endian)

// float[] = {2.938736E-39,0,0}

profinet_write_output(676,var_data,11)

var_fa = profinet_read_output_float(676,12)

```

```

// byte[] = {0x00,0x00,0x20,0xC1,0x33,0x33,0x53,0x40,0x66,0xE6,0xF6,0x00} (Little Endian)    to float[]
// float[] = {0xC1200000,0x40533333,0x00F6E666} (Little Endian)
// float[] = {-10.3.3,2.267418E-38}

profinet_write_output(676,var_data,4,4)

var_fa = profinet_read_output_float(676,12)
// byte[] = {0x33,0x33,0x53,0x40,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)    to float[]
// float[] = {0x40533333,0x00000000,0x00000000} (Little Endian)
// float[] = {3.3,0,0}

string var_data = "abcd 達明機器人 1234"

profinet_write_output(540,var_data,32)
string var_s = profinet_read_output_string(540,32)
// byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,
0xE4,0xBA,0xBA,0x31,0x32,0x33,0x34,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
// 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}
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
// 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}
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
// 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"

```

語法 4

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

參數

string 項目名稱
int 項目的起始偏移位址
? 寫入資料
* 支援型別包含 byte,byte[],int,int[],float,float[],string
int 寫入資料的起始位址
int 寫入資料的位址個數

傳回值

bool True 寫入成功
False 寫入失敗 1. 如果 ? 寫入資料值為空字串或空陣列
2. 無法正確發送或接收通訊

說明

* 依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 寫入資料。

語法 5

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

參數

string 項目名稱
int 項目的起始偏移位址
? 寫入資料
* 支援型別包含 byte,byte[],int,int[],float,float[],string
int 寫入資料的起始位址

傳回值

bool	True	寫入成功
	False	寫入失敗
		1. 如果 ? 寫入資料值為空字串或空陣列
		2. 無法正確發送或接收通訊

說明

*與語法 4 相同，預設將寫入資料剩餘部分完整寫入

*依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 寫入資料。

語法 6

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

參數

string	項目名稱
int	項目的起始偏移位址
?	寫入資料
*支援型別包含 byte,byte[],int,int[],float,float[],string	

傳回值

bool	True	寫入成功
	False	寫入失敗
		1. 如果 ? 寫入資料值為空字串或空陣列
		2. 無法正確發送或接收通訊

說明

*與語法 4 相同，預設將寫入資料的起始位址填入 0；預設依寫入資料的完整長度寫入。

*依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 寫入資料。

語法 7

```
bool profinet_write_output(  
    string,  
    ?  
)
```

參數

string	項目名稱
?	寫入資料
*支援型別包含 byte,byte[],int,int[],float,float[],string	

傳回值

bool	True	寫入成功
	False	寫入失敗
	1. 如果 ? 寫入資料值為空字串或空陣列	
	2. 無法正確發送或接收通訊	

說明

*與語法 4 相同，預設將項目的起始偏移位址填入 0；寫入資料的起始位址填入 0；依寫入資料的完整長度寫入。

*依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 寫入資料。

```
int[] var_data = {32767,99999,-32768}

profinet_write_output("Register_Int",0,var_data,0,12)

int[] var_ia = profinet_read_output_int(556,12)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]

// int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)

// int[] = {32767,99999,-32768}

profinet_write_output("Register_Int",4,var_data,4,4)

var_ia = profinet_read_output_int(556,12)

// byte[] = {0x00,0x00,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x00,0x00,0x00} (Little Endian)    to int[]

// int[] = {0x00000000,0x0001869F,0x00000000} (Little Endian)

// int[] = {0,99999,0}

profinet_write_output("Register_Int",4,var_data)

var_ia = profinet_read_output_int(556,20)

// byte[] = {0x00,0x00,0x00,0x00,0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF,
//            0x00,0x00,0x00,0x00} (Little Endian)    to int[]

// int[] = {0x00000000,0x00007FFF,0x0001869F,0xFFFF8000,0x00000000} (Little Endian)

// int[] = {0,32767,99999,-32768,0}

float[] var_data = {-10.0,3.3,123.45}

profinet_write_output("Register_Float",0,var_data,0,12)

float[] var_fa = profinet_read_output_float(676,12)

// byte[] = {0x00,0x00,0x20,0xC1,0x33,0x33,0x53,0x40,0x66,0xE6,0xF6,0x42} (Little Endian)    to float[]

// float[] = {0xC1200000,0x40533333,0x42F6E666} (Little Endian)

// float[] = {-10,3.3,123.45}
```

```
profinet_write_output("Register_Float",4,var_data,4,8)
var_fa = profinet_read_output_float(676,12)
// byte[] = {0x00,0x00,0x00,0x00,0x33,0x33,0x53,0x40,0x66,0xE6,0xF6,0x42} (Little Endian)    to float[]
// float[] = {0x00000000,0x40533333,0x42F6E666} (Little Endian)
// float[] = {0,3.3,123.45}

profinet_write_output("Register_Float",8,var_data)
var_fa = profinet_read_output_float(676,20)
// byte[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x20,0xC1,0x33,0x33,0x53,0x40,
//            0x66,0xE6,0xF6,0x42} (Little Endian)    to float[]
// float[] = {0x00000000,0x00000000,0xC1200000,0x40533333,0x42F6E666} (Little Endian)
// float[] = {0,0,-10,3.3,123.45}
```

12.12 profinet_write_output_bit()

寫入輸出表內容，設定 byte 資料中 bit 的數值

語法 1

```
bool profinet_write_output_bit(  
    int,  
    int,  
    int  
)
```

參數

int 起始位址
int byte 資料中的第 n 個 bit
int 寫入資料
*bit 資料將會以 int 型式寫入。

傳回值

bool True 寫入成功
False 寫入失敗 1. 無法正確發送或接收通訊

說明

```
byte var_data = 240  
  
profinet_write_output(540,var_data)  
  
byte var_b = profinet_read_output(540)  
  
    // 0xF0  
  
profinet_write_output_bit(540,1,1)  
  
var_b = profinet_read_output_bit(540,1)  
  
    // 0xF2    get bit: "1"  
  
    // 1  
  
profinet_write_output_bit(540,7,0)  
  
var_b = profinet_read_output_bit(540,7)  
  
    // 0x72    get bit: "7"  
  
    // 0
```

語法 2

```
bool profient_write_output_bit(  
    string,  
    int,  
    int  
)
```

參數

int 項目名稱
int 第 n 個 bit
int 寫入資料
*bit 資料將會以 int 型式寫入。

傳回值

bool True 寫入成功
False 寫入失敗 1. 無法正確發送或接收通訊

說明

```
byte var_data = 240  
  
profinet_write_output(540,var_data)  
  
byte var_b = profinet_read_output(540)  
    // 0xF0  
  
profinet_write_output_bit("Register_Bit",1,1)  
  
var_b = profinet_read_output_bit(540,1)  
    // 0xF2    get bit: "1"  
    // 1  
  
profinet_write_output_bit("Register_Bit",7,0)  
  
var_b = profinet_read_output_bit(540,7)  
    // 0x72    get bit: "7"  
    // 0
```

語法 3

```
bool profinet_write_output_bit(  
    int,  
    int,  
    byte[],  
    int,  
    int  
)
```

參數

int 起始位址
int 起始 bit
byte[] 寫入資料
*bit 資料將會以 byte 型式寫入，即 byte[0] 填入 bit[0]， byte[1] 填入 bit[1]
int 寫入資料的起始 bit
int 寫入資料的 bit 個數

傳回值

bool True 寫入成功
False 寫入失敗 1. 無法正確發送或接收通訊

說明

byte 數值 ≥ 1 ，則 bit 數值 = 1，
byte 數值 = 0，則 bit 數值 = 0

語法 4

```
bool profinet_write_output_bit(  
    int,  
    int,  
    byte[],  
    int  
)
```

參數

int 起始位址
int 起始 bit
byte[] 寫入資料
*bit 資料將會以 byte 型式寫入，即 byte[0] 填入 bit[0]， byte[1] 填入 bit[1]
int 寫入資料的起始 bit

傳回值

bool True 寫入成功
False 寫入失敗 1. 無法正確發送或接收通訊

說明

*與語法 3 相同，將寫入資料剩餘部分完整寫入。

byte 數值 ≥ 1 ，則 bit 數值 = 1，
byte 數值 = 0，則 bit 數值 = 0

語法 5

```
bool profinet_write_output_bit(  
    int,  
    int,  
    byte[]  
)
```

參數

int 起始位址

int 起始 bit

byte[] 寫入資料

*bit 資料將會以 byte 型式寫入，即 byte[0] 填入 bit[0]，byte[1] 填入 bit[1]

傳回值

bool	True	寫入成功
	False	寫入失敗 1. 無法正確發送或接收通訊

說明

*與語法 3 相同，預設將寫入資料的起始 bit 填入 0；預設依寫入資料的完整長度寫入。

byte 數值 ≥ 1 ，則 bit 數值 = 1，
byte 數值 = 0，則 bit 數值 = 0

```
byte[] var_data = {1,0,0,1,1,0,0,0,0,1,1,1,0,1,0,0,1,1}
```

```
profinet_write_output_bit(540,0,var_data,0,20)
```

```
byte[] var_ba = profinet_read_output (540,0,3)
```

```
// byte[] = {0x39,0xB8,0x0C}
```

```
var_ba = profinet_read_output_bit(540,0,20)
```

```
// byte[] = {1,0,0,1,1,0,0,0,0,1,1,1,0,1,0,0,1,1}
```

```
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}
```

語法 6

```
bool profinet_write_output_bit(  
    string,  
    int,  
    byte[],  
    int,  
    int  
)
```

參數

string 項目名稱
int 起始 bit
byte[] 寫入資料
*bit 資料將會以 byte 型式寫入，即 byte[0] 填入 bit[0]， byte[1] 填入 bit[1]
int 寫入資料的起始 bit
int 寫入資料的 bit 個數

傳回值

bool True 寫入成功
False 寫入失敗 1. 無法正確發送或接收通訊

說明

byte 數值 ≥ 1 ，則 bit 數值 = 1，
byte 數值 = 0，則 bit 數值 = 0

語法 7

```
bool profinet_write_output_bit(  
    string,  
    int,  
    byte[],  
    int  
)
```

參數

string 項目名稱
int 起始 bit
byte[] 寫入資料
*bit 資料將會以 byte 型式寫入，即 byte[0] 填入 bit[0]， byte[1] 填入 bit[1]
int 寫入資料的起始 bit

傳回值

bool True 寫入成功
False 寫入失敗 1. 無法正確發送或接收通訊

說明

*與語法 6 相同，將寫入資料剩餘部分完整寫入。

byte 數值 ≥ 1 ，則 bit 數值 = 1，
byte 數值 = 0，則 bit 數值 = 0

語法 8

```
bool profinet_write_output_bit(  
    string,  
    int,  
    byte[]  
)
```

參數

string 起始位址

int 起始 bit

byte[] 寫入資料

*bit 資料將會以 byte 型式寫入，即 byte[0] 填入 bit[0]，byte[1] 填入 bit[1]

傳回值

bool	True	寫入成功
	False	寫入失敗 1. 無法正確發送或接收通訊

說明

*與語法 6 相同，預設將寫入資料的起始 bit 填入 0；預設依寫入資料的完整長度寫入。

byte 數值 ≥ 1 ，則 bit 數值 = 1，
byte 數值 = 0，則 bit 數值 = 0

```
byte[] var_data = {1,0,0,1,1,0,0,0,0,1,1,1,0,1,0,0,1,1}  
  
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[] = {1,0,0,1,1,0,0,0,0,1,1,1,0,1,0,0,1,1}  
  
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}
```

13. EtherNet/IP 函式

機器人與外部控制器以EtherNet/IP通信協定進行通訊。在EtherNet/IP通信協定的機制下，機器人做為EtherNet/IP IO裝置，提供外部設備讀寫機器人資料；同時TMflow可透過EtherNet/IP函式監控機器人「自外部設備接收的資料表」與「傳送給外部設備的資料表」，亦可改變「傳送給外部設備的資料表」中的使用者自定義區域。

通訊資料表：

資料表類型分為輸入/輸出兩式，輸入資料表是自外部設備發送至機器人，輸出資料表是自機器人發送至外部設備；此二資料表內容皆包含「系統定義區」和「使用者定義區」兩大資料區。

3. 系統定義區：由機器人系統所定義的項目與設定，並由機器人系統或外部裝置更新資料內容。定義的項目與機器人狀態相關，如手臂座標、專案狀態、電控箱狀態等等；或是與 IO 輸入/輸出相關狀態，如數位輸入/數位輸出、類比輸入/類比輸出等等。使用者可透過EtherNet/IP函式讀取「輸入資料表」與「輸出資料表」中的「系統定義區」。
4. 使用者定義區：由使用者所定義的項目與設定，並由使用者或外部裝置更新資料內容。使用者可在專案編程中透過EtherNet/IP函式寫入/讀取「輸出資料表中」的「使用者定義區」，或是讀取「輸入資料表中」的「使用者定義區」。藉由使用者定義區，可使專案編程與外部通訊裝置，做為資料交換的寄存器。

通訊資料表 (機器人觀點)	資料區	TMflow EtherNet/IP函式權限	外部裝置權限
輸入資料表	系統定義區	讀取	寫入
	使用者定義區	讀取	寫入
輸出資料表	系統定義區	讀取	讀取
	使用者定義區	讀取/寫入	讀取

13.1 eip_read_input()

讀取輸入表內容

語法 1

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

參數

int 起始位址
int 讀取位址個數

傳回值

byte[] 傳回資料 byte 陣列

說明

```
byte[] var_ba = eip_read_input(104,8)  
// {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32}
```

語法 2

```
byte eip_read_input(  
    int,  
)
```

參數

int 起始位址

傳回值

byte 傳回資料 byte

說明

```
byte var_b = eip_read_input(104)  
// 0x30
```

語法 3

```
? eip_read_input(  
    string,  
    int,  
    int  
)
```

參數

string 項目名稱
int 項目的起始偏移位址
int 讀取位址個數

傳回值

? 依據通訊資料表所定義的項目決定回傳資料型別。
* 支援型別包含 byte,byte[],int,int[],float,float[],string

語法 4

```
? eip_read_input(  
    string,  
    int,  
)
```

參數

string 項目名稱
int 項目的起始偏移位址

傳回值

? 依據通訊資料表所定義的項目決定回傳資料型別。
* 支援型別包含 byte,byte[],int,int[],float,float[],string

說明

*與語法 3 相同，預設讀取到項目結尾。
*依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 寫入資料。

語法 5

```
? eip_read_input(  
    string  
)
```

參數

string 項目名稱

傳回值

? 依據通訊資料表所定義的項目決定回傳資料型別和資料長度。

* 支援型別包含 byte,byte[],int,int[],float,float[],string

說明

*與語法 3 相同，預設將項目的起始偏移位址填入 0；預設讀取到項目結尾。

*依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 寫入資料。

```
byte var_b = eip_read_input("O2T_StickStatus",0,1)
```

// 0x02

```
var_b = eip_read_input("O2T_StickStatus",0)
```

// 0x02

```
var_b = eip_read_input("O2T_StickStatus")
```

// 0x02

```
byte[] var_ba = eip_read_input("O2T_CtrlBox_DO",0,2)
```

// {0x00,0x04}

```
var_ba = eip_read_input("O2T_CtrlBox_DO")
```

// {0x00,0x04}

```

int[] var_ia = eip_read_input("O2T_Register_Int",0,12)
// byte[] = {0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x02,0x00,0x00,0x00,0x03} (Little Endian)    to int[]
// int[] = {0x00000001,0x00000002,0x00000003} (Little Endian)
// int[] = {1,2,3}

var_ia = eip_read_input("O2T_Register_Int",12)
// byte[] = {0x00,0x00,0x00,0x04,0x00,0x00,0x00,0x05,0x00,0x00,0x00,0x06,0x00,0x00,0x00,0x07,
0x00,0x00,0x00,0x08,0x00,0x00,0x00,0x09,0x00,0x00,0x00,0x0A,0x00,0x00,0x00,0x0B,
0x00,0x00,0x00,0x0C,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)
to int[]
// int[] = {0x00000004,0x00000005,0x00000006,0x00000007,0x00000008,0x00000009,
0x0000000A,0x0000000B,0x0000000C,0x00000000,0x00000000,0x00000000} (Little Endian)
// int[] = {4,5,6,7,8,9,10,11,12,0,0,0}

var_ia = eip_read_input("O2T_Register_Int")
// byte[] = {0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x02,0x00,0x00,0x00,0x03,0x00,0x00,0x00,0x04,
0x00,0x00,0x00,0x05,0x00,0x00,0x00,0x06,0x00,0x00,0x00,0x07,0x00,0x00,0x00,0x08,
0x00,0x00,0x00,0x09,0x00,0x00,0x00,0x0A,0x00,0x00,0x00,0x0B,0x00,0x00,0x00,0x0C,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)    to int[]
// int[] = {0x00000001,0x00000002,0x00000003,0x00000004,0x00000005,0x00000006,
0x00000007,0x00000008,0x00000009,0x0000000A,0x0000000B,0x0000000C,
0x00000000,0x00000000,0x00000000} (Little Endian)
// int[] = {1,2,3,4,5,6,7,8,9,10,11,12,0,0,0}

```

```

float[] var_fa = eip_read_input("O2T_Register_Float",4,12)
// byte[] = {0x3F,0x99,0x99,0x9A,0x3F,0xA6,0x66,0x66,0x40,0x06,0x66,0x66} (Big Endian)      to float[]
// float[] = {0x3F99999A,0x3FA66666,0x40066666} (Big Endian)
// float[] = {1.2,1.3,2.1}

var_fa = eip_read_input("O2T_Register_Float",12)
// byte[] = {0x40,0x06,0x66,0x66,0x40,0x0C,0xCC,0xCD,0x40,0x13,0x33,0x33,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Big Endian)
// to float[]
// float[] = {0x40066666,0x400CCCCD,0x40133333,0x00000000,0x00000000,0x00000000,
0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000} (Big Endian)
// float[] = {2.1,2.2,2.3,0,0,0,0,0,0,0,0,0,0,0,0,0,0}

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} (Big Endian)      to float[]
// float[] = {0x3F8CCCCD,0x3F99999A,0x3FA66666,0x40066666,0x400CCCCD,0x40133333,
0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,
0x00000000,0x00000000,0x00000000} (Big Endian)
// float[] = { 1.1,1.2,1.3,2.1,2.2,2.3,0,0,0,0,0,0,0,0,0}

```

13.2 eip_read_input_int()

讀取輸入表內容並轉換資料為 32 位元整數型別

語法 1

```
int[] eip_read_input_int(  
    int,  
    int,  
    int  
)
```

參數

int	起始位址
int	讀取位址個數
int	讀出後的資料是依照 Little-Endian (DCBA) 或 Big-Endian (ABCD) 轉成 int 陣列。 0 Little-Endian 1 Big-Endian 2 依照設定檔

傳回值

int[] 傳回資料 int 陣列

語法 2

```
int[] eip_read_input_int(  
    int,  
    int  
)
```

參數

int	起始位址
int	讀取位址個數

說明

與語法 1 相同，預設將最後一個參數 int 填入 2。

* 讀出的資料是依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 轉成 int 陣列。

```

int[] var_ia = eip_read_input_int(112,12,0)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)      to int[]
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)
    // int[] = {32767,99999,-32768}

var_ia = eip_read_input_int(112,11,0)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF} (Little Endian)      to int[]
    // int[] = {0x00007FFF,0x0001869F,0x00FF8000} (Little Endian)
    // int[] = {32767,99999,16744448}

var_ia = eip_read_input_int(112,10,0)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80} (Little Endian)      to int[]
    // int[] = {0x00007FFF,0x0001869F,0x00008000} (Little Endian)
    // int[] = {32767,99999,32768}

var_ia = eip_read_input_int(112,12,1)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Big Endian) to int[]
    // int[] = {0xFF7F0000,0x9F860100,0x0080FFFF} (Big Endian)
    // int[] = {-8454144,-1618607872,8454143}

var_ia = eip_read_input_int(112,12,2)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)      to int[]
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)
    // int[] = {32767,99999,-32768}

var_ia = eip_read_input_int(112,12)
    // byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)      to int[]
    // int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)
    // int[] = {32767,99999,-32768}

```

語法 3

```
int eip_read_input_int(  
    int  
)
```

參數

int 起始位址

* 讀出的資料是依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 轉成 int。

傳回值

int 傳回資料 int

說明

```
int var_i = eip_read_input_int(112)  
  
    // byte[] = {0xE4,0x07,0x00,0x00} (Little Endian)      to int  
  
    // int = 0x000007E4 (Little Endian)  
  
    // int = 2020  
  
var_i = eip_read_input_int(112)  
  
    // byte[] = {0x00,0x00,0x07,0xE4} (Big Endian)      to int  
  
    // int = 0x000007E4 (Big Endian)  
  
    // int = 2020
```

13.3 eip_read_input_float()

讀取輸入表內容並轉換資料為 32 位元浮點數型別

語法 1

```
float[] eip_read_input_float(  
    int,  
    int,  
    int  
)
```

參數

int	起始位址
int	讀取位址個數
int	讀出後的資料是依照 Little-Endian (DCBA) 或 Big-Endian (ABCD) 轉成 float 陣列。 0 Little-Endian 1 Big-Endian 2 依照設定檔

傳回值

float[] 傳回資料 float 陣列

語法 2

```
float[] eip_read_input_float(  
    int,  
    int  
)
```

參數

int	起始位址
int	讀取位址個數

說明

與語法 1 相同，預設將最後一個參數 int 填入 2。

* 讀出的資料是依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 轉成 float 陣列。

```

float[] var_fa = eip_read_input_float(172,12,0)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)    to float[]
// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)
// float[] = {1.0,2.0,3.0}

var_fa = eip_read_input_float(172,11,0)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40} (Little Endian)    to float[]
// float[] = {0x3F800000,0x40000000,0x00400000} (Little Endian)
// float[] = {1.0,2.0,5.877472E-39}

var_fa = eip_read_input_float(172,10,0)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00} (Little Endian)    to float[]
// float[] = {0x3F800000,0x40000000,0x00000000} (Little Endian)
// float[] = {1.0,2.0,0.0}

var_fa = eip_read_input_float(172,12,1)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)    to float[]
// float[] = {0x0000803F,0x00000040,0x00004040} (Big Endian)
// float[] = {4.600603E-41,8.96831E-44,2.304856E-41}

var_fa = eip_read_input_float(172,12,2)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)    to float[]
// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)
// float[] = {1.0,2.0,3.0}

var_fa = eip_read_input_float(172,12)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)    to float[]
// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)
// float[] = {1.0,2.0,3.0}

```

語法 3

```
float eip_read_input_float(  
    int  
)
```

參數

int 起始位址

* 讀出的資料是依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 轉成 float。

傳回值

float 傳回資料 float

說明

```
float var_f = eip_read_input_float(172)  
  
// byte[] = {0x00,0x00,0x80,0x3F} (Little Endian) to float  
  
// float = 0x3F800000 (Little Endian)  
  
// float = 1.0  
  
var_f = eip_read_input_float(172)  
  
// byte[] = {0x3F,0x80,0x00,0x00} (Big Endian) to float  
  
// float = 0x3F800000 (Big Endian)  
  
// float = 1.0
```

13.4 eip_read_input_string()

讀取輸入表內容並轉換資料陣列為 UTF8 編碼的字串文本

語法 1

```
string eip_read_input_string(  
    int,  
    int  
)
```

參數

int 起始位址
int 讀取位址個數

傳回值

string 傳回資料 string UTF8 字串格式 (遇到 0x00 結束)

說明

```
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[] = {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  
//             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,0x31,0x32,0x33,0x34,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}  
  
// string = "abcd 達明機器人 1234"
```

```
var_s = eip_read_input_string(104,10)  
  
// byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6,0x98,0x8E}  
  
// string = "abcd 達明"
```

```
var_s = eip_read_input_string(104,8)  
  
// byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6}  
  
// string = "abcd 達◆"
```

13.5 eip_read_input_bit()

讀取輸入表內容，取得 byte 資料中 bit 的數值

語法 1

```
byte eip_read_input_bit(  
    int,  
    int  
)
```

參數

int 起始位址
int byte 資料中的第 n 個 bit 數值

傳回值

byte 傳回資料 byte 型別。
bit 數值 == 1，回傳 1
bit 數值 == 0，回傳 0
*bit 資料將會以 byte 型式回傳。

說明

```
byte var_b = eip_read_input_bit(104,0)  
    // 0x30    get bit: "0"  
    // 0  
  
var_b = eip_read_input_bit(104,5)  
    // 0x30    get bit: "5"  
    // 1
```

語法 2

```
byte eip_read_input_bit(  
    string,  
    int  
)
```

參數

string 項目名稱
int 第 n 個 bit 數值

傳回值

`byte` 傳回資料 byte 型別。
bit 數值 == 1，回傳 1
bit 數值 == 0，回傳 0
`*bit` 資料將會以 `byte` 型式回傳。

說明

```
byte var_b = eip_read_input_bit("O2T_Register_Bit",0)  
// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total  
// 1 get bit: "0"  
  
var_b = eip_read_input_bit("O2T_Register_Bit",17)  
// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total  
// 0 get bit: "17"
```

語法 3

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

參數

`int` 起始位址
`int` 起始 bit
`int` 讀取 bit 個數

傳回值

`byte[]` 傳回資料 `byte[]` 型別。
bit 數值 == 1，回傳 1
bit 數值 == 0，回傳 0
`*bit` 資料將會以 `byte` 型式回傳，即 `bit[0]` 讀為 `byte[0]`，`bit[1]` 讀為 `byte[1]`。

說明

```
byte[] var_ba = eip_read_input_bit(104,0,20)  
// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total  
// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1}  
  
var_ba = eip_read_input_bit(104,12,8)  
// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total  
// byte[] = {1,1,0,1,0,0,1,1}
```

語法 4

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

參數

string 項目名稱
int 起始 bit
int 讀取 bit 個數

傳回值

byte[] 傳回資料 byte[] 型別。
bit 數值 == 1，回傳 1
bit 數值 == 0，回傳 0
*bit 資料將會以 byte 型式回傳，即 bit[0] 讀為 byte[0]， bit[1] 讀為 byte[1]。

說明

```
byte[] var_ba = eip_read_input_bit("O2T_Register_Bit",0,20)  
// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total  
// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1}  
  
var_ba = eip_read_input_bit("O2T_Register_Bit",12,8)  
// byte[] = {1,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1,...} // total  
// byte[] = {1,1,0,1,0,0,1,1}
```

13.6 eip_read_output()

讀取輸出表內容

語法 1

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

參數

int 起始位址
int 讀取位址個數

傳回值

byte[] 傳回資料 byte 陣列

說明

```
byte[] var_ba = eip_read_output(300,8)  
// {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32}
```

語法 2

```
byte eip_read_output(  
    int  
)
```

參數

int 起始位址

傳回值

byte 傳回資料 byte

說明

```
byte var_b = eip_read_output(300)  
// 0x30
```

語法 3

```
? eip_read_output(  
    string,  
    int,  
    int  
)
```

參數

string 項目名稱
int 項目的起始偏移位址
int 讀取位址個數

傳回值

? 依據通訊資料表所定義的項目決定回傳資料型別。
* 支援型別包含 byte,byte[],int,int[],float,float[],string

語法 4

```
? eip_read_output(  
    string,  
    int,  
)
```

參數

string 項目名稱
int 項目的起始偏移位址

傳回值

? 依據通訊資料表所定義的項目決定回傳資料型別。
* 支援型別包含 byte,byte[],int,int[],float,float[],string

說明

*與語法 3 相同，預設讀取到項目結尾。
*依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 寫入資料。

語法 5

```
? eip_read_output(  
    string  
)
```

參數

string 項目名稱

傳回值

? 依據通訊資料表所定義的項目決定回傳資料型別和資料長度。

* 支援型別包含 byte,byte[],int,int[],float,float[],string

說明

* 與語法 3 相同，預設將項目的起始偏移位址填入 0；預設讀取到項目結尾。

* 依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 寫入資料。

```
byte var_b = eip_read_output("ManualAuto",0,1)  
// 0x02  
  
var_b = eip_read_output("ManualAuto",0)  
// 0x02  
  
var_b = eip_read_output("ManualAuto")  
// 0x02
```

```
byte[] var_ba = eip_read_output("Error_Code",0,4)  
// {0x00,0x04,0x80,0x0C}  
  
var_ba = eip_read_output("Error_Code",0,2)  
// {0x00,0x04}  
  
var_ba = eip_read_output("Error_Code")  
// {0x00,0x04,0x80,0x0C}
```

```
int var_i = eip_read_output("Current_Time_Year")  
// byte[] = {0x00,0x00,0x07,0xE4} (Little Endian)      to int  
// int = 0x000007E4 (Little Endian)  
// int = 2020
```

```

int[] var_ia = eip_read_output("T2O_Register_Int",0,12)
    // byte[] = {0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x02,0x00,0x00,0x00,0x03} (Little Endian)    to int[]
    // int[] = { 0x00000001,0x00000002,0x00000003} (Little Endian)
    // int[] = {1,2,3}

var_ia = eip_read_output("T2O_Register_Int",12)
    // byte[] = {0x00,0x00,0x00,0x04,0x00,0x00,0x00,0x05,0x00,0x00,0x00,0x06,0x00,0x00,0x00,0x07,
        0x00,0x00,0x00,0x08,0x00,0x00,0x00,0x09,0x00,0x00,0x00,0x0A,0x00,0x00,0x00,0x0B,
        0x00,0x00,0x00,0x0C,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)
    to int[]
    // int[] = {0x00000004,0x00000005,0x00000006,0x00000007,0x00000008,0x00000009,
        0x0000000A,0x0000000B,0x0000000C,0x00000000,0x00000000,0x00000000} (Little Endian)
    // int[] = {4,5,6,7,8,9,10,11,12,0,0,0}

var_ia = eip_read_output("T2O_Register_Int")
    // byte[] = {0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x02,0x00,0x00,0x00,0x03,0x00,0x00,0x00,0x04,
        0x00,0x00,0x00,0x05,0x00,0x00,0x00,0x06,0x00,0x00,0x00,0x07,0x00,0x00,0x00,0x08,
        0x00,0x00,0x00,0x09,0x00,0x00,0x00,0x0A,0x00,0x00,0x00,0x0B,0x00,0x00,0x00,0x0C,
        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)    to int[]
    // int[] = {0x00000001,0x00000002,0x00000003,0x00000004,0x00000005,0x00000006,
        0x00000007,0x00000008,0x00000009,0x0000000A,0x0000000B,0x0000000C,
        0x00000000,0x00000000,0x00000000} (Little Endian)
    // int[] = {1,2,3,4,5,6,7,8,9,10,11,12,0,0,0}

float var_f = eip_read_output("Current_TCP_Mass")
    // byte[] = {0x40,0x40,0x00,0x00} (Big Endian)      to float
    // float = 0x40400000 (Big Endian)
    // float = 3.0

float[] var_fa = eip_read_output("Current_TCP_Value",4,12)
    // byte[] = {0x3F,0x99,0x99,0x9A,0x3F,0xA6,0x66,0x66,0x40,0x06,0x66,0x66} (Big Endian)      to float[]
    // float[] = {0x3F9999A,0x3FA66666,0x40066666} (Big Endian)
    // float[] = {1.2,1.3,2.1}

```

```

var_fa = eip_read_output("Current_TCP_Value",12)

// byte[] = { 0x40,0x06,0x66,0x66,0x40,0x0C,0xCC,0xCD,0x40,0x13,0x33,0x33} (Big Endian)    to float[]

// float[] = {0x40066666,0x400CCCCD,0x40133333} (Big Endian)

// float[] = {2.1,2.2,2.3}

var_fa = eip_read_output("Current_TCP_Value")

// byte[] = {0x3F,0x8C,0xCC,0xCD,0x3F,0x99,0x99,0x9A,0x3F,0xA6,0x66,0x66,0x40,0x06,0x66,0x66,
           0x40,0x0C,0xCC,0xCD,0x40,0x13,0x33,0x33} (Big Endian)    to float[]

// float[] = {0x3F8CCCCD,0x3F99999A,0x3FA66666,0x40066666,0x400CCCCD,0x40133333} (Big Endian)

// float[] = { 1.1,1.2,1.3,2.1,2.2,2.3}

string var_s = eip_read_output ("ControlBoxID",0,16)

// byte[] = {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}

// string = "01060112"

var_s = eip_read_output ("ControlBoxID",4,3)

// byte[] = {0x30,0x31,0x31}

// string = "011"

var_s = eip_read_output ("ControlBoxID")

// byte[] = {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}

// string = "01060112"

```

13.7 eip_read_output_int()

讀取輸出表內容並轉換資料為 32 位元整數型別

語法 1

```
int[] eip_read_output_int(  
    int,  
    int,  
    int  
)
```

參數

int	起始位址
int	讀取位址個數
int	讀出後的資料是依照 Little-Endian (DCBA) 或 Big-Endian (ABCD) 轉成 int 陣列。 0 Little-Endian 1 Big-Endian 2 依照設定檔

傳回值

int[] 傳回資料 int 陣列

語法 2

```
int[] eip_read_output_int(  
    int,  
    int  
)
```

參數

int	起始位址
int	讀取位址個數

說明

與語法 1 相同，預設將最後一個參數 int 填入 2。

* 讀出的資料是依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 轉成 int 陣列。

```

int[] var_ia = eip_read_output_int(308,12,0)
// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]
// int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)
// int[] = {32767,99999,-32768}

var_ia = eip_read_output_int(308,11,0)
// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF} (Little Endian)    to int[]
// int[] = {0x00007FFF,0x0001869F,0x00FF8000} (Little Endian)
// int[] = {32767,99999,16744448}

var_ia = eip_read_output_int(308,10,0)
// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80} (Little Endian)    to int[]
// int[] = {0x00007FFF,0x0001869F,0x00008000} (Little Endian)
// int[] = {32767,99999,32768}

var_ia = eip_read_output_int(308,12,1)
// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]
// int[] = {0xFF7F0000,0x9F860100,0x0080FFFF} (Big Endian)
// int[] = {-8454144,-1618607872,8454143}

var_ia = eip_read_output_int(308,12,2)
// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]
// int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)
// int[] = {32767,99999,-32768}

var_ia = eip_read_output_int(308,12)
// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]
// int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)
// int[] = {32767,99999,-32768}

```

語法 3

```
int eip_read_output_int(  
    int  
)
```

參數

int 起始位址

* 讀出的資料是依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 轉成 int。

傳回值

int 傳回資料 int

說明

```
int var_i = eip_read_output_int(308)  
  
    // byte[] = {0xE4,0x07,0x00,0x00} (Little Endian)      to int  
  
    // int = 0x000007E4 (Little Endian)  
  
    // int = 2020  
  
var_i = eip_read_output_int(308)  
  
    // byte[] = {0x00,0x00,0x07,0xE4} (Big Endian)      to int  
  
    // int = 0x000007E4 (Big Endian)  
  
    // int = 2020
```

13.8 eip_read_output_float()

讀取輸出表內容並轉換資料為 32 位元浮點數型別

語法 1

```
float[] eip_read_output_float(  
    int,  
    int,  
    int  
)
```

參數

int	起始位址
int	讀取位址個數
int	讀出後的資料是依照 Little-Endian (DCBA) 或 Big-Endian (ABCD) 轉成 float 陣列。 0 Little-Endian 1 Big-Endian 2 依照設定檔

傳回值

float[] 傳回資料 float 陣列

語法 2

```
float[] eip_read_output_float(  
    int,  
    int  
)
```

參數

int	起始位址
int	讀取位址個數

說明

與語法 1 相同，預設將最後一個參數 int 填入 2。

* 讀出的資料是依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 轉成 float 陣列。

```

float[] var_fa = eip_read_output_float(368,12,0)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)    to float[]
// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)
// float[] = {1.0,2.0,3.0}

var_fa = eip_read_output_float(368,11,0)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40} (Little Endian)    to float[]
// float[] = {0x3F800000,0x40000000,0x00400000} (Little Endian)
// float[] = {1.0,2.0,5.877472E-39}

var_fa = eip_read_output_float(368,10,0)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00} (Little Endian)    to float[]
// float[] = {0x3F800000,0x40000000,0x00000000} (Little Endian)
// float[] = {1.0,2.0,0.0}

var_fa = eip_read_output_float(368,12,1)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)    to float[]
// float[] = {0x0000803F,0x00000040,0x00004040} (Big Endian)
// float[] = {4.600603E-41,8.96831E-44,2.304856E-41}

var_fa = eip_read_output_float(368,12,2)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)    to float[]
// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)
// float[] = {1.0,2.0,3.0}

var_fa = eip_read_output_float(368,12)
// byte[] = {0x00,0x00,0x80,0x3F,0x00,0x00,0x00,0x40,0x00,0x00,0x40,0x40} (Little Endian)    to float[]
// float[] = {0x3F800000,0x40000000,0x40400000} (Little Endian)
// float[] = {1.0,2.0,3.0}

```

語法 3

```
float eip_read_output_float(  
    int  
)
```

參數

int 起始位址

* 讀出的資料是依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 轉成 float。

傳回值

float 傳回資料 float

說明

```
float var_f = eip_read_output_float(368)  
  
// byte[] = {0x00,0x00,0x80,0x3F} (Little Endian) to float  
  
// float = 0x3F800000 (Little Endian)  
  
// float = 1.0  
  
var_f = eip_read_output_float(368)  
  
// byte[] = {0x3F,0x80,0x00,0x00} (Big Endian) to float  
  
// float = 0x3F800000 (Big Endian)  
  
// float = 1.0
```

13.9 eip_read_output_string()

讀取輸出表內容並轉換資料陣列為 UTF8 編碼的字串文本

語法 1

```
string eip_read_output_string(  
    int,  
    int  
)
```

參數

int 起始位址
int 讀取位址個數

傳回值

string 傳回資料 string UTF8 字串格式 (遇到 0x00 結束)

說明

```
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[] = {0x30,0x31,0x30,0x36,0x30,0x31,0x31,0x32,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  
//             0x54,0x4D,0x35,0x2D,0x37,0x30,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}  
  
// 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,0x31,0x32,0x33,0x34,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}  
  
// string = "abcd 達明機器人 1234"  
  
var_s = eip_read_output_string(300,10)  
  
// byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6,0x98,0x8E}  
  
// string = "abcd 達明"  
  
var_s = eip_read_output_string(300,8)  
  
// byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6}  
  
// string = "abcd 達◆"
```

13.10 eip_read_output_bit()

讀取輸出表內容，取得 byte 資料中 bit 的數值

語法 1

```
byte eip_read_output_bit(  
    int,  
    int  
)
```

參數

int 起始位址
int byte 資料中的第 n 個 bit 數值

傳回值

byte 傳回資料 byte 型別。
bit 數值 == 1，回傳 1
bit 數值 == 0，回傳 0
*bit 資料將會以 byte 型式回傳。

說明

```
byte var_b = eip_read_output_bit(300,0)  
// 0x30    get bit: "0"  
// 0  
  
var_b = eip_read_output_bit(300,5)  
// 0x30    get bit: "5"  
// 1
```

語法 2

```
byte eip_read_output_bit(  
    string,  
    int  
)
```

參數

string 項目名稱
int 第 n 個 bit 數值

傳回值

byte 傳回資料 byte 型別。
bit 數值 == 1，回傳 1

bit 數值 == 0，回傳 0
*bit 資料將會以 byte 型式回傳。

說明

```
byte[] var_data = {57,184,12}  
  
eip_write_output(300,var_data,3)  
// {00111001,10111000,00001100} (binary)  
  
byte var_b = eip_read_output_bit("T2O_Register_Bit",0)  
// 1  
  
var_b = eip_read_output_bit("T2O_Register_Bit",17)  
// 0
```

語法 3

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

參數

int 起始位址
int 起始 bit
int 讀取 bit 個數

傳回值

byte[] 傳回資料 byte[] 型別。
bit 數值 == 1，回傳 1
bit 數值 == 0，回傳 0
*bit 資料將會以 byte 型式回傳，即 bit[0] 讀為 byte[0]， bit[1] 讀為 byte[1]。

說明

```
byte[] var_data = {57,184,12}  
  
eip_write_output(300,var_data,3)  
// {00111001,10111000,00001100} (binary)  
  
byte[] var_ba = eip_read_output_bit(300,0,20)  
// byte[] = {1,0,0,1,1,1,0,0,0,0,1,1,1,0,1,0,0,1,1}  
  
var_ba = eip_read_output_bit(300,12,8)  
// byte[] = {1,1,0,1,0,0,1,1}
```

語法 4

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

參數

string 項目名稱
int 起始 bit
int 讀取 bit 個數

傳回值

byte[] 傳回資料 byte[] 型別。
bit 數值 == 1，回傳 1
bit 數值 == 0，回傳 0
*bit 資料將會以 byte 型式回傳，即 bit[0] 讀為 byte[0]， bit[1] 讀為 byte[1]。

說明

```
byte[] var_data = {57,184,12}  
  
eip_write_output(300,var_data,3)  
  
// {00111001,10111000,00001100} (binary)  
  
byte[] var_ba = eip_read_output_bit("T2O_Register_Bit",0,20)  
  
// byte[] = {1,0,0,1,1,0,0,0,0,1,1,1,0,1,0,0,1,1}  
  
var_ba = eip_read_output_bit("T2O_Register_Bit",12,8)  
  
// byte[] = {1,1,0,1,0,0,1,1}
```

13.11 eip_write_output()

寫入輸出表內容

語法 1

```
bool eip_write_output(  
    int,  
    ?,  
    int  
)
```

參數

int	起始位址
?	寫入資料
	* 支援型別包含 byte,byte[],int,int[],float,float[],string
int	最大寫入位址個數
>0	合法位址長度，依位址個數寫入
<=0	非法位址長度，依寫入資料的完整長度寫入

傳回值

bool	True	寫入成功
	False	寫入失敗

1. 如果 ? 寫入資料值為空字串或空陣列
2. 無法正確發送或接收通訊

說明

* 依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 寫入資料。

語法 2

```
bool eip_write_output(  
    int,  
    ?  
)
```

參數

int	起始位址
?	寫入資料
	* 支援型別包含 byte,byte[],int,int[],float,float[],string

傳回值

bool	True	寫入成功
	False	寫入失敗

1. 如果 ? 寫入資料值為空字串或空陣列
2. 無法正確發送或接收通訊

說明

與語法 1 相同，預設依寫入資料的完整長度寫入。

*依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 寫入資料。

語法 3

```
bool eip_write_output(  
    int,  
    ?,  
    int,  
    int  
)
```

參數

int	起始位址
?	寫入資料
*支援型別包含 byte,byte[],int,int[],float,float[],string	
int	寫入資料的起始位址
int	寫入資料的位址個數

傳回值

bool	True	寫入成功
	False	寫入失敗
1. 如果 ? 寫入資料值為空字串或空陣列		
2. 無法正確發送或接收通訊		

說明

*依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 寫入資料。

```
byte var_data = 255  
eip_write_output(300,var_data,1)  
byte var_b = eip_read_output(300)  
// 0xFF
```

```
byte[] var_data = {1,127,255}  
eip_write_output(300,var_data,3)  
byte[] var_ba = eip_read_output(300,3)  
// {0x01,0x7F,0xFF}  
eip_write_output(300,var_data,2)  
var_ba = eip_read_output(300,3)
```

```

// {0x01,0x7F,0x00}

eip_write_output(300,var_data,-1)

var_ba = eip_read_output(300,3)

// {0x00,0x7F,0xFF}

int var_data = 32767

eip_write_output(308,var_data,4)

int var_i = eip_read_output_int(308)

// byte[] = {0xFF,0x7F,0x00,0x00} (Little Endian)      to int

// int = 0x00007FFF (Little Endian)

// int = 32767

eip_write_output(308,var_data,1)

var_i = eip_read_output_int(308)

// byte[] = {0xFF,0x00,0x00,0x00} (Little Endian)      to int

// int = 0x000000FF (Little Endian)

// int = 255

int[] var_data = {32767,99999,-32768}

eip_write_output(308,var_data,12)

int[] var_ia = eip_read_output_int(308,12)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)      to int[]

// int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)

// int[] = {32767,99999,-32768}

eip_write_output(308,var_data,3)

var_ia = eip_read_output_int(308,12)

// byte[] = {0xFF,0x7F,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)      to int[]

// int[] = {0x00007FFF,0x00000000,0x00000000} (Little Endian)

// int[] = {32767,0,0}

eip_write_output(308,var_data,11)

var_ia = eip_read_output_int(308,12)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0x00} (Little Endian)      to int[]

// int[] = {0x00007FFF,0x0001869F,0x00FF8000} (Little Endian)

```

```

// int[] = {32767,99999,16744448}

eip_write_output(308,var_data,4,4)

var_ia = eip_read_output_int(308,12)

// byte[] = {0x9F,0x86,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)    to int[]

// int[] = {0x0001869F,0x00000000,0x00000000} (Little Endian)

// int[] = {99999,0,0}

float var_data = -10.0

eip_write_output(368,var_data,4)

float var_f = eip_read_output_float(368)

// byte[] = {0x00,0x00,0x20,0xC1} (Little Endian)      to float

// float = 0xC1200000 (Little Endian)

// float = -10.0

eip_write_output(368,var_data,1)

var_f = eip_read_output_float(368)

// byte[] = {0x00,0x00,0x00,0x00} (Little Endian)      to float

// float = 0x00000000 (Little Endian)

// float = 0

float[] var_data = {-10.0,3.3,123.45}

eip_write_output(368,var_data,12)

float[] var_fa = eip_read_output_float(368,12)

// byte[] = {0x00,0x00,0x20,0xC1,0x33,0x33,0x53,0x40,0x66,0xE6,0xF6,0x42} (Little Endian)    to float[]

// float[] = {0xC1200000,0x40533333,0x42F6E666} (Little Endian)

// float[] = {-10,3.3,123.45}

eip_write_output(368,var_data,3)

var_fa = eip_read_output_float(368,12)

// byte[] = {0x00,0x00,0x20,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)    to float[]

// float[] = {0x00200000,0x00000000,0x00000000} (Little Endian)

// float[] = {2.938736E-39,0,0}

eip_write_output(368,var_data,11)

var_fa = eip_read_output_float(368,12)

```

```

// byte[] = {0x00,0x00,0x20,0xC1,0x33,0x33,0x53,0x40,0x66,0xE6,0xF6,0x00} (Little Endian)    to float[]
// float[] = {0xC1200000,0x40533333,0x00F6E666} (Little Endian)
// float[] = {-10.3.3,2.267418E-38}

eip_write_output(368,var_data,4,4)

var_fa = eip_read_output_float(368,12)

// byte[] = {0x33,0x33,0x53,0x40,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00} (Little Endian)    to float[]
// float[] = {0x40533333,0x00000000,0x00000000} (Little Endian)
// float[] = {3.3,0,0}

string var_data = "abcd 達明機器人 1234"

eip_write_output(300,var_data,32)

string var_s = eip_read_output_string(300,32)

// byte[] = {0x61,0x62,0x63,0x64,0xE9,0x81,0x94,0xE6,0x98,0x8E,0xE6,0xA9,0x9F,0xE5,0x99,0xA8,
//            0xE4,0xBA,0xBA,0x31,0x32,0x33,0x34,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
// 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}
//          0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
// 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}
//            0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
// 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"

```

語法 4

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

參數

string 項目名稱
int 項目的起始偏移位址
? 寫入資料
* 支援型別包含 byte,byte[],int,int[],float,float[],string
int 寫入資料的起始位址
int 寫入資料的位址個數

傳回值

bool True 寫入成功
False 寫入失敗 1. 如果 ? 寫入資料值為空字串或空陣列
2. 無法正確發送或接收通訊

說明

* 依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 寫入資料。

語法 5

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

參數

string 項目名稱
int 項目的起始偏移位址
? 寫入資料
* 支援型別包含 byte,byte[],int,int[],float,float[],string
int 寫入資料的起始位址

傳回值

bool True 寫入成功	
False 寫入失敗	1. 如果 ? 寫入資料值為空字串或空陣列
	2. 無法正確發送或接收通訊

說明

*與語法 4 相同，預設將寫入資料剩餘部分完整寫入

*依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 寫入資料。

語法 6

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

參數

string 項目名稱	
int 項目的起始偏移位址	
? 寫入資料	
* 支援型別包含 byte,byte[],int,int[],float,float[],string	

傳回值

bool True 寫入成功	
False 寫入失敗	1. 如果 ? 寫入資料值為空字串或空陣列
	2. 無法正確發送或接收通訊

說明

*與語法 4 相同，預設將寫入資料的起始位址填入 0；預設依寫入資料的完整長度寫入。

*依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 寫入資料。

語法 7

```
bool eip_write_output(  
    string,  
    ?  
)
```

參數

string 項目名稱	
? 寫入資料	
* 支援型別包含 byte,byte[],int,int[],float,float[],string	

傳回值

bool	True	寫入成功
	False	寫入失敗
1. 如果 ? 寫入資料值為空字串或空陣列		
2. 無法正確發送或接收通訊		

說明

*與語法 4 相同，預設將項目的起始偏移位址填入 0；寫入資料的起始位址填入 0；依寫入資料的完整長度寫入。

*依照設定檔 Little Endian (DCBA) 或 Big Endian (ABCD) 寫入資料。

```
int[] var_data = {32767,99999,-32768}

eip_write_output("T2O_Register_Int",0,var_data,0,12)

int[] var_ia = eip_read_output_int(308,12)

// byte[] = {0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF} (Little Endian)    to int[]

// int[] = {0x00007FFF,0x0001869F,0xFFFF8000} (Little Endian)

// int[] = {32767,99999,-32768}

eip_write_output("T2O_Register_Int",4,var_data,4,4)

var_ia = eip_read_output_int(308,12)

// byte[] = {0x00,0x00,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x00,0x00,0x00} (Little Endian)    to int[]

// int[] = {0x00000000,0x0001869F,0x00000000} (Little Endian)

// int[] = {0,99999,0}

eip_write_output("T2O_Register_Int",4,var_data)

var_ia = eip_read_output_int(308,20)

// byte[] = {0x00,0x00,0x00,0x00,0xFF,0x7F,0x00,0x00,0x9F,0x86,0x01,0x00,0x00,0x80,0xFF,0xFF,
//            0x00,0x00,0x00,0x00} (Little Endian)    to int[]

// int[] = {0x00000000,0x00007FFF,0x0001869F,0xFFFF8000,0x00000000} (Little Endian)

// int[] = {0,32767,99999,-32768,0}

float[] var_data = {-10.0,3.3,123.45}

eip_write_output("T2O_Register_Float",0,var_data,0,12)

float[] var_fa = eip_read_output_float(368,12)

// byte[] = {0x00,0x00,0x20,0xC1,0x33,0x33,0x53,0x40,0x66,0xE6,0xF6,0x42} (Little Endian)    to float[]

// float[] = {0xC1200000,0x40533333,0x42F6E666} (Little Endian)

// float[] = {-10,3.3,123.45}
```

```
eip_write_output("T2O_Register_Float",4,var_data,4,8)
var_fa = eip_read_output_float(368,12)
// byte[] = {0x00,0x00,0x00,0x00,0x33,0x33,0x53,0x40,0x66,0xE6,0xF6,0x42} (Little Endian)    to float[]
// float[] = {0x00000000,0x40533333,0x42F6E666} (Little Endian)
// float[] = {0,3.3,123.45}

eip_write_output("T2O_Register_Float",8,var_data)
var_fa = eip_read_output_float(368,20)
// byte[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x20,0xC1,0x33,0x33,0x53,0x40,
//            0x66,0xE6,0xF6,0x42} (Little Endian)    to float[]
// float[] = {0x00000000,0x00000000,0xC1200000,0x40533333,0x42F6E666} (Little Endian)
// float[] = {0,0,-10,3.3,123.45}
```

13.12 eip_write_output_bit()

寫入輸出表內容，設定 byte 資料中 bit 的數值

語法 1

```
bool eip_write_output_bit(  
    int,  
    int,  
    int  
)
```

參數

int 起始位址
int byte 資料中的第 n 個 bit
int 寫入資料
*bit 資料將會以 int 型式寫入。

傳回值

bool True 寫入成功
False 寫入失敗 1. 無法正確發送或接收通訊

說明

```
byte var_data = 240  
  
eip_write_output(300,var_data)  
  
byte var_b = eip_read_output(300)  
    // 0xF0  
  
eip_write_output_bit(300,1,1)  
  
var_b = eip_read_output_bit(300,1)  
    // 0xF2    get bit: "1"  
    // 1  
  
eip_write_output_bit(300,7,0)  
  
var_b = eip_read_output_bit(300,7)  
    // 0x72    get bit: "7"  
    // 0
```

語法 2

```
bool eip_write_output_bit(  
    string,  
    int,  
    int  
)
```

參數

int 項目名稱
int 第 n 個 bit
int 寫入資料
*bit 資料將會以 int 型式寫入。

傳回值

bool True 寫入成功
False 寫入失敗 1. 無法正確發送或接收通訊

說明

```
byte var_data = 240  
eip_write_output(300,var_data)  
byte var_b = eip_read_output(300)  
    // 0xF0  
eip_write_output_bit("T2O_Register_Bit",1,1)  
var_b = eip_read_output_bit(300,1)  
    // 0xF2    get bit: "1"  
    // 1  
eip_write_output_bit("T2O_Register_Bit",7,0)  
var_b = eip_read_output_bit(300,7)  
    // 0x72    get bit: "7"  
    // 0
```

語法 3

```
bool eip_write_output_bit(  
    int,  
    int,  
    byte[],  
    int,  
    int  
)
```

參數

int 起始位址
int 起始 bit
byte[] 寫入資料
*bit 資料將會以 byte 型式寫入，即 byte[0] 填入 bit[0]， byte[1] 填入 bit[1]
int 寫入資料的起始 bit
int 寫入資料的 bit 個數

傳回值

bool True 寫入成功
False 寫入失敗 1. 無法正確發送或接收通訊

說明

byte 數值 ≥ 1 ，則 bit 數值 = 1，
byte 數值 = 0，則 bit 數值 = 0

語法 4

```
bool eip_write_output_bit(  
    int,  
    int,  
    byte[],  
    int  
)
```

參數

int 起始位址
int 起始 bit
byte[] 寫入資料
*bit 資料將會以 byte 型式寫入，即 byte[0] 填入 bit[0]， byte[1] 填入 bit[1]
int 寫入資料的起始 bit

傳回值

bool True 寫入成功
False 寫入失敗 1. 無法正確發送或接收通訊

說明

*與語法 3 相同，將寫入資料剩餘部分完整寫入。

byte 數值 ≥ 1 ，則 bit 數值 = 1，
byte 數值 = 0，則 bit 數值 = 0

語法 5

```
bool eip_write_output_bit(  
    int,  
    int,  
    byte[]  
)
```

參數

int 起始位址

int 起始 bit

byte[] 寫入資料

*bit 資料將會以 byte 型式寫入，即 byte[0] 填入 bit[0]，byte[1] 填入 bit[1]

傳回值

bool	True	寫入成功
	False	寫入失敗 1. 無法正確發送或接收通訊

說明

*與語法 3 相同，預設將寫入資料的起始 bit 填入 0；預設依寫入資料的完整長度寫入。

byte 數值 ≥ 1 ，則 bit 數值 = 1，
byte 數值 = 0，則 bit 數值 = 0

```
byte[] var_data = {1,0,0,1,1,0,0,0,0,1,1,1,0,1,0,0,1,1}
```

```
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[] = {1,0,0,1,1,0,0,0,0,1,1,1,0,1,0,0,1,1}
```

```
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[] = {0,0,0,1,0,0,0,0,1,1,1,0,0,0,0,0,0,0}
```

語法 6

```
bool eip_write_output_bit(  
    string,  
    int,  
    byte[],  
    int,  
    int  
)
```

參數

string 項目名稱
int 起始 bit
byte[] 寫入資料
*bit 資料將會以 byte 型式寫入，即 byte[0] 填入 bit[0]， byte[1] 填入 bit[1]
int 寫入資料的起始 bit
int 寫入資料的 bit 個數

傳回值

bool True 寫入成功
False 寫入失敗 1. 無法正確發送或接收通訊

說明

byte 數值 ≥ 1 ，則 bit 數值 = 1，
byte 數值 = 0，則 bit 數值 = 0

語法 7

```
bool eip_write_output_bit(  
    string,  
    int,  
    byte[],  
    int  
)
```

參數

string 項目名稱
int 起始 bit
byte[] 寫入資料
*bit 資料將會以 byte 型式寫入，即 byte[0] 填入 bit[0]， byte[1] 填入 bit[1]
int 寫入資料的起始 bit

傳回值

bool True 寫入成功
False 寫入失敗 1. 無法正確發送或接收通訊

說明

*與語法 6 相同，將寫入資料剩餘部分完整寫入。

byte 數值 ≥ 1 ，則 bit 數值 = 1，
byte 數值 = 0，則 bit 數值 = 0

語法 8

```
bool eip_write_output_bit(  
    string,  
    int,  
    byte[]  
)
```

參數

string 起始位址
int 起始 bit
byte[] 寫入資料
*bit 資料將會以 byte 型式寫入，即 byte[0] 填入 bit[0]，byte[1] 填入 bit[1]

傳回值

bool True 寫入成功
False 寫入失敗 1. 無法正確發送或接收通訊

說明

*與語法 6 相同，預設將寫入資料的起始 bit 填入 0；預設依寫入資料的完整長度寫入。

byte 數值 ≥ 1 ，則 bit 數值 = 1，
byte 數值 = 0，則 bit 數值 = 0

```
byte[] var_data = {1,0,0,1,1,0,0,0,0,1,1,1,0,1,0,0,1,1}  
  
eip_write_output_bit("T2O_Register_Bit",0,var_data,0,20)  
  
byte[] var_ba = eip_read_output(300,3)  
  
// byte[] = {0x39,0xB8,0x0C}  
  
var_ba = eip_read_output_bit(300,0,20)  
  
// byte[] = {1,0,0,1,1,0,0,0,0,1,1,1,0,1,0,0,1,1}  
  
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}
```

TECHMAN
ROBOT



www.tm-robot.com