18.巨集指令說明

本章介紹巨集指令的語法、編輯、及使用方法。

18.1. 概要

巨集指令提供了應用程式之外所需的附加功能。在 HMI 人機界面運行時，巨集指令可以自動的執行這些命令。它可以擔負執行譬如複雜的運算、字串處理，和使用者與工程之間的交流等功能。本章主要介紹巨集指令的語法、如何使用和編輯方法等功能。希望通過本章的說明，能夠使各位能夠快速的掌握 EasyBuilder Pro 軟體提供的強大的巨集指令功能。

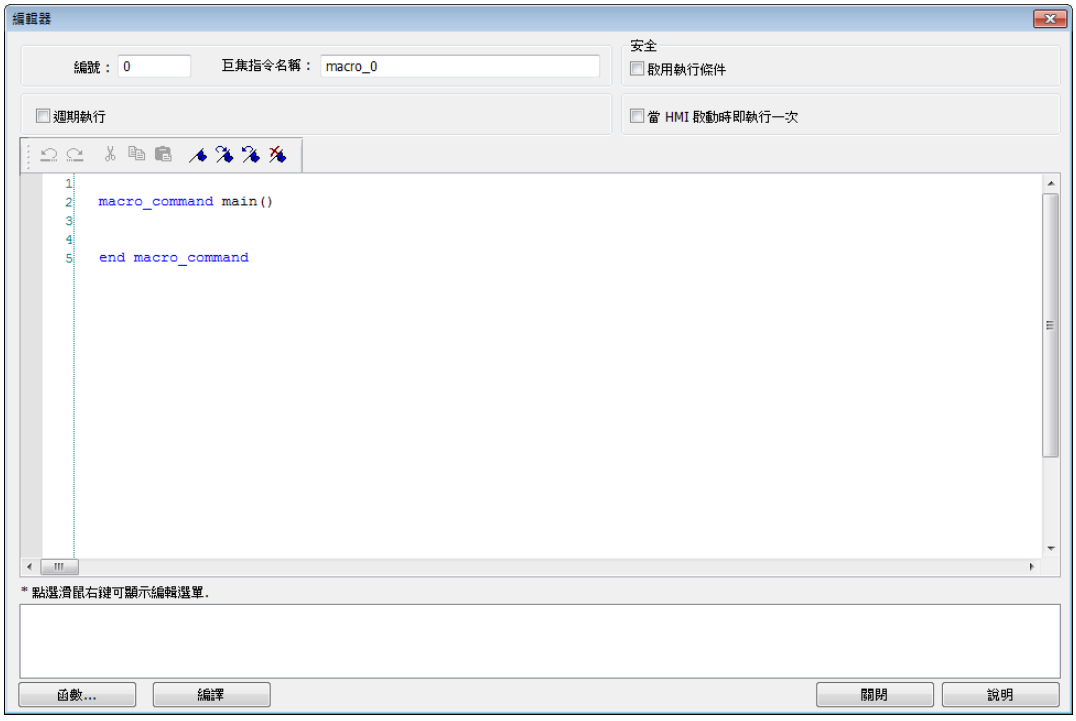
18.2. 巨集指令編輯器功能使用說明

巨集指令編輯器提供下列新功能：

- 顯示行號
- 復原 (Undo) / 重複 (Redo)
- 剪下 (Cut) / 複製 (Copy) / 貼上 (Paste)
- 全選 (Select All)
- 建立 / 取消書籤 (Toggle Bookmark) / 上一個書籤 (Previous Bookmark) / 下一個書籤 (Next Bookmark) / 清除全部書籤 (Clear All Bookmarks)
- 程式碼摺疊 (Toggle All Outlining)
- 安全 -> 啟用執行條件
- 週期執行
- 當 HMI 啟動時即執行一次

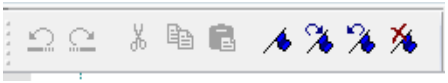
以下將詳細描述如何使用各項功能。

1. 打開巨集指令編輯器，可以看到編輯區左邊將自動顯示行號。

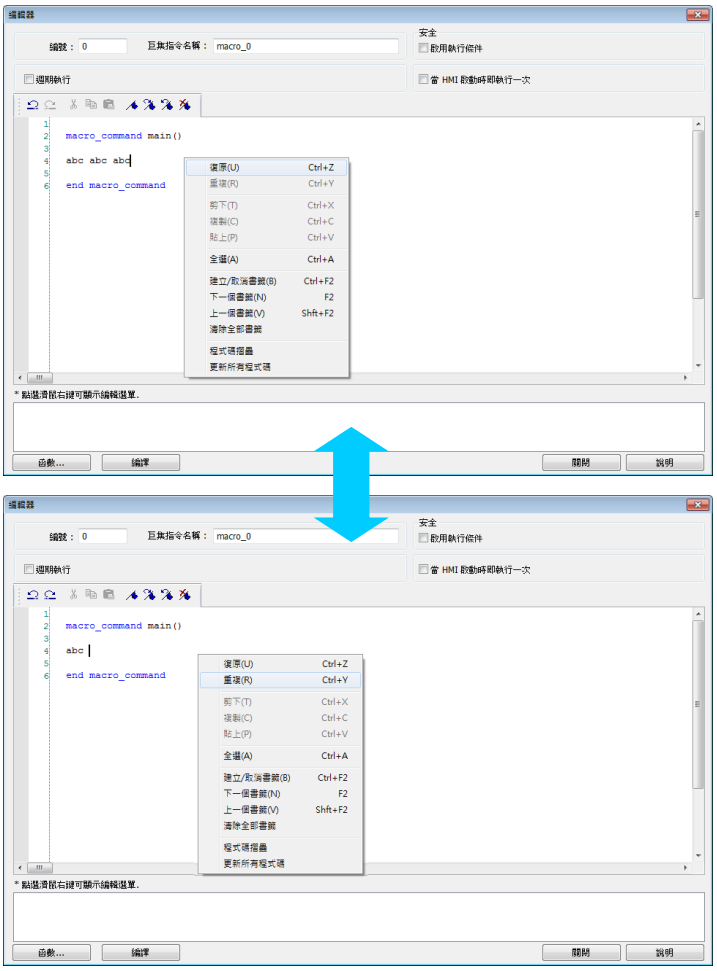


2. 編輯區中按滑鼠右鍵，呼叫出右鍵選單如下圖。目前的狀態無法使用的功能將顯示灰色。例如必須在編輯區中選取一段文字才會開啟複製功能，因此未選取任何文字的狀態下，複製功能暫不開啟。提供快速鍵，如選單內所提示。

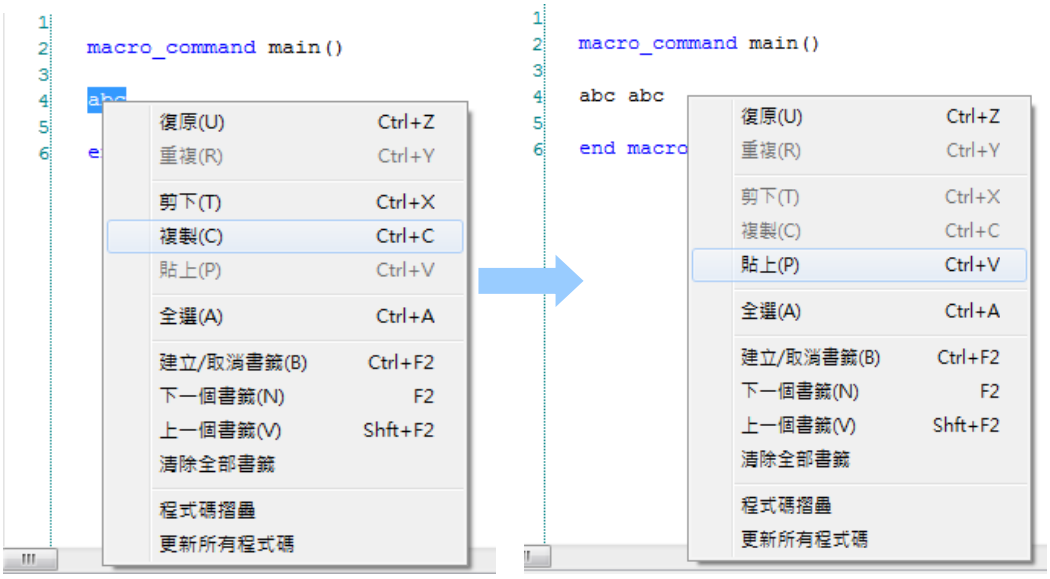
3. 編輯區上方有工具列，提供 [復原]、[重複]、[剪下]、[複製]、[貼上]、[建立/取消書籤]、[下一個書籤]、[上一個書籤]、[清除全部書籤] 等按鈕方便快速選取。



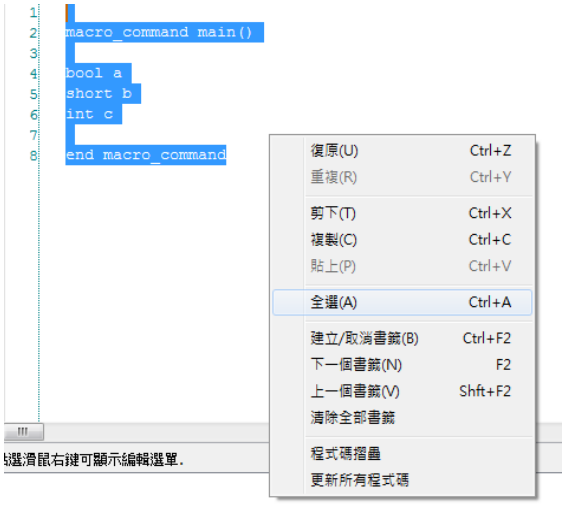
4. 改變編輯區內容將開啟 [復原] 功能，使用者執行復原後可用 [重複] 復原。使用者可從右鍵選單或是利用熱鍵 (Undo : Ctrl+Z，Redo : Ctrl+Y) 執行此功能。



5. 在編輯區選取一段文字後可進行 [剪下]、[複製]，之後可用 [貼上] 將選取的文字貼上。

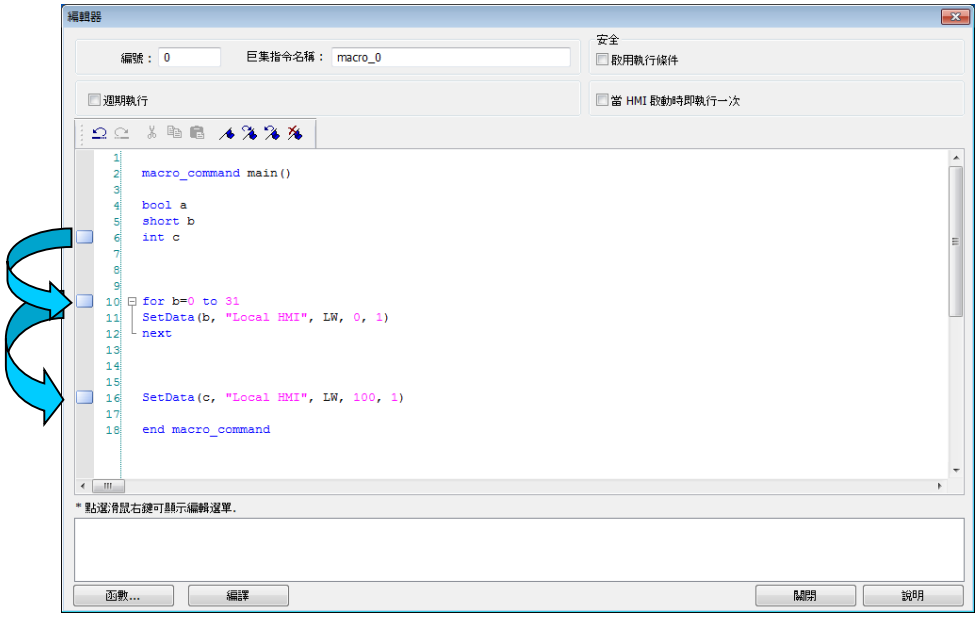


6. 選擇 [全選] 可選取編輯區全部內容。

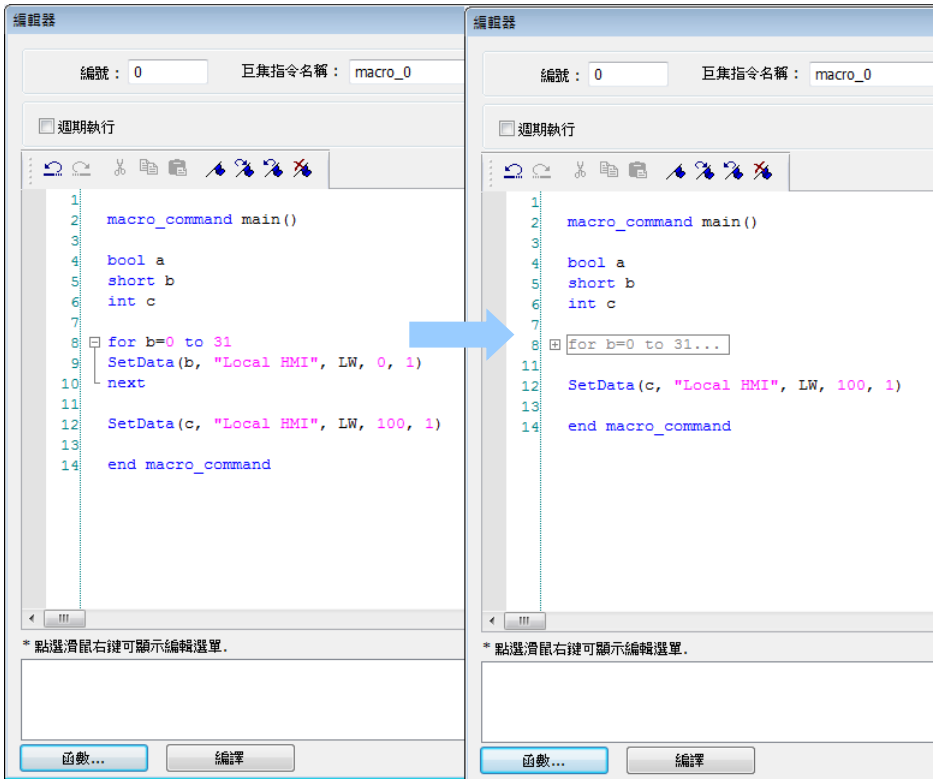


7. 當程式碼很長的時候，為方便使用者閱讀，提供了書籤功能。下面說明如何使用此功能。
8. 將游標移至編輯區中想要插入書籤的位置，按右鍵，選擇 [建立 / 取消書籤]。編輯區左邊將會看到一個代表書籤的藍色小方塊。

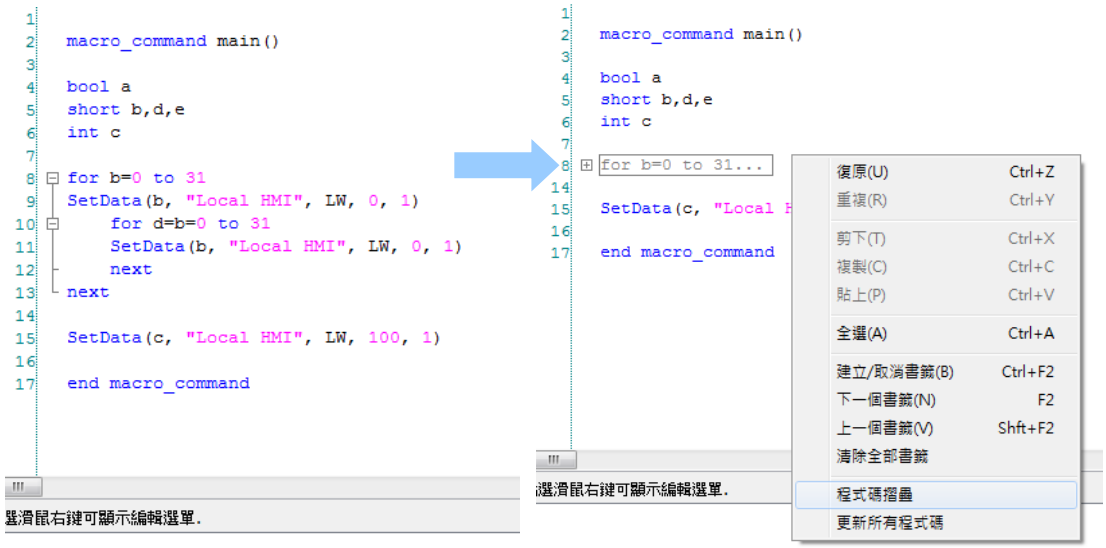
9. 若游標所在位置已存在書籤，選擇 [建立 / 取消書籤] 可將其關閉，反之，選擇 [建立 / 取消書籤] 可將其開啟。
10. 右鍵選擇 [下一個書籤] 游標將會移至下一個書籤所在位置。選擇 [上一個書籤] 游標將會移至上一個書籤所在位置。



11. 選擇 [清除全部書籤] 將關閉所有書籤。
12. 巨集指令編輯器提供程式碼折疊功能，方便使用者瀏覽程式碼。所謂程式碼摺疊，是指編輯器可將屬於同一區塊的程式碼隱藏起來，被隱藏起來的程式碼在編輯區裡會顯示 ⊞ ... 成。編輯區左側會顯示樹狀圖，使用者可按下 ⊟ 隱藏程式區塊，按下 ⊞ 展開程式區塊。如下圖所示：



13. 右鍵選擇 [程式碼折疊] 可展開所有程式碼區塊。



14. 有時候程式碼區塊可能會誤判。這種誤判起因於編輯器沒有辦法區分當前輸入的關鍵字是否存在於注釋中。例如下圖。使用者可以從右鍵選單選擇 [更新所有程式碼] 來更正這個錯誤。

15. 包圍在特定關鍵字內的程式碼稱為一程式碼區塊。內定的程式碼區塊如下列：

- 子函數： sub – end sub
- 迴圈語句：
  - i. for – next
  - ii. while – wend
- 邏輯運算語句：
  - i. if – end if
- 多重判斷語句： select case – end select

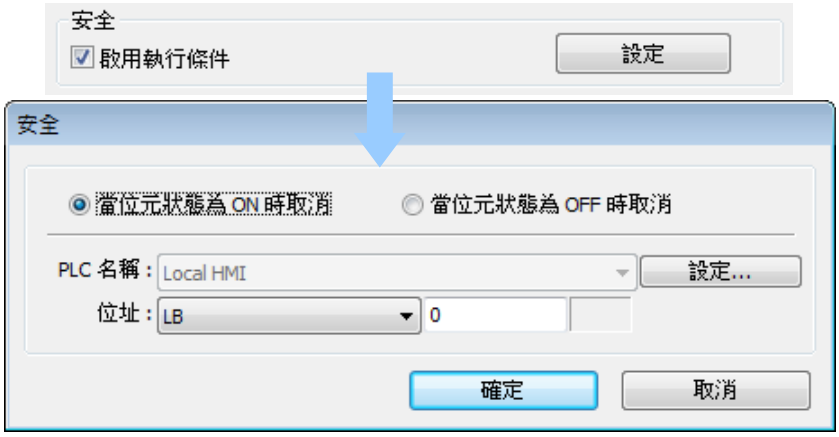
16. 巨集指令編輯窗口非壟斷屬性，開啟巨集編輯窗口後，可繼續回到主畫面同時進行編輯，也可直接在巨集視窗中進行連線/離線模擬。

17. 巨集編輯器提供尋找 / 替換文字的功能。

18. 使用者勾選 [週期執行] 時，會週期性的觸發此巨集。

19. 使用者勾選 [安全] -> [啟用執行條件] -> [設定] 後,可以進行安全設定：

- 當位元狀態 ON 時取消:當位元狀態 ON 時禁止執行此巨集。
- 當位元狀態 OFF 時取消: 當位元狀態 OFF 時禁止執行此巨集。



20. 使用者勾選 [當 HMI 啟動時即執行一次] 時，在 HMI 啟動時會自動執行巨集一次。

18.3. 巨集指令的結構

巨集指令是由各種語句組成的。這些語句包含常數、變數和各種運算符號。這些語句放置在特定的順序位置以便執行後達到一個希望的執行結果。

巨集指令的結構一般為以下格式：

```
總體變數聲明----------------------------------------------------------------可選

Sub Function Block Declarations (子函數聲明)----------------------可選
   局部變數聲明
End Sub (結束子函數)

macro_command main() [主函數]------------------------------------ 必須
   局部變數聲明
   [各式語句]
end macro_command     [結束主函數]------------------------------ 必須
```

一個巨集指令必須有一個且只有一個主函數，用來開始巨集指令的執行。格式為：

macro_command 函數名稱()

end macro_command

變數聲明必須放在巨集指令語句的前面，否則如果語句放置在變數聲明的前面，將會造成巨集指令無法編譯通過。

局部變數一般用在巨集指令主函數或者自定義的子函數中。它的合法性只在指定的函數中有效。

總體變數一般是定義在所有巨集指令函數的前面，且它在整個巨集指令中均具有有效性。當局部變數和總體變數被定義為相同的名稱時，只有局部變數有效。

下面就是一個簡單的巨集指令，其中就包含了變數聲明和函數呼叫。雙斜線 "//" 代表程式註解，在它後面的文字不會被執行。

```
macro_command main()
 short pressure = 10                                    //   局部變數聲明
 SetData(pressure, "Allen-Bradley DF1", N7, 0, 1)       //   函數呼叫
end macro_command
```

18.4. 巨集指令的語法

18.4.1. 常數和變數

18.4.1.1. 常數

常數是一個可以被各式語句直接使用的固定的資料。有如下格式：

常數類型	使用說明	舉例
十進位整數		345, -234, 0, 23456
十六進位數	必須以 0x 開頭	0x3b, 0xffff, 0x237
字元型	字元必須使用單引號；字串使用雙引號，雙引號內如果要顯示"請使用\"表示。	'a', "data", "函數名稱"
布林型		true, false

下面即為一個簡單的常數使用的範例。

```
macro_command main()
 short A, B     //   聲明 A 和 B 為短整型變數
 A = 1234
 B = 0x12       //   1234 和 0x12 即為常數
end macro_command
```

18.4.1.2. 變數

變數是一個代表著各種資料的名稱。在巨集指令中，這些資料可以隨著巨集指令語句執行的結果改變而改變。

變數的命名規則

- 必須以英文字母開頭
- 變數名稱長度不超過 32 個字元
- 系統保留暫存器名稱不能作為變數名稱。

下面為 8 種不同的變數類型，前 5 種為有號數值型態，後 3 種為無號數值型態：

變數類型	描述	範圍
bool 布林型	1 bit (一個位)	0, 1
char 字元型	8 bits (一個位元組)	+127 ~ -128
short 短整型	16 bits (一個字組)	+32767 ~ -32768
int 雙整型	32 bits (雙字組)	+2147483647 ~ -2147483648
float 浮點型	32 bits (雙字組)	
unsigned char 字元型	8 bits (一個位元組)	0 ~ 255
unsigned short 短整型	16 bits (一個字組)	0 ~ 65535
unsigned int 雙整型	32 bits (雙字組)	0 ~ 4,294,967,295
long 長整數	64 bits (四個字組) (只支援於 cMT / cMT X)	+281474976710655 ~ -281474976710655
unsigned long 長整數	64 bits (四個字組) (只支援於 cMT / cMT X)	0 ~ 281474976710655
double 雙精度浮點數	64 bits (四個字組) (只支援於 cMT / cMT X)	

變數聲明

變數必須在使用前聲明。所以，在巨集指令，所有的變數都必須在語句使用前都被聲明完成。聲明變數時，先定義變數的類型，後面再跟著變數名稱。

如下範例:

```
int             a
short    b, switch
float           pressure
unsigned short     c
```

陣列聲明

巨集指令支援一維陣列 (下標從 0 開始)。聲明陣列變數時，先定義陣列變數的類型，變數名稱，接著就是該陣列變數的個數，變數個數必須放置在“[]”符號中。陣列變數的長度為 1 ~ 4096。一個巨集指令中最多只支援 4096 個變數。

如下範例:

```
int             a[10]
short    b[20], switch[30]
float           pressure[15]
```

陣列的下標最小為 0，最大下標為(陣列的長度-1)

如下範例:

char data[100] // 陣列變數的長度是 100

所以: 最小的陣列為 “data[0]”，最大的陣列為 “data[99]”，即 100 – 1 = 99。

變數和陣列初始化

有兩種方法可以讓變數初始化：

- 使用語句中的賦值語句 (=)

如下範例:

```
int a
float b[3]
a = 10
b[0] = 1
```

- 聲明變數時直接賦值

```
char a = ‘5’, b = 9
```

陣列變數的聲明是一個特殊的情況。一個完整的陣列被初始化時，可以在陣列變數聲明時，將資料放置在波形括弧**“{}”**裡面，各資料使用逗號分開。

如下所示:

```
float data[4] = {11, 22, 33, 44}  // 這樣 data[0] = 11, data[1] = 22….
```

18.4.2. 運算符號

運算符通常被用來指定資料是如何被操作和運算，如下：(在任何一個語句中，運算符左邊的變數結果均依據運算符右邊的條件而獲得。)

運算符號	描述	舉例
=	賦值運算符號	pressure = 10

數學運算符號	描述	舉例
+	加	A = B + C
-	減	A = B – C
*	乘	A = B * C
/	除	A = B / C
% 或 mod	求餘 (返回剩餘數)	A = B % 5 或 A = B mod 5

注意：

由於整數的預設變數類型為整數型，因此在使用除法時，若除數與被除數皆是整數，且計算出的結果含有小數點，則小數點會自動被捨去。若要避免小數點被捨去，在除數或被除數加上.0 即可將數值的變數類型轉為浮點型。

範例：

A = 3 / 2 = 1 » 3 和 2 皆是整數型，因此運算結果也是整數型。

B = 3 / 2.0 = 1.5 » 3 是整數型，2.0 是浮點型，因此運算結果是浮點型。

C = 3.0 / 2 = 1.5 » 3.0 是浮點型，2 是整數型，因此運算結果是浮點型。

比較運算符號	描述	舉例
<	小於	if A < 10 then B = 5
<=	小於或者等於	if A <= 10 then B = 5
>	大於	if A > 10 then B = 5
>=	大於或者等於	if A >= 10 then B = 5
==	等於	if A == 10 then B = 5
<>	不等於	if A <> 10 then B = 5

邏輯運算符號	描述	舉例
And	與	if A < 10 and B > 5 then C = 10
Or	或	if A >= 10 or B > 5 then C = 10
Xor	異或	if A xor 256 then B = 5
Not	非	if not A then B = 5

移位元和位元運算符號通常被用來操作字元型變數、短整型變數和雙整型變數的位元。在一個語句中，這些運算符號的優先權是在從該語句的左邊到右邊依此執行的。即在語句中左邊位置的優先執行，依次從左到右執行。

移位運算符號	描述	舉例
<<	往左移動指定的位數	A = B << 8
>>	往右移動指定的位數	A = B >> 8

位運算符號	描述	舉例
&	位與運算	A = B & 0xf
|	位或運算	A = B | C
^	位異或運算	A = B ^ C
~	位取反運算	A = ~B

所有運算符號的優先權

上述所有運算符號的優先權從高到低詳細如下所述：

1. 位於圓括號裡面的運算符號最優先
2. 數學運算符號
3. 移位和位元運算符號
4. 比較運算符號
5. 邏輯運算符號
6. 賦值運算符號

關鍵字

下面的關鍵字為巨集指令保留使用。這些均不能用來作為變數名稱、陣列名稱或者函數名稱等。

+, -, *, /, %, >=, >, <=, <, <>, ==, and, or, xor, not, <<, >>,=, &, |, ^, ~

exit, macro_command, for, to, down, step, next, return, bool, short, int, char, float, void, if, then, else, break, continue, set, sub, end, while, wend, true, false

SQRT, CUBERT, LOG, LOG10, SIN, COS, TAN, COT, SEC, CSC, ASIN, ACOS, ATAN, BIN2BCD, BCD2BIN, DATE2ASCII, DATE2DEC, DEC2ASCII, FLOAT2ASCII, HEX2ASCII, DOUBLE2ASCII, ASCII2DEC, ASCII2FLOAT, ASCII2HEX, ASCII2DOUBLE, FILL, RAND, DELAY, SWAPB, SWAPW, LOBYTE, HIBYTE, LOWORD, HIWORD, GETBIT, SETBITON, SETBITOFF, INVBIT, ADDSUM, XORSUM, CRC, CRC8, CRC16_CCITT, CRC16_CCITT_FALSE, CRC16_X25, CRC16_XMODEM, INPORT, OUTPORT, POW, GetCnvTagArrayIndex , GetError, GetData, GetDataEx, SetData, SetDataEx, SetRTS, GetCTS, Beep, SYNC_TRIG_MACRO, ASYNC_TRIG_MACRO, TRACE, FindDataSamplingDate, FindDataSamplingIndex, FindEventLogDate, FindEventLogIndex

StringGet, StringGetEx, StringSet, StringSetEx, StringCopy, StringMid, StringMD5, StringDecAsc2Bin, StringBin2DecAsc, StringDecAsc2Float, StringFloat2DecAsc, StringHexAsc2Bin, StringBin2HexAsc, StringLength, StringCat, StringCompare, StringCompareNoCase, StringFind, StringReverseFind, StringFindOneOf, StringIncluding, StringExcluding, StringToUpper, StringToLower, StringToReverse, StringTrimLeft, StringTrimRight, StringInsert, String2Unicode, Unicode2Utf8, UnicodeCat, UnicodeCompare, UnicodeCopy, UnicodeExcluding, Uft82Unicode

18.5. 語句

18.5.1. 定義語句

這個定義語句包含了變數和陣列的聲明。正式的格式如下：

類型 名稱

定義一個變數的名稱為"名稱"且類型為"類型"。

舉例:

```
int A        //定義了變數 A 為雙整型格式
```

類型 陣列名稱[陣列長度]

定義一個陣列變數為"名稱"，大小為"陣列長度"且類型為"類型"時。

舉例:

```
int B[10]    //定義了一維陣列變數 B 的長度為 10，類型為雙整型
```

18.5.2. 賦值語句

賦值語句使用賦值運算符號將賦值運算符號右邊運算式運算的結果放置到運算符號左邊的變數中。一個運算式是由變數、常數和各種運算符號組成，執行後產生一個新的資料。

類型	運算式

舉例:

```
A = 2        //這樣變數 A 就被賦值為 2
```

18.5.3. 邏輯運算語句

邏輯運算語句是根據邏輯 (布林) 運算式的結果來執行相應的動作。它的語句如下所示：

單行格式

```
If <Condition> then
   [Statements]
else
   [Statements]
end if
```

舉例:

```
if a = = 2 then
        b = 1
else
        b = 2
end if
```

區塊格式

```
If <Condition> then
   [Statements]
else if <Condition-n> then
[Statements]
else
   [Statements]
end if
```

舉例:

```
if a == 2 then
        b = 1
else if a = = 3 then
        b = 2
else
        b = 3
end if
```

語法描述

if	必須用在該語句的開始部分。
<Condition>	必要條件。 這是一個控制語句。當 <Condition> 為 0 時，即為“FALES”，(條件為假)；當 <Condition> 為非 0 時，即為“True”(條件為真)。
then	當 <Condition> 執行為 “TRUE” (真) 時，必須放置在需要執行的語句之前。
[Statements]	在區塊形式中是可選擇的參數，在單行形式中，且沒有 else 子句時，為必要參數，該語句在 <Condition> 為真時執行。
else if	可選，一條或多條語句，在相對應的 <Condition – n> 為 true 時執行。
<Condition-n>	可選，解釋同 Condition
else	可選，在上述 Condition 和 Condition – n 都不為 true 時執行。

end if	必須。在一個 if-then 語句中使用這個來結束 if-then 語句。

18.5.4. 多重判斷語句

Select-case 可用來處理多重判斷的敘述，其功能類似 if-else 語句。根據所指定變數的值，分別對應到符合該值的 case，並執行 case 下面的敘述，直到遇到 break 敘述時，才跳到結束符號 end select 處。語法結構如下：

沒有預設 case 的形式：

```
Select Case [variable]
Case [value]
    [Statements]
    break
end Select
```

舉例:

```
Select Case A
    Case 1
        b=1
    break
end Select
```

有預設 case 的形式：

```
Select Case [variable]
Case [value]
    [Statements]
    break
    Case else
    [Statements]
    break
end Select
```

舉例:

```
Select Case A
    Case 1
        b=1
    break
    Case else
        b=0
```

```
        break
    end Select
```

多個不同 case 對應到相同區塊：

```
Select Case [variable]
Case [value1]
    [Statements]
Case [value2]
    [Statements]
    break

end Select
```

舉例:

```
    Select Case A
        Case 1
        break
        Case 2
            b=2
        break
        Case 3
            b=3
        break
    end Select
```

語法描述

Select Case	必須用在該語句的開始部分。
[variable]	必要條件。此變數將會與每一個 case 做比較。
Case else	可選。代表預設 case。當 [variable] 的值不符合任何一個 case 時，將會執行此敘述下面的區塊。在沒有預設 case 的情況，當 [variable] 的值不符合任何一個 case 時，將不會做任何動作而直接跳出 select 控制結構。
break	可選。跳到某一個 case 下面執行時，將一句一句執行 case 語句下面的敘述直到遇到 break 命令才結束，並跳到 end select 敘述。當 case 敘述下面沒有任何 break 命令時，流程將不斷往下執行，直到遇到 end select 敘述，才結束並跳出 select 控制結構。
end Select	select-case 語句的結束標誌。

18.5.5. 迴圈語句

迴圈語句依據迴圈條件來反復的執行一個任務。迴圈語句有兩種表達方式。

18.5.5.1. for next 語句

For-next 語句通常用來執行次數固定的迴圈任務。一個變數用作為任務執行次數的計數器和結束迴圈任務執行的條件。這個變數為固定執行的次數。語法結構如下：

```
for [Counter] = <StartValue> to <EndValue> [step <StepValue>]
    [Statements]
next [Counter]
```

或者

```
for [Counter] = <StartValue> down <EndValue> [step <StepValue>]
    [Statements]
next [Counter]
```

舉例:

```
for a = 0 to 10 step 2
    b = a
next a
```

語法描述

for	必須用在該語句的開始部分。
[Counter]	必要，迴圈計數器的數值變數，該變數的結果用來計數迴圈的次數。
<StartValue>	必要，Counter 的初值。
to/down	必要。用來決定步長是遞增還是遞減。 “to” 以 <StepValue> 為步長遞增 <Counter> “down” 以 <StepValue> 為步長遞減 <Counter>
<EndValue>	必要，Counter 的終值、測試點。當 <Counter> 大於該值時，巨集指令將結束這個迴圈任務。
step	可選，指定 <Step Value> 的步長，指定為 1 以外的數值。
[StepValue]	可選，Counter 的步長，只能是數值，如果沒有指定，則預設為 1。
[Statements]	可選，for 和 next 之間的語句區塊，該語句區塊將執行所指定的次數。
next	必須的。
[Counter]	可選。

18.5.5.2. while-wend 語句

While-wend 語句是用來執行不確定次數的迴圈任務。設置一個變數用來判斷結束迴圈的條件。當條件為“True”時，該語句將一直迴圈執行直到條件變為 “False”。語法結構如下：

```
while <Condition>
        [Statements]
wend
```

舉例:

```
while a < 10
        a = a + 10
wend
```

語法描述

while	必須用在該語句的開始部分。
continue	必要條件。 這是一個控制語句。當為 “True” 時，開始執行迴圈命令，當為 “False” 時，結束執行迴圈命令。
wend	While-wend 語句的結束標誌。

18.5.5.3. 其他控制命令

break	用在 for-next 和 while-wend 語句中。當遇到此語句時，立即跳到語句的結束部分。
continue	用在 for-next 和 while-wend 語句中。當遇到此語句時，立即結束當前迴圈命令而開始執行下一個迴圈命令。
return	可用在自訂 function 的回傳值敘述。寫在主函數裡面時，用來強制跳出主函數。

18.6. 子函數

使用子函數可以有效的減少迴圈命令的代碼，子函數必須在使用前被定義，且可以使用任何變數和語句類型。在主函數中，將子函數的參數放置在子函數名稱後面的圓括號中，即可調用子函數。子函數被執行後，將執行後的結果返回到主函數需要的賦值語句或者條件中。定義子函數時，不一定要有返回值，且參數部分可以為空。在主函數中調用子函數時，調用方式應符合其定義。語法結構如下：

有返回值的子函數語法：

```
sub type <函數名稱> [(parameters)]
        Local variable declarations
        [Statements]
        [return [value]]
end sub
```

舉例:

```
sub int Add(int x, int y)
        int result
        result = x +y
        return result
    end sub

macro_command main()
        int    a = 10, b = 20, sum
        sum = Add(a, b)
    end macro_command
```

或:

```
sub int Add()
        int result, x=10, y=20
        result = x +y
        return result
    end sub

    macro_command main()
        int    sum
        sum = Add()
    end macro_command
```

沒有返回值的子函數語法：

```
sub <函數名稱> [(parameters)]
    Local variable declarations
    [Statements]
end sub
```

舉例:

```
sub Add(int x, int y)
        int result
        result = x +y
    end sub

    macro_command main()
        int    a = 10, b = 20
        Add(a, b)
    end macro_command
```

或:

```
sub Add()
        int result, x=10, y=20
        result = x +y
    end sub

    macro_command main()
        Add()
    end macro_command
```

語法描述

sub	必須用在該子函數的開始部分。
type	可選。用來定義子函數執行後返回的資料類型。子函數也可以不回傳任何值。
(parameters)	可選。這些參數保留了從主函數傳入的數值。這些被傳入的參數必須使用與在參數變數聲明的類型一致。 舉例： sub int MyFunction(int x, int y). x 和 y 必須為從主函數中傳過來的雙整型資料格式的資料。調用此子函數的語句格式大致為這樣：ret = MyFunction(456, pressure)，其中 pressure 需為雙整型資料格式方符合子函數參數變數的聲明。 請注意調用語句的參數部分可以是常數也可以是變數。當執行這個子函數後，一個雙整型資料將會返回給變數 “ret”。
Local variable declaration	除了被傳遞的參數之外，子函數中使用的變數必須事先聲明。在上面的“舉例”中，X 和 Y 就是子函數可以使用的變數。總體變數也可以用在子函數中。
[Statements]	需要執行的語句。
[return [value]]	可選。用來將執行的結果返回給調用語句。這個結果可以是一個常數或者變數。返回後同時也結束了子函數的執行。子函數也可以不回傳任何值，但是當 type 部分有定義時，則必須加上此 return 敘述。
end sub	必須的。用來結束子函數。

18.7. 內置函數功能

EasyBuilder Pro 軟體巨集指令中本身提供了一些內建的函數用來從設備獲取資料和傳輸資料到設備、資料處理和數學運算等。

18.7.1. 函數一覽表

點選以下表格中的函數名稱可檢視其相關詳細資訊。

函數名稱	簡述
設備函數	
GetData	讀取設備的資料。
GetDataEx	讀取設備的資料，不等待設備回應，逕自往下執行。
GetError	取得錯誤碼。
SetData	將數據寫到設備中。
SetDataEx	將數據寫到設備中，不等待設備回應，逕自往下執行。
Free Protocol 函數	
GetCTS	偵測 RS-232 之 CTS 訊號。
INPORT	從串列埠/乙太網口讀取數據。
INPORT2	從串列埠/乙太網口讀取數據並在讀取完畢後，等待指定的時間。
INPORT3	從串列埠/乙太網口讀取數據讀取指定的長度。
INPORT4	從串列埠/乙太網口讀取數據，直到讀取到特定結尾符號後即停止讀取。
OUTPORT	將數據從串列埠/乙太網口傳送給設備或者控制器。
PURGE	清空 COM port 的輸出入緩衝區。
SetRTS	拉高或拉低 RS-232 之 RTS 訊號。
巨集控制函數	
ASYNC_TRIG_MACRO	觸發指定巨集指令並同時執行。
SYNC_TRIG_MACRO	觸發指定巨集指令執行，並等待其執行完畢後，才往下執行。
DELAY	讓巨集指令暫停執行指定時間。
資料操作函數	
FILL	將數據放置到陣列中。
SWAPB	將一個 16 bit 字的高低位元組顛倒。
SWAPW	將一個 32 bit 雙整型資料的高位字組和低位字組顛倒。
LOBYTE	讀取一個 16 bit 資料的低位元組。
HIBYTE	讀取一個 16 bit 資料的高位元組。
LOWORD	讀取一個 32 bit 資料的低位字組。
HIWORD	讀取一個 32 bit 資料的高位字組。
INVBIT	將資料的指定的位元狀態相反。
SETBITON	將資料的指定位元設置為 1。
SETBITOFF	將資料的指定位元設置為 0。
GETBIT	讀取資料的指定位元的狀態。
資料轉換函數	
ASCII2DEC	將字元型 ASCII 資料轉換為十進位格式的資料。

ASCII2FLOAT	將字元型 ASCII 資料轉換為浮點數格式的資料。
ASCII2HEX	將 ASCII 字元型資料轉換為十六進位的資料。
ASCII2DOUBLE	將 ASCII 字元型資料 (source) 轉換為雙精度浮點數的資料。 此函數僅支援 cMT/cMT X 系列。
BIN2BCD	將 BIN 格式的資料轉換為 BCD 格式的資料。
BCD2BIN	將 BCD 格式的資料轉換為 BIN 格式的資料。
DATE2ASCII	將現在日期資料轉換為 ASCII 格式的資料。
DATE2DEC	將現在日期資料轉換為十進位格式的資料。
DEC2ASCII	將十進位的資料轉換為 ASCII 格式的資料。
FLOAT2ASCII	將浮點數格式資料轉換為 ASCII 格式的資料。
HEX2ASCII	將十六進位格式資料轉換為 ASCII 格式的資料。
DOUBLE2ASCII	雙精度浮點數格式資料 (source) 轉換為 ASCII 格式的資料。 此函數僅支援 cMT/cMT X 系列。
StringDecAsc2Bin	將十進位字串轉換成整數。
StringBin2DecAsc	將整數轉換成十進位字串。
StringDecAsc2Float	將十進位字串轉換成浮點數。
StringFloat2DecAsc	將浮點數轉換成十進位字串。
StringHexAsc2Bin	將十六進位字串轉換成整數。
StringBin2HexAsc	將整數轉換成十六進位字串。
字串處理函數	
String2Unicode	將字串轉換為 Unicode 字串。
StringCat	連接兩字串。
StringCompare	比較兩字串的內容是否相等。此函數將大小寫視為不同。
StringCompareNoCase	比較兩字串的內容是否相等。此函數將大小寫視為相同。
StringCopy	複製字串。
StringExcluding	從某字串第一個字元開始提取不包含特定字元的一段字串。
StringFind	在某個字串中尋找另一個字串。
StringFindOneOf	尋找某字串中任一個字元在另一個字串中第一次出現的位置。
StringGet	讀取設備的字元陣列資料。
StringGetEx	讀取設備的字元陣列資料，不等待設備回應，逕自往下執行。
StringIncluding	從某字串第一個字元開始提取包含特定字元的一段字串。
StringInsert	將字串插入到另一字串中的特定位置。
StringLength	取得字串的長度。

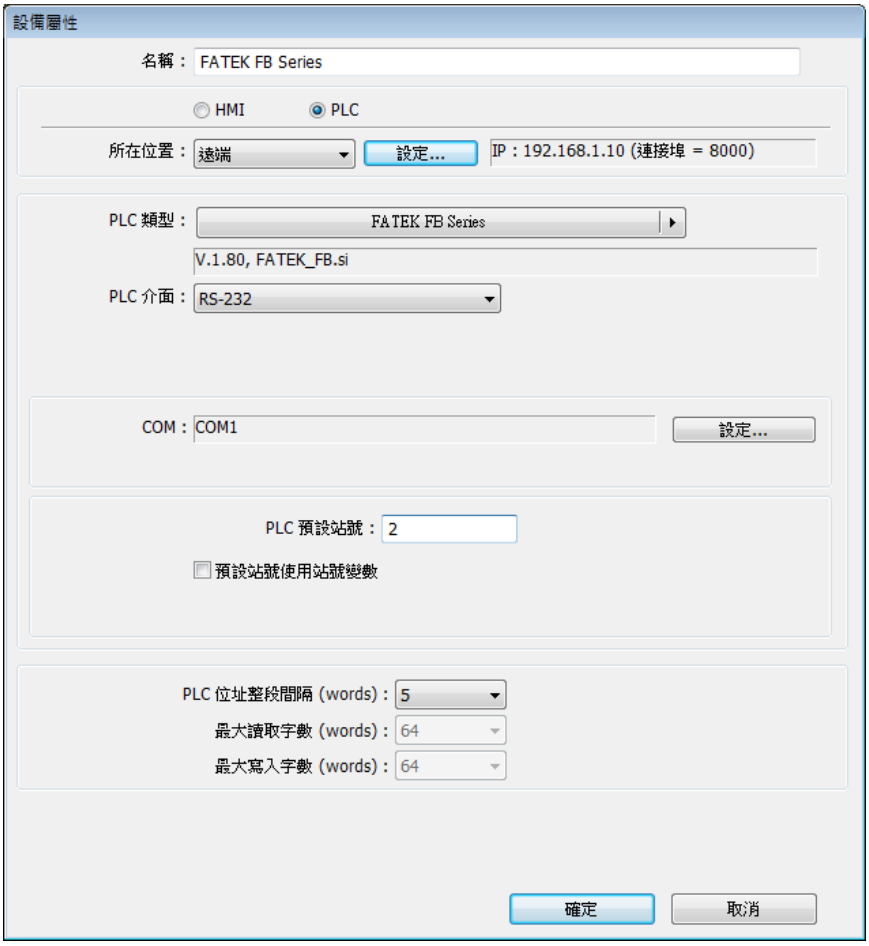
StringMD5	產生一組透過 MD5 訊息摘要演算法的字串。
StringMid	利用此函數可將一個字串中的某一段子字串提取出來。
StringReverseFind	在某個字串中尋找另一個字串，從後面開始找。
StringSet	將字元陣列資料寫到設備中。
StringSetEx	將字元陣列資料寫到設備中，不等待設備回應，逕自往下執行。
StringToUpper	將字串的字元全部轉換成大寫。
StringToLower	將字串的字元全部轉換成小寫。
StringToReverse	將字串反轉。
StringTrimLeft	裁剪字串開頭的特定字元。
StringTrimRight	裁剪字串尾端的特定字元。
Unicode2Utf8	將 Unicode 字串轉換為 UTF8 格式字串。
UnicodeCat	連接兩 Unicode 字串。
UnicodeCompare	比較兩 Unicode 字串的內容是否相等。大小寫視為不同。
UnicodeCopy	複製 Unicode 字串。
UnicodeExcluding	從某 Unicode 字串第一個字元開始提取不包含特定字元的一段字串。
UnicodeLength	取得 Unicode 字串長度。
Utf82Unicode	將 Utf8 格式字串轉換為 Unicode 字串。
數學運算函數	
SQRT	開平方根。
CUBERT	開三次方根。
POW	指數計算。
SIN	三角函數的正弦計算。
COS	三角函數的餘弦計算。
TAN	三角函數的正切計算。
COT	三角函數的餘切計算。
SEC	三角函數的正割計算。
CSC	三角函數的餘割計算。
ASIN	反三角函數中反正弦計算。
ACOS	反三角函數中反餘弦計算。
ATAN	反三角函數中反正切計算。
LOG	取得自然對數。
LOG10	取得以 10 為基底的對數。
RAND	產生一個隨機數。
CEIL	向上取得整數。
FLOOR	向下取得整數。

ROUND	取得最接近整數。
統計函數	
AVERAGE	從陣列中計算平均值。
HARMEAN	從陣列中計算調和平均值。
MAX	從陣列中取最大值。
MEDIAN	從陣列中取中位數。
MIN	從陣列中取最小值。
STDEVP	從陣列中計算標準差。
STDEVS	從陣列中計算樣本標準差。
配方資料庫函數	
RecipeGetData	取得配方中的資料。
RecipeQuery	查詢配方中的資料。
RecipeQueryGetData	取得 RecipeQuery 的查詢結果。
RecipeQueryGetRecordID	取得 RecipeQuery 查詢結果的資料列編號。
RecipeSetData	將資料寫入配方資料庫中。
RecipeTransactionBegin	開啟批次寫入配方的功能。需與 RecipeTransactionCommit 或 RecipeTransactionRollback 搭配使用。 此函數僅支援 cMT/cMT X 系列。
RecipeTransactionCommit	執行批次寫入配方。 此函數僅支援 cMT/cMT X 系列。
RecipeTransactionRollback	取消執行批次寫入配方。 此函數僅支援 cMT/cMT X 系列。
資料取樣/事件記錄函數	
FindDataSamplingDate	查詢資料取樣檔的日期。
FindDataSamplingIndex	查詢資料取樣檔的檔案索引值。
FindEventLogDate	查詢事件登錄檔的日期。
FindEventLogIndex	查詢事件登錄檔的檔案索引值。
校驗函數	
ADDSUM	計算一維陣列累加的 checksum。
XORSUM	計算一維陣列異或運算的 checksum。
BCC	等同 XORSUM 函數。
CRC	計算一維陣列的資料進 16-bit CRC 計算，以獲得 checksum (校驗和)。
CRC8	計算一維陣列的資料進 8-bit CRC 計算，以獲得 checksum (校驗和)。
CRC16_CCITT	計算一維陣列的資料進 CRC16_CCITT 計算，以獲得 checksum (校驗和)。
CRC16_CCITT_FALSE	計算一維陣列的資料進 CRC16_CCITT_FALSE 計算，以獲得 checksum (校驗和)。
CRC16_X25	計算一維陣列的資料進 CRC16_X25計算，以獲得 checksum

	(校驗和)。
CRC16_XMODEM	計算一維陣列的資料進 CRC16_XMODEM 計算，以獲得 checksum (校驗和)。
其他函數	
Beep	發出系統警示音。
Buzzer	開啟 / 關閉蜂鳴器。
TRACE	監視並列印巨集中的變數內容。
GetCnvTagArrayIndex	使用者定義標籤使用【讀取轉換】並為陣列時，在巨集副函數中，使用此函數可取得陣列的索引。

18.7.2. 設備

函數名稱	GetData
語法	GetData(read_data[start], device_name, device_type, address_offset, data_count) or GetData(read_data, device_name, device_type, address_offset, 1)
描述	獲取設備的資料，若讀取失敗，後續的程式碼將不會繼續執行。資料是存儲在 read_data[start]~read_data[start+data_count-1] 這些一維陣列變數中。 data_count 是設定的讀取數據的個數。一般來說，read_data 是一個一維陣列，但是如果 data_count 是 1，read_data 可以是一個一維陣列，也可以是一個普通的變數。下面是兩種從設備中讀取一個字的方法。 macro_command main() short read_data_1[2], read_data_2 GetData(read_data_1[0], "FATEK FB Series", RT, 5, 1) GetData(read_data_2, "FATEK FB Series", RT, 5, 1) end macro_command 此處的 device_name，即為在 “系統參數” 中建立設備類型時，設定的“設備名稱”。在此，設備名稱被設定為 “FATEK FB Series”，如下圖所示。  device_type 是設備類型和設備中資料的編碼方式。例如：如果 device_type 是 LW_BIN，那麼讀取的設備類型為 LW，資料編碼方式為 BIN。如果使用 BIN 編碼方式，"_BIN" 可以忽略。 如果 device_type 是 LW_BCD，表示設備類型 LW，資料的編碼方式為 BCD 格式。 address_offset 是設備中的地址偏移量。 例如，GetData(read_data_1[0], "FATEK FB Series", RT, 5, 1) 代表讀取的設備位址偏移量為 5。 如果 address_offset 使用格式為 "N#AAAAA"，N 表示設備的站號，AAAAA 表示位址偏移量。此情況一般使用在同一個串列埠上連接有多台設備或者控制器的情況下。例如：GetData(read_data_1[0], "FATEK FB Series", RT, 2#5, 1) 表示讀取站號為 2 的設備的資料。如果 GetData() 使用 “系統參數 / 設備列表” 中設定的默認的站號，在此可以不填這個站號。



從設備中讀取的資料個數，根據 read_data 變數的類型和 data_count 的值來決定的。如下表所示：

read_data的類型	data_count的值	讀取16位元數據的個數
char (8-bit)	1	1
char (8-bit)	2	1
bool (8-bit)	1	1
bol (8-bit)	2	1
short (16-bit)	1	1
short (16-bit)	2	2
int (32-bit)	1	2
int (32-bit)	2	4
float (32-bit)	1	2
float(3-bit)	2	4

當 Getdata() 函數讀取 32 位元的資料類型 (int 或者float型) 時，此函數會自動的轉換這個資料。例如：

	macro_command main()
	float f
	GetData(f, "MODBUS", 6x, 2, 1) // f 中將會是浮點型的數據
	end macro_command

舉例	macro_command main()
	bool a
	bool b
	bool b_array[30]
	char c
	char c_array[20]
	short s
	short s_array[50]
	int i
	int i_array[10]
	float f
	float f_array[15]
	
	// 讀取 LB2 的狀態到變數 a 中
	GetData(a, "Local HMI", LB, 2, 1)
	
	// 讀取 LB0 ~ LB29共 30 個狀態，到變數 b_array[0] ~ b_array[29] 中
	GetData(b_array[0], "Local HMI", LB, 0, 30)
	
	// 讀取 LW-0 的低位元組到變數 c 中
	// 注意char為1 Byte，LW一個定址佔2 Byte(1字組)，讀取字組暫存器的第一個位元組會讀取到該字組的低位元組
	// Ex: 假設LW-0的值為0x0201，則c會讀取到0x01
	GetData(c, "Local HMI", LW, 0, 1)
	
	// 讀取 LW-1 ~ LW-10的資料到c_array[0] 至 c_array[19] 中
	GetData(c_array[0], "Local HMI", LW, 0, 20)
	
	// 讀取 LW-2 的資料到變數 s 中
	GetData(s, "Local HMI", LW, 2, 1)
	
	// 讀取 LW-0 ~ LW-49 共 50 個字組到變數 s_array[0] 到 s_array[49] 中
	GetData(s_array[0], "Local HMI", LW, 0, 50)
	
	// 讀取兩個字 LW-6 ~ LW-7 到變數 e 中
	// Ex: 假設LW-6的值為0x0002, LW-7的值為0x0001, 則i會讀取到0x00010002(65538)
	// 注意一個 int 資料將佔據 2 個字組(32-bit)
	GetData(i, "Local HMI", LW, 6, 1)
	
	// 讀取 LW-0 ~ LW-19 共 20 個字組到變數 i_array[0] ~ i_array[9] 中 (共 10 個 int 型變數)，陣列 i_array[10] 的變數類型定義為 int。
	GetData(i_array[0], "Local HMI", LW, 0, 10)
	
	// 讀取 LW-10 ~ LW-11 到變數 f 中

	// 注意此時變數 f 的類型為 float
	GetData(f, "Local HMI", LW, 10, 1)
	
	// 讀取 LW-0 ~ LW-29 共 30 個字到變數 f_array[0] ~ f_array[14] 中 (共 15 個 float 型變數)，陣列 f_array[15] 的變數類型定義為 float。
	// 注意一個 float 資料將佔據 2 個字組(32-bit)
	GetData(f_array[0], "Local HMI", LW, 0, 15)
	
	end macro_command

函數名稱	GetDataEx
語法	GetDataEx (read_data[start], device_name, device_type, address_offset, data_count) or GetDataEx (read_data, device_name, device_type, address_offset, 1)
描述	獲取設備的資料，若讀取失敗，仍會繼續執行後續的程式碼。 read_data、device_name、device_type、address_offset 和 data_count的說明和 GetData 相同。

舉例	macro_command main()
	bool a
	bool b
	bool b_array[30]
	char c
	char c_array[20]
	short s
	short s_array[50]
	int i
	int i_array[10]
	float f
	float f_array[15]
	
	// 讀取 LB2 的狀態到變數 a 中
	GetDataEx(a, "Local HMI", LB, 2, 1)
	
	// 讀取 LB0 ~ LB29共 30 個狀態，到變數 b_array[0] ~ b_array[29] 中
	GetDataEx(b_array[0], "Local HMI", LB, 0, 30)
	
	// 讀取 LW-0 的低位元組到變數 c 中
	// 注意char為1 Byte，LW一個定址佔2 Byte(1字組)，讀取字組暫存器的第一個位元組會讀取到該字組的低位元組
	// Ex: 假設LW-0的值為0x0201，則c會讀取到0x01
	GetDataEx(c, "Local HMI", LW, 0, 1)
	
	// 讀取 LW-1 ~ LW-10的資料到c_array[0] 至 c_array[19] 中
	GetDataEx(c_array[0], "Local HMI", LW, 0, 20)
	
	// 讀取 LW-2 的資料到變數 s 中
	GetDataEx(s, "Local HMI", LW, 2, 1)

	// 讀取 LW-0 ~ LW-49 共 50 個字組到變數 s_array[0] 到 s_array[49] 中
	GetDataEx(s_array[0], "Local HMI", LW, 0, 50)
	
	// 讀取兩個字 LW-6 ~ LW-7 到變數 e 中
	// Ex: 假設LW-6的值為0x0002, LW-7的值為0x0001, 則i會讀取到0x00010002(65538)
	// 注意一個 int 資料將佔據 2 個字組(32-bit)
	GetDataEx(i, "Local HMI", LW, 6, 1)
	
	// 讀取 LW-0 ~ LW-19 共 20 個字組到變數 i_array[0] ~ i_array[9] 中 (共 10 個 int 型變數)，陣列 i_array[10] 的變數類型定義為 int。
	GetDataEx(i_array[0], "Local HMI", LW, 0, 10)
	
	// 讀取 LW-10 ~ LW-11 到變數 f 中
	// 注意此時變數 f 的類型為 float
	GetDataEx(f, "Local HMI", LW, 10, 1)
	
	// 讀取 LW-0 ~ LW-29 共 30 個字到變數 f_array[0] ~ f_array[14] 中 (共 15 個 float 型變數)，陣列 f_array[15] 的變數類型定義為 float。
	// 注意一個 float 資料將佔據 2 個字組(32-bit)
	GetDataEx(f_array[0], "Local HMI", LW, 0, 15)
	
	end macro_command

函數名稱	GetError
語法	GetError(err)
描述	取得錯誤碼。

舉例	macro_command main()
	short err
	char byData[10]
	
	GetDataEx(byData[0], "MODBUS RTU", 4x, 1, 10) // 讀取 10 byte資料
	
	// 當錯誤碼 (err) 為 0，表示 GetDataEx 成功執行
	GetErr(err) // 讀取錯誤碼，存入變數err
	
	end macro_command
	
	錯誤碼:
	0: 執行正常
	1: GetDataEx 錯誤
	2: SetDataEx 錯誤

函數名稱	SetData
語法	SetData(send_data[start], device_name, device_type, address_offset, data_count) or SetData(send_data, device_name, device_type, address_offset, 1)

描述

將數據寫到設備中，若寫入失敗，後續的程式碼將不會繼續執行。資料保存在 send_data[start]~send_data[start+data_count-1] 中。
data_count 是寫入到設備中資料的個數。一般來說，send_data 是一個陣列。但是如果 data_count 是 1，send_data 可以是一個陣列也可以是一個普通的變數。下面是寫一個資料到設備中的方法。

```
macro_command main()
        short send_data_1[2] = { 5, 6}, send_data_2 = 5
        SetData(send_data_1[0], "FATEK FB Series", RT, 5, 1)
        SetData(send_data_2,     "FATEK FB Series", RT, 5, 1)
end macro_command
```

device_name 詳見上面的說明，在此不在說明。
device_type 是設備類型和設備中資料的編碼方式。例如：如果 device_type 是 LW_BIN，那麼讀取的設備類型為 LW，資料編碼方式為 BIN。如果使用 BIN 編碼方式，"_BIN" 可以忽略。
如果 device_type 是 LW_BCD，表示設備類型 LW，資料的編碼方式為 BCD 格式。
address_offset 是設備中的地址偏移量。
例如，SetData (read_data_1[0], "FATEK FB Series", RT, 5, 1) 代表讀取的設備位址偏移量為 5。
如果 address_offset 使用格式為 "N#AAAAA"，N 表示設備的站號，AAAAA 表示位址偏移量。此情況一般使用在同一個串列埠上連接有多台設備或者控制器的情況下。例如：SetData(send_data_1[0], "FATEK FB Series", RT, 2#5, 1) 表示設定站號為 2 的設備的資料。如果 SetData()使用“系統參數 / 設備列表”中設定的默認的站號，在此可以不填這個站號。

設定到設備的資料個數，根據 sead_data 變數的類型和 data_count 的值來決定的。如下表所示：

sead_data 的類型	data_count 的值	設定 16 位元數據的個數
char (8-bit)	1	1
char (8-bit)	2	1
bool (8-bit)		1
bol (8-bit)	2	
short 16-bit)	1	1
shrt (16-bit)	2	2
int (32-it)	1	2
int (32-bit)	2	4
float (32-bit)	1	2
float (32-bit)	2	4

當 Setdata() 函數寫入 32 位元的資料類型 (int 或者 float 型) 到設備時，此函數會自動的轉換這個資料。例如：

```
macro_command main()
float f = 2.6
SetData(f, "MODBUS", 6x, 2, 1) //  在此將會設定一個浮點數到設備中
end macro_command
```

舉例

```
macro_command main()
int i
bool a = true
bool b[30]
short c = false
short d[50]
int e = 5
int f[10]

for i = 0 to 29
b[i] = true
next i

for i = 0 to 49
d[i] = i * 2
next i

for i = 0 to 9
f [i] = i * 3
next i

//  變數 a 的數值設定到 LB2 中
SetData(a, "Local HMI", LB, 2, 1)


// 設定 LB0 ~ LB29 共 30 個位的狀態
SetData(b[0], "Local HMI", LB, 0, 30)

// 將變數 c 的值設定到 LW-2 中
SetData(c, "Local HMI", LW, 2, 1)

// 設定 LW-0 ~ LW-49 共 50 個數據
SetData(d[0], "Local HMI", LW, 0, 50)

// 將變數 e 的值寫入到 LW-6 ~ LW-7 兩個暫存器中，注意變數 e 的類型為int。
SetData(e, "Local HMI", LW, 6, 1)

// 設定 LW-0 ~ LW-19 共 20 個字的數據
// 10 個雙整型數據等於 20 個 16 位元整型數 (1個字組)，因一個 int 資料將佔據 2 個字組
SetData(f[0], "Local HMI", LW, 0, 10)
```

	end macro_command

函數名稱	SetDataEx
語法	SetDataEx (send_data[start], device_name, device_type, address_offset, data_count) or SetDataEx (send_data, device_name, device_type, address_offset, 1)
描述	將數據寫到設備中，若寫入失敗，仍會繼續執行後續的程式碼。 send_data、device_name、device_type、address_offset和data_count的說明和 SetData 相同。

舉例	macro_command main()
	int i
	bool a = true
	bool b[30]
	short c = false
	short d[50]
	int e = 5
	int f[10]
	
	for i = 0 to 29
	b[i] = true
	next i
	
	for i = 0 to 49
	d[i] = i * 2
	next i
	
	for i = 0 to 9
	f [i] = i * 3
	next i
	
	// 將變數 a 的數值設定到 LB2 中
	SetDataEx(a, "Local HMI", LB, 2, 1)
	
	// 設定 LB0 ~ LB29 共 30 個位的狀態
	SetDataEx (b[0], "Local HMI", LB, 0, 30)
	
	// 將變數 c 的值設定到 LW-2 中
	SetDataEx (c, "Local HMI", LW, 2, 1)
	
	// 設定 LW-0 ~ LW-49 共 50 個數據
	SetDataEx (d[0], "Local HMI", LW, 0, 50)
	
	// 將變數 e 的值寫入到 LW-6 ~ LW-7 兩個暫存器中，注意變數 e 的類型為int。
	SetDataEx (e, "Local HMI", LW, 6, 1)

	// 設定 LW-0 ~ LW-19 共 20 個字的數據 // 10 個雙整型數據等於 20 個 16 位元整型數 (1個字組)，因一個 int 資料將佔據 2 個字組 SetDataEx (f[0], "Local HMI", LW, 0, 10) end macro_command

18.7.3. Free Protocol

函數名稱	GetCTS
語法	GetCTS(com_port, result)
描述	偵測 RS-232 之 CTS 訊號。 com_port 表示序列埠編號，支援 COM1。此參數可以為變數或常數。result 參數必須為變數。 此函數可偵測 CTS 電位值，當 CTS 在高電位時，result 將寫入 1，否則寫入 0。

舉例	macro_command main()
	char com_port=1
	char result
	
	GetCTS(com_port, result) // 偵測 COM1 之 CTS 電位值
	
	GetCTS (1, result) // 偵測 COM1 之 CTS 電位值
	
	end macro_command

函數名稱	INPORT
語法	INPORT(read_data[start], device_name, read_count, return_value)
描述	從串列埠或者乙太網口讀數據到人機界面上。這些資料保存在read-data[start]~read-data[start+read-count-1] 這個一維陣列中。 device_name是在“設備列表”中定義的“設備名稱”，而這個 device 必須選擇為 “Free Protocol” 這個設備類型。 read-count 是設定的需要讀取的命令的位元組長度，它可以是一個常數或變數。 如果這個函數能夠成功的從設備或者控制器中讀取到資料，則return_value將回傳讀取到的word長度。

舉例	下面就是一個使用 INPORT 函數讀取一個 MODBUS 設備保持暫存器資料的範例。
	
	// 讀取保持暫存器資料
	macro_command main()
	char command[32], response[32]
	short address, checksum
	short read_no, return_value, read_data[2]
	
	FILL(command[0], 0, 32) // 命令初始化

```
FILL(response[0], 0, 32)

command[0] = 0x1      // 站號
command[1] = 0x3      // 功能碼：讀取保持暫存器

address = 0
HIBYTE(address, command[2])
LOBYTE(address, command[3])

read_no = 2      // read 2 words (4x_1 and 4x_2)
HIBYTE(read_no, command[4])
LOBYTE(read_no, command[5])

CRC(command[0], checksum, 6)

LOBYTE(checksum, command[6])
HIBYTE(checksum, command[7])

// 使用 OUTPORT 函數將命令送出去
OUTPORT(command[0], "MODBUS RTU Device", 8)

// 使用 INPORT 函數讀取返回的命令
INPORT(response[0], "MODBUS RTU Device", 9, return_value)

if return_value > 0 then
   read_data[0] = response[4] + (response[3] << 8)      // data in 4x_1
   read_data[1] = response[6] + (response[5] << 8)      // data in 4x_2

   SetData(read_data[0], "Local HMI", LW, 100, 2)
end if

end macro_command
```

函數名稱	INPORT2
語法	INPORT2(response[start], device_name, receive_len, wait_time)
描述	從串列埠或者乙太網口讀數據到人機界面上。這些資料保存在 response 這個一維陣列中。device_name是在“設備列表”中定義的“設備名稱”，而這個 device 必須選擇為 “Free Protocol” 這個設備類型。 receive_len 存放所接收到的資料位元組長度，必須為變數。receive_len 的最大長度將不會超過 response 陣列宣告的大小。wait_time 表示等待時間 (單位是millisecond)，可以是一個常數或變數。當資料讀取完畢後，若在指定的等待時間內未再收到任何資料，此函數便結束執行並回傳結果。
舉例	macro_command main() short wResponse[6], receive_len, wait_time=20 INPORT2(wResponse[0], "Free Protocol", receive_len, wait_time)

	// wait_time 單位 : millisecond
	
	if receive_len > 0 then
	SetData(wResponse[0], "Local HMI", LW, 0, 6)
	// 把回應的資料寫入LW0
	end if
	
	end macro_command

函數名稱	INPORT3
語法	INPORT3(response[start], device_name, read_count, receive_len)
描述	從串列埠或者乙太網口讀數據到人機界面上。這些資料保存在 response 這個一維陣列中，讀取數據時會讀取指定的數量，未讀取的數據會保留於HMI記憶體緩衝區中，留待下次讀取，避免數據遺失。device_name是在“設備列表”中定義的“設備名稱”，而這個 device 必須選擇為 “Free Protocol” 這個設備類型。read_count是每次讀取的數據的長度。receive_len 存放所接收到的資料位元組長度，必須為變數。receive_len 的最大長度將不會超過 response 陣列宣告的大小。

舉例	macro_command main()
	
	short wResponse[6], receive_len
	
	INPORT3(wResponse[0], "Free Protocol", 6, receive_len) // 讀取 6 words
	
	if receive_len >= 6 then
	
	SetData(wResponse[0], "Local HMI", LW, 0, 6) // 把回應的資料寫入LW0
	
	end if
	
	end macro_command

函數名稱	INPORT4
語法	INPORT4(response[start], device_name, receive_len, tail_ascii)
描述	從串列埠或者乙太網口讀數據到人機界面上。這些資料保存在 response 這個一維陣列中。tail_ascii 是結尾符號。讀取數據時，只要讀取到結尾符號後即停止讀取。device_name是在“設備列表”中定義的“設備名稱”，而這個 device 必須選擇為 “Free Protocol” 這個設備類型。。receive_len 存放所接收到的資料位元組長度，必須為變數。receive_len 的最大長度將不會超過 response 陣列宣告的大小。
舉例	macro_command main() char tail_ascii = 0x03// == ETX short wResponse[1024], receive_len

	INPORT4(wResponse[0], "Free Protocol", receive_len, 0x0d)// 0x0d == CR
	
	INPORT4(wResponse[0], "Free Protocol", receive_len, tail_ascii)
	
	if receive_len >= 6 then
	
	  SetData(wResponse[0], "Local HMI", LW, 0, 6)// set responses to LW0
	
	end if
	end macro_command

函數名稱	OUTPORT
語法	OUTPORT(source[start], device_name, data_count)
描述	將放置在從 source[start] 到 source[start+data_count-1] 的所有資料通過串列埠或者乙太網口傳送給設備或者控制器中。 device_name 是在“設備列表”中定義的“設備名稱”，而這個 device 必須選擇為 “Free Protocol” 這個設備類型。 Data_count 是發送資料的個數，可以是常數或變數。

舉例	要使用 OUTPORT 函數，必須要在設備類型中選擇 “Free Protocol”，如下圖所示。
	 
	
	這裡的 device_name 即為 “MODBUD RTU Device”。埠的屬性也是依據在這個“系統參數” 中的設定，譬如，在此設定為 (9600，E，8，1…)
	
	下面是一個範例程式，使用自由協定，以 MODBUS RTU 的協定格式，將單個暫存器設置為 ON。
	
	macro_command main()
	
	char command[32]
	short address, checksum
	
	FILL(command[0], 0, 32) // 初始化命令
	
	command[0] = 0x1 // 站號
	command[1] = 0x5 // 功能碼 ： 寫單個位
	
	address = 0
	HIBYTE(address, command[2])

	LOBYTE(address, command[3])
	
	command[4] = 0xff // 使該bit設置為ON
	command[5] = 0
	
	CRC(command[0], checksum, 6)
	
	LOBYTE(checksum, command[6])
	HIBYTE(checksum, command[7])
	
	// 將命令通過串列埠送出去
	OUTPORT(command[0], "MODBUS RTU Device", 8)
	
	end macro_command

函數名稱	PURGE
語法	PURGE (com_port)
描述	com_port 表示序列埠編號，支援 COM1 ~ COM3。此參數可以為變數或常數。 可利用這個指令來清空 COM port 的輸出入緩衝區。
舉例	macro_command main() int com_port=3 PURGE (com_port) PURGE (1) end macro_command

函數名稱	SetRTS
語法	SetRTS(com_port, source)
描述	拉高或拉低 RS-232 之 RTS 訊號。 com_port 表示序列埠編號，支援 COM1。此參數可以為變數或常數。source 參數也可以為變數或常數。 當傳入的 source 參數值大於 0 時，將拉高 RTS 電位，當 source 參數值等於 0 時，將拉低 RTS 電位。

舉例	macro_command main()
	char com_port=1
	char value=1
	
	SetRTS(com_port, value) // value > 0，拉高 COM1 之 RTS 電位
	
	SetRTS(1, 0) // 拉低 COM1 之 RTS 電位
	
	end macro_command

18.7.4. 巨集控制

函數名稱	ASYNC_TRIG_MACRO
語法	ASYNC_TRIG_MACRO(macro_id or name)
描述	一個執行中的巨集指令可以使用此函數利用非同步的方式觸發執行其他的巨集指令。可使用其他巨集指令的 macro_id 或名稱作為參數，指定被觸發的巨集指令 [編號] 或 [名稱]。 巨集指令在使用此函數後將立即執行後續的命令，不需等待被觸發的巨集指令執行完成。 macro_id 可以是常數或使用變數來表示。

舉例	macro_command main()
	char ON = 1, OFF = 0
	
	SetData(ON, "Local HMI", LB, 0, 1)
	
	ASYNC_TRIG_MACRO(5) // call a macro (its ID is 5)
	
	ASYNC_TRIG_MACRO("macro_1") // call a macro (its name is macro_1)
	
	SetData(OFF, "Local HMI", LB, 0, 1)
	
	end macro_command

函數名稱	DELAY
語法	DELAY(time)
描述	讓巨集指令暫停執行，持續的時間至少是指定的這個時間。時間的單位為毫秒。 time 可以是常數或者變數。

舉例	macro_command main()
	int time = 500
	
	DELAY(100) // delay 100 ms
	DELAY(time) // delay 500 ms
	
	end macro_command

函數名稱	SYNC_TRIG_MACRO
語法	SYNC_TRIG_MACRO(macro_id or name)
描述	一個執行中的巨集指令可以使用此函數利用同步的方式觸發執行其他的巨集指令。可使用其他巨集指令的 macro_id 或名稱作為參數，指定被觸發的巨集指令 [編號] 或 [名稱]。 使用此函數的巨集指令將暫停執行，直到被觸發的巨集指令執行完成才會繼續執行後續的命令。 macro_id 可以是常數或使用變數來表示。
舉例	macro_command main() char ON = 1, OFF = 0

	SetData(ON, "Local HMI", LB, 0, 1)
	
	SYNC_TRIG_MACRO(5) // call a macro (its ID is 5)
	
	SYNC_TRIG_MACRO("macro_1") // call a macro (its name is macro_1)
	
	SetData(OFF, "Local HMI", LB, 0, 1)
	
	end macro_command

18.7.5. 資料操作

函數名稱	FILL
語法	FILL(source[start], preset, count)
描述	依序將預設值 (preset) 放置到一維陣列 source[start] 開始的陣列中，放置的資料個數由 count 決定。 source 和 start 必須為變數，preset 可以為一個常數或者變數。

舉例	macro_command main()
	char result[4]
	char preset
	
	FILL(result[0], 0x30, 4)
	// result[0] is 0x30, result[1] is 0x30, , result[2] is 0x30, , result[3] is 0x30
	
	preset = 0x31
	FILL(result[0], preset, 2) // result[0] is 0x31, result[1] is 0x31
	
	end macro_command

函數名稱	SWAPB
語法	SWAPB(source, result)
描述	將一個 16 位元字的高低位元組顛倒，並將結果存放在 result 變數中。 source 可以是常數或者是變數，單數 result 必須為變數。

舉例	macro_command main()
	short source, result
	
	SWAPB(0x5678, result) // result is 0x7856
	
	source = 0x123
	SWAPB(source, result) // result is 0x2301
	
	end macro_command

函數名稱	SWAPW
語法	SWAPW(source, result)

描述	將一個 32 位元雙整型資料的高位字組和低位字組顛倒，並將結果存放在 result 變數中。source 可以是常數或者變數，但是 result 必須為變數。

舉例	macro_command main()
	int source, result
	
	SWAPW(0x12345678, result) // result is 0x56781234
	
	source = 0x12345
	SWAPW(source, result) // result is 0x23450001
	
	end macro_command

函數名稱	LOBYTE
語法	LOBYTE(source, result)
描述	獲取一個 16 位元資料的低位元組，並且放置在 result 變數中。 source 可以是常數或者變數，但是 result 必須為變數。

舉例	macro_command main()
	short source, result
	
	LOBYTE(0x1234, result) // result is 0x34
	
	source = 0x123
	LOBYTE(source, result) // result is 0x23
	
	end macro_command

函數名稱	HIBYTE
語法	HIBYTE(source, result)
描述	獲取一個 16 位元資料的高位元組，並且放置在 result 變數中。 source 可以是常數或者變數，但是 result 必須為變數。

舉例	macro_command main()
	short source, result
	
	HIBYTE(0x1234, result) // result is 0x12
	
	source = 0x123
	HIBYTE(source, result) // result is 0x01
	
	end macro_command

函數名稱	LOWORD
語法	LOWORD(source, result)
描述	獲取一個 32 位元資料的低位字組，並將結果放置在 result 變數中。 source 可以是常數或者變數，但是 result 必須為變數。
舉例	macro_command main() int source, result

	LOWORD(0x12345678, result) // result is 0x5678
	
	source = 0x12345
	LOWORD(source, result) // result is 0x2345
	
	end macro_command

函數名稱	HIWORD
語法	HIWORD(source, result)
描述	獲取一個 32 位元資料的高位字組，並將結果放置在 result 變數中。 source 可以是常數或者變數，但是 result 必須為變數。

舉例	macro_command main()
	int source, result
	
	HIWORD(0x12345678, result) // result is 0x1234
	
	source = 0x12345
	HIWORD(source, result) // result is 0x0001
	
	end macro_command

函數名稱	INVBIT
語法	INVBIT(source, result, bit_pos)
描述	將資料或者變數 (source) 指定的位元位址狀態相反，並將改變後的資料存放在 result 變數中。 source 和 bit-pos 可以是常數或者變數，但是 result 必須為變數。

舉例	macro_command main()
	int source, result
	short bit_pos
	
	INVBIT(4, result, 1) // result = 6
	
	source = 6
	bit_pos = 1
	INVBIT(source, result, bit_pos) // result = 4
	
	end macro_command

函數名稱	SETBITON
語法	SETBITON(source, result, bit_pos)
描述	將資料或者變數 (source) 指定的位元位址設置為 1，並將改變後的資料存放在 result 變數中。 source 和 bit-pos 可以是常數或者變數，但是 result 必須為變數。
舉例	macro_command main() int source, result

	short bit_pos
	
	SETBITON(1, result, 3) // result is 9
	
	source = 0
	bit_pos = 2
	SETBITON (source, result, bit_pos) // result is 4
	
	end macro_command

函數名稱	SETBITOFF
語法	SETBITOFF(source, result, bit_pos)
描述	將資料或者變數 (source) 指定的位元位址設置為 0，並將改變後的資料存放在 result 變數中。 source 和 bit-pos 可以是常數或者變數，但是 result 必須為變數。

舉例	macro_command main()
	int source, result
	short bit_pos
	
	SETBITOFF(9, result, 3) // result is 1
	
	source = 4
	bit_pos = 2
	SETBITOFF(source, result, bit_pos) // result is 0
	
	end macro_command

函數名稱	GETBIT
語法	GETBIT(source, result, bit_pos)
描述	獲取資料或者變數 (source) 指定的位元的狀態，並將結果放置在 result 變數中。result 的資料將為 1 或者 0。 source 和 bit_pos 可以是常數或者變數，但是 result 必須為變數。

舉例	macro_command main()
	int source, result
	short bit_pos
	
	GETBIT(9, result, 3) // result is 1
	
	source = 4
	bit_pos = 2
	GETBIT(source, result, bit_pos) // result is 1
	
	end macro_command

18.7.6. 資料轉換

函數名稱	ASCII2DEC
語法	ASCII2DEC(source[start], result, len)
描述	將字元型 ASCII 資料 (source) 轉換為十進位格式的資料，並存放在 result 變數中。ASCII 的長度即為 len，第一個字元的位置即為 source[start] 的資料。 source 和 len 可以是常數或者變數，單數 result 必須為變數。start 必須為常數。
舉例	macro_command main() char source[4] short result source[0] = '5' source[1] = '6' source[2] = '7' source[3] = '8' ASCII2DEC(source[0], result, 4) // result is 5678 end macro_command

函數名稱	ASCII2FLOAT
語法	ASCII2FLOAT (source[start], result, len)
描述	將字元型 ASCII 資料 (source) 轉換為浮點數格式的資料，並存放在 result 變數中。ASCII 的長度即為 len，第一個字元的位置為 source[start] 的資料。 source 和 len 可以是常數或者變數，單數 result 必須為變數。start 必須為常數。
舉例	macro_command main() char source[4] float result source[0] = '5' source[1] = '6' source[2] = '.' source[3] = '8' ASCII2FLOAT(source[0], result, 4) // result is 56.8 end macro_command

函數名稱	ASCII2HEX
語法	ASCII2HEX (source[start], result, len)
描述	將 ASCII 字元型資料 (source) 轉換為十六進位的資料，並存放在 result 變數中。字元的長度即為 len 的資料。第一個字元存放在 source[start] 中。 source 和 len 可以是常數或者變數，單數 result 必須為變數。start 必須為常數。
舉例	macro_command main() char source[4] short result

	source[0] = '5' source[1] = '6' source[2] = '7' source[3] = '8' ASCII2HEX(source[0], result, 4) // result is 0x5678 end macro_command

函數名稱	ASCII2DOUBLE
語法	ASCII2DOUBLE(source[start], result, count)
描述	將 ASCII 字元型資料 (source) 轉換為雙精度浮點數的資料，並存放在 result 變數中。字元的長度即為 count 的資料。第一個字元存放在 source[start] 中。 source 和 count 可以是常數或者變數，單數 result 必須為變數。start 必須為常數。 此函數僅支援 cMT/cMT X 系列。
舉例	macro_command main() char source[4] = {'5', '6', '.', '8'} double result ASCII2DOUBLE(source[0], result, 4)// result == 56.8 SetData(result, "Local HMI", LW, 100, 1) end macro_command

函數名稱	BIN2BCD
語法	BIN2BCD(source, result)
描述	將 BIN 格式的資料 (source) 轉換為 BCD 格式的資料 (result)。資料來源 source 可以是常數或者變數，但是存放結果的 result 必須為變數。

舉例	macro_command main()
	
	short source, result
	
	BIN2BCD(1234, result) // result is 0x1234
	
	source = 5678
	BIN2BCD(source, result) // result is 0x5678
	
	end macro_command

函數名稱	BCD2BIN
語法	BCD2BIN(source, result)
描述	將 BCD 格式的資料 (source) 轉換為 BIN 格式的資料 (result)。資料來源 source 可以是常數或者變數，但是存放結果的 result 必須為變數。
舉例	macro_command main()

	short source, result
	
	BCD2BIN(0x1234, result) // result is 1234
	
	source = 0x5678
	BCD2BIN(source, result) // result is 5678
	
	end macro_command

函數名稱	DATE2ASCII
語法	DATE2ASCII(day_offset, date[start], count, *[separator]*)
描述	將現在日期資料經過 day_offset 的時間調整後轉換為 ASCII 格式的資料，並存放在一個一維陣列 (date) 中。count 表示這個轉換後的字串的長度，同時這個長度也取決於存放結果。separator 則是分隔年月日的分隔符號，預設為 “/” 。 day_offset 和 count 可以是常數或者變數，start 與 separator 必須為常數。
舉例	macro_command main() char result[10] DATE2ASCII(5, result[0], 10) // result[0]~[9] == "2019/02/16"// today is 2019/02/11 DATE2ASCII(5, result[0], 10,”-“) // result[0]~[9] == "2019-02-16"// today is 2019/02/11 end macro_command

函數名稱	DATE2DEC
語法	DATE2DEC(day_offset, date)
描述	將現在日期資料經過 day_offset 的時間調整後轉換為十進位格式的資料，並存放在 date 變數中。 day_offset 可以是常數或者變數，date 必須為變數。

舉例	macro_command main()
	int day_offset = 5, date
	
	DATE2DEC(0, date) // date == 20190211 (Today is 2019/02/11)
	DATE2DEC(day_offset, date) // date == 20190216 (20190211 + 5)
	
	end macro_command

函數名稱	DEC2ASCII
語法	DEC2ASCII(source, result[start], len)
描述	將十進位的資料 (source) 轉換為 ASCII 格式的資料，並存放在一個一維陣列 (result) 中。len 表示這個轉換後的字串的長度，同時這個長度也取決於存放結果的一維陣列的資料格式。例如：如果 result 一維陣列的格式為 “char” (字元型，長度為一個位元組) ，則長度為 “位元組數*len” 。如果 result 一維陣列的格式

	為 “short” (短整型資料，2 個位元組) ，則長度為 “word*len” 。依此類推。轉換後的第一個字元放在 result[start] 中，第二個字元放在 result[start+1]中，最後一個字元放在 result[start+(len-1)]中。 source 和 len 可以是常數或者變數，單數 result 必須為變數。start 必須為常數。

舉例	macro_command main()
	short source
	char result1[4]
	short result2[4]
	char result3[6]
	source = 5678
	
	DEC2ASCII(source, result1[0], 4)
	// result1[0] is '5', result1[1] is '6', result1[2] is '7', result1[3] is '8'
	// the length of the string (result1) is 4 bytes(= 1 * 4)
	DEC2ASCII(source, result2[0], 4)
	// result2[0] is '5', result2[1] is '6', result2[2] is '7', result2[3] is '8'
	// the length of the string (result2) is 8 bytes(= 2 * 4)
	
	source=-123
	DEC2ASCII(source, result3[0], 6)
	// result1[0] is '-', result1[1] is '0', result1[2] is '0', result1[3] is '1'
	// result1[4] is '2', result1[5] is '3'
	// the length of the string (result1) is 6 bytes(= 1 * 6)
	
	end macro_command

函數名稱	FLOAT2ASCII
語法	FLOAT2ASCII (source, result[start], len)
描述	浮點數格式資料 (source) 轉換為 ASCII 格式的資料，並將結果存放在一個一維陣列 (result) 中。len 表示這個轉換後的字串的長度，同時這個長度也取決於存放結果的一維陣列的資料格式。例如：如果 result 一維陣列的格式為 “char” (字元型，長度為一個位元組)，則長度為 “位元組數*len” 。如果 result 一維陣列的格式為 “short” (短整型資料，2 個位元組)，則長度為 “word*len” 。依此類推。 source 和 len 可以是常數或者變數，單數 result 必須為變數。start 必須為常數。
舉例	macro_command main() float source char result[4] source = 56.8 FLOAT2ASCII (source, result[0], 4) // result[0] is '5', result[1] is '6', result[2] is '.', result[3] is '8' end macro_command

函數名稱	HEX2ASCII

語法	HEX2ASCII(source, result[start], len)
描述	十六進位格式資料 (source) 轉換為 ASCII 格式的資料，並將結果存放在一個一維陣列 (result) 中。len 表示這個轉換後的字串的長度，同時這個長度也取決於存放結果的一維陣列的資料格式。例如：如果 result 一維陣列的格式為 “char” (字元型，長度為一個位元組)，則長度為 “位元組數*len”。如果 result 一維陣列的格式為 “short” (短整型資料，2 個位元組)，則長度為 “word*len”。依此類推。 source 和 len 可以是常數或者變數，單數 result 必須為變數。start 必須為常數。
舉例	macro_command main() short source char result[4] source = 0x5678 HEX2ASCII(source, result[0], 4) // result[0] is '5', result[1] is '6', result[2] is '7', result[3] is '8' end macro_command

函數名稱	DOUBLE2ASCII
語法	DOUBLE2ASCII(source, result[start], count)
描述	雙精度浮點數格式資料 (source) 轉換為 ASCII 格式的資料，並將結果存放在一個一維陣列 (result) 中。count 表示這個轉換後的字串的長度，同時這個長度也取決於存放結果的一維陣列的資料格式。例如：如果 result 一維陣列的格式為 “char” (字元型，長度為一個位元組)，則長度為 “位元組數*count”。如果 result 一維陣列的格式為 “short” (短整型資料，2 個位元組)，則長度為 “word* count”。依此類推。 source 和 count 可以是常數或者變數，單數 result 必須為變數。start 必須為常數。 此函數僅支援 cMT/cMT X 系列。
舉例	macro_command main() double source = 56.8 char result[4] DOUBLE2ASCII(source, result[0], 4) // result[0] == '5', result[1] == '6', result[2] == '.', result[3] == '8' end macro_command

函數名稱	StringDecAsc2Bin
語法	success = StringDecAsc2Bin(source[start], destination) 或 success = StringDecAsc2Bin("source", destination)
描述	此函數將十進位字串轉換成整數。 來源字串 source 可以為靜態字串 (如 "source")或是一維字元陣列變數 (如 source[start])。 destination 必須為一變數，用以存放轉換後的整數值。

	執行完畢會回傳一 bool 型態的值給 success 欄位。當轉換成功，success等於 true，否則等於 false。 來源字串必需為十進位字串，若其包含正負號或 '0' ~ '9' 以外的字元，函數將會回傳false。 success 欄位可寫可不寫。

舉例	macro_command main()
	char src1[5] = "12345"
	int result1
	bool success1
	success1 = StringDecAsc2Bin(src1[0], result1)
	// success1 = true，result1 為 "12345"
	
	char src2[5] = "-6789"
	short result2
	bool success2
	success2 = StringDecAsc2Bin(src2[0], result2)
	// success2 = true，result2 為 "-6789"
	
	char result3
	bool success3
	success3= StringDecAsc2Bin("32768", result3)
	// success3 = true，但結果超出 result3 所能表達的範圍
	
	char src4[2] = "4b"
	char result4
	bool success4
	success4 = StringDecAsc2Bin (src4[0], result4)
	// success4= false，因 src4 包含正負號 或 '0' ~ '9' 以外的字元
	
	end macro_command

函數名稱	StringBin2DecAsc
語法	success = StringBin2DecAsc (source, destination[start])
描述	此函數將整數轉換成十進位字串。 來源 source 可以為常數或變數。 destination[start] 必須為一維字元陣列變數，用以存放轉換後的十進位字串。 執行完畢會回傳一 bool 型態的值給 success 欄位。當轉換成功，success等於 true，否則等於 false。 若轉換後的十進位字串長度大於目標陣列的大小，函數將會回傳 false。 success 欄位可寫可不寫。 注意: 此函數不能轉換負值。
舉例	macro_command main() int src1 = 2147483647 char dest1[20] bool success1

	success1 = StringBin2DecAsc(src1, dest1[0])
	// success1 = true，dest1為 "2147483647"
	
	short src2 = 0x3c
	char dest2[20]
	bool success2
	success2 = StringBin2DecAsc(src2, dest2[0])
	// success2 = true，dest2為 "60"
	
	int src3 = 2147483647
	char dest3[5]
	bool success3
	success3 = StringBin2DecAsc(src3, dest3[0])
	// success3 = false，dest3 內容不變
	
	end macro_command

函數名稱	StringDecAsc2Float
語法	success = StringDecAsc2Float (source[start], destination) 或 success = StringDecAsc2Float ("source", destination)
描述	此函數將十進位字串轉換成浮點數。 來源字串 source 可以為靜態字串 (如 "source")或是一維字元陣列變數 (如 source[start])。 destination 必須為一變數，用以存放轉換後的浮點數值。 執行完畢會回傳一 bool 型態的值給 success 欄位。當轉換成功，success等於 true，否則等於 false。 來源字串必需為十進位字串，若其包含 '0' ~ '9' 或 '.' 以外的字元，函數將會回傳 false。 success 欄位可寫可不寫。

舉例	macro_command main()
	char src1[10] = "12.345"
	float result1
	bool success1
	success1 = StringDecAsc2Float(src1[0], result1)
	// success1 = true，result1 為 "12.345"
	
	float result2
	bool success2
	success2 = StringDecAsc2Float("1.234567890", result2)
	// success2 = true，但結果超出 result2 所能表達的範圍，可能喪失精確度
	
	char src3[2] = "4b"
	float result3
	bool success3
	success3 = StringDecAsc2Float(src3[0], result3)
	// success3 = false，因 src3 包含 '0' ~ '9' 或 '.' 以外的字元

	end macro_command

函數名稱	StringFloat2DecAsc
語法	success = StringFloat2DecAsc(source, destination[start])
描述	此函數將浮點數轉換成十進位字串。 來源 source 可以為常數或變數。 destination[start] 必須為一維字元陣列變數，用以存放轉換後的十進位字串。 執行完畢會回傳一 bool 型態的值給 success 欄位。當轉換成功，success等於 true，否則等於 false。 若轉換後的十進位字串長度大於目標陣列的大小，函數將會回傳 false。 success 欄位可寫可不寫。

舉例	macro_command main()
	float src1 = 1.2345
	char dest1[20]
	bool success1
	success1 = StringFloat2DecAsc(src1, dest1[0])
	// success1 = true，dest1為 "1.2345"
	
	float src2 = 1.23456789
	char dest2 [20]
	bool success2
	success2 = StringFloat2DecAsc(src2, dest2 [0])
	// success2 = true，但可能喪失精確度
	
	float src3 = 1.2345
	char dest3[5]
	bool success3
	success3 = StringFloat2DecAsc(src3, dest3 [0])
	// success3 = false，dest3 內容不變
	
	end macro_command

函數名稱	StringHexAsc2Bin
語法	success = StringHexAsc2Bin (source[start], destination) 或 success = StringHexAsc2Bin ("source", destination)
描述	此函數將十六進位字串轉換成整數。 來源字串 source 可以為靜態字串 (如 "source")或是一維字元陣列變數 (如 source[start])。 destination 必須為一變數，用以存放轉換後的整數值。 執行完畢會回傳一 bool 型態的值給 success 欄位。當轉換成功，success等於 true，否則等於 false。

	來源字串必需為十六進位字串，若其包含 ‘0’ ~ ‘9’ 或 ‘a’ ~ ‘f’ 或 ‘A’ ~ ‘F’ 以外的字元，函數將會回傳 false。 success 欄位可寫可不寫。

舉例	macro_command main()
	char src1[5]="0x3c"
	int result1
	bool success1
	success1 = StringHexAsc2Bin(src1[0], result1)
	// success1 = true，result1 為 3c
	
	short result2
	bool success2
	success2 = StringDecAsc2Bin("1a2b3c4d", result2)
	// success2 = true，但結果超出 result2 所能表達的範圍，result2 = 3c4d
	
	char src3[2] = "4g"
	char result3
	bool success3
	success3 = StringDecAsc2Bin (src3[0], result3)
	// success3=false，因 src3包含 ‘0’ ~ ‘9’ 或 ‘a’ ~ ‘f’ 或 ‘A’ ~ ‘F’ 以外的字元
	
	end macro_command

函數名稱	StringBin2HexAsc
語法	success = StringBin2HexAsc (source, destination[start])
描述	此函數將整數轉換成十六進位字串。 來源source可以為常數或變數。 destination[start] 必須為一維字元陣列變數，用以存放轉換後的十六進位字串。 執行完畢會回傳一 bool 型態的值給 success 欄位。當轉換成功，success等於 true，否則等於 false。 若轉換後的十六進位字串長度大於目標陣列的大小，函數將會回傳 false。 success 欄位可寫可不寫。 注意: 此函數不能轉換負值。

舉例	macro_command main()
	int src1 = 20
	char dest1[20]
	bool success1
	success1 = StringBin2HexAsc(src1, dest1[0])
	// success1 = true，dest1 為 "14"
	
	short src2 = 0x3c
	char dest2[20]
	bool success2
	success2 = StringBin2HexAsc(src2, dest2[0])
	// success2 = true，dest2 為 "3c"

	int src3 = 0x1a2b3c4d char dest3[6] bool success3 success3 = StringBin2HexAsc(src3, dest3[0]) // success3 = false，dest3 內容不變 end macro_command

請點選此圖示下載範例程式。下載範例程式前，請先確定已連上網路線。

18.7.7. 字串處理

函數名稱	String2Unicode
語法	result = String2Unicode("source", destination[start])
描述	將source字串轉換為Unicode字串，存放到destination，轉換後的Unicode字串長度會存到result。Source來源必須為常數，不可為變數。

舉例	macro_command main()
	
	char dest[20]
	int result
	result = String2Unicode("abcde", dest[0]) // "result" will be set to 10.
	result = String2Unicode("abcdefghijklmno", dest[0]) // "result" will be set to 20.
	// "result" will be the length of converted Unicode string
	
	end macro_command

函數名稱	StringCat
語法	success = StringCat (source[start], destination[start]) 或 success = StringCat ("source", destination[start])
描述	利用此函數將來源字串銜接於目標字串之後。此函數執行成功後，目標字串將等於兩字串銜接後的結果。 來源字串source可以為靜態字串 (如 "source") 或是一維字元陣列變數 (如 source[start])。 destination[start]必須為一維字元陣列變數。 執行完畢會回傳一 bool 型態的值給 success 欄位。當執行成功，success等於 true，否則等於 false。當兩字串銜接後的長度超過目標陣列的長度時，目標字串將保留銜接以前的內容，不做任何改變，並回傳 false。
舉例	macro_command main() char src1[20] = "abcdefghij" char dest1[20] = "1234567890" bool success1 success1= StringCat(src1[0], dest1[0])

	// success1 = true，dest1 = "123456790abcdefghij"
	
	char dest2 [10] = "1234567890"
	bool success2
	success2= StringCat("abcde", dest2 [0])
	// success2 = false，dest2 內容不變
	
	char src3[20] = "abcdefghij"
	char dest3[20]
	bool success3
	success3 = StringCat(src3[0], dest3[15])
	// success3 = false，dest3 內容不變
	
	end macro_command

函數名稱	StringCompare
語法	ret = StringCompare (str1[start], str2[start]) ret = StringCompare ("string1", str2[start]) ret = StringCompare (str1[start], "string2") ret = StringCompare ("string1", "string2")
描述	比較兩字串的內容是否相等。此函數將大小寫視為不同。 兩個傳入的字串皆可以為靜態字串 (如 "source")或是一維字元陣列變數 (如 source[start])。 執行完畢會回傳一 bool型態的值給 ret 欄位。若兩字串相等，ret 為 true，否則為 false。

舉例	macro_command main()
	char a1[20] = "abcde"
	char b1[20] = "ABCDE"
	bool ret1
	ret1 = StringCompare(a1[0], b1[0])
	// ret1 = false
	
	char a2[20] = "abcde"
	char b2[20] = "abcde"
	bool ret2
	ret2 = StringCompare(a2[0], b2[0])
	// ret2 = true
	
	char a3 [20] = "abcde"
	char b3[20] = "abcdefg"
	bool ret3
	ret3 = StringCompare(a3[0], b3[0])
	// ret3 = false
	
	end macro_command

函數名稱	StringCompareNoCase

語法	ret = StringCompareNoCase(str1[start], str2[start]) ret = StringCompareNoCase("string1", str2[start]) ret = StringCompareNoCase(str1[start], "string2") ret = StringCompareNoCase("string1", "string2")
描述	比較兩字串的內容是否相等。此函數將大小寫視為相同。 兩個傳入的字串皆可以為靜態字串 (如 "source") 或是一維字元陣列變數(如 source[start])。 執行完畢會回傳一 bool 型態的值給 ret 欄位。若兩字串相等，ret 為 true，否則為 false。

舉例	macro_command main()
	char a1[20] = "abcde"
	char b1[20] = "ABCDE"
	bool ret1
	ret1 = StringCompareNoCase(a1[0], b1[0])
	// ret1 = true
	
	char a2[20] = "abcde"
	char b2[20] = "abcde"
	bool ret2
	ret2 = StringCompareNoCase(a2[0], b2[0])
	// ret2 = true
	
	char a3 [20] = "abcde"
	char b3[20] = "abcdefg"
	bool ret3
	ret3 = StringCompareNoCase(a3[0], b3[0])
	// ret3 = false
	
	end macro_command

函數名稱	StringCopy
語法	success = StringCopy ("source", destination[start]) 或 success = StringCopy (source[start], destination[start])
描述	利用此函數進行字串的複製。此函數可以將傳入的靜態字串 (以雙引號" "括起來的字串)，或是存放在字元陣列中的字串複製到目的 buffer。 來源字串 sourc e可以為靜態字串 (如 "source")或是一維字元陣列變數 (如 source[start])。 destination[start] 必須為一維字元陣列變數。 複製完畢會回傳一 bool 型態的值給 success 欄位。當複製成功，success等於 true，否則等於 false。當來源字串的長度大於目的 buffer 的大小時，將不做任何處理，並回傳 false 到 success 欄位。 success 欄位可寫可不寫。
舉例	macro_command main() char src1[5] = "abcde" char dest1[5]

	bool success1
	success1 = StringCopy(src1[0], dest1[0])
	// success1=true，dest1為 "abcde"
	char dest2[5]
	bool success2
	success2 = StringCopy("12345", dest2[0])
	// success2 = true，dest2為 "12345"
	char src3[10] = "abcdefghij"
	char dest3[5]
	bool success3
	success3 = StringCopy(src3[0], dest3[0])
	// success3 = false，dest3 內容不變
	char src4[10] = "abcdefghij"
	char dest4[5]
	bool success4
	success4 = StringCopy(src4[5], dest4[0])
	// success4=true，dest4為 "fghij"
	
	end macro_command

函數名稱	StringExcluding
語法	success = StringExcluding (source[start], set[start], destination[start]) success = StringExcluding ("source", set[start], destination[start]) success = StringExcluding (source[start], "set", destination[start]) success = StringExcluding ("source", "set", destination[start])
描述	提取 source 字串中某個以索引值 0 的字元 (第一個字元) 開頭的子字串，而且此子字串的每個字元必不存在於set字串中。此函數將會從 source字串的第一個字元開始尋找，直到找到存在於 set 字串中的字元為止。 source 字串與 set 字串皆可以為靜態字串 (如 "source") 或是一維字元陣列變數 (如 source[start])。 執行完畢會回傳一 bool 型態的值給 success 欄位。當執行成功，success等於 true，否則等於 false。當提取出來的字串長度大於目標陣列的大小，將會回傳 false。

舉例	macro_command main()
	char src1[20] = "cabbageabc"
	char set1[20] = "ge"
	char dest1[20]
	bool success1
	success1 = StringExcluding(src1[0], set1[0], dest1[0])
	// success1 = true，dest1 = "cabba"
	
	char src2[20] = "cabbage"
	char dest2[20]
	bool success2
	success2 = StringExcluding(src2[0], "abc", dest2[0])
	// success2 = true，dest2 =" "

	char set3[20] = "ge" char dest3[4] bool success3 success3 = StringExcluding("cabbage", set3[0], dest3[0]) // success3 = false，dest3 內容不變 end macro_command

函數名稱	StringFind
語法	position = StringFind (source[start], target[start]) position = StringFind ("source", target[start]) position = StringFind (source[start], "target") position = StringFind ("source", "target")
描述	尋找某一字串 (target) 在另一個字串 (source) 裡第一次出現的位置。 兩個傳入的字串皆可以為靜態字串 (如 "source") 或是一維字元陣列變數(如 source[start])。 執行完畢會回傳target字串在 source 字串裡出現的位置。Source 字串由 0 開始遞增為字元編索引值。若 source 字串存在一個子字串，其所包含的字元與排列順序跟 target 字串完全相等，則函數會回傳此子字串開頭的索引值，若沒有找到，則回傳 -1。

舉例	macro_command main()
	char src1[20] = "abcde"
	char target1[20] = "cd"
	short pos1
	pos1 = StringFind(src1[0], target1[0])
	// pos1 = 2
	
	char target2[20] = "ce"
	short pos2
	pos2 = StringFind("abcde", target2[0])
	// pos2 = -1
	
	char src3[20] = "abcde"
	short pos3
	pos3 = StringFind(src3[3], "cd")
	// pos3 = -1
	
	end macro_command

函數名稱	StringFindOneOf
語法	position = StringFindOneOf (source[start], target[start]) position = StringFindOneOf ("source", target[start]) position = StringFindOneOf (source[start], "target") position = StringFindOneOf ("source", "target")
描述	尋找 target 字串中任一個字元在 source 字串中第一次出現的位置。

	兩個傳入的字串皆可以為靜態字串 (如 "source") 或是一維字元陣列變數(如 source[start])。 執行完畢會回傳 target 字串中任一個字元在 source 字串裡第一次出現的位置，即 source 字串中為該字元編的索引值 (由 0 開始)。若沒有找到，則回傳 -1。

舉例	macro_command main()
	char src1[20] = "abcdeabcde"
	char target1[20] = "sdf"
	short pos1
	pos1 = StringFindOneOf(src1[0], target1[0])
	// pos1 = 3
	
	char src2[20] = "abcdeabcde"
	short pos2
	pos2 = StringFindOneOf(src2[1], "agi")
	// pos2 = 4
	
	char target3 [20] = "bus"
	short pos3
	pos3 = StringFindOneOf("abcdeabcde", target3[1])
	// pos3 = -1
	
	end macro_command

函數名稱	StringGet
語法	StringGet(read_data[start], device_name, device_type, address_offset, data_count)
描述	獲取設備的資料。字串的資料型態為字元陣列，是存儲在 read_data[start]~read_data[start+data_count-1] 這些一維陣列變數中。read_data 必須為一維字元陣列。 data_count 是設定的讀取字元的個數，可以是常數也可以是變數。 此處的 device_name，即為在“系統參數” 中建立設備類型時，設定的“設備名稱”。在此，設備名稱被設定為 “FATEK FB Series”，如下圖所示。  device_type 是設備類型和設備中資料的編碼方式。例如：如果 device_type 是 LW_BIN，那麼讀取的設備類型為 LW，資料編碼方式為 BIN。如果使用 BIN 編碼方式，"_BIN" 可以忽略。 如果 device_type 是 LW_BCD，表示設備類型 LW，資料的編碼方式為 BCD 格式。

address_offset 是設備中的地址偏移量。
例如，StringGet(read_data_1[0], "FATEK FB Series", RT, 5, 1) 代表讀取的設備位址偏移量為 5。
如果 address_offset 使用格式為 "N#AAAAA"，N 表示設備的站號，AAAAA 表示位址偏移量。此情況一般使用在同一個串列埠上連接有多台設備或者控制器的情況下。例如：StringGet(read_data_1[0], "FATEK FB Series", RT, 2#5, 1) 表示讀取站號為 2 的設備的資料。如果 StringGet()使用“系統參數 / 設備列表”中設定的默認的站號，在此可以不填這個站號。

從設備中讀取的資料個數，由 data_count 的值來決定，因 read_data 變數僅接受 char 陣列型態。如下表所示：

read_data 的類型	data_count 的值	讀取 16 位元數據的個數
char (8-bit)	1	1
char (8-bit)	2	1

因為一個 WORD (16 位元) 等於 2 個 ASCII 字元的長度，當設備類型長度為 WORD 時，根據上表，讀取 2 個 ASCII 字元實際上是讀 1 個 WORD 的數據。

舉例

```
macro_command main()
char str1[20]

// 讀取 LW-0 ~ LW-9 共 10 個 WORD 到變數 str1[0] 到 str1[19] 中
// 因 1 WORD 可存放 2 個 ASCII 字元，欲讀取 20 個 ASCII 字元
```

	// 實際上共讀取了 10 個WORD
	StringGet(str1[0], "Local HMI", LW, 0, 20)
	
	end macro_command

函數名稱	StringGetEx
語法	StringGetEx (read_data[start], device_name, device_type, address_offset, data_count)
描述	獲取設備的資料，不等待設備回應，逕自往下執行。 read_data、device_name、device_type、address_offset 和 data_count 的說明和 GetData 相同。

舉例	macro_command main()
	char str1[20]
	short test=0
	
	// 當 MODBUS 設備未回應，test = 1將照常執行
	StringGetEx(str1[0], "MODBUS RTU", 4x, 0, 20)
	test = 1
	
	// 當 MODBUS 設備未回應，test = 2 將不會被執行，直到得到回應
	StringGet(str1[0], "MODBUS RTU", 4x, 0, 20)
	test = 2
	
	end macro_command

函數名稱	StringIncluding
語法	success = StringIncluding (source[start], set[start], destination[start]) success = StringIncluding ("source", set[start], destination[start]) success = StringIncluding (source[start], "set", destination[start]) success = StringIncluding ("source", "set", destination[start])
描述	提取 source 字串中某個以索引值 0 的字元 (第一個字元) 開頭的子字串，而且此子字串的每個字元都能在 set 字串中找到相同的字元。此函數將會從 source 字串的第一個字元開始尋找，直到找到不存在於 set 字串中的字元為止。 source 字串與 set 字串皆可以為靜態字串 (如 "source") 或是一維字元陣列變數 (如 source[start])。 執行完畢會回傳一 bool 型態的值給 success 欄位。當執行成功，success等於 true，否則等於 false。當提取出來的字串長度大於目標陣列的大小，將會回傳 false。
舉例	macro_command main() char src1[20] = "cabbageabc" char set1[20] = "abc" char dest1[20] bool success1 success1 = StringIncluding(src1[0], set1[0], dest1[0]) // success1 = true，dest1 = "cabba"

	char src2[20] = "gecabba"
	char dest2[20]
	bool success2
	success2 = StringIncluding(src2[0], "abc", dest2[0])
	// success2 = true，dest2 = "　"
	
	char set3[20] = "abc"
	char dest3[4]
	bool success3
	success3 = StringIncluding("cabbage", set3[0], dest3[0])
	// success3 = false，dest3內容不變
	
	end macro_command

函數名稱	StringInsert
語法	success = StringInsert (pos, insert[start], destination[start]) success = StringInsert (pos, "insert", destination[start]) success = StringInsert (pos, insert[start], length, destination[start]) success = StringInsert (pos, "insert", length, destination[start])
描述	將 insert 字串插入到目標字串中的特定位置，插入位置由 pos 所指定。Insert 字串可以為靜態字串 (如 "insert") 或是一維字元陣列變數 (如 insert[start])。 使用者亦可以在 length 欄位指定 insert 字串的長度。 執行完畢會回傳一 bool 型態的值給 success 欄位。當執行成功，success等於 true，否則等於 false。當插入完成後的字串長度大於目標陣列的大小，將會回傳 false。

舉例	macro_command main()
	
	char str1[20] = "but the question is"
	char str2[10] = ", that is"
	char dest[40] = "to be or not to be"
	bool success
	
	success = StringInsert(18, str1[3], 13, dest[0])
	// success = true, dest = "to be or not to be the question"
	
	success = StringInsert(18, str2[0], dest[0])
	// success=true, dest="to be or not to be, that is the question"
	
	success = StringInsert(0, "Hamlet:", dest[0])
	// success = false, dest 內容不變
	
	end macro_command

函數名稱	StringLength
語法	length = StringLength (source[start]) 或

	length = StringLength ("source")
描述	取得字串的長度。 來源字串 source 可以為靜態字串 (如 "source") 或是一維字元陣列變數(如 source[start])。 函數回傳值代表來源字串的長度。

舉例	macro_command main()
	char src1[20] = "abcde"
	int length1
	length1= StringLength(src1[0])
	// length1 = 5
	
	char src2[20] = { 'a', 'b' , 'c' , 'd' , 'e' }
	int length2
	length2= StringLength(src2[0])
	// length2 = 5
	
	char src3[20] = "abcdefghij"
	int length3
	length3 = StringLength(src3 [2])
	// length3 = 8
	
	end macro_command

函數名稱	StringMD5
語法	result = StringMD5(source[start], destination[start]) 或 result = StringMD5("source", destination[start])
描述	利用此函數可產生一組透過MD5訊息摘要演算法的字串。 來源字串 source 可以為靜態字串 (如 "source")或是一維字元陣列變數 (如 source[start])。當來源字串為字元陣列變數時，由陣列下標決定子字串起始位置。 destination[start] 必須為一維字元陣列變數，用以存放產生出來的子字串。 執行完畢會回傳MD5字串長度給 result 欄位。

舉例	macro_command main()
	char source[32] = "password", dest[32]
	int result
	
	result = StringMD5(source[0], dest[0])
	result = StringMD5("password", dest[0]) // "result" will be set to 32.
	// "result" will be the length of MD5 string.
	// dest[] = 5f4dcc3b5aa765d61d8327deb882cf99
	
	end macro_command

函數名稱	StringMid
語法	success = StringMid (source[start], count, destination[start])

	或 success = StringMid ("string", start, count, destination[start])
描述	利用此函數可將一個字串中的某一段子字串提取出來。 來源字串 source 可以為靜態字串 (如 "source")或是一維字元陣列變數 (如 source[start])。當來源字串為字元陣列變數時，由陣列下標決定子字串起始位置。當來源字串為靜態字串時，由第二個參數 start 決定子字串起始位置。 count 決定要提取的子字串長度。 destination[start] 必須為一維字元陣列變數，用以存放提取出來的子字串。 執行完畢會回傳一 bool 型態的值給 success 欄位。當執行成功，success等於 true，否則等於 false。 若提取出來的子字串長度大於目標陣列的大小，函數將會回傳false。 success 欄位可寫可不寫。

舉例	macro_command main()
	char src1[20] = "abcdefghijklmnopqrst"
	char dest1[20]
	bool success1
	success1 = StringMid(src1[5], 6, dest1[0])
	// success1 = true，dest1 為 "fghijk"
	
	char src2[20] = "abcdefghijklmnopqrst"
	char dest2[5]
	bool success2
	success2 = StringMid(src2[5], 6, dest2[0])
	// success2 = false，dest2 內容不變
	
	char dest3[20] = "12345678901234567890"
	bool success3
	success3 = StringMid("abcdefghijklmnopqrst", 5, 5, dest3[15])
	// success3 = true，dest3 = "123456789012345fghij"
	
	end macro_command

函數名稱	StringReverseFind
語法	position = StringReverseFind (source[start], target[start]) position = StringReverseFind ("source", target[start]) position = StringReverseFind (source[start], "target") position = StringReverseFind ("source", "target")
描述	尋找某一字串 (target) 在另一個字串 (source) 裡最後一次出現的位置。 兩個傳入的字串皆可以為靜態字串 (如 "source") 或是一維字元陣列變數(如 source[start])。 執行完畢會回傳 target 字串在 source 字串裡最後一次出現的位置。source 字串由 0 開始遞增為字元編索引值。若 source 字串存在一個子字串，其所包含的字元與排列順序跟 target 字串完全相等，則函數會回傳此子字串開頭的索引值，若沒有找到，則回傳 -1。若 source 字串中存在多個與 target 字串相等的子字串，則回傳最後一個出現的子字串的位置。

舉例	macro_command main()
	char src1[20] = "abcdeabcde"
	char target1[20] = "cd"
	short pos1
	pos1 = StringReverseFind(src1[0], target1[0])
	// pos1 = 7
	
	char target2[20] = "ce"
	short pos2
	pos2 = StringReverseFind("abcdeabcde" , target2[0])
	// pos2 = -1
	
	char src3[20] = "abcdeabcde"
	short pos3
	pos3 = StringReverseFind(src3[6], "ab")
	// pos3 = -1
	
	end macro_command

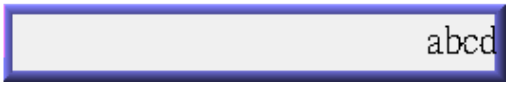
函數名稱	StringSet
語法	StringSet (send_data[start], device_name, device_type, address_offset, data_count)
描述	將數據寫到設備中。字串資料保存在 send_data[start]~send_data[start+data_count-1] 中，send_data 必須為一維字元陣列型態。 data_count 是寫入到設備中字元資料的個數，可以是常數也可以是變數。 device_name 詳見上面的說明，在此不在說明。 device_type 是設備類型和設備中資料的編碼方式。例如：如果 device_type 是 LW_BIN，那麼讀取的設備類型為 LW，資料編碼方式為 BIN。如果使用 BIN 編碼方式，"_BIN" 可以忽略。 如果 device_type 是 LW_BCD，表示設備類型 LW，資料的編碼方式為 BCD 格式。 address_offset 是設備中的地址偏移量。 例如，StringSet(read_data_1[0], "FATEK FB Series", RT, 5, 1) 代表讀取的設備位址偏移量為 5。 如果 address_offset 使用格式為 "N#AAAAA"，N 表示設備的站號， AAAAA 表示位址偏移量。此情況一般使用在同一個串列埠上連接有多台設備或者控制器的情況下。例如：StringSet(send_data_1[0], "FATEK FB Series", RT, 2#5, 1) 表示設定站號為 2 的設備的資料。如果 StringSet() 使用 “系統參數 / 設備列表” 中設定的默認的站號，在此可以不填這個站號。 設定到設備的資料個數，根據 data_count 的值來決定，因 send_data 變數僅接受 char 陣列型態。如下表所示： sead_data 的類型 | data_count 的值 | 設定 16 位元數據的個數

char (8-bit)	1	1
char (8-bit)	2	1

因為一個 WORD (16 位元) 等於 2 個 ASCII 字元的長度，當設備類型長度為 WORD 時，根據上表，寫入 2 個 ASCII 字元實際上是寫 1 個 WORD 的數據。巨集指令會以先寫 low byte 再寫 hight byte 的順序，依序將 ASCII 字元寫入。使用 [字元顯示] 物件顯示數據時，data_count 必須填入 2 的倍數才能正確顯示。例如：

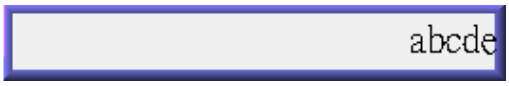
```
macro_command main()
char src1[10]="abcde"
StringSet(src1[0], "Local HMI", LW, 0, 5)
end macro_command
```

[字元顯示] 物件顯示如下：



當 data_count 為一個大於或等於字串長度的偶數時，可以完整顯示整個字串：

```
macro_command main()
char src1[10]="abcde"
StringSet(src1[0], "Local HMI", LW, 0, 6)
end macro_command
```



舉例

```
macro_command main()
char str1[10]= "abcde"

// 字串 str1 寫入 LW-0 ~ LW-2 共三個 WORD
// 即使 data_count 為 10，寫到第 3 個 WORD 時發現字串已結束，
// 便不再寫入後面的數據
StringSet(str1[0], "Local HMI", LW, 0, 10)

end macro_command
```

函數名稱	StringSetEx
語法	StringSetEx(send_data[start], device_name, device_type, address_offset, data_count)
描述	將數據寫到設備中，不等待設備回應，逕自往下執行。 send_data、device_name、device_type、address_offset 和 data_count的說明和 StringSet 相同。

舉例	macro_command main()
	char str1[20]= "abcde"
	short test=0
	
	// 當 MODBUS 設備未回應，test = 1 將照常執行
	StringSetEx(str1[0], "MODBUS RTU", 4x, 0, 20)
	test = 1
	
	// 當 MODBUS 設備未回應，test = 2 將不會被執行，直到得到回應
	StringSet(str1[0], "MODBUS RTU", 4x, 0, 20)
	test = 2
	
	end macro_command

函數名稱	StringToUpper
語法	success = StringToUpper (source[start], destination[start]) success = StringToUpper ("source", destination[start])
描述	將 source 字串的字元全部轉換成大寫，並寫入 destination 字串。 source 字串可以為靜態字串 (如 "source") 或是一維字元陣列變數 (如 source[start])。 執行完畢會回傳一 bool 型態的值給 success 欄位。當執行成功，success等於 true，否則等於 false。source 字串長度大於目標陣列的大小，將會回傳 false。

舉例	macro_command main()
	char src1[20] = "aBcDe"
	char dest1[20]
	bool success1
	success1 = StringToUpper(src1[0], dest1[0])
	// success1 = true，dest1 = "ABCDE"
	
	char dest2[4]
	bool success2
	success2 = StringToUpper("aBcDe", dest2[0])
	// success2 = false，dest2 內容不變
	
	end macro_command

函數名稱	StringToLower
語法	success = StringToLower (source[start], destination[start]) success = StringToLower ("source", destination[start])
描述	將 source 字串的字元全部轉換成小寫，並寫入 destination 字串。 source 字串可以為靜態字串 (如 "source")或是一維字元陣列變數 (如 source[start])。 執行完畢會回傳一 bool 型態的值給 success 欄位。當執行成功，success等於 true，否則等於 false。source 字串長度大於目標陣列的大小，將會回傳 false。

舉例	macro_command main()
	char src1[20] = "aBcDe"
	char dest1[20]
	bool success1
	success1 = StringToLower (src1[0], dest1[0])
	// success1 = true，dest1 = “abcde”
	
	char dest2[4]
	bool success2
	success2 = StringToLower ("aBcDe", dest2[0])
	// success2 = false，dest2 內容不變
	
	end macro_command

函數名稱	StringToReverse
語法	success = StringToReverse (source[start], destination[start]) success = StringToReverse ("source", destination[start])
描述	將 source 字串反轉，並寫入 destination 字串。 source 字串可以為靜態字串 (如 "source") 或是一維字元陣列變數 (如 source[start])。 執行完畢會回傳一 bool 型態的值給 success 欄位。當執行成功，success等於 true，否則等於 false。source 字串長度大於目標陣列的大小，將會回傳 false。

舉例	macro_command main()
	char src1[20] = "abcde"
	char dest1[20]
	bool success1
	success1 = StringToReverse (src1[0], dest1[0])
	// success1 = true，dest1 = “edcba”
	
	char dest2[4]
	bool success2
	success2 = StringToReverse ("abcde", dest2[0])
	// success2 = false，dest2 內容不變
	
	end macro_command

函數名稱	StringTrimLeft
語法	success = StringTrimLeft (source[start], set[start], destination[start]) success = StringTrimLeft ("source", set[start], destination[start]) success = StringTrimLeft (source[start], "set", destination[start]) success = StringTrimLeft ("source", "set", destination[start])
描述	從 source 字串的第一個字元開始往後尋找，若找到與 set 字串相同的字元便裁剪掉該字元，直到遇到不存在於 set 字串中的字元為止。 source 字串與 set 字串皆可以為靜態字串 (如 "source") 或是一維字元陣列變數 (如 source[start])。

	執行完畢會回傳一 bool 型態的值給 success 欄位。當執行成功，success等於 true，否則等於 false。當裁剪完後的字串長度大於目標陣列的大小，將會回傳 false。

舉例	macro_command main()
	char src1[20] = "# *a*#bc"
	char set1[20] = "# *"
	char dest1[20]
	bool success1
	success1 = StringTrimLeft (src1[0], set1[0], dest1[0])
	// success1 = true，dest1 = "a*#bc"
	
	char set2[20] = {'#', ' ', '*'}
	char dest2[4]
	bool success2
	success2 = StringTrimLeft ("# *a*#bc", set2[0], dest2[0])
	// success2 = false，dest2 內容不變
	
	char src3[20] = "abc *#"
	char dest3[20]
	bool success3
	success3 = StringTrimLeft (src3[0], "# *", dest3[0])
	// success3 = true，dest3 = "abc *#"
	
	end macro_command

函數名稱	StringTrimRight
語法	success = StringTrimRight (source[start], set[start], destination[start]) success = StringTrimRight ("source", set[start], destination[start]) success = StringTrimRight (source[start], "set", destination[start]) success = StringTrimRight ("source", "set", destination[start])
描述	從 source 字串的最後一個字元開始往前尋找，若找到與 set 字串相同的字元便裁剪掉該字元，直到遇到不存在於 set 字串中的字元為止。 source 字串與 set 字串皆可以為靜態字串 (如 "source") 或是一維字元陣列變數 (如 source[start])。 執行完畢會回傳一 bool 型態的值給 success 欄位。當執行成功，success等於 true，否則等於 false。當裁剪完後的字串長度大於目標陣列的大小，將會回傳 false。
舉例	macro_command main() char src1[20] = "# *a*#bc# * " char set1[20] = "# *" char dest1[20] bool success1 success1 = StringTrimRight(src1[0], set1[0], dest1[0]) // success1 = true，dest1= "# *a*#bc" char set2[20] = {'#', ' ', '*'} char dest2[20]

	bool success2
	success2 = StringTrimRight("# *a*#bc", set2[0], dest2[0])
	// success2 = true，dest2 = "# *a*#bc"
	
	char src3[20] = "ab**c *#"
	char dest3[4]
	bool success3
	success3 = StringTrimRight(src3[0], "# *", dest3[0])
	// success3 = false，dest3 內容不變
	
	end macro_command

函數名稱	Unicode2Utf8
語法	result = Unicode2Utf8(source[start], destination[start])
描述	將source的Unicode字串轉換為Utf8字串，存放到destination。 執行完畢會回傳一 bool 型態的值給 result 欄位。當執行成功，result 等於 true，否則等於 false。
舉例	macro_command main() char unicode_str[20] char utf8_str[20] String2Unicode("ABC", unicode_str[0]) bool result result = Unicode2Utf8(unicode_str[0], utf8_str[0]) // "result" will be set to true. "utf8_str" will equal "ABC" encoded in UTF8 StringCat("DEF", utf8_str[0]) // "utf8_str" will equal "ABCDEF" encoded in UTF8 char dst[20] bool result2 result2 = Utf82Unicode(utf8_str[0], dst[0]) // "result" will be set to true. "dst" will equal "ABCDEF" encoded in Unicode. end macro_command

函數名稱	UnicodeCat
語法	result = UnicodeCat(source[start], destination[start]) 或 result = UnicodeCat("source", destination[start])
描述	利用此函數將來源字串銜接於目標字串之後。此函數執行成功後，目標字串將等於兩字串銜接後的結果。 來源字串source可以為靜態字串 (如 "source") 或是一維字元陣列變數 (如 source[start])。 destination[start]必須為一維字元陣列變數。

	執行完畢會回傳一 bool 型態的值給 result 欄位。當執行成功，result等於 true，否則等於 false。當兩字串銜接後的長度超過目標陣列的長度時，目標字串將保留銜接以前的內容，不做任何改變，並回傳 false。

舉例	macro_command main()
	
	char strSrc[12]="αθβγθδ"
	char strDest[28]="ζηθλ1234"
	bool result
	
	result = UnicodeCat(strSrc[0], strDest[0]) // "result" will be set to true
	//"strDest" will be set to "ζηθλ1234αθβγθδ"
	
	result = UnicodeCat("ζηθλ", strDest[0]) // the function fails.
	// "result" will be set to false due to insufficient destination buffer size.
	// In this case, the content of "strDest" remains the same.
	
	end macro_command

函數名稱	UnicodeCompare
語法	result = UnicodeCompare(string1[start], string2[start]) result = UnicodeCompare("string1", string2[start]) result = UnicodeCompare(string1[start], "string2") result = UnicodeCompare("string1", "string2")
描述	比較兩字串的內容是否相等。此函數將大小寫視為不同。 兩個傳入的字串皆可以為靜態字串 (如 "string")或是一維字元陣列變數 (如 string[start])。 執行完畢會回傳一 bool型態的值給 result 欄位。若兩字串相等，result 為 true，否則為 false。

舉例	macro_command main()
	char str1[10]="θαβθγ"
	char str2[8]="αβγδ"
	bool result
	
	result = UnicodeCompare(str1[0], str2[0]) // "result" will be set to false.
	result = UnicodeCompare(str1[0], "θαβθγ") // "result" will be set to true.
	
	end macro_command

函數名稱	UnicodeCopy
語法	result = UnicodeCopy("source", destination[start]) 或 result = UnicodeCopy(source[start], destination[start])
描述	利用此函數進行字串的複製。此函數可以將傳入的靜態字串 (以雙引號" "括起來的字串)，或是存放在字元陣列中的字串複製到目標陣列。 來源字串 sourc e可以為靜態字串 (如 "source")或是一維字元陣列變數 (如 source[start])。

	destination[start] 必須為一維字元陣列變數。 複製完畢會回傳一 bool 型態的值給 result 欄位。當複製成功，result等於 true，否則等於 false。當來源字串的長度大於目標陣列的大小時，將不做任何處理，並回傳 false 到 result 欄位。 result 欄位可寫可不寫。

舉例	macro_command main()
	char strSrc[14]="α β θ γ δ θ ε" //α β θ γ δ θ ε
	char strDest[14]
	bool result
	
	result = UnicodeCopy(strSrc[0], strDest[0]) // "result" will be set to true.
	result = UnicodeCopy("α β θ γ δ θ ε", strDest[0])
	// "result" will be set to true, strDest = α β θ γ δ θ ε"
	result = UnicodeCopy("α β γ δ ε ζ α β γ δ ε ζ", strDest[0])
	// "result" will be set to false.
	// The size of source string exceeds the size of destination string.
	
	end macro_command

函數名稱	UnicodeExcluding
語法	result = UnicodeExcluding(source[start], set[start], destination[start]) result = UnicodeExcluding("source", set[start], destination[start]) result = UnicodeExcluding(source[start], "set", destination[start]) result = UnicodeExcluding("source", "set", destination[start])
描述	提取 source 字串中某個以索引值 0 的字元 (第一個字元) 開頭的子字串，而且此子字串的每個字元必不存在於set字串中。此函數將會從 source字串的第一個字元開始尋找，直到找到存在於 set 字串中的字元為止。 source 字串與 set 字串皆可以為靜態字串 (如 "source") 或是一維字元陣列變數 (如 source[start])。 執行完畢會回傳一 bool 型態的值給 result 欄位。當執行成功，result等於 true，否則等於 false。當提取出來的字串長度大於目標陣列的大小，將會回傳 false。

舉例	macro_command main()
	char source[14]="γ δ ξ κ θ λ θ, dest[8]
	char set[4]="λ θ"
	bool result
	
	result = UnicodeExcluding(source[0], set[0], dest[0]) // the function succeeds.
	// "result" will be set to true and "dest" will be set to "γ δ ξ κ".
	
	result = UnicodeExcluding(source[0], set[0], dest[4]) // the function fails.
	// "result" will be set to false due to insufficient destination buffer size.
	
	end macro_command

函數名稱	UnicodeLength

語法	result = UnicodeLength(source[start]) 或 result = UnicodeLength("source")
描述	取得 Unicode 字串長度。 來源字串 source 可以為靜態字串 (如 "source") 或是一維字元陣列變數(如 source[start])。 函數回傳值代表來源 Unicode 字串的長度。

舉例	macro_command main()
	
	char strSrc[6]="ÅÈÑ"
	int result1, result2
	
	result1 = UnicodeLength(strSrc[0]) // "result1" is equal to 3
	result2 = UnicodeLength("ÅÈÑ") // "result2" is equal to 3
	
	end macro_command

函數名稱	Utf82Unicode
語法	result = Utf82Unicode(source[start], destination[start])
描述	將source的Utf8字串轉換為Unicode字串，存放到destination。 執行完畢會回傳一 bool 型態的值給 result 欄位。當執行成功，result 等於 true，否則等於 false。
舉例	macro_command main() char unicode_str[20] char utf8_str[20] String2Unicode("ABC", unicode_str[0]) bool result result = Unicode2Utf8(unicode_str[0], utf8_str[0]) // "result" will be set to true. "utf8_str" will equal "ABC" encoded in UTF8 StringCat("DEF", utf8_str[0]) // "utf8_str" will equal "ABCDEF" encoded in UTF8 char dst[20] bool result2 result2 = Utf82Unicode(utf8_str[0], dst[0]) // "result" will be set to true. "dst" will equal "ABCDEF" encoded in Unicode. end macro_command

請點選此圖示下載範例程式。下載範例程式前，請先確定已連上網路線。

18.7.8. 數學運算

函數名稱	SQRT
語法	SQRT(source, result)
描述	開平方根。資料來源 source 可以是常數或者變數，但是存放結果的 result 必須為變數。資料來源必須為一個正數。
舉例	macro_command main() float source, result SQRT(15, result) source = 9.0 SQRT(source, result)// 執行後 result = 3.0 end macro_command

函數名稱	CUBERT
語法	CUBERT (source, result)
描述	開三次方根。資料來源 source 可以是常數或者變數，但是存放結果的 result 必須為變數。資料來源必須為一個正數。

舉例	macro_command main()
	float source, result
	
	CUBERT (27, result) // 執行後 result = 3.0
	
	source = 27.0
	CUBERT (source, result) // 執行後 result = 3.0
	
	end macro_command

函數名稱	POW
語法	POW (source1, source2, result)
描述	計算 source1 的某次方 (source2) 。資料來源 source1 和 source2 可以是常數或者變數，但是存放結果的 result 必須為變數。資料來源必須為一個正數。
舉例	macro_command main() float y, result y = 0.5 POW (25, y, result) // 執行後 result = 5 end macro_command

函數名稱	SIN
語法	SIN(source, result)
描述	三角函數的正弦計算。資料來源 source 可以是常數或者變數，但是存放結果的 result 必須為變數。
舉例	macro_command main()

	float source, result
	
	SIN(90, result) // result is 1
	
	source = 30
	SIN(source, result) // result is 0.5
	
	end macro_command

函數名稱	COS
語法	COS(source, result)
描述	三角函數的餘弦計算。資料來源 source 可以是常數或者變數，但是存放結果的 result 必須為變數。

舉例	macro_command main()
	float source, result
	
	COS(90, result) // result is 0
	
	source = 60
	COS(source, result) // result is 0.5
	
	end macro_command

函數名稱	TAN
語法	TAN(source, result)
描述	三角函數的正切計算。資料來源 source 可以是常數或者變數，但是存放結果的 result 必須為變數。

舉例	macro_command main()
	float source, result
	
	TAN(45, result) // result is 1
	
	source = 60
	TAN(source, result) // result is 1.732
	
	end macro_command

函數名稱	COT
語法	COT(source, result)
描述	三角函數的餘切計算。資料來源 source 可以是常數或者變數，但是存放結果的 result 必須為變數。
舉例	macro_command main() float source, result COT(45, result) // result is 1

	source = 60 COT(source, result) // result is 0.5774 end macro_command

函數名稱	SEC
語法	SEC(source, result)
描述	三角函數的正割計算。資料來源 source 可以是常數或者變數，但是存放結果的 result 必須為變數。

舉例	macro_command main()
	float source, result
	
	SEC(45, result) // result is 1.414
	
	source = 60
	SEC(source, result) // if source is 60, result is 2
	
	end macro_command

函數名稱	CSC
語法	CSC(source, result)
描述	三角函的餘割計算。資料來源 source 可以是常數或者變數，但是存放結果的 result 必須為變數。

舉例	macro_command main()
	float source, result
	
	CSC(45, result) // result is 1.414
	
	source = 30
	CSC(source, result) // result is 2
	
	end macro_command

函數名稱	ASIN
語法	ASIN(source, result)
描述	反三角函數中反正弦計算。資料來源 source 可以是常數或者變數，但是存放結果的 result 必須為變數。

舉例	macro_command main()
	float source, result
	
	ASIN(0.8660, result) // result is 60
	
	source = 0.5
	ASIN(source, result) // result is 30

	end macro_command

函數名稱	ACOS
語法	ACOS(source, result)
描述	反三角函數中反餘弦計算。資料來源 source 可以是常數或者變數，但是存放結果的 result 必須為變數。

舉例	macro_command main()
	float source, result
	
	ACOS(0.8660, result) // result is 30
	
	source = 0.5
	ACOS(source, result) // result is 60
	
	end macro_command

函數名稱	ATAN
語法	ATAN(source, result)
描述	反三角函數中反正切計算。資料來源 source 可以是常數或者變數，但是存放結果的 result 必須為變數。

舉例	macro_command main()
	float source, result
	
	ATAN(1, result) // result is 45
	
	source = 1.732
	ATAN(source, result) // result is 60
	
	end macro_command

函數名稱	LOG
語法	LOG (source, result)
描述	取得 source 的自然對數，並存入 result 變數。 source 可為變數或常數。 存放結果的 result 必須為變數。
舉例	macro_command main() float source=100, result LOG (source, result) // result 約等於 4.6052 end macro_command

函數名稱	LOG10
語法	LOG10 (source, result)
描述	取得 source 的以 10 為基底的對數，並存入 result 變數。 source 可為變數或常數。

	存放結果的 result 必須為變數。
舉例	macro_command main() float source=100, result LOG10 (source, result) // result 等於 2 end macro_command

函數名稱	RAND
語法	RAND(result)
描述	產生一個隨機數。(範圍: 0 ~ 32766) 存放結果的 result 必須為變數。
舉例	macro_command main() short result RAND (result) // result is not a fixed value when executes macro every time end macro_command

函數名稱	CEIL
語法	result=CEIL(source)
描述	取得不小於 source 的最小整數值。
舉例	macro_command main() float x = 3.8 int result result = CEIL(x) // result = 4 end macro_command

函數名稱	FLOOR
語法	result=FLOOR(source)
描述	取得不大於 source 的最大整數值。
舉例	macro_command main() float x = 3.8 int result result = FLOOR(x) // result = 3 end macro_command

函數名稱	ROUND
語法	result=ROUND(source)
描述	取得最接近 source 的整數數值。

舉例	macro_command main() float x = 5.55 int result result = ROUND(x) // result = 6 end macro_command

 請點選此圖示下載範例程式。下載範例程式前，請先確定已連上網路線。

18.7.9. 統計

函數名稱	AVERAGE
語法	AVERAGE(source[start], result, count)
描述	從陣列中計算平均值

舉例	int data[5] = {1, 2, 3, 4, 5}
	float result
	
	AVERAGE(data[0], result, 5) // result 等於3
	AVERAGE(data[2], result, 3) // result 等於4

函數名稱	HARMEAN
語法	HARMEAN(source[start], result, count)
描述	從陣列中計算調和平均值
舉例	int data[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} float result HARMEAN(data[0], result, 10) // result 等於3.414

函數名稱	MAX
語法	MAX(source[start], result, count)
描述	從陣列中取最大值

舉例	int data[5] = {1, 2, 3, 4, 5}
	int result
	
	MAX(data[0], result, 5) // result 等於5
	MAX(data[1], result, 3) // result 等於4

函數名稱	MEDIAN
語法	MEDIAN(source[start], result, count)
描述	從陣列中取中位數
舉例	int data[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} float result MEDIAN(data[0], result, 10) // result 等於 5.5

函數名稱	MIN
語法	MIN(source[start], result, count)
描述	從陣列中取最小值
舉例	int data[5] = {1, 2, 3, 4, 5} int result

	MIN(data[0], result, 5) // result 等於 1
	MIN(data[1], result, 3) // result 等於 2

函數名稱	STDEVP
語法	STDEVP(source[start], result, count)
描述	從陣列中計算標準差
舉例	nt data[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} float result STDEVP(data[0], result, 10) // result 等於2.872

函數名稱	STDEVS
語法	STDEVS(source[start], result, count)
描述	從陣列中計算樣本標準差
舉例	int data[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} float result STDEVS(data[0], result, 10) // result 等於 3.027


請點選此圖示下載範例程式。下載範例程式前，請先確定已連上網路線。

18.7.10. 配方資料庫

函數名稱	RecipeGetData
語法	RecipeGetData (destination, recipe_address, record_ID)
描述	取得配方中的資料。所取得的資料存放在destination且必須為變數。 recipe_address由配方名稱與項目名稱組成："recipe_name.item_name"。 record_ID指定欲取得配方中的第幾筆資料，也就是資料列編號。

舉例	macro_command main()
	int data = 0
	char str[20]
	int recordID
	bool result
	
	recordID = 0
	result = RecipeGetData(data, "TypeA.item_weight", recordID)
	// 從配方 "TypeA" 中取得第0筆，且item name為 "item_weight" 的資料
	
	recordID = 1
	result = RecipeGetData(str[0], "TypeB.item_name", recordID)
	// 從配方 "TypeB" 中取得第1筆，且 item name為 "item_name" 的資料
	
	end macro_command

函數名稱	RecipeQuery
語法	RecipeQuery (SQL command, destination)
描述	透過SQL語句，對配方中的資料進行查詢。查詢結果的資料筆數存放在 destination，必須為變數。SQL command 的部份可為靜態文字或傳入字元陣列，例如： RecipeQuery("SELECT * FROM TypeA", destination) 或 RecipeQuery(sql[0], destination) SQL 語句必須以 "SELECT * FROM" 開頭，後面接配方名稱及查詢條件。

舉例	macro_command main()
	
	int total_row = 0
	char sql[100] = "SELECT * FROM TypeB"
	short var
	bool result
	
	result = RecipeQuery("SELECT * FROM TypeA", total_row)
	// 查詢配方 "TypeA"。查詢結果的資料筆數存放在 total_row.
	
	result = RecipeQuery(sql[0], total_row)
	// 查詢配方 "TypeB"。查詢結果的資料筆數存放在 total_row.
	
	result = RecipeQuery("SELECT * FROM Recipe WHERE Item >%(var)", total_row)
	// 查詢配方 "Recipe" 裡的 "Item" 欄位數據大於 var 的資料筆數。查詢結果的資料筆數存放在 total_row.
	
	end macro_command

函數名稱	RecipeQueryGetData
語法	RecipeQueryGetData (destination, recipe_address, result_row_no)
描述	取得 RecipeQuery 的查詢結果。呼叫此函數前必須先呼叫 RecipeQuery，且 recipe_address 中需指定與 RecipeQuery 相同的配方名稱。 result_row_no 指定欲取得查詢結果中的第幾筆資料。

舉例	macro_command main()
	
	int data = 0
	int total_row = 0
	int row_number = 0
	bool result_query
	bool result_data
	
	result_query = RecipeQuery("SELECT * FROM TypeA", total_row)
	// 查詢配方 "TypeA"。查詢結果的資料筆數存放在 total_row.
	
	if (result_query) then

	for row_number = 0 to total_row - 1 result_data = RecipeQueryGetData(data, "TypeA.item_weight", row_number) next row_number end if end macro_command

函數名稱	RecipeQueryGetRecordID
語法	RecipeQueryGetRecordID (destination, result_row_no)
描述	取得 RecipeQuery 查詢結果的資料列編號。呼叫此函數前必須先呼叫 RecipeQuery。 result_row_no 指定欲取得查詢結果中的第幾筆資料的資料列編號，並將資料列編號寫入 destination。

舉例	macro_command main()
	
	int recordID = 0
	int total_row = 0
	int row_number = 0
	bool result_query
	bool result_id
	
	result_query = RecipeQuery("SELECT * FROM TypeA", total_row)
	// 查詢配方 "TypeA"。查詢結果的資料筆數存放在 total_row.
	
	if (result_query) then
	for row_number = 0 to total_row - 1
	result_id = RecipeQueryGetRecordID(recordID, row_number)
	next row_number
	end if
	
	end macro_command

函數名稱	RecipeSetData
語法	RecipeSetData(*source*, *recipe_address*, *record_ID*)
描述	將資料寫入配方資料庫中。如果成功將傳回 True，否則將傳回 False。 recipe_address 由配方名稱與項目名稱組成："recipe_name.item_name"。 record_ID 指定欲修改配方中的第幾筆資料，也就是資料列編號。
舉例	macro_command main() int data=99 char str[20]="abc" int recordID bool result recordID = 0 result = RecipeSetData(data, "TypeA.item_weight", recordID)

	// set data to recipe "TypeA", where item name is "item_weight" and the record ID is 0.
	
	recordID = 1
	result = RecipeSetData(str[0], "TypeB.item_name", recordID)
	// set data to recipe "TypeB", where item name is "item_name" and the record ID is 1.
	
	end macro_command

函數名稱	RecipeTransactionBegin
語法	RecipeTransactionBegin ()
描述	開啟批次寫入配方的功能。需配合RecipeTransactionCommit或RecipeTransactionRollback進行使用。 RecipeTransactionBegin 到 RecipeTransactionCommit之間的所有Recipe寫入動作將會等到Commit命令後一次執行。 RecipeTransactionBegin 到 RecipeTransactionRollback之間的所有Recipe寫入動作將會等到Rollback命令後全數取消。 **警告提示** 若在巨集結束前，沒有呼叫RecipeTransactionCommit或RecipeTransactionRollback，則系統將會自動呼叫RecipeTransactionRollback進行寫入取消，並在cMT診斷器上出現以下警示：DB Transaction ended without commit/rollback, and rolled back all changes automatically. 若重複呼叫RecipeTransactionBegin()，則會在cMT診斷器上出現以下警示：Cannot start a transaction within a transaction. **注意** 當使用RecipeTransactionBegin時，請盡量縮短RecipeTransactionBegin至RecipeTransactionCommit/RecipeTransactionRollback的執行時間，避免其他物件同時操作配方資料庫造成系統異常。

舉例	macro_command main()
	int data=99
	char str[20]="abc"
	int recordID = 0
	bool result
	result = RecipeSetData(data, "TypeA.item_weight", recordID)
	//將資料寫入配方“TypeA”其中的欄位“item_weight”，Record ID為0。
	RecipeTransactionBegin()
	recordID = 1
	result = RecipeSetData(str[0], "TypeB.item_name", recordID)
	RecipeTransactionCommit()
	//將資料寫入配方“TypeB”其中的欄位“item_name”，Record ID為1。

	RecipeTransactionBegin() recordID = 2 result = RecipeSetData(str[0], "TypeB.item_name", recordID) RecipeTransactionRollback() //因為取消批次寫入配方，所以Record ID仍為1。 end macro_command

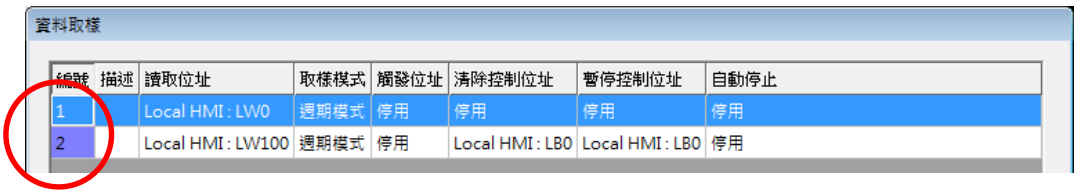
函數名稱	RecipeTransactionCommit
語法	RecipeTransactionCommit ()
描述	執行批次寫入配方。需配合RecipeTransactionBegin使用。 RecipeTransactionBegin 到 RecipeTransactionCommit之間的所有Recipe寫入動作將會等到Commit命令後一次執行。 **警告提示** 若在沒有呼叫RecipeTransactionBegin就先呼叫RecipeTransactionCommit，則系統將會在cMT診斷器上出現以下警示：Cannot commit - no transaction is active.
舉例	請參考RecipeTransactionBegin範例。

函數名稱	RecipeTransactionRollback
語法	RecipeTransactionRollback ()
描述	取消執行批次寫入配方。需配合RecipeTransactionBegin使用。 RecipeTransactionBegin 到 RecipeTransactionRollback之間的所有Recipe寫入動作將會等到Rollback命令後全數取消。 **警告提示** 若在沒有呼叫RecipeTransactionBegin就先呼叫RecipeTransactionRollback，則系統將會在cMT診斷器上出現以下警示：Cannot rollback - no transaction is active.
舉例	請參考RecipeTransactionBegin範例。

請點選此圖示下載範例程式。下載範例程式前，請先確定已連上網路線。

18.7.11. 資料取樣/事件記錄

函 數名稱	FindDataSamplingDate
語法	return_value = FindDataSamplingDate (data_log_number, index, year, month, day) or FindDataSamplingDate (data_log_number, index, year, month, day)
描述	利用此函數查詢資料取樣檔的日期。根據所輸入的資料取樣編號 (data_log_number) 與資料取樣檔案索引 (index) 可查詢該資料取樣的日期，依照年、月、日的順序寫入 year、month、day 的欄位中。



資料取樣檔的儲存方式為：[保存位置] \ [取樣資料夾名稱] \ yyyymmdd.dtl。而資料取樣檔案索引 (index) 指的是取樣資料夾下面所有資料取樣檔案依檔名排序後的順位值 (從0開始)。Index 值越小者，日期越新。例如假設某取樣資料夾下面有四個資料取樣檔：

20101210.dtl
20101230.dtl
20110110.dtl
20110111.dtl

則index依序為：

20101210.dtl -> index 為 3
20101230.dtl -> index 為 2
20110110.dtl -> index 為 1
20110111.dtl -> index 為 0

當成功搜尋到所欲查詢的資料取樣檔時，將回傳 1 至 return_value，否則將回傳 0。

data_log_number 與 index 可以為常數或是變數，但 year、month、day與 return_value 必須為變數。

return_value 為可選。

舉例

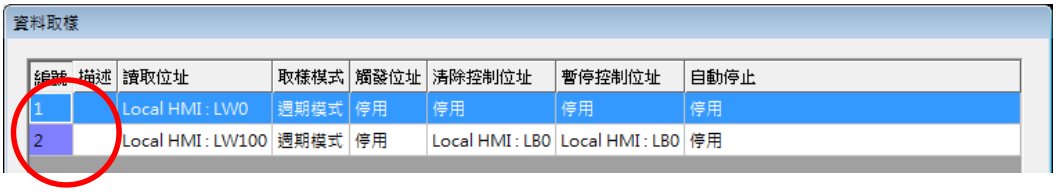
```
macro_command main()
short data_log_number = 1, index = 2, year, month, day
short success

// 若存在一資料取樣檔 20101230.dtl，其資料取樣編號為 1，檔案索引為 2
// 則success == 1，year == 2010，month == 12，day == 30
success = FindDataSamplingDate(data_log_number, index, year, month, day)

end macro_command
```

請點選此圖示下載範例程式。下載範例程式前，請先確定已連上網路線。

函數名稱	FindDataSamplingIndex
語法	return_value = FindDataSamplingIndex (data_log_number, year, month, day, index) or FindDataSamplingIndex (data_log_number, year, month, day, index)
描述	利用此函數查詢資料取樣檔的檔案索引值。根據所輸入的資料取樣編號 (data_log_number) 與日期可查詢該資料取樣檔的檔案索引，並將其寫入 index。year、month、day 三個欄位依序代表年、月、日，其輸入格式為 YYYY (年)、MM (月)、DD (日)。

	 資料取樣檔的儲存方式為：[保存位置] \ [取樣資料夾名稱] \ yyyymmdd.dtl。而資料取樣檔案索引 (index) 指的是取樣資料夾下面所有資料取樣檔案依檔名排序後的順位值 (從 0 開始)。Index 值越小者，日期越新。例如假設某取樣資料夾下面有四個資料取樣檔： 20101210.dtl 20101230.dtl 20110110.dtl 20110111.dtl 則 index 依序為： 20101210.dtl -> index 為 3 20101230.dtl -> index 為 2 20110110.dtl -> index 為 1 20110111.dtl -> index 為 0 當成功搜尋到所欲查詢的資料取樣檔時，將回傳 1 至 return_value，否則將回傳 0。 data_log_number 與 year、month、day 可以為常數或是變數，但 index 與 return_value 必須為變數。 return_value 為可選。

舉例	macro_command main()
	short data_log_number = 1, year = 2010, month = 12, day = 10, index
	short success
	
	// 若存在一資料取樣檔 20101210.dtl，其資料取樣編號為 1，檔案索引為 2
	// 則success == 1，index == 2
	success = FindDataSamplingIndex (data_log_number, year, month, day, index)
	
	end macro_command

請點選此圖示下載範例程式。下載範例程式前，請先確定已連上網路線。

函數名稱	FindEventLogDate
語法	return_value = FindEventLogDate (index, year, month, day) or FindEventLogDate (index, year, month, day)
描述	利用此函數查詢事件登錄檔的日期。根據所輸入的事件登錄檔案索引(index)可查詢該事件登錄的日期，依照年、月、日的順序寫入 year、month、day的欄位中。

	事件登錄檔案索引 (index) 指的是指定的保存位置 (HMI、SD 卡或 USB)下事件登錄檔案依檔名排序後的順位值 (從 0 開始)。Index 值越小者，日期越新。例如假設有四個事件登錄檔： EL_20101210.evt EL_20101230.evt EL_20110110.evt EL_20110111.evt 則index依序為： EL_20101210.evt -> index為3 EL_20101230.evt -> index為2 EL_20110110.evt -> index為1 EL_20110111.dtl -> index為0 當成功搜尋到所欲查詢的事件登錄檔時，將回傳 1 至return_value，否則將回傳 0。 Index 可以為常數或是變數，但 year、month、day與return_value 必須為變數。 return_value 為可選。

舉例	macro_command main()
	short index = 1, year, month, day
	short success
	
	// 若存在一事件登錄檔 EL_20101230.evt，檔案索引為 1
	// 則 success == 1，year == 2010，month == 12，day == 30
	success = FindEventLogDate (index, year, month, day)
	
	end macro_command

請點選此圖示下載範例程式。下載範例程式前，請先確定已連上網路線。

函數名稱	FindEventLogIndex
語法	return_value = FindEventLogIndex (year, month, day, index) or FindEventLogIndex (year, month, day, index)
描述	利用此函數查詢事件登錄檔的檔案索引值。根據所輸入的日期可查詢該事件登錄檔的檔案索引，並將其寫入 index。year、month、day 三個欄位依序代表年、月、日，其輸入格式為 YYYY (年)、MM (月)、DD (日)。 事件登錄檔案索引 (index) 指的是指定的保存位置 (HMI、SD 卡或 USB)下事件登錄檔案依檔名排序後的順位值 (從 0 開始)。Index 值越小者，日期越新。例如假設有四個事件登錄檔： EL_20101210.evt EL_20101230.evt EL_20110110.evt EL_20110111.evt 則 index 依序為： EL_20101210.evt -> index 為 3

	EL_20101230.evt -> index 為 2 EL_20110110.evt -> index 為 1 EL_20110111.dtl -> index 為 0 當成功搜尋到所欲查詢的事件登錄檔時，將回傳 1 至 return_value，否則將回傳 0。 year、month、day 可以為常數或是變數，但 index 與 return_value 必須為變數。 return_value 為可選。

舉例	macro_command main()
	short year = 2010, month = 12, day = 10, index
	short success
	
	// 若存在一事件登錄檔 EL_20101210.evt，檔案索引為 2
	// 則success == 1，index == 2
	success = FindEventLogIndex (year, month, day, index)
	
	end macro_command

請點選此圖示下載範例程式。下載範例程式前，請先確定已連上網路線。

18.7.12. 校驗

函數名稱	ADDSUM
語法	ADDSUM(source[start], result, data_count)
描述	將 source[start] 到 source[start+data-count-1] 的所有一維陣列的資料累加起來，以獲得 checksum (校驗和)，並將結果存放在 result 變數中。 result 必須為變數，data_count 是進行累加的資料的個數，可以是常數或者是變數。
舉例	macro_command main() char data[5] short checksum data[0] = 0x1 data[1] = 0x2 data[2] = 0x3 data[3] = 0x4 data[4] = 0x5 ADDSUM(data[0], checksum, 5) // checksum is 0xf end macro_command

函數名稱	XORSUM
語法	XORSUM(source[start], result, data_count)
描述	將 source[start] 到 source[start+data_count-1] 的所有一維陣列的資料進行異或運算，以獲得 checksum (校驗和)，並將結果存放在 result 變數中。 result 必須為變數，data_count 是進行異或計算的資料的個數，可以是常數或者是

	變數。
舉例	macro_command main() char data[5] = {0x1, 0x2, 0x3, 0x4, 0x5} short checksum XORSUM(data[0], checksum, 5) // checksum is 0x1 end macro_command

函數名稱	BCC
語法	BCC(source[start], result, data_count)
描述	等同 XORSUM 函數。
舉例	macro_command main() char data[5] = {0x1, 0x2, 0x3, 0x4, 0x5} char checksum BCC(data[0], checksum, 5) // checksum is 0x1 end macro_command

函數名稱	CRC
語法	CRC(source[start], result, data_count)
描述	將 source[start] 到 source[start+data-count-1] 的所有一維陣列的資料進 16-bit CRC 計算，以獲得 checksum (校驗和)，並將結果存放在 result 變數中。 result 必須為變數，data_count 是進行計算的資料的個數，可以是常數或者是變數。
舉例	macro_command main() char data[5] = {0x1, 0x2, 0x3, 0x4, 0x5} short checksum CRC(data[0], checksum, 5) // checksum is 0xbb2a, 16-bit CRC end macro_command

函數名稱	CRC8
語法	CRC8(source[start], result, data_count)
描述	將 source[start] 到 source[start+data-count-1] 的所有一維陣列的資料進 8-bit CRC 計算，以獲得 checksum (校驗和)，並將結果存放在 result 變數中。 result 必須為變數，data_count 是進行計算的資料的個數，可以是常數或者是變數。

舉例	macro_command main()
	  char source[5] = {1, 2, 3, 4, 5}
	  short CRC8_result
	
	  CRC8(source[0], CRC8_result, 5)
	  // CRC8_result = 188
	end macro_command

函數名稱	CRC16_CCITT
語法	CRC16_CCITT(source[start], result, data_count)
描述	將 source[start] 到 source[start+data-count-1] 的所有一維陣列的資料進 16-bit CRC 計算，以獲得 CRC-16/CCITT 的 checksum (校驗和)，並將結果存放在 result 變數中。 result 必須為變數，data_count 是進行計算的資料的個數，可以是常數或者是變數。
舉例	macro_command main() char source[5] = “12345” short crc_result CRC16_CCITT(source[0], crc_result, 5) //crc_result = 0xA5A2 end macro_command

函數名稱	CRC16_CCITT_FALSE
語法	CRC16_CCITT_FALSE (source[start], result, data_count)
描述	將 source[start] 到 source[start+data-count-1] 的所有一維陣列的資料進 16-bit CRC 計算，以獲得 CRC-16/CCITT-FALSE 的 checksum (校驗和)，並將結果存放在 result 變數中。 result 必須為變數，data_count 是進行計算的資料的個數，可以是常數或者是變數。
舉例	macro_command main() char source[5] = “12345” short crc_result CRC16_CCITT_FALSE(source[0], crc_result, 5) //crc_result = 0x4560 end macro_command

函數名稱	CRC16_X25
語法	CRC16_X25 (source[start], result, data_count)
描述	將 source[start] 到 source[start+data-count-1] 的所有一維陣列的資料進 16-bit CRC 計算，以獲得 CRC-16/X25 的 checksum (校驗和)，並將結果存放在 result 變數中。 result 必須為變數，data_count 是進行計算的資料的個數，可以是常數或者是變數。
舉例	macro_command main() char source[5] = “12345” short crc_result CRC16_X25(source[0], crc_result, 5) //crc_result = 0xBB40 end macro_command

函數名稱	CRC16_XMODEM
語法	CRC16_ XMODEM (source[start], result, data_count)
描述	將 source[start] 到 source[start+data-count-1] 的所有一維陣列的資料進 16-bit CRC 計算，以獲得 CRC-16/XMODEM 的 checksum (校驗和)，並將結果存放在 result 變數

	中。 result 必須為變數，data_count 是進行計算的資料的個數，可以是常數或者是變數。
舉例	macro_command main() char source[5] = “12345” short crc_result CRC16_ XMODEM(source[0], crc_result, 5) //crc_result = 0x546C end macro_command

18.7.13. 其他

函數名稱	Beep
語法	Beep ()
描述	發出系統警示音。 此函數可發出 800 赫茲，30 毫秒的系統警示音。
舉例	macro_command main() Beep() end macro_command

函數名稱	Buzzer
語法	Buzzer (state)
描述	開啟 / 關閉 蜂鳴器

舉例	macro_command main()
	
	char on = 1, off = 0
	Buzzer(on) // turn on the buzzer
	
	DELAY(1000) // delay 1 second
	
	Buzzer(off) // turn off the buzzer
	
	DELAY(500) // delay 500ms
	
	Buzzer(1) // turn on the buzzer
	
	DELAY(1000) // delay 1 second
	
	Buzzer(0) // turn off the buzzer
	
	end macro_command

函數名稱	TRACE
語法	TRACE(format, argument)

描述	一個執行中的 巨集指令可以使用此函數，監視變數內容的變化，並列印字串，以協助除錯。使用者應開啟 EasyDiagnoser/cMT Diagnoser 觀看此函數的輸出結果。 當 TRACE 函數抓取一個%開頭的特殊字元，將同時從 argument 抓取一個參數做格式化後輸出。 Format 代表列印格式，支援 % 開頭的特殊字元。特殊字元格式如下，其中方括號內的欄位為可選，紅色文字欄位為必需： %[flags] [width] [.precision] type 每個欄位的意義如下所述： flags (可選)： - ：靠左對齊。指定寬度後，不足的左邊補空格 + ：輸出正負號 width (可選)： 十進位正整數，指定應預留的字元寬度，不足部份補空白字元。 precision (可選)： 十進位正整數，指定精確度，以及輸出字元數。 type： C 或 c ： 以字元方式輸出 d ： 以 signed 十進位整數輸出 i ： 以 signed 十進位整數輸出 o ： 以 unsigned 八進位整數輸出 u ： 以 unsigned 十進位整數輸出 X 或 x ： 以 unsigned 十六進位整數輸出 lld ： 以長整數 (64bit) signed 變數輸出 (只支援於cMT / cMT X) llu ： 以長整數 (64bit) unsigned 變數輸出 (只支援於cMT / cMT X) f ： 以單倍精確度浮點數輸出。 llf ： 以雙倍精確度浮點數輸出。 E 或 e ： 以科學表示法輸出。格式為 “[-] d.dddd e[sign]ddd” 。其中欄位d是十進位數字，欄位dddd是一至多個十進位數字，欄位ddd必須是三個十進位數字。sign是+或-。 *Format* 字串最長支援 256 個字元，多出的字元將被忽略。 *Argument* 部份可寫可不寫。但一個特殊字元應搭配一個變數。

舉例	macro_command main()
	char c1 = 'a'
	short s1 = 32767
	float f1 = 1.234567
	
	TRACE("The results are") // 輸出：The results are
	TRACE("c1 = %c, s1 = %d, f1 = %f", c1, s1, f1)
	// 輸出：c1 = a, s1 = 32767, f1 = 1.234567
	
	end macro_command

請點選此圖示下載範例程式。下載範例程式前，請先確定已連上網路線。

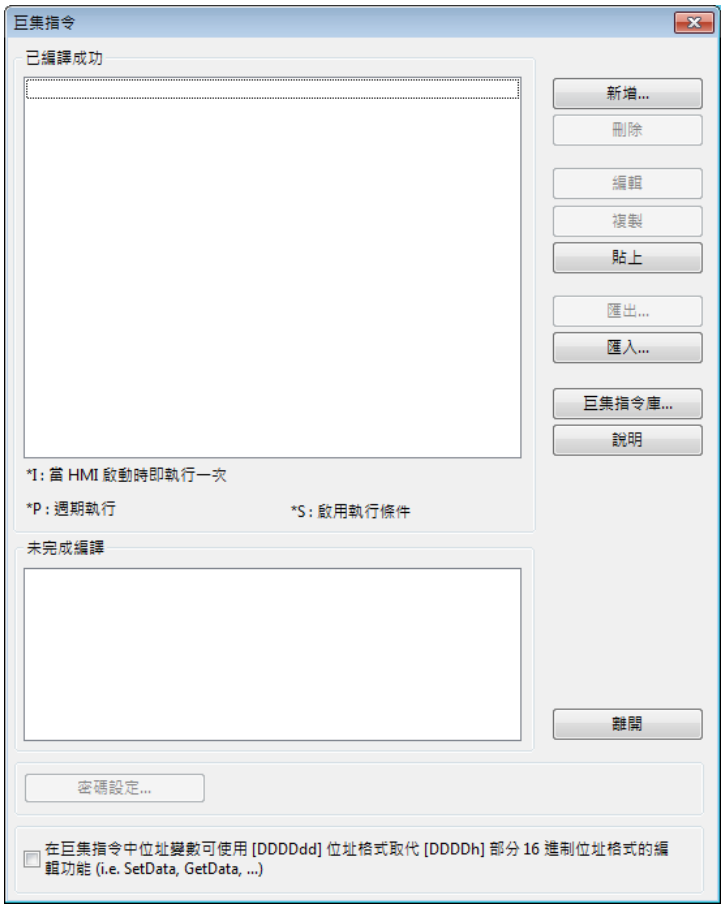
函數名稱	GetCnvTagArrayIndex
語法	GetCnvTagArrayIndex(array_index)
描述	使用者定義標籤使用 [讀取轉換] 並為陣列時，在巨集副函數中，使用此函數可取得陣列的索引。 當資料取樣使用 [讀取轉換] 時，可用來處理陣列索引的數據後再取樣。
舉例	Sub short newfun(short param) Int index GetCnvTagArrayIndex(index) //如果index的數據是2，表示將使用 [讀取轉換] 陣列中的第三個資料 return param end sub

18.8. 怎樣建立和執行巨集指令

18.8.1. 怎樣建立一個巨集指令

按照以下步驟以建立一個巨集指令。

1. 點選 [工程檔案]上的巨集指令圖示  打開巨集指令管理對話窗。

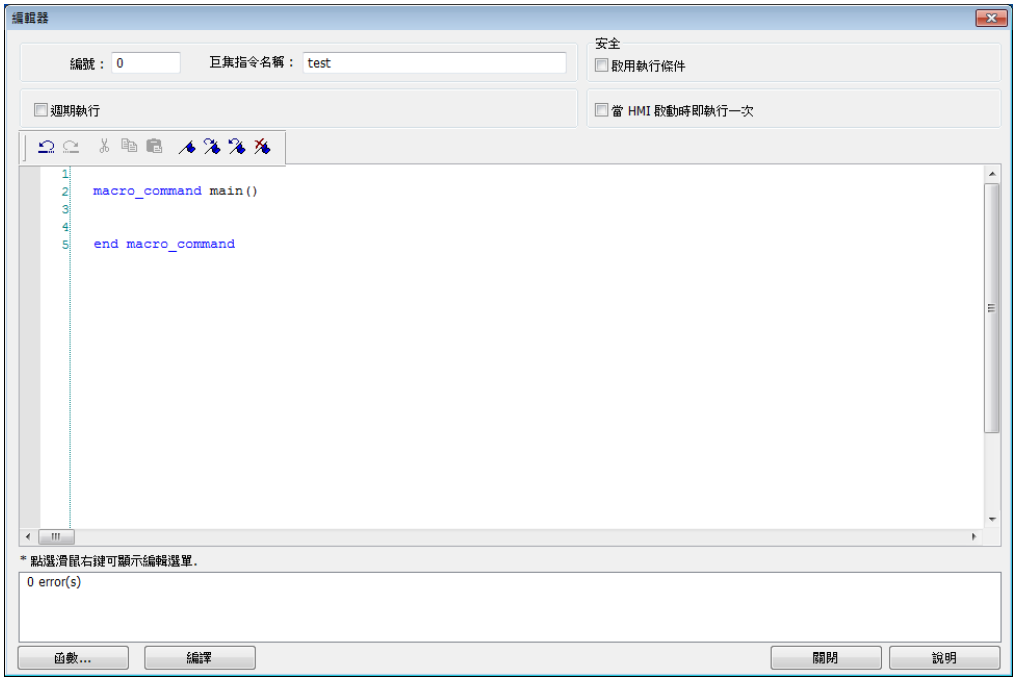


在巨集指令管理對話方塊中，已經編譯成功的巨集指令會出現在 [已編譯成功] 列表上，未完成編譯的會出現在 [未完成編譯] 列表中。下面是巨集指令管理對話方塊中各按鍵的功能描述。

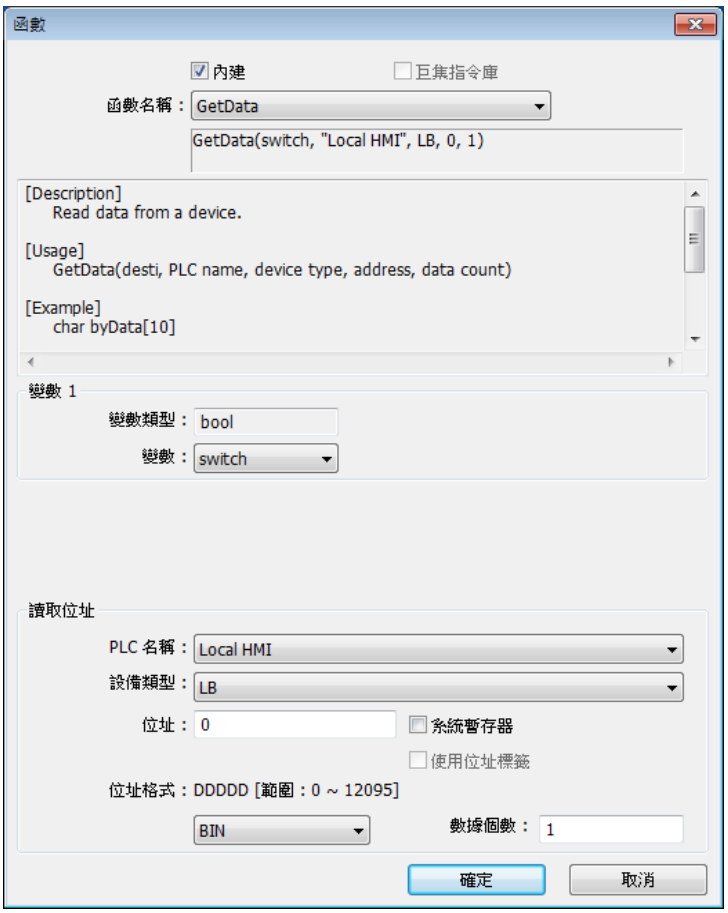
設定	描述
新增	新增一個巨集指令，並開啟新建的巨集指令編輯區。
刪除	刪除選擇的巨集指令。
編輯	打開巨集指令編輯區，並開啟選擇的巨集指令。
複製	複製選擇的巨集指令。
貼上	將剛剛選擇需要複製的巨集指令，貼至“已編譯完成”列表區，並產生一個新的巨集指令名稱。

匯出	將勾選的巨集指令儲存為*.ebm 檔案。
匯入	將*.ebm 檔案匯入至此工程檔案。
確認	確認此次所編輯的所有巨集指令後離開此巨集指令管理對話方塊，必須按此鍵才會儲存　此次編輯內容。
取消	取消此次所編輯的所有巨集指令，離開巨集指令管理對話方塊。
巨集指令庫	進入巨集函數庫管理對話方塊。

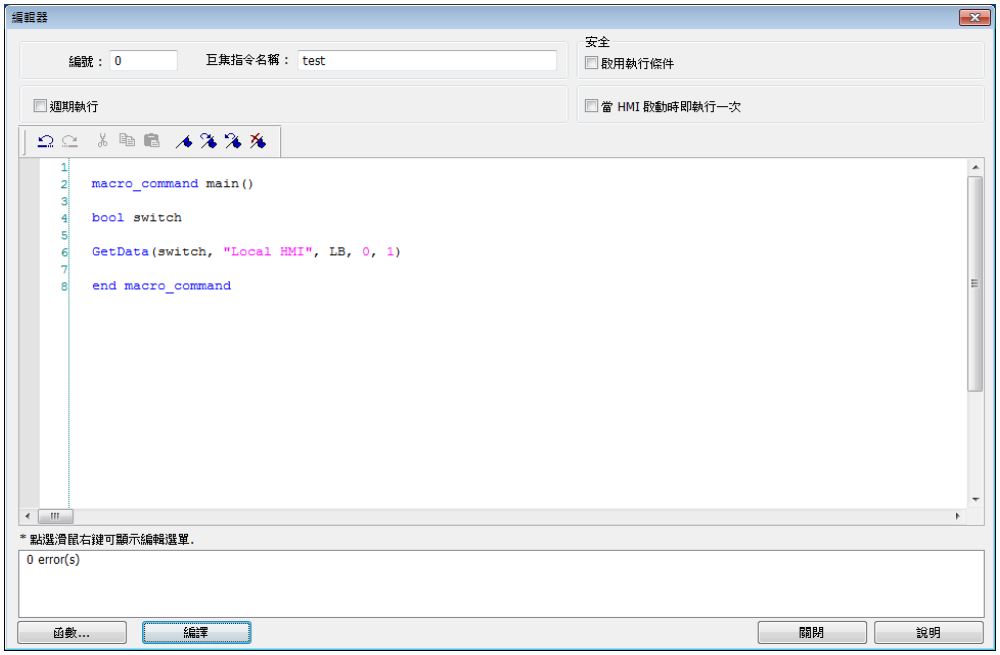
2. 按下 [新增] 按鈕，打開一個新增的巨集指令編輯區。每一個巨集指令都有一個唯一的編號，定義在 [編號] 這個位置。在 [巨集指令名稱] 這個欄目中也必須輸入巨集指令的名稱，否則編譯將無法通過。



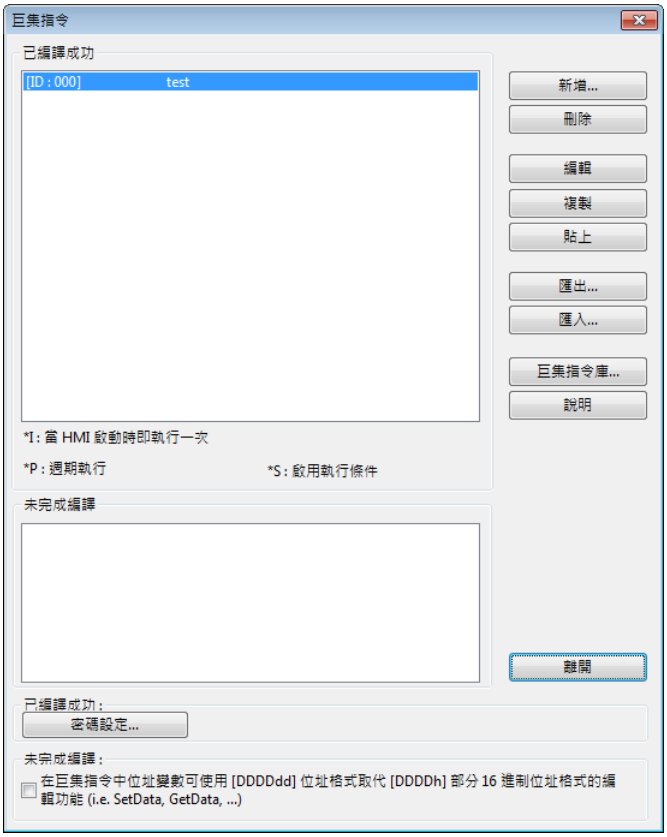
3. 設計屬於您的巨集指令程式。如果要使用內建的函數，譬如 Setdata() 或者 Getdata() 等函數，單擊 [函數] 按鍵開啟函數列表對話方塊，選擇需要的函數，並設定必要的參數。



4. 編輯完成一個新建的巨集指令程式後，單擊 [編譯] 按鍵，對該巨集指令進行編譯工作。



5. 如果沒有錯誤，單擊 [關閉] 按鍵，這樣在 [已編譯成功] 區會發現新增了一個 “test” 這個名稱的巨集指令。


巨集指令
已編譯成功
[ID : 000] test
新增...
刪除
編輯
複製
貼上
匯出...
匯入...
巨集指令庫...
說明
*I : 當 HMI 啟動時即執行一次
*P : 週期執行
*S : 啟用執行條件
未完成編譯
離開
已編譯成功 :
密碼設定...
未完成編譯 :
在巨集指令中位址變數可使用 [DDDDdd] 位址格式取代 [DDDDh] 部分 16 進制位址格式的編輯功能 (i.e. SetData, GetData, ...)

18.8.2. 執行巨集指令

執行巨集指令有多種不同的方法，下面分別說明。

- 使用 [PLC 控制] 物件

1. 打開 [PLC 控制] 物件，並設定屬性為 [執行巨集指令]。
2. 選擇需要執行的巨集指令名稱。選擇一個位元作為觸發巨集指令並設定觸發巨集指令的條件。在條件滿足時，該巨集指令將會被重複執行。為了每次只讓巨集指令執行一次，設計時需在巨集指令將該觸發位復位。
3. 使用一個 [位元狀態設定] 物件或者 [位元狀態切換開關] 物件作為這個位元的控制開關。

- 使用 [位元狀態設定] 物件或者 [位元狀態切換開關] 物件

1. 在 [位元狀態設定] 物件或者 [位元狀態切換開關] 物件的一般屬性頁中，勾選 [使用巨集指令]。
2. 選擇要執行的巨集指令。當這個物件被執行時，選擇的巨集指令就會被執行一次。

- 使用 [功能鍵]

1. 在 [功能鍵] 的一般屬性頁對話方塊中，勾選 [觸發巨集指令]。
2. 選擇需要執行的巨集指令。每按一下這個功能鍵時，選擇的巨集指令就會被執行一次。

- 使用 [巨集指令編輯器] 設定條件

1. [週期執行]：可以設定每間隔幾秒自動執行巨集指令一次。
2. [當 HMI 啟動時即執行一次]：當 HMI 重新上電或重新啟動時會執行此巨集指令一次。

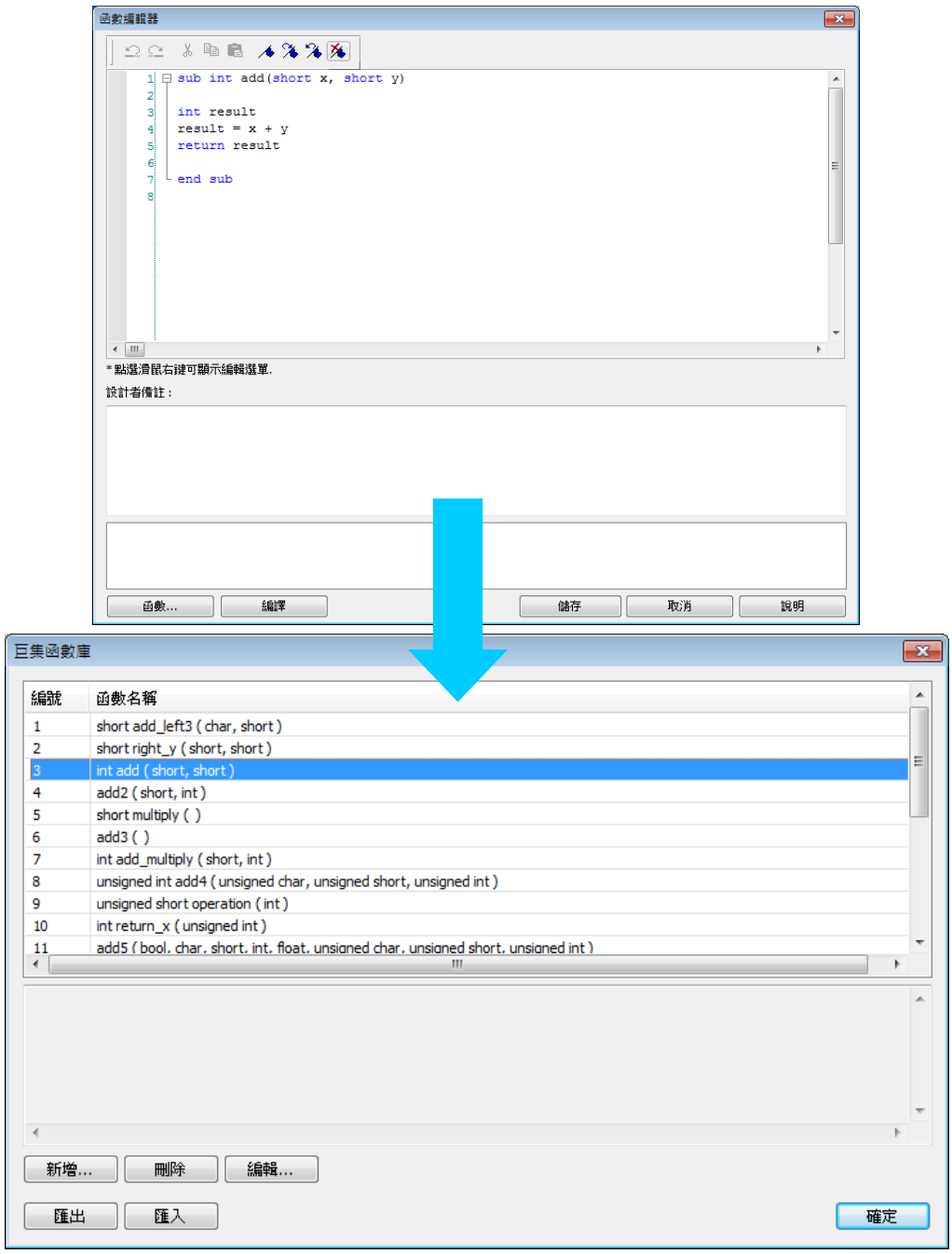
- 在 [視窗設定] 設定巨集的執行時機

1. [開啟視窗時執行]：當開啟此視窗時即執行指定的巨集指令一次。
2. [循環執行]：當開啟此視窗時，即每 0.5 秒循環執行指定的巨集指令。
3. [關閉視窗時執行]：當離開此視窗時即執行指定的巨集指令一次。

 請點選此圖示觀看教學影片，請先確定已連上網路線。

18.9. 使用者自定義函數功能

在使用巨集編輯器時，為了減少定義函數的時間，使用者可從內建的函數庫搜尋所需的函數。然而，當使用者編輯巨集時，若有某些特定的函數常常使用卻無法從內建函數庫搜尋到時，就可以自行定義所需的函數並儲存起來。當下次需要再定義相同的函數時，可由 [巨集指令庫] 呼叫出已儲存的函數，方便函數編輯。另外，[巨集指令庫] 也大幅提升了使用者自訂函數之可攜性。建立函數前可先查看現有內建函數或線上函數庫是否有現成函數可使用。

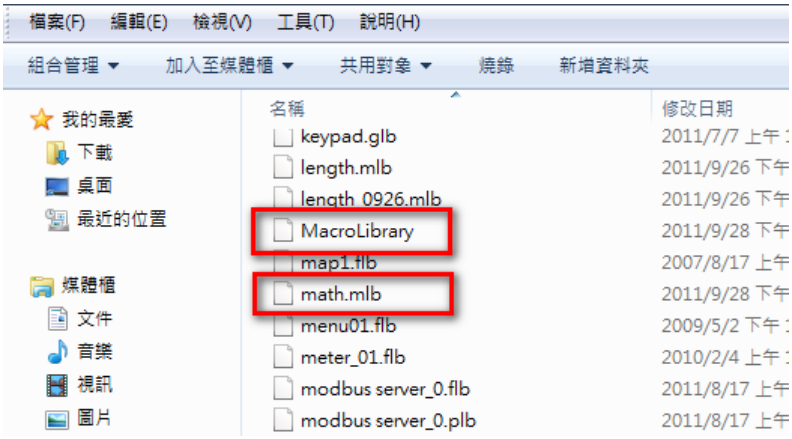


18.9.1. 匯入函數庫檔案

HMI 編輯軟體開啟一個工程檔時，會自動去讀入一個預設函數庫檔案，並載入函數資訊。當開啟一專案有呼叫到使用者定義函數時必須先載入相關的 .mlb 檔案。

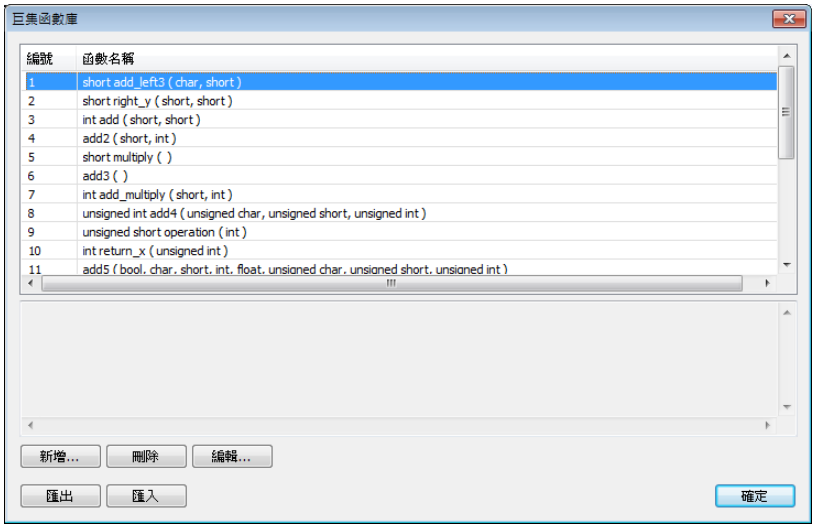
1. 預設函數庫檔名：MacroLibrary (沒有副檔名)
2. 函數庫路徑：HMI 編輯軟體的安裝目錄的 \library 資料夾下面。
3. \library 資料夾下面可以看到兩種函數庫檔案：
   沒有副檔名： MacroLibrary，為預設函數庫， HMI 編輯軟體指定讀入此檔案。
   有副檔名 (.mlb)：例如 math.mlb，使用者匯入/匯出時會去讀 / 寫的檔案，具可攜性，欲使用時再從資料夾呼叫出檔案即可。

4. EasyBuilder Pro 開啟時，只會去載入預設函數庫中的函數，使用者若需要用到 .mlb 檔內的函數時，必須自行將其匯入。

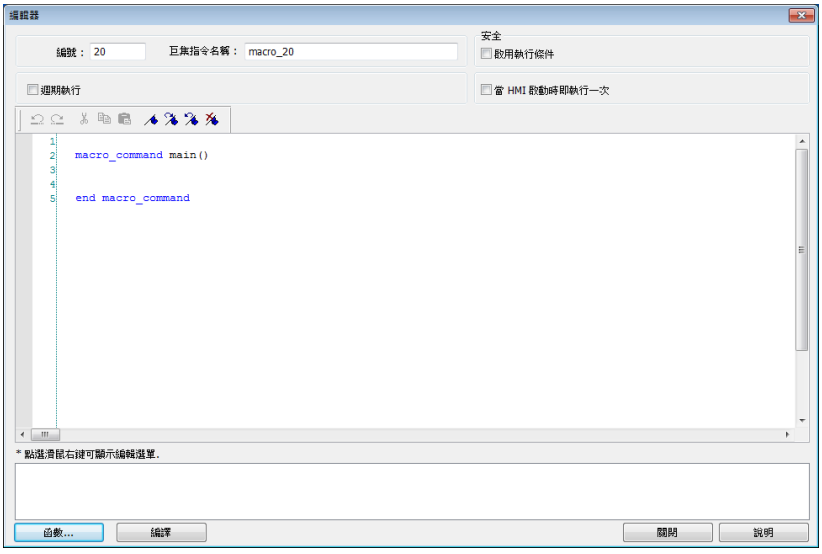


18.9.2. 如何使用巨集函數庫

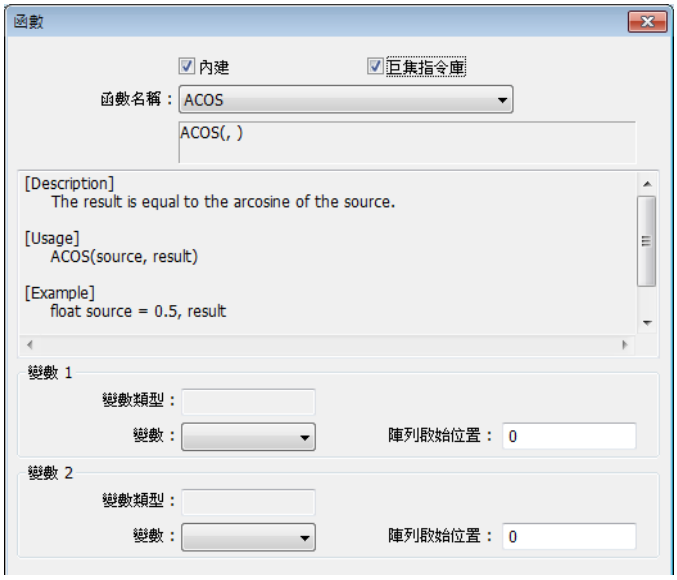
1. 在巨集編輯器中直接呼叫函數。



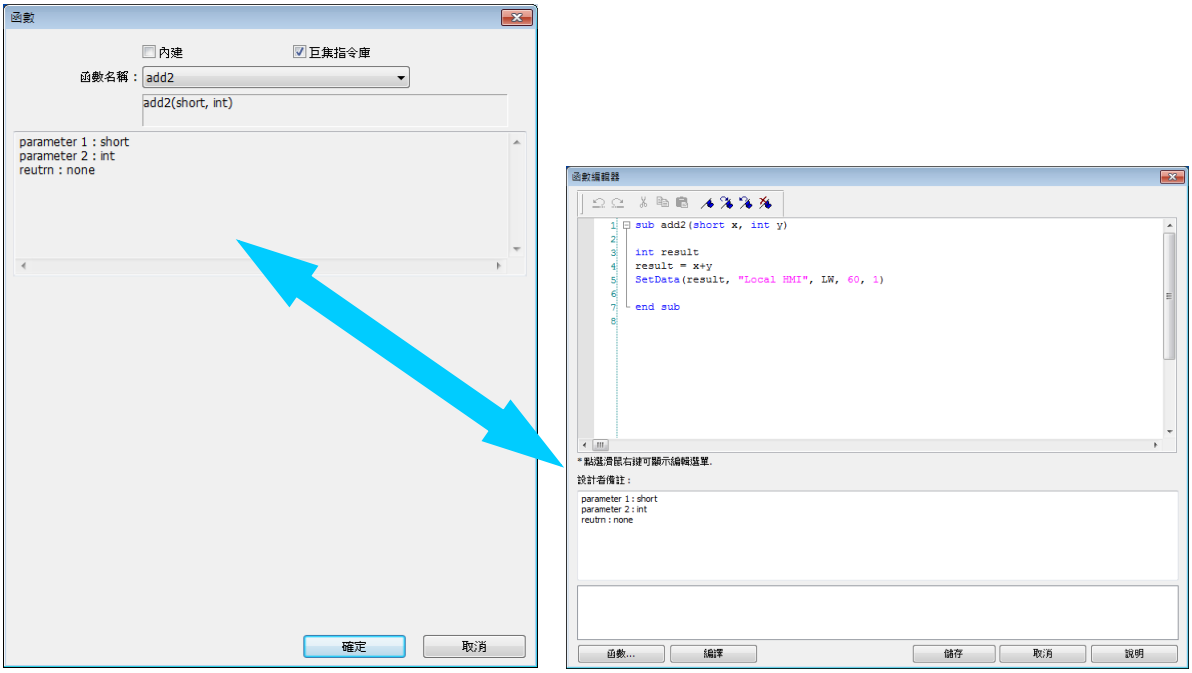
2. 按下巨集編輯器左下角的 [函數] 按鈕開啟函數對話框。



3. 須至少勾選一項 [巨集指令庫] 或 [內建]，並選擇欲使用的函數。



4. 顯示函數說明文字，即是使用者在函數編輯器中編輯的說明文字。



5. 選取欲使用的函數，將函數庫中的 [變數類型] 修改為指定的變數名稱。

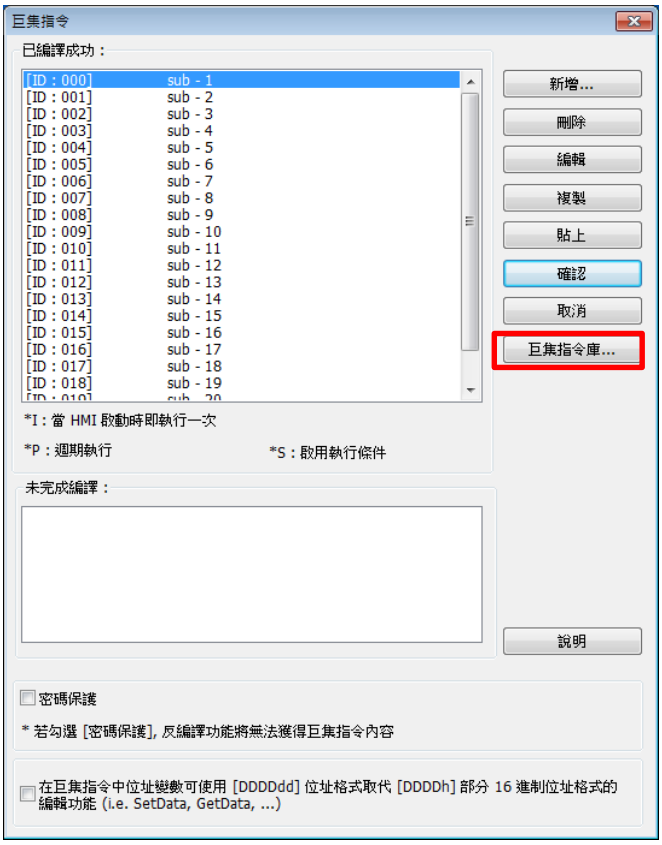
```
1
2   macro_command main()
3
4   short a
5   int b,result
6
7   add2(short, int)
8
9   end macro_command
```

```
1
2   macro_command main()
3
4   short a
5   int b,result
6
7   result = add2(a, b)
8
9   end macro_command
```

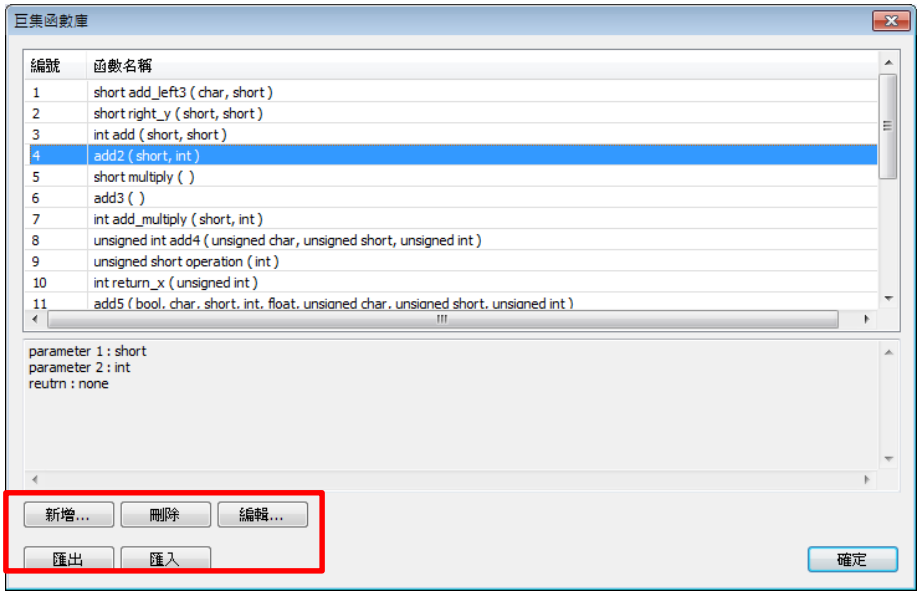
6. 使用者根據以上的步驟，即可完成使用自訂函數的功能，有效節省了重複定義相同的函數。

18.9.3. 函數庫管理介面

1. 開啟巨集管理對話框，按下右下角 [巨集指令庫] 按鈕，進入函數庫管理對話框介面。



2. 函數庫管理對話框中有函數列表。此列表列出工程檔開啟時， HMI 編輯軟體會從預設函數庫載入所有公共巨集庫。



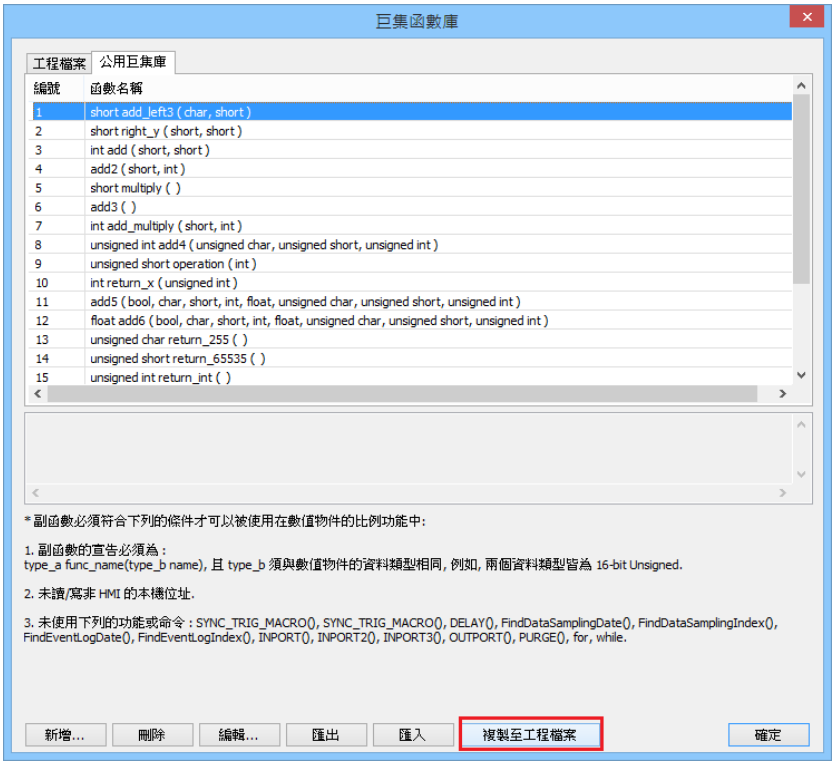
3. 函數列表中每一行的格式如下：

return_type function_name (*parameter_type1*, …, *parameter_typeN*)

return_type 表示回傳值型態，若無回傳值則此欄位省略；function_name 表示函數名稱。
parameter_typeN 表示第 N 個參數型態，若此函數不接受任何參數，此欄位省略。

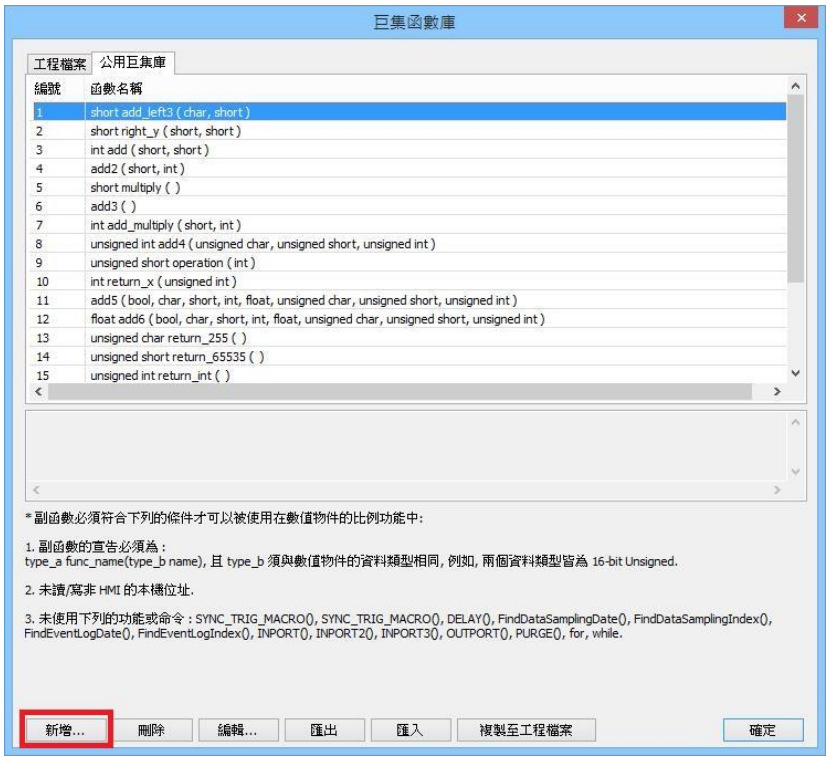
```
1  sub int ADD(int a, int b)
2    int ret
3    ret = a+b
4    return ret
5  end sub
6
```

4. 函數可以內崁到工程檔案中。選擇函數項目再點選 [複製至工程檔案]，在 [工程檔案] 標籤中就可以找到此函數，如此當使用者將專案拿到別台電腦中開啟，仍然可使用此函數。編譯工程檔案時會將工程檔案中有使用到的內崁函數編譯進 .exob 檔案中，若工程檔案中都未使用的函數，反編譯後會發現找不到此函數內容。

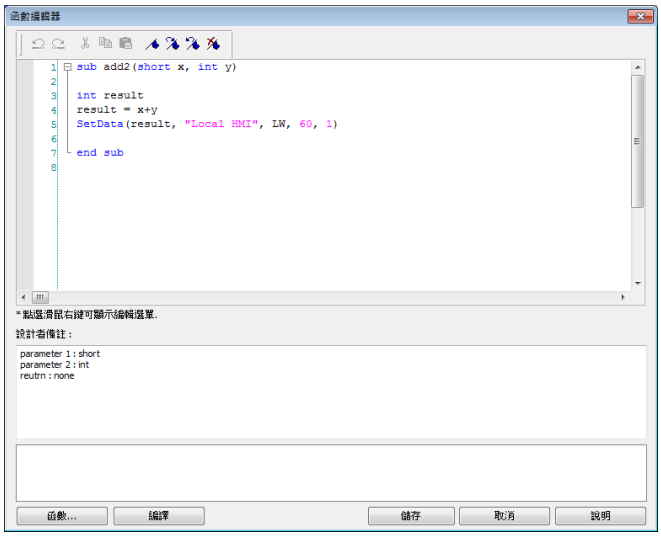


18.9.3.1. 新建函數

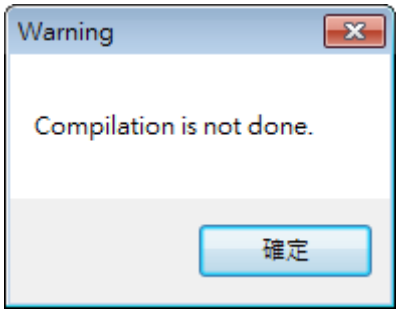
1. 按下 [新增] 進入函數編輯器。



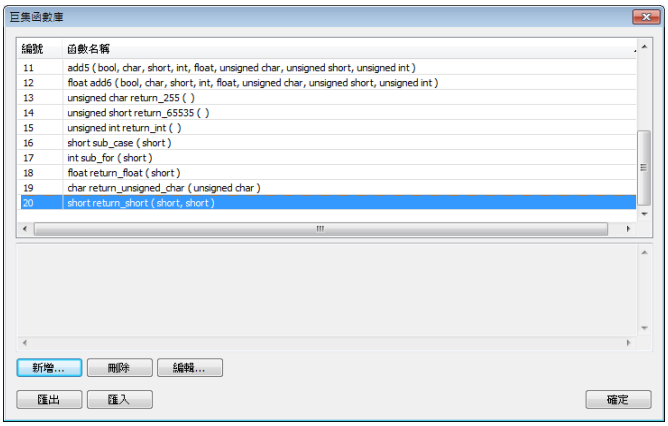
2. 在函數編輯區編輯函數，接著按 [編譯] 再按 [儲存]。



3. 使用者可以在函數說明編輯區中編輯說明文字，說明此函數的規格、使用方法、作者聲明等等。
4. 函數編輯完成後，必須通過編譯，才可以按下 [儲存] 寫入函數庫。否則會出現下圖彈出視窗，警告使用者此函數未完成編譯。



5. 成功加入函數庫。

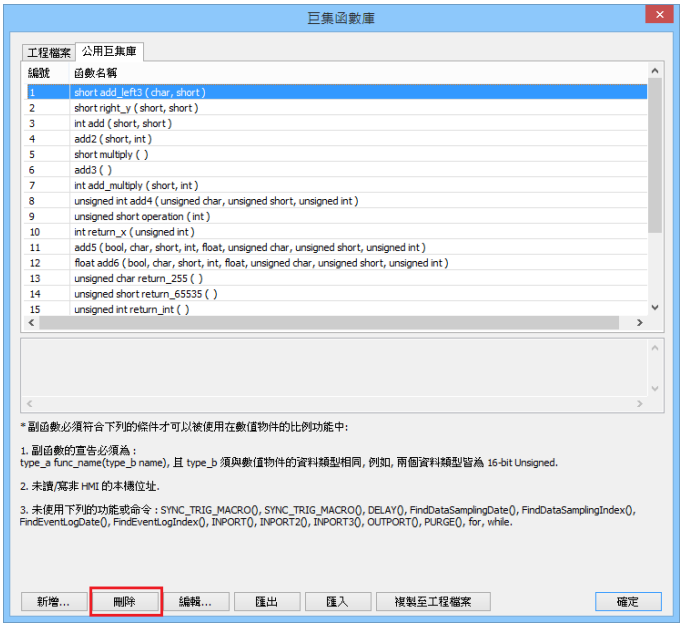


Note

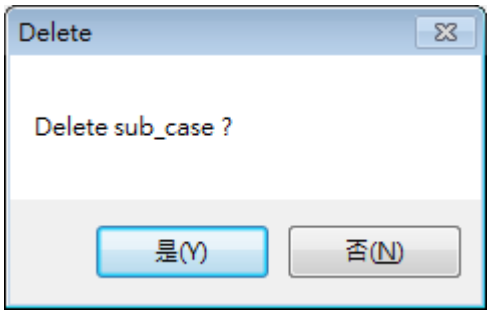
- 函數中可宣告的資料型態總數為 4096 個位元組。
- 函數名稱必須為英數字元，且不可為數字開頭。

18.9.3.2. 刪除函數

1. 在函數列表中選定要刪除的函數，按下 [刪除] 按鈕，即可刪除該函數。

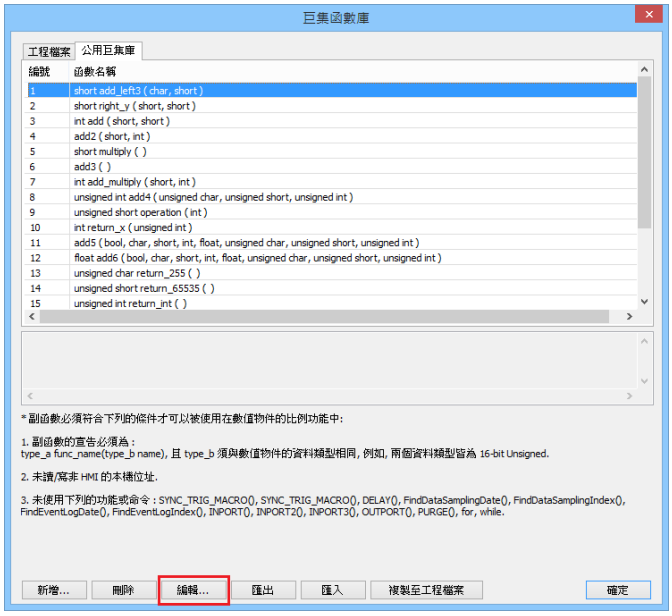


2. 按下 [是] 確認刪除，按下 [否] 取消刪除。按下 [是] 刪除 MAX_SHORT 此函數。

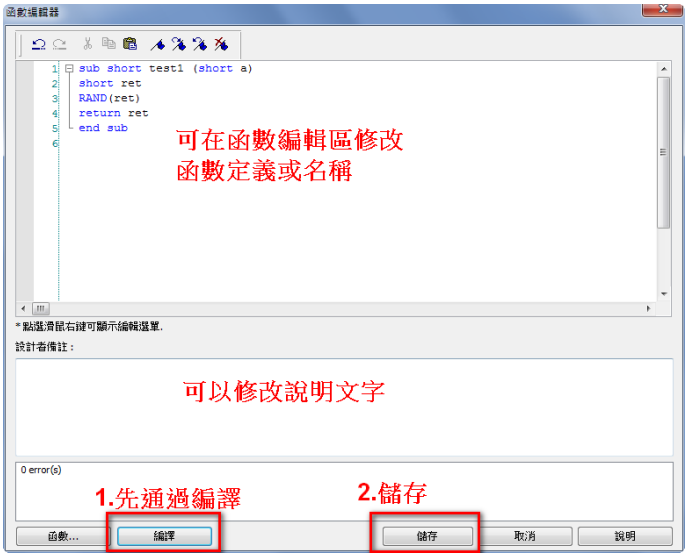


18.9.3.3. 修改函數

1. 使用者可以修改函數庫中的函數。
2. 選定要修改的函數，按下 [編輯]按鈕，進入函數編輯器。



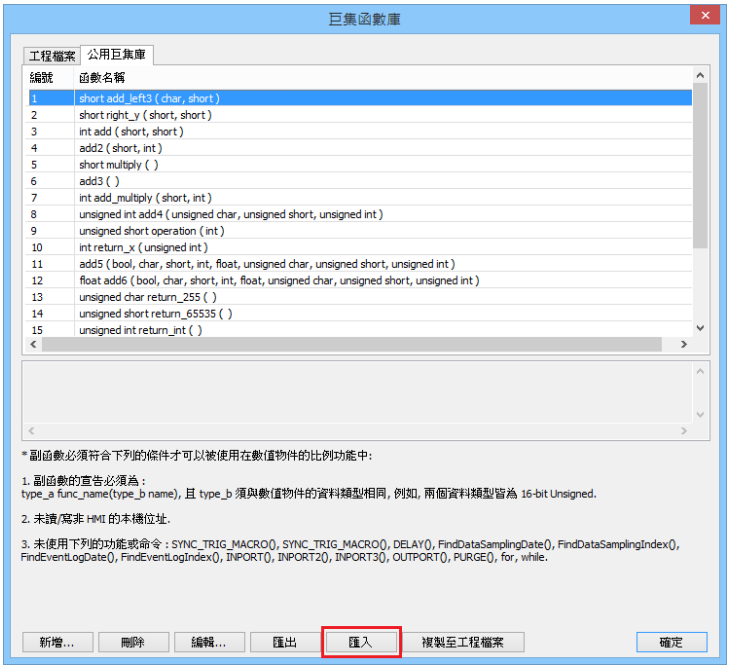
3. 也可直接在該函數上雙點滑鼠左鍵，進入函數編輯器。



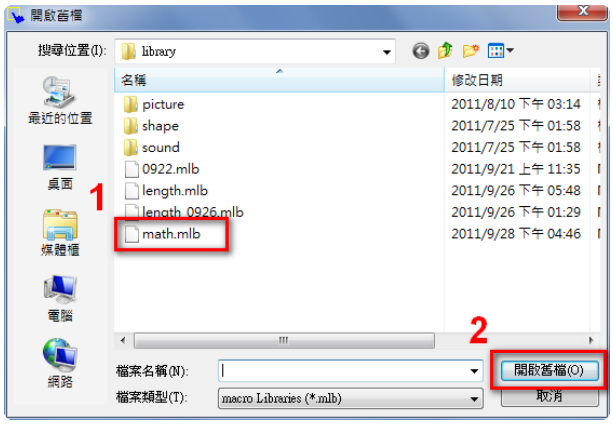
4. 修改完成後，一樣要先通過編譯，才可以儲存離開。

18.9.3.4. 匯入函數

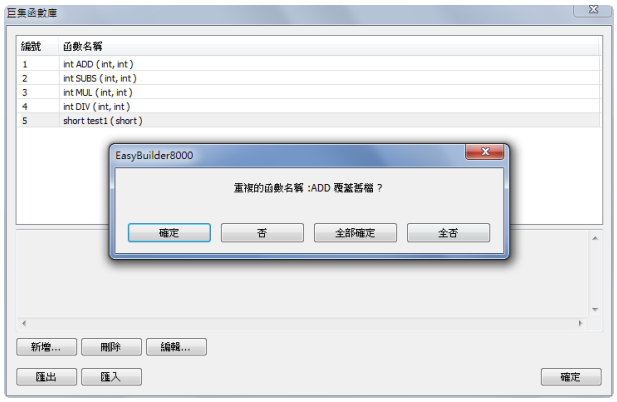
1. 使用者可以從外部的 .mlb 檔案將函數匯入。



2. 假設欲加入函數庫 math.mlb，此函數庫內含有一函數 test1。按下[開啟舊檔]按鈕。



3. 匯入函數時，若檔案庫中已存在相同名稱的函數，會出現下面的彈出視窗。



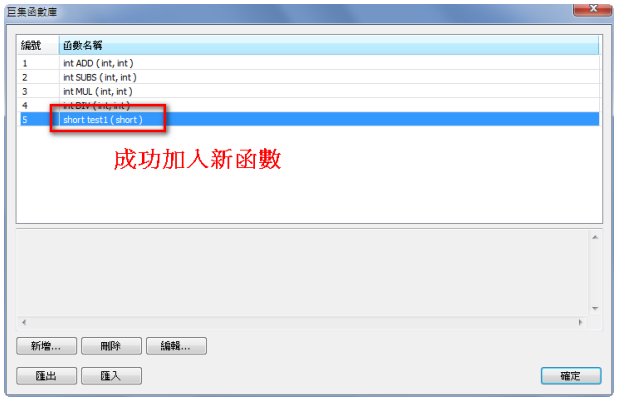
[確定]：用外部匯入的函數覆蓋函數庫中現有的同名函數。

[否]：放棄外部匯入此同名函數。

[全部確定]：全部用外部匯入的同名函數覆蓋所有同名函數。

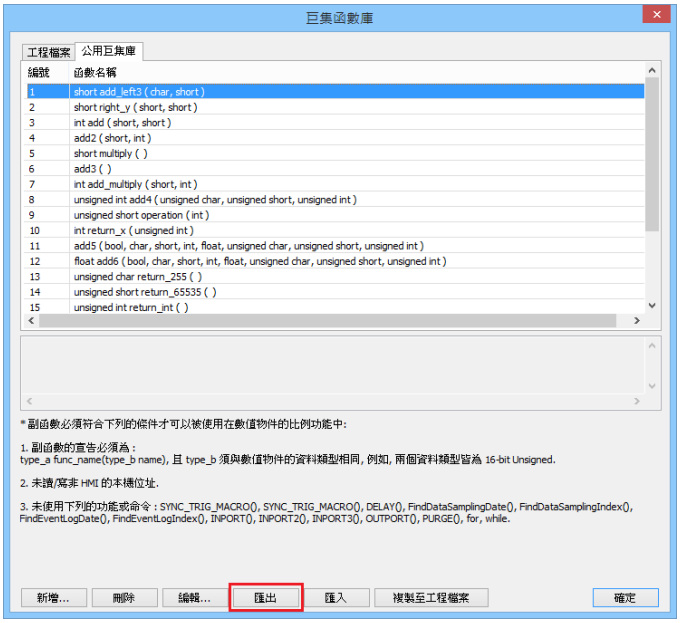
[全否]：全部放棄覆蓋匯入同名的函數。

4. 完成匯入後，匯入的函數都已寫入到預設函數庫中，因此使用者可以將 math.mlb 檔案刪除，而 test1 仍會存在函數庫中，即使重新開啟 HMI 編輯軟體亦然。

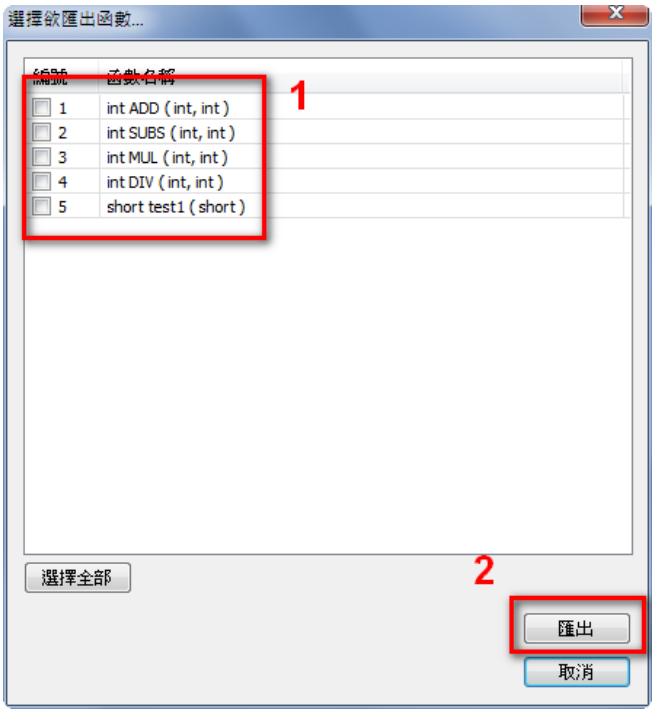


18.9.3.5. 匯出函數

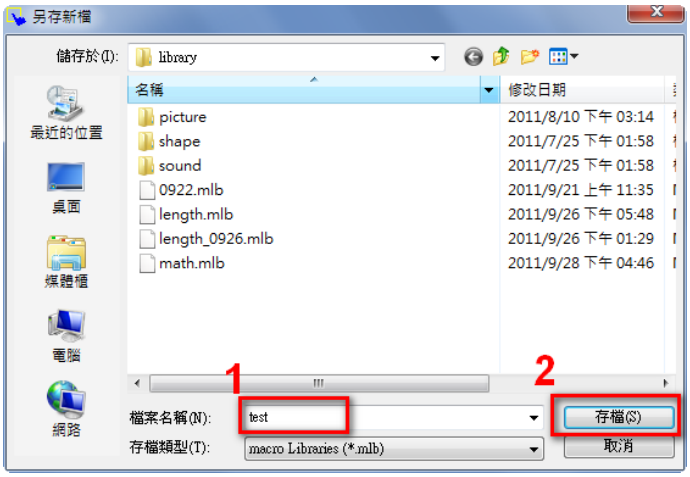
1. 使用者可以將函數庫中的函數匯出到 .mlb 檔。按下 [匯出] 按鈕。



2. 選擇要匯出的函數，然後按 [匯出]。



3. 匯出的目錄下出現一個 math.mlb 的函數庫檔案，其中包含 ADD、SUBS、MUL、DIV 這四支函數。
4. 匯出的 .mlb 檔可以攜帶到別的電腦上，只要使用者開啟 HMI 編輯軟體並完成匯入動作後，即可使用此檔案內所提供的所有函數。



18.10. 使用巨集指令時的注意事項

1. 儲存局部變數的空間是 4KB，所以各種不同變數類型的最大陣列大小為如下：

char	a[4096]
bool	b[4096]
short	c[2048]
int	d[1024]
float	e[1024]

long f[512]
double g[512]

2. 一個 EasyBuider Pro 工程檔案中最多包含 255 個巨集指令，但 cMT X 系列可以支援到 500 個巨集指令。
3. 巨集指令有可能造成 HMI 當機，可能的原因為：

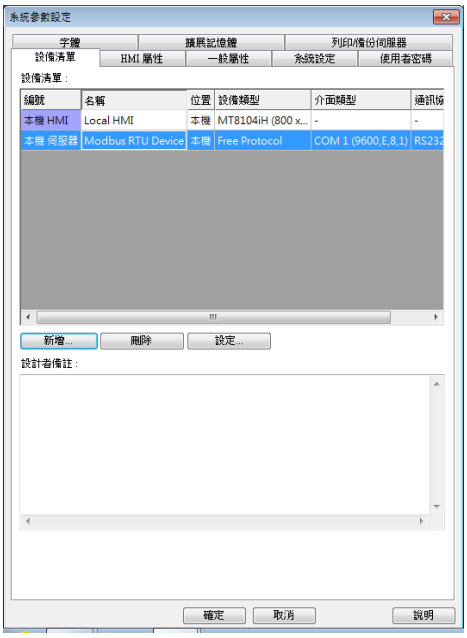
- 巨集指令中執行了一個不正確的無限迴圈
- 陣列的大小超過了巨集指令的變數容量

4. 設備的通訊速度可能影響巨集指令的執行速度。相對的，使用過多的巨集指令，可能會造成與設備的通訊速度變慢。

18.11. 使用自由協定去控制一個設備

當 EasyBuilder Pro 沒有內建與某一個設備通訊的驅動程式時，使用者也可以使用巨集指令中的 OUTPORT 和 INPORT 函數來實現與該設備的通訊。使用 OUTPORT 和 INPORT 函數發送和接收的資料，必須遵行該設備的通訊協定。下面的範例程式說明了如何使用這兩個函數來控制一個 MODBUS RTU 設備。

1. 首先，在系統參數/設備列表中建立一個新的設備。這個新建的 [設備類型] 設置為 “Free Protocol”，[設備名稱] 設置為 “MODBUS RTU Device”，如下圖所示。



2. 這裡的設備連接使用 RS232，如果連接一個 MODBUS TCP / IP 設備，這個介面類型必須設定為 “乙太網路”，同時必須設定正確的 IP 位址和連接埠號，如下圖所示。

假設 HMI 人機界面將要讀取設備中的 4x_1 和 4x_2 兩個資料暫存器。首先，使用 OUTPORT 函數發送讀命令給這個設備，OUTPORT 函數的寫法為：

OUTPORT(command[start], device_name, cmd_count)

因為 “MODBUS RTU Device” 是一個 MODBUS RTU 設備，讀數據的命令必須遵行 MODBUS RTU 協定的命令規則。所以必須使用 0x03 這個命令去讀取 4x_1 和 4x_2 這兩個資料暫存器的資料。下圖說明瞭讀命令的格式。(省略了設備的站號和最後的兩個 CRC 位元組)

Request

Function code	1 Byte	0x03
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Registers	2 Bytes	1 to 125 (0x7D)

Response

Function code	1 Byte	0x03
Byte count	1 Byte	2 x N*
Register value	N* x 2 Bytes	

*N = Quantity of Registers

Error

Error code	1 Byte	0x83
Exception code	1 Byte	01 or 02 or 03 or 04

根據這個 MODBUS RTU 協定，發送命令的內容如下所示：（總共 8 個位元組）

response[0]: station number (BYTE 0) 站號

response[1]: function code (BYTE 1) 功能碼

response[2]: high byte of starting address (BYTE 2) 起始位址高位元

response[3]: low byte of starting address (BYTE 3) 起始位址低位元

response[4]: high byte of quantity of registers (BYTE 4) 資料暫存器的高位元

response[5]: low byte of quantity of registers (BYTE 5) 資料暫存器的低位元

response[6]: low byte of 16-bit CRC (BYTE 6) CRC 的低位元組

response[7]: high byte of 16-bit CRC (BYTE 7) CRC 的高位元組

所以讀數據的命令巨集指令程式設計如下：

```
char command[32]
short address, checksum

FILL(command[0], 0, 32)    //    initialize command[0]~command[31] to 0

command[0] = 0x1    //    station number
command[1] = 0x3    //    read holding registers (function code    is 0x3)

address =    //    starting address (4x_1) is 0
HIBYTE(address, command[2])
LOBYTE(address, command[3])

read_no = 2    //    the total words of rading is 2 words
HIBYTE(read_no, command[4])
LOBYTE(read_no, command[5])

CRC(command[0], checksum, 6)    //    calculate 16-bit CRC

LOBYTE(checksum, command[6])
HIBYTE(checksum, command[7])
```

最後，使用 OUTPORT 函數將這個讀命令發送給這個 MODBUS RTU 設備。

```
OUTPORT(command[0], “MODBUS RTU Device”, 8)    //    發送命令
```

發送完這個命令後，使用 INPORT 函數讀取該 MODBUS RTU 設備返回的命令。根據 MODBUS RTU 協定，這個回復的命令內容為如下 (總共 9 個位元組)：

command[0]: station number (BYTE 0) 站號
command[1]: function code (BYTE 1) 功能碼
command[2]: byte count (BYTE 2) 位元組長度
command[3]: high byte of 4x_1 (BYTE 3) 第一個資料的高位元組
command[4]: low byte of 4x_1 (BYTE 4) 第一個資料的低位元組
command[5]: high byte of 4x_2 (BYTE 5) 第二個資料的高位元組
command[6]: high byte of 4x_2 (BYTE 6) 第二個資料的低位元組
command[7]: low byte of 16-bit CRC (BYTE 7) CRC 的低位元組
command[8]: high byte of 16-bit CRC (BYTE 8) CRC 的高位元組

此時，INPORT 函數的語句如下：

```
INPORT(response[0], “MODBUS RTU Device”, 9, return_value)    //    讀取回復命令
```

函數中，實際讀取到的位元組長度存放在變數 return_value (變數類型為位元組) 中。如果 return_value 的資料為 0，則表示使用 INPORT 讀取命令失敗。

根據 MODBUS RTU 協定，如果命令回復的正確，則 response[1] 必須為 0x03。當讀取到正確的命令後，計算出 4x_1 和 4x_2 這兩個暫存器的值，並將這兩個資料送到人機界面的 LW-100 和

LW-101 暫存器中。

```
If (return_value) >0 and response[1] == 0x3) then
   read_data[0] = response[4] + (response[3] << 8) //  計算 4x_1  的資料
   read_data[1] = response[6] + (response[5] << 8) //  計算 4x_2  的資料

   SetData(read_data[0], “Local HMI”, LW, 100, 2)    //    將資料存至 HMI 上
endif
```

完整的巨集指令程式如下：

```
// Read Holding Registers
macro_command main()

   char command[32], response[32]
   short address, checksum
   short read_no, return_value, read_data[2], i

   FILL(command[0], 0, 32)//    initialize command[0]~command[31] to 0
   FILL(response[0], 0, 32)

   command[0] = 0x1//    station number
   command[1] = 0x3//    read holding registers (function code is 0x3)

   address = 0
   address =    0//    starting address (4x_1) is 0
   HIBYTE(address, command[2])
   LOBYTE(address, command[3])

   read_no = 2/    the total words of reading is 2 words
   HIBYTE(read_no, command[4])
   LOBYTE(read_no, command[5])

   CRC(command[0], checksum, 6)//    calculate 16-bit CRC

   LOBYTE(checksum, command[6])
   HIBYTE(checksum, command[7])

   OUTPORT(command[0], "MODBUS RTU Device", 8 )//    send request
   INPORT(response[0], "MODBUS RTU Device", 9, return_value)//    read response

   if (return_value > 0 and response[1] == 0x3) then
      read_data[0] = response[4] + (response[3] << 8)//    4x_1
      read_data[1] = response[6] + (response[5] << 8)//    4x_2

      SetData(read_data[0], "Local HMI", LW, 100, 2)
   end if

   end macro_command
```

下面的舉例說明如何使用自由協定設定 MODBUS RTU 設備中 0x_1 的狀態。這個是使用 MODBUS RTU 協議中的“寫單個暫存器”的功能碼“0x05”來實現的。

Request

Function code	1 Byte	**0x05**
Output Address	2 Bytes	0x0000 to 0xFFFF
Output Value	2 Bytes	0x0000 or 0xFF00

Response

Function code	1 Byte	**0x05**
Output Address	2 Bytes	0x0000 to 0xFFFF
Output Value	2 Bytes	0x0000 or 0xFF00

Error

Error code	1 Byte	**0x85**
Exception code	1 Byte	01 or 02 or 03 or 04

完整的巨集指令程式如下：

```
//    Write Single Coil (ON)
macro_command main()

char command[32], response[32]
short address, checksum
short i, return_value

FILL(command[0], 0, 32)//    initialize command[0]~ command[31] to 0
FILL(response[0], 0, 32)

command[0] = 0x1//    station number
command[1] = 0x5//    function code : write single coil

address = 0
HIBYTE(address, command[2])
LOBYTE(address, command[3])

command[4] = 0xff//    force 0x_1 on
command[5] = 0

CRC(command[0], checksum, 6)

LOBYTE(checksum, command[6])
HIBYTE(checksum, command[7])

OUTPORT(command[0], "MODBUS RTU Device", 8)//    send request
INPORT(response[0], "MODBUS RTU Device", 8, return_value)//    read response

end macro_command
```

請點選此圖示下載範例程式。下載範例程式前，請先確定已連上網路線。

18.12. 編譯錯誤提示資訊

- 錯誤資訊格式：

error C# : 錯誤描述

(# 是錯誤資訊編號)

舉例：error C37：undeclared identifier：i

當編譯後提示有錯誤資訊時，這個錯誤的描述內容可以參考錯誤資訊編號。

- Error 描述

(C1) syntax error：'identifier'

出現這個資訊時，有許多種可能。

舉例:

```
macro_command main()
char i, 123xyz     // 不支援這個變數類型
end macro_command
```

(C2) 'identifier' used without having been initialized

巨集指令必須定義聲明的陣列變數的大小。

舉例:

```
macro_command main()
char i
int g[i]     // i 必須為一個數值常數
end macro_command
```

(C3) redefinition error : 'identifier'

函數名稱和變數名稱在有效範圍內，必須是唯一的。

舉例:

```
macro_command main()
int g[10]，g     // 重複定義錯誤
end macro_command
```

(C4) function name error : 'identifier'

保留的關鍵字和常數，不能被定義為函數名稱。

舉例：

```
sub int if()    // 函數名稱定義錯誤
```

(C5) parentheses have not come in pairs

語句中缺少了“(“ or “)” 。

舉例：

```
macro_command main    )    // 圓括號沒有成對的出現
```

(C6) illegal expression without matching ‘if’

‘if’ 語句中沒有合法的運算式。

(C7) illegal expression (no ‘then’) without matching ‘if’

‘if’ 語句中缺少了 ‘then’，也就是 ‘if’ 和 ‘then’ 沒有成對。

(C8) illegal expression (no ‘end if’)

‘if’ 語句中缺少了 ‘end if’。

(C9) illegal ‘end if’ without matching ‘if’

‘end if’ 語句前缺少了 ‘if’ 語句。

(C10) illegal ‘else’

‘if’ 語句的標準格式請參考 CH 18.5.3 邏輯運算語句，任何與以上格式不符合的語句，在編譯時就會錯誤。

(C11)'case' expression not constant

‘case’ 後方的數值判斷必須為一個定值常數。

舉例：

```
macro_command main()
int a = 0
int b
select case a
case b // case 內容非常數
break
end select
end macro_command
```

(C12) 'select' statement contains no 'case'

‘select’ 後方缺少了 ‘case’ 語句。

舉例：

```
macro_command main()
int a = 0
int b
select a // ‘select’ 後方缺少了 ‘case’ 語句
case 1
break
end select
end macro_command
```

(C13) illegal expression without matching 'select case'

‘select’ 和 ‘case’ 沒有成對。

(C14) 'select' statement contains no 'end select'

‘select’ 語句中缺少了 ‘end select’。

(C15) illegal ‘case’

‘case’ 語句的標準格式請參考 CH 18.5.4 多重判斷語句，任何與以上格式不符合的語句，在編譯時就會錯誤。

(C16) illegal ‘case else’

‘case else’ 語句的標準格式請參考 CH 18.5.4 多重判斷語句，任何與以上格式不符合的語句，在編譯時就會錯誤。

(C17) illegal expression (no 'for') without matching ‘next’

‘for’ 語句錯誤：在 ‘next’ 前，缺少了 ‘for’ 語句。

(C19) variable data type error

語句中變數的資料型態錯誤。

(C20) must be keyword ‘to’ or ‘down’

缺少了關鍵字 ‘to’ 或者 ‘down’。

(C21) illegal expression (no 'next')

‘for’ 語句的標準格式為：

for [變數] = [初始值] to [結束值] [step]

next [變數]

任何與上述格式不符合的語句，編譯時會錯誤。

(C22) 'wend' statement contains no 'while'

'while' 語句錯誤：在 'wend' 前，缺少了 'while' 語句。

(C23) illegal expression without matching 'wend'

缺少 'wend' 關鍵字。

'while' 語句的標準格式為：

while [邏輯運算式]

wend

任何與上述格式不符合的語句，編譯時會錯誤。

(C24) syntax error : 'break'

不合法的 'break' 語句。'break' 語句只會在 'for' 或者 'while' 語句中出現。

(C25) syntax error : 'continue'

不合法的 'continue' 語句。'continue' 語句只會在 'for' 或者 'while' 語句中出現。

(C28) must be 'macro_command'

此處應該為 'macro_command' 。

(C29) must be key word 'sub'

子函數的定義格式為：

sub [data type] function_函數名稱 (…)

……..

end sub

舉例:

sub int pow (int exp)

…….

end sub

任何不符合上述語法結構的，在編譯時會錯誤。

(C30) number of parameters is incorrect

參數數量不匹配。

(C31) parameter type is incorrect

參數資料類型不相配。調用函數時，參數必須在資料類型、個數上一一對應才能通過編譯，否則編譯時將出現此項錯誤訊息。

(C33) function name : undeclared function

沒有定義的函數名稱。

(C34) expected constant expression

不合法的陣列下標表達形式。

(C35) invalid array declaration

不合法的陣列定義。

(C37) undeclared identifier : i 'identifier'

使用沒有定義的變數。只能使用已經定義的變數和函數，否則編譯時將出現此項錯誤訊息。

(C38) device encoding method is not supported

通訊函數 GetData(…)、SetData(…)的參數中有包含設備位址類型資訊，當設備位址類型不是此種設備支援的位址類型時，編譯時將出現此項錯誤訊息。

(C39) array index must be integer, short, char or constant

陣列的格式為：

宣告：函數名稱[constant] (constant is the size of the array)

使用：函數名稱[integer, character or constant]

任何不符合上述規則的陣列運算式，編譯時將會錯誤。

(C40) execution syntax should not exist before variable declaration or constant definition

變數定義語句的前面不能有執行語句。

舉例 ：

```
macro_command main()
int a, b
for a = 0 To 2
b = 4 + a
int h , k   // 定義變數語句在此處是錯誤的，在一個函數內定義變數語句的前面
不能有執行語句，例如 b = 4 + a
next a
```

end macro_command

(C41) float variables cannot be contained in shift calculation
移位運算中，運算元不能為浮點數。

(C42) function must return a value
函數應有返回值。

(C43) function should not return a value
函數不應有返回值。

(C44) float variables cannot be contained in calculation
運算中不能有浮點數的資料。

(C45) device address error/tag name does not exist
設備位址錯誤。

(C46) size of function variables is too large (max. 4k bytes)
一維陣列的大小超過 4k。

(C47) macro command entry function is not only one
巨集指令程式入口只能有一個，形式為：
macro_command function_函數名稱( )
end macro_command

(C49) an extended addressee's station number must be between 0 and 255
在巨集指令中，擴展地址內的站號大小只能從 0 到 255。

舉例：
SetData(bits[0] , "PLC 1", LB , 300#123, 100)
// illegal : 300#123 意思是站號為 300, 但是最大值是 255

(C50) an invalid device name
在巨集指令中，設備的名稱並未定義在系統參數的設備列表中。

(C51) macro command do not control a remote device
巨集指令只能控制本機連接的設備。

舉例 :
SetData(bits[0] , “PLC 1” , LB , 300#123, 100)
“PLC1” 連接在遠端的 HMI 上，所以它不能被執行。

(C52) GetData/GetDataEx/StringGet/StringGetEx cannot use a broadcast station no.
上述語法無法使用廣播站號。

(C53) INPORT() must use a "Free Protocol" device
INPORT 語法必須使用在 “Free Protocol” 的設備上。

(C54) OUTPORT() must use a "Free Protocol" device
OUTPORT 語法必須使用在 “Free Protocol” 的設備上。

(C55) Recipe Database is not supported on this HMI model
該機型不支援配方資料庫功能。

(C56) the data type of ‘identifier’ must be "unsigned"
資料型態必須為 "unsigned"。

(C57) Recipe bit position is out of range
使用到的配方資料的 Recipe_bit 設定超出範圍。

(C58) assignment is out of range
變數的賦值超出資料型態定義的上下限範圍。

(C59) declaration of global variables in macro library is not allowed
巨集指令庫中不允許宣告全域變數。

(C60) illegal expression following the keyword "step" in the for-loop
在 ‘for’ 語句中的關鍵字 ‘step’ 後方有不合法的表達式。

(C61) nested call to sub function is not allowed
不允許子函數間的巢狀呼叫。

(C62) case else must be placed at the end of the select case
‘case else’ 必須位在 ‘select case’ 語句中的最後一項。

(C63) array index exceeds array size

陣列索引超出陣列定義的大小。

(C64) data count exceeds the size of read/write buffer
讀/寫指令超過 4k 位元組。

(C65) SQL syntax not accepted
不支援的 SQL 語法。

(C66) recipe tag not found
配方標籤名稱不存在於配方資料庫。

(C67) counter variable of for-loop doesn't support unsigned data type
'for' 語句中的計數器變數不支援 unsigned 資料型別。

舉例：

```
macro_command main()
unsigned int i
for i = 5 down 0 step 1    // 不支援 unsigned 資料型別
next
end macro_command
```

(C68) Conversion Tag size error
使用轉換標籤相關語法時長度錯誤。

(C69) Macro name : 'identifier' not found
使用的巨集名稱不存在。

(C70) Macro undefined : Macro ID = 'identifier'
使用的巨集 ID 不存在。

(C71) syntax error (or number of characters exceeds 2048)
語法錯誤（或字元數超過 2048）。

(C72) parameter value is out of range : 'identifier'
參數值超出範圍。

(C73) 'identifier' does not support GetData/SetData/GetDataEx/SetDataEx/StringGet/StringGetEx/StringSet/StringSetEx

識別字不支援使用於上述語法。

(C74) station no. variable must be between var0 ~ var15
站號變數必須在 var0 ~ var15 之間。

(C75) Macro function is not supported on this HMI model
該機型不支援巨集相關功能。

(C76) the "unsigned" keyword must be followed by a data type
關鍵字 "unsigned" 後面必須跟著資料類型。

(C77) index register syntax error
索引暫存器語法錯誤。

(C78) this tag does not support index register
此標籤不支援索引暫存器。

(C79) index register is not supported on this HMI model
該機型不支援索引暫存器。

(C80) function does not support if/while/for/switch statement
函數不支援 if/while/for/switch 語句。

(C82) string must be declared as a char or unsigned char array
字串必須宣告為 char 或 unsigned char 陣列。

(C83) 'identifier' must be a constant data
識別字必須是常數資料。

(C84) array data exceeds array size
陣列資料超出陣列大小。

(C85) illegal expression (no 'select') without matching 'end select'
'end select' 缺少 'select' 關鍵字。

(C95) total number of characters exceeds 100000
巨集指令總字元數不得超過 100,000。

若使用者欲使用超過字元數量限制的巨集指令時，需要先將該巨集指令的總字元數調整至 100,000 以下，並透過 SYNC_TRIG_MACRO 函數利用同步的方式觸發執行其他的巨集指令完成總字元數超過 100,000 的巨集指令。

18.13. 巨集指令範例程式

- for 迴圈，各種運算式 (算術，移位元，邏輯，關係運算式)

```
macro_command main()
   int a[10], b[10], i

   b[0] = (400 + 400 << 2) / 401
   b[1] = 22 *2 - 30 % 7
   b[2] = 111 >> 2
   b[3] = 403 > 9 + 3 >= 9 + 3 < 4 + 3 <= 8 + 8 == 8
   b[4] = not 8 + 1 and 2 + 1 or 0 + 1 xor 2
   b[5] = 405 and 3 and not 0
   b[6] = 8 & 4 + 4 & 4 + 8 | 4 + 8 ^ 4
   b[7] = 6 – (~4)
   b[8] = 0x11
   b[9] = 409

   for i = 0 to 4 step 1
      if (a[0] == 400) then
         GetData(a[0],”Device 1”, 4x, 0,9)
         GetData(b[0],”Device 1”, 4x, 11,10)
      end If
   next i
end macro_command
```

- while, if, break 語句

```
macro_command main()
   int b[10], i
   i = 5
   while i == 5 - 20 % 3
      GetData(b[1], ”Device 1”, 4x, 11, 1)

      if b[1] == 100 then
         break
      end if
   wend
end macro_command
```

- 總體變數和子函數調用

```
char g
sub int fun(int j, int k)
   int y

   SetData(j, "Local HMI", LB, 14, 1)
   GetData(y, "Local HMI", LB, 15, 1)
   g = y

   return y
end Sub

macro_command main()
   int a, b, i

   a = 2
   b = 3
   i = fun(a, b)
   SetData(i, "Local HMI", LB, 16, 1)
end macro_command
```

- if 結構語句

```
macro_command main()
   int k[10], j

   for j = 0 to 10
      k[j] = j
   next j

   if k[0] == 0 then
      SetData(k[1], "Device 1", 4x, 0, 1)
   end if

    if k[0] == 0 then
      SetData(k[1], "Device 1", 4x, 0, 1)
   else
      SetData(k[2], "Device 1", 4x, 0, 1)
   end if
```

```
    if k[0] == 0 then
        SetData(k[1], "Device 1", 4x, 1, 1)
    else if k[2] == 1 then
        SetData(k[3], "Device 1", 4x, 2, 1)
    end If

    if k[0] == 0 then
        SetData(k[1], "Device 1", 4x, 3, 1)
    else if k[2] == 2 then
        SetData(k[3], "Device 1", 4x, 4, 1)
    else
        SetData(k[4], "Device 1", 4x, 5, 1)
    end If
end macro_command
```

- while 和 wend 結構語句

```
macro_command main()
    char i = 0
    int a[13], b[14], c = 4848

    b[0] = 13

    while b[0]
        a[i] = 20 + i * 10

        if a[i] == 120 then
            c =200
            break
    end if

    i = i + 1
    wend

    SetData(c, "Device 1", 4x, 2, 1)
end macro_command
```

- break 和 continue 語句結構

```
macro_command main()
  char i = 0
  int a[13], b[14], c = 4848
  b[0] = 13

  while b[0]
      a[i] = 20 + i * 10

      if a[i] == 120 then
      c =200
        i = i + 1
        continue
      end if
      i = i + 1

    if c == 200 then
    SetData(c, "Device 1", 4x, 2, 1)
    break
    end if
  wend
end macro_command
```

- 陣列結構

```
macro_command main()
  int a[25], b[25], i

  b[0] = 13

  for i = 0 to b[0] step 1
    a[i] = 20 + i * 10
  next i

  SetData(a[0], "Device 1", 4x, 0, 13)
end macro_command
```

- 字串相關語法內使用”符號的方法，也可使用在變數宣告及函數參數的時候

```
macro_command main()
char data[40]= "\"Note\" "
```

```
StringCopy("This is a \"test\" for weintek", data[7])
//字串內容為"Note" This is a "test" for weintek
end macro_command
```

18.14. 巨集指令 TRACE 函數

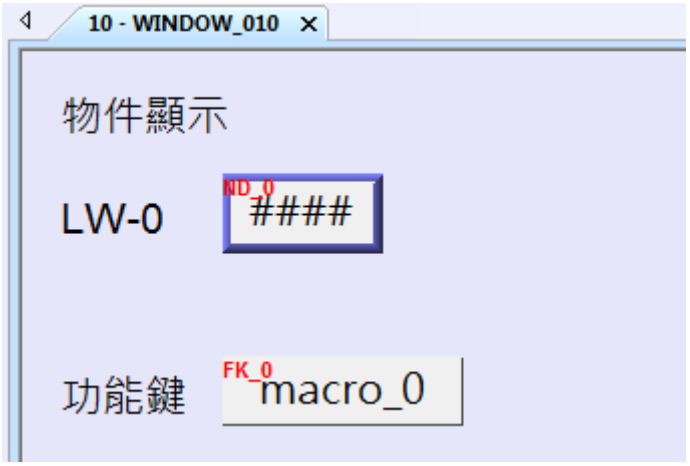
巨集指令的 TRACE 函數，搭配使用 EasyDiagnoser/cMT Diagnoser，可用來檢視所使用變數目前的內容。使用 cMT / cMT X 系列的用戶，建議使用 cMT Diagnoser 的內建巨集除錯工具進行除錯，無需特別使用 TRACE 函數。

下面示範如何在巨集指令中利用 TRACE 命令並使用 EasyDiagnoser 進行監控。

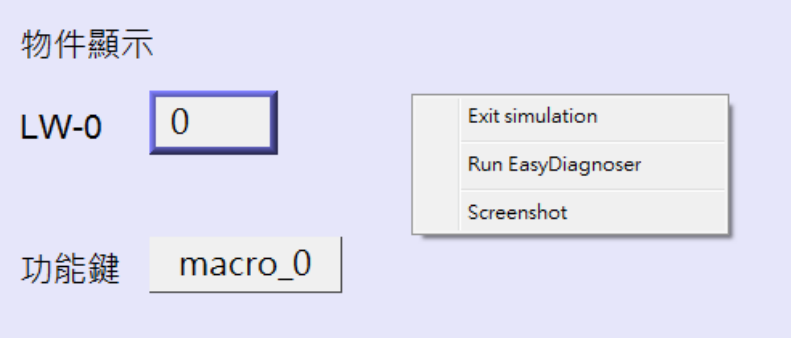
1. 首先，請在工程檔案中新增 macro_0，並在 macro_0 的內容中加入 TRACE("LW = %d", a)，"%d"表示使用 10 進制顯示 LW 目前的數值。macro_0 的內容如下：

```
1
2   macro_command main()
3
4   short a
5   GetData(a, "Local HMI", LW, 0, 1)
6   a=a+1
7   SetData(a, "Local HMI", LW, 0, 1)
8   TRACE ("LW0 = %d" , a)
9
10  end macro_command
```

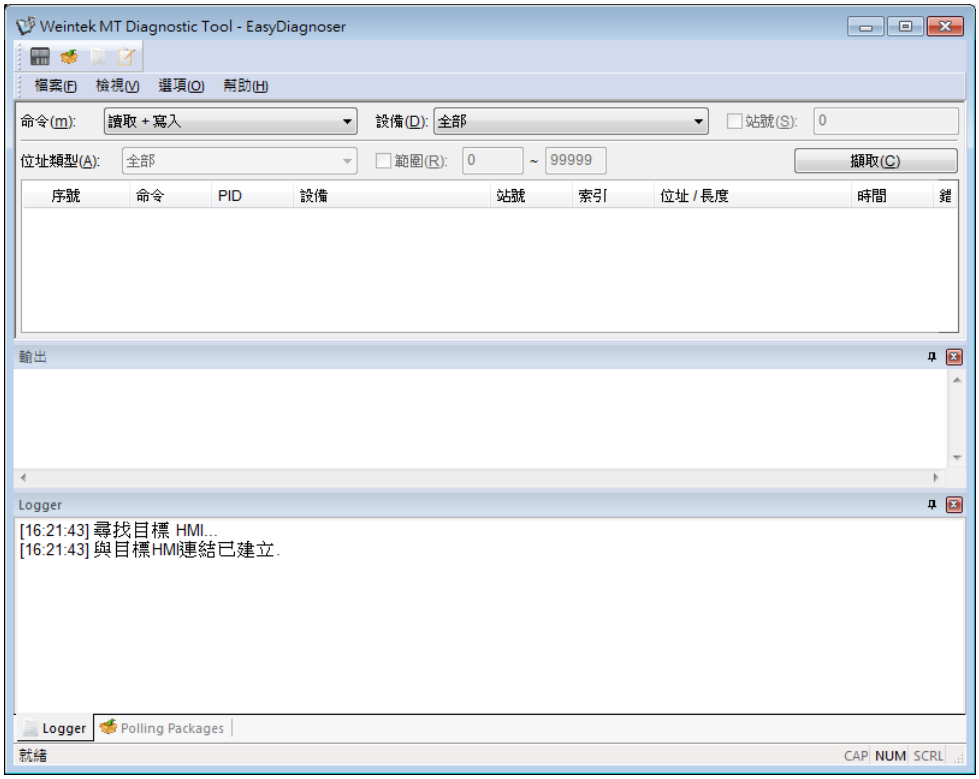
2. 接著在工程檔案的第 10 頁分別加上 [數值顯示] 與 [功能鍵] 物件，[功能鍵] 物件用來執行 macro_0。



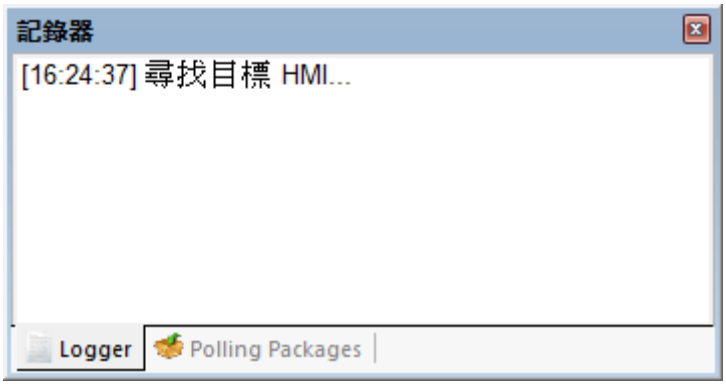
3. 最後編譯已完成的工程檔案並執行離線或線上模擬。
4. 在 PC 上進行模擬功能時，點擊滑鼠右鍵後選擇選單上的 "Run EasyDiagnoser"。



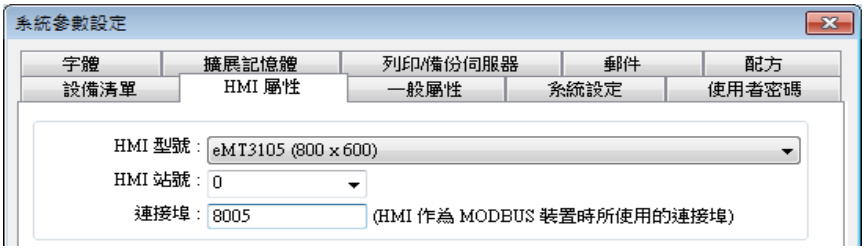
5. 此時即會出現 EasyDiagnoser 的畫面，[記錄器] 窗口用來顯示 EasyDiagnoser 是否可以連接上需要監視的 HMI，[輸出] 窗口用來顯示 TRACE 的執行結果，下圖表示 EasyDiagnoser 已成功連接上 HMI。



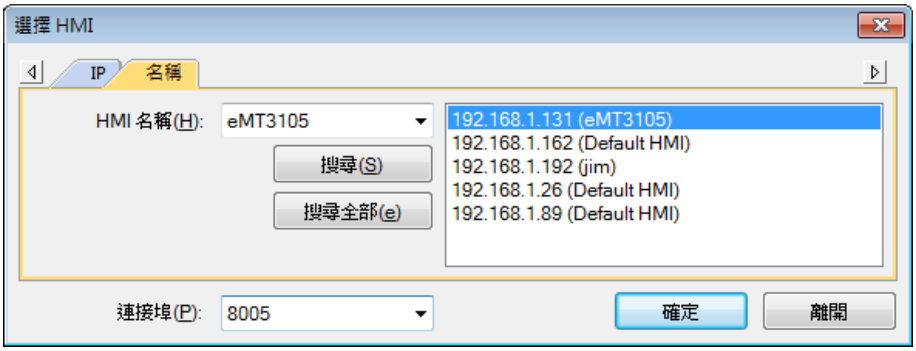
若未成功連接上 HMI，[記錄器] 的窗口會顯示下面的內容：



6. 未連線成功的可能原因是 PC 未成功執行模擬功能；另一個原因是在 PC 執行模擬功能的工程檔案所使用的 [連接埠] 不正確 (可能已被系統佔用)，此時請更改工程檔案的 [連接埠] (請參考下圖)，再次編譯重新執行模擬功能即可。



7. 開啟 EasyDiagnoser 時，應設定與工程檔案相同的連接埠。



從所設定的 [連接埠] 算起連續 3 個 port 將保留給 HMI 做通訊用。例如，上圖中設定 [連接埠] 為 8005，則 8005、8006、8007 三個 port 將被保留。因此在 PC 上模擬時，應確定這些被保留的 port 未被其他程式所佔用。

TRACE 命令語法如下：

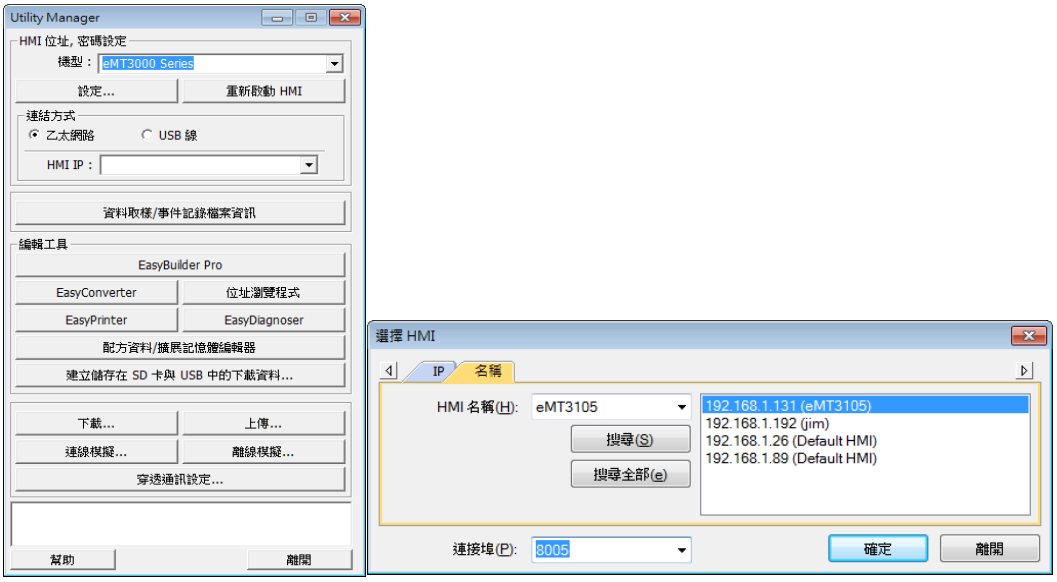
函數名稱	TRACE
語法	TRACE(format, argument)
描述	一個執行中的 巨集指令可以使用此函數，監視變數內容的變化，並列印字串，以協助除錯。使用者應開啟 EasyDiagnoser /cMT Diagnoser觀看此函數的輸出結果。 當 TRACE 函數抓取一個%開頭的特殊字元，將同時從 argument 抓取一個參數做格式化後輸出。 Format 代表列印格式，支援 % 開頭的特殊字元。特殊字元格式如下，其中方括號內的欄位為可選，紅色文字欄位為必需： %[flags] [width] [.precision] type 每個欄位的意義如下所述： flags (可選)： - ：靠左對齊。指定寬度後，不足的左邊補空格 + ：輸出正負號 width (可選)： 十進位正整數，指定應預留的字元寬度，不足部份補空白字元。 precision (可選)： 十進位正整數，指定精確度，以及輸出字元數。 type： C 或 c ： 以字元方式輸出 d ： 以 signed 十進位整數輸出 i ： 以 signed 十進位整數輸出 o ： 以 unsigned 八進位整數輸出 u ： 以 unsigned 十進位整數輸出 X 或 x ： 以 unsigned 十六進位整數輸出 lld ： 以長整數 (64bit) signed 變數輸出 (只支援於cMT / cMT X) llu ： 以長整數 (64bit) unsigned 變數輸出 (只支援於cMT / cMT X)

	f : 以單倍精確度浮點數輸出。 llf : 以雙倍精確度浮點數輸出。 E 或 e : 以科學表示法輸出。格式為[–] d.dddd e [sign]ddd。其中欄位d是十進位數字，欄位dddd是一至多個十進位數字，欄位ddd必須是三個十進位數字。sign是+或-。 *Format* 字串最長支援 256 個字元，多出的字元將被忽略。 *Argument* 部份可寫可不寫。但一個特殊字元應搭配一個變數。

舉例	macro_command main()
	char c1 = 'a'
	short s1 = 32767
	float f1 = 1.234567
	
	TRACE("The results are") // 輸出：The results are
	TRACE("c1 = %c, s1 = %d, f1 = %f", c1, s1, f1)
	// 輸出： c1 = a, s1 = 32767, f1 = 1.234567
	
	end macro_command

8. 新增 LB-9059 –關閉巨集指令 TRACE 功能 (當狀態為 ON)。當設定為 ON 時，TRACE 將不會把數據輸出到 EasyDiagnoser。

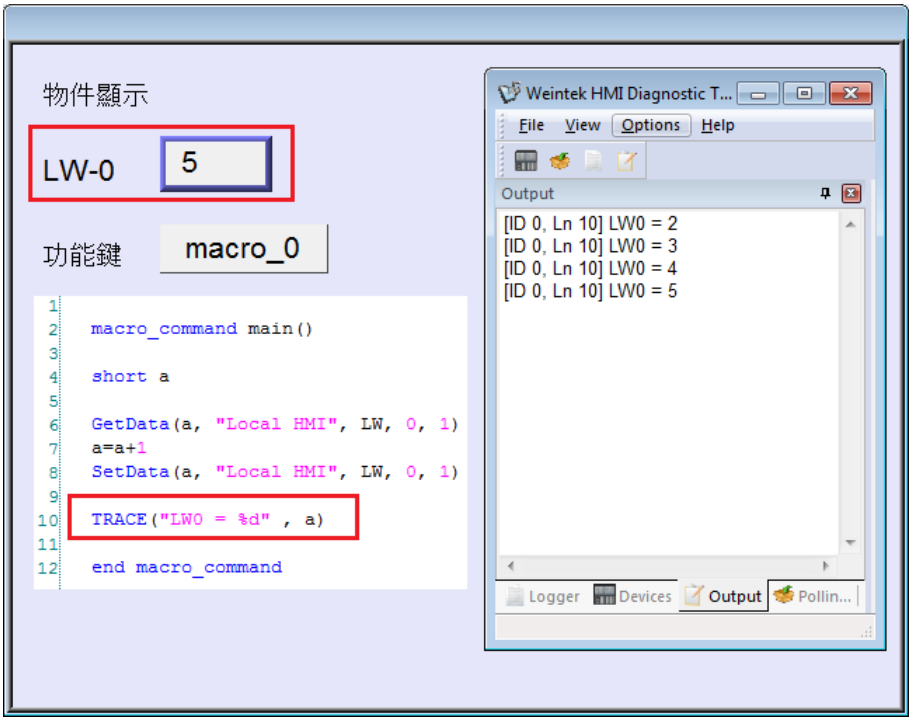
9. 也可以直接利用 Utility Manager 執行 EasyDiagnoser.exe，Utility Manager 將會顯示網路上目前存在的 HMI，此時只要選擇要監看通訊狀態的 HMI 即可。請注意 Project Port 的部份應設定與工程檔案所使用的 [連接埠] 相同。



10. 將工程檔下載到 HMI 實際操作。當 EasyDiagnoser 無法連接上需要監視的 HMI 時，一般可能的原因是 HMI 未上電。或是 [連接埠] 不正確，可能發生 EasyDiagnoser 不斷連上 HMI 又斷線的情況。請確定 EasyDiagnoser 應設定與工程檔案相同的 Port No.，更改方式如前面所述。

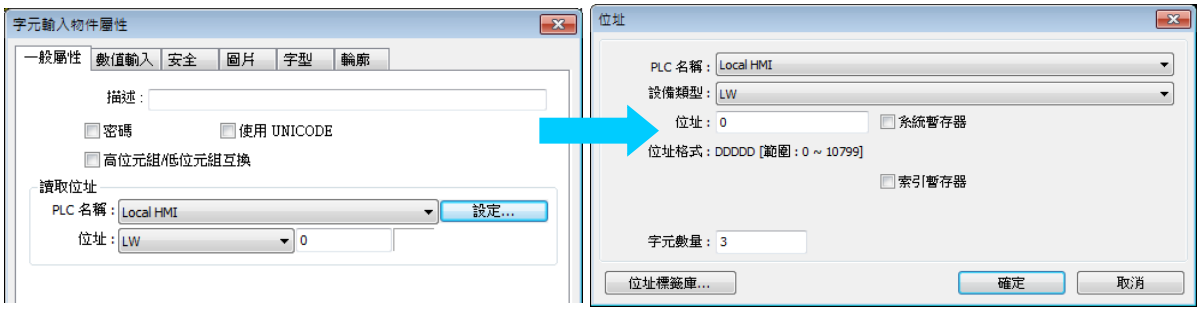
11. 當 EasyDiagnoser 成功與 HMI 連結後，只要執行 macro_0，即可發現 [輸出] 窗口顯示目

前 TRACE 的執行結果。



18.15. 字串處理函式使用方法

巨集指令提供字串處理函式，令使用者可以很方便的對字串進行各種操作。所謂字串，是由一連串的 ASCII 字元組成，每一個 ASCII 字元佔用 1 位元組 (byte)，在 16 位元暫存器中是以低位元組優先方式儲存。舉例來說，我們利用一個 [文字輸入] 物件在 LW-0 ~ LW-2 的位置，寫入一條字串 "abcdef"，設定如下：



在此 [文字輸入] 物件輸入"abcdef"：

此字串在 LW-0 ~ LW-2 中的存放方式如下圖所示 (LB 代表低位元組，HB 代表高位元組)：

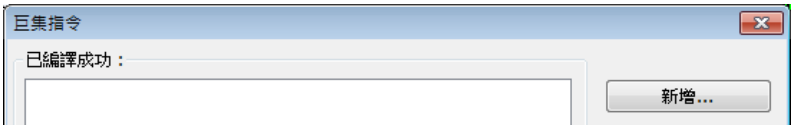
	HB	LB
LW0	'B'	'A'
LW1	'D'	'C'
LW2	'F'	'E'
LW3		
LW4		
LW5		

由於 [文字輸入] 物件顯示的資料長度單位為 word，而每個 ASCII 字元長度為一個 byte，所以每次最少會顯示兩個字元，此部份在 [文字輸入] 物件的章節有詳細的說明。因此上面的例子中，設定 [文字輸入] 物件的字組數量為 3，表示最多可輸入 / 顯示 6 個 ASCII 字元。

下面將舉例說明如何從新增一個巨集指令的方法開始，一直到執行模擬為止，帶您一步步建立一個可執行的工程檔，以實際體會字串處理函數強大的功能。

1. 以下說明如何從暫存器讀入與寫入一個字串。

請新增一個巨集指令：



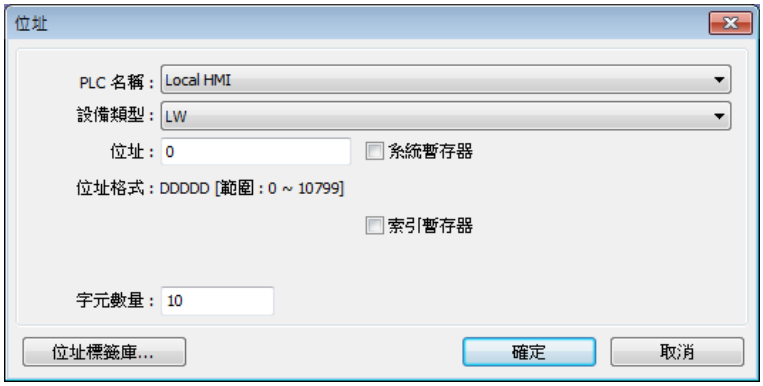
在巨集指令編輯器編輯以下內容：

```
1
2   macro_command main()
3
4   char str[20]
5
6   StringGet(str[0], "Local HMI", LW, 0, 20)
7   StringSet(str[0], "Local HMI", LW, 50, 20)
8
9   end macro_command
```

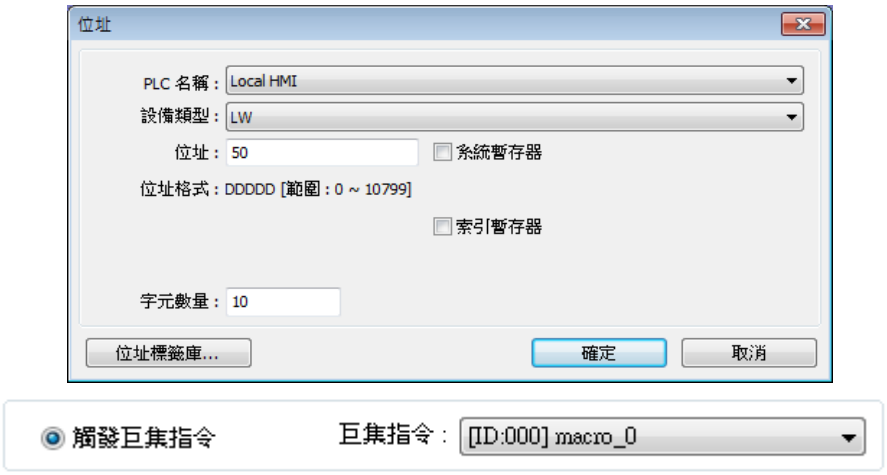
此巨集指令的目的是利用 StringGet 函數從暫存器中讀入一條字串放入字元陣列 str 中，並利用 StringSet 輸出 str 的內容。

接著在工程檔案的第 10 頁分別加上 [文字輸入] 與 [功能鍵] 物件，物件的設定內容請參考下圖，[功能鍵] 物件用來執行 macro_0。

[文字輸入] 物件



[功能鍵] 物件

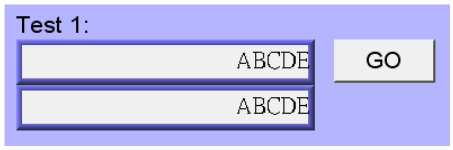


最後編譯 已完成的工程檔案並執行離線 (off-line) 或線上 (on-line) 模擬，並在模擬畫面按照以下步驟操作：

Step 1. 輸入字串

Step 2. 按下 “GO” 按鈕

Step 3. 顯示結果

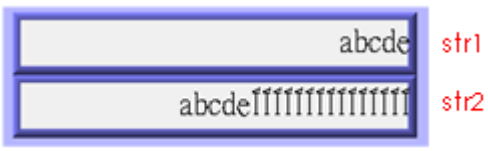


2. 字串之初始化。

在巨集指令編輯器編輯以下內容：

```
1
2   macro_command main()
3
4   char str1[20]="abcde"
5   char str2[20]={'a','b','c','d','e'}
6
7   StringSet(str1[0], "Local HMI", LW, 0, 20)
8   StringSet(str2[0], "Local HMI", LW, 50, 20)
9
10  end macro_command
```

用雙引號 (" ") 刮起來的內容視為一個字串。str1 是以字串方式初始化，而 str2 是以字元陣列方式初始化。



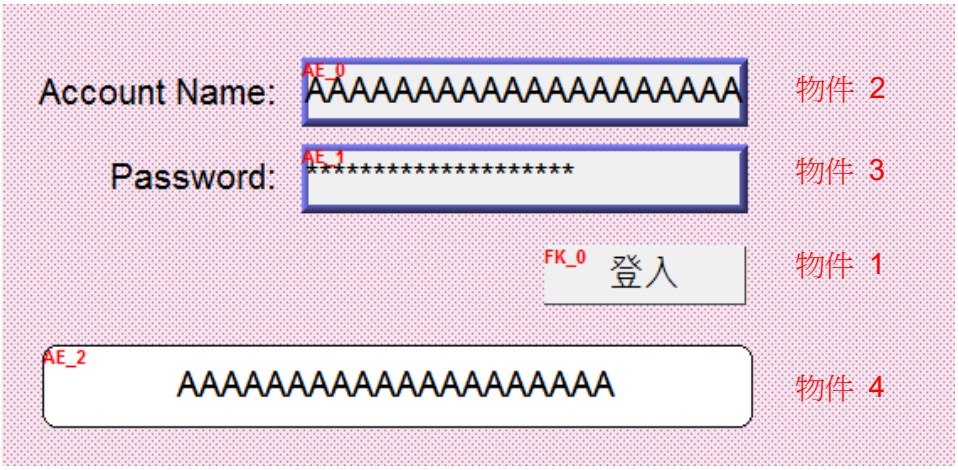
以字串方式初始化時巨集編譯器會在字串結尾後面加上結束符號 ’\0’ 代表字串結束。利用

StringSet 將字串寫入暫存器時遇到結束符號便會停止繼續寫入陣列後面的內容。即使 data count 被設為大於字串長度的數值，顯示字串時只會顯示結束符號之前的內容。
以字元陣列方式初始化時陣列內容並不會被視為一個字串，因此也不會加上結束符號 '\0'。
寫入暫存器的字元數會依據 data count 所設定的數值決定。

3. 一個簡單的帳號密碼登入頁面。
在巨集指令編輯器編輯以下內容(巨集指令 [ID:001] macro_1)：

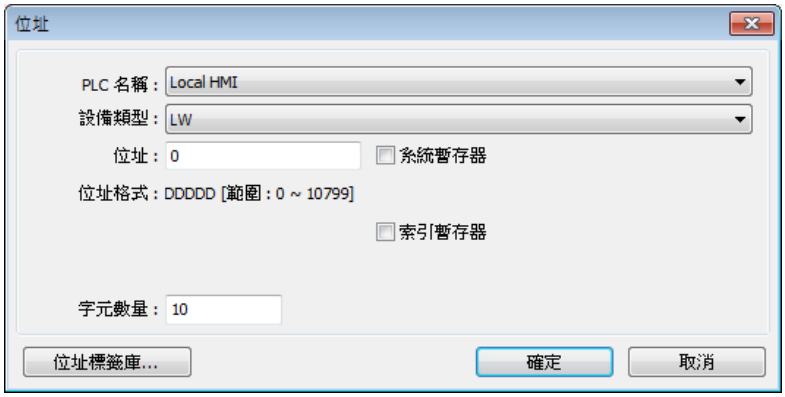
```
1
2   macro_command main()
3
4   char name[20]="admin"
5   char password[20]="123456"
6   char name_input[20], password_input[20]
7   char message_success[40]="Success! Access Accepted."
8   char message_fail[40]="Fail! Access Denied."
9   char message_clear[40]
10  bool name_match=false, password_match=false
11
12  StringGet(name_input[0], "Local HMI", LW, 0, 20)
13  StringGet(password_input[0], "Local HMI", LW, 50, 20)
14
15  name_match = StringCompare(name_input[0], name[0])
16  password_match = StringCompare(password_input[0], password[0])
17
18  FILL(message_clear[0], 0x20, 40) //FILL with white space
19  StringSet(message_clear[0], "Local HMI", LW, 100, 40)
20
21  if (name_match==true and password_match==true) then
22      StringSet(message_success[0], "Local HMI", LW, 100, 40)
23  else
24      StringSet(message_fail[0], "Local HMI", LW, 100, 40)
25  end if
26
27  end macro_command
```

此巨集將利用 [StringGet] 取得使用者輸入的帳號密碼的字串，分別放入 name_input 與 password_input 兩個陣列中。接著利用 [StringCompare] 比對帳號密碼，若帳號相符，則 name_match 設為 true；若密碼相符，則 password_match 設為 true。最後檢查 name_match 與 password_match 的值，若同時為 true，表示登入成功，並印出 "Success! Access Accepted." 字串。若其中之一不相符，則印出 "Fail! Access Denied. " 字串。
接著在工程檔案的第 10 頁分別加上 [文字輸入] 與 [功能鍵] 物件，物件的設定內容請參考下圖，[功能鍵] 物件用來執行 macro_1。

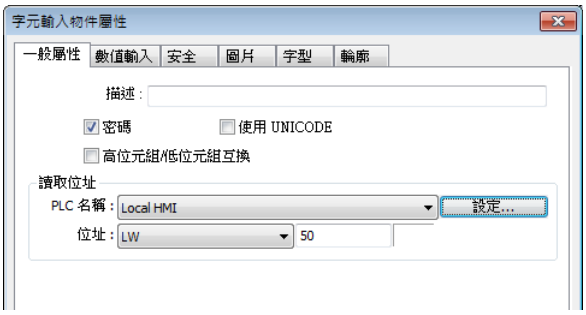


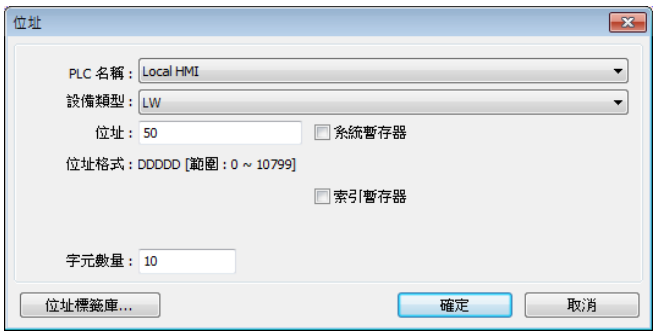
物件 1：[功能鍵] 物件 ，點選 [觸發巨集指令] 並選擇巨集指令 [ID:001] macro_1。

物件 2：[文字輸入] 物件



物件 3：[文字輸入] 物件

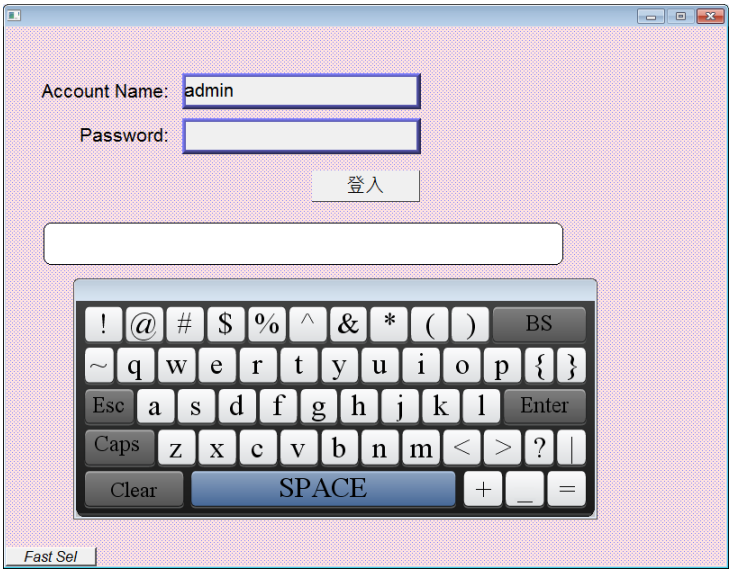




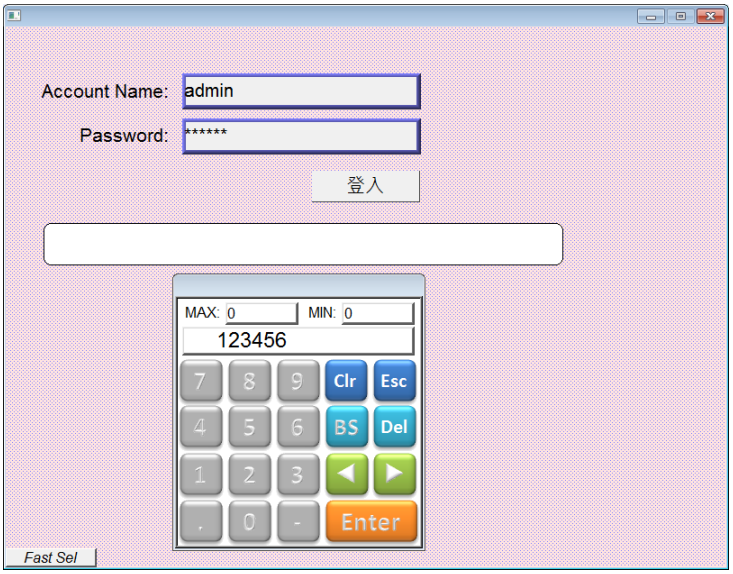
物件 4：[文字顯示] 物件

最後編譯 已完成的工程檔案並執行離線 (off-line) 或線上(on-line) 模擬，並在模擬畫面按照以下步驟操作：

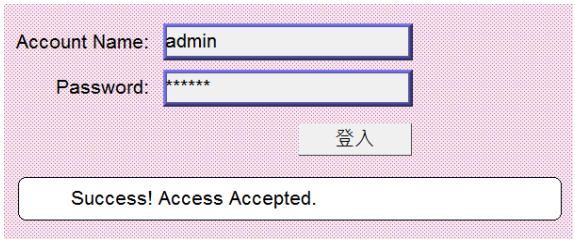
1. 輸入帳號。



2. 輸入密碼並按下登入鍵。



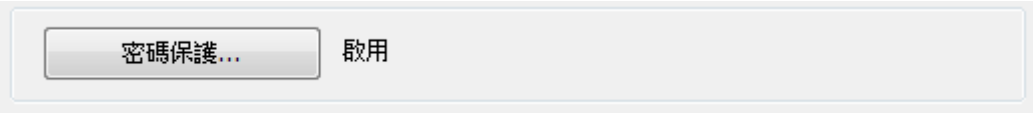
3. 顯示登入成功或登入失敗。



18.16. 巨集指令密碼保護

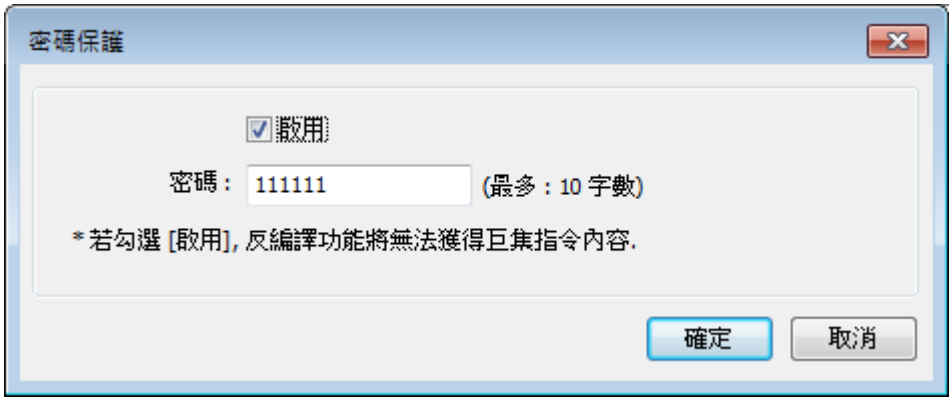
巨集指令密碼保護密碼保護分為兩層：所有巨集密碼保護與單一巨集密碼保護

所有巨集保護：

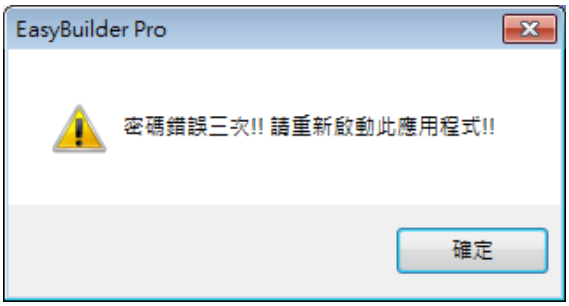


在所有巨集列表提供 [密碼保護] 選項，當點選 [密碼保護] 後勾選 [啟用] 按鈕可以設定密碼，密碼最多不可超過 10 個字元 (只支援 ASCII 文字，並區分大小寫，例如可以輸入 "a$#*hFds")。

當設定所有巨集的 [密碼保護] 功能後，用戶欲開啟巨集時，需先輸入正確的密碼。



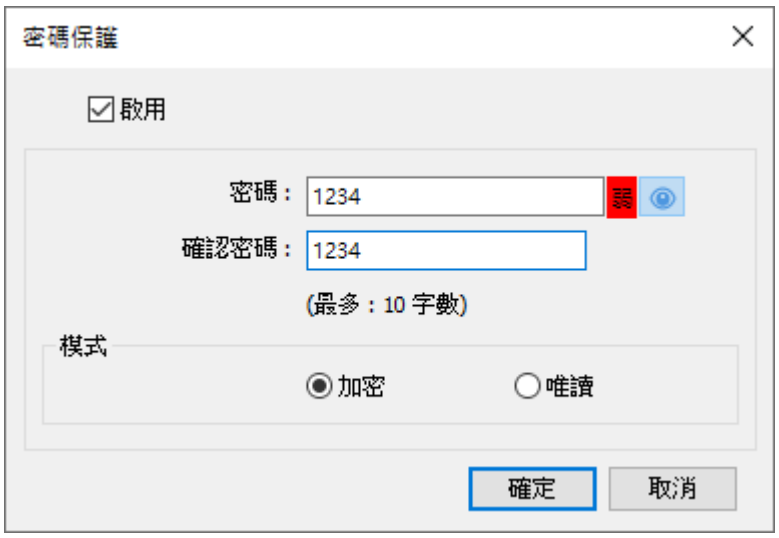
當輸入不正確的密碼三次，需重新啟動 EasyBuilder Pro，才能重新輸入密碼。



單一巨集密碼保護：

在巨集指令編輯器裡提供[密碼保護] 選項，當點選 [密碼保護] 後勾選 [啟用] 按鈕可以設定密

碼。密碼最多不可超過 10 個字元 (只支援 ASCII 文字，並區分大小寫，例如可以輸入 "a$#*hFds")。



密碼保護方式分為 [加密] 與 [唯讀]。

[加密]

對巨集內容加密，從巨集列表點選巨集欲進入編輯窗口時需先正確輸入密碼。

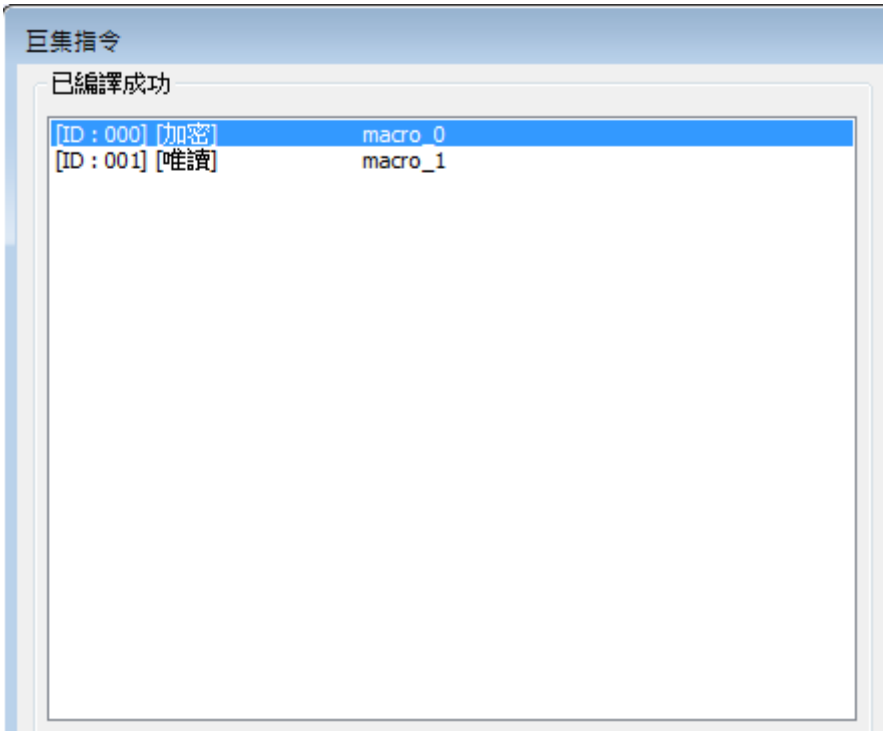
密碼輸入錯誤三次，將提醒需重啟 EasyBuilder Pro (每一項巨集所需次數各自獨立) 。

[唯讀]

用戶將僅能檢視巨集的內容，無法編輯。

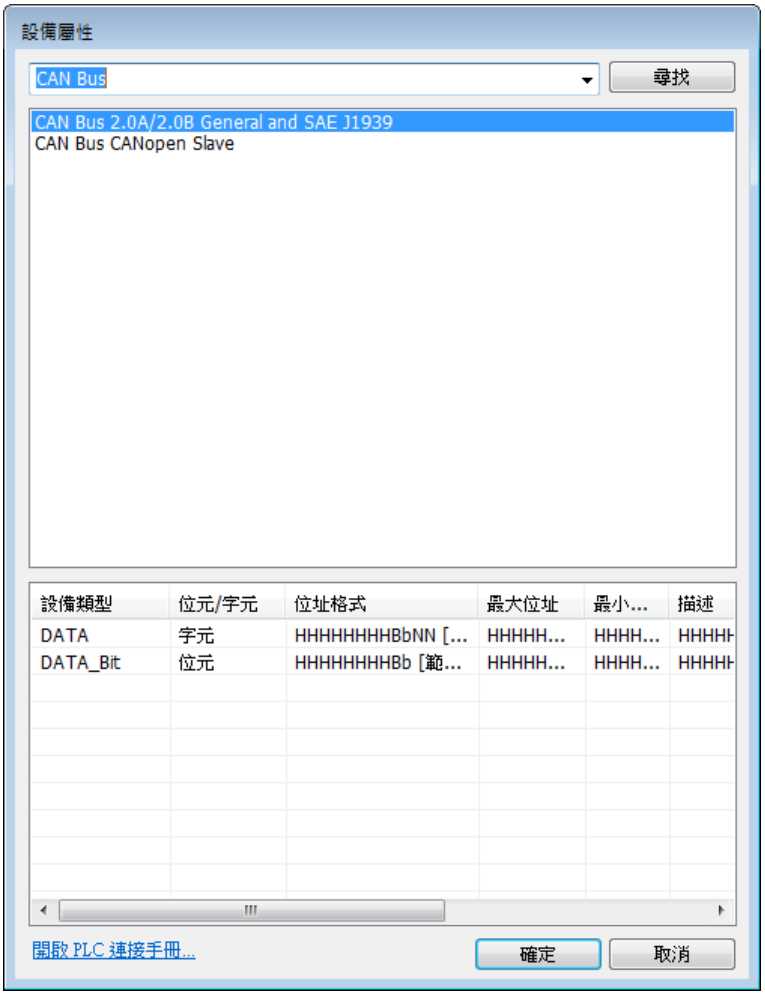
選用此模式時可以由巨集列表直接進入巨集指令編輯器， 但在進到[密碼保護]設定窗口時就需要輸入正確的密碼。密碼的輸入規則與 [加密] 相同，限制只能錯誤輸入三次。

巨集列表中會顯示單一巨集所使用的保護模式。



18.17. 巨集支援使用變數讀寫 CANbus 地址

當使用 CAN Bus 2.0A/2.0B General and SAE J1939 驅動程式時，會發現該驅動程式擁有兩種地址類型：DATA 與 DATA_Bit，資料格式的描述如下表所示。



資料格式	描述
DATA HHHHHHHHBbNN	HHHHHHHH: ID B: Byte 位置(1~8) b: Bit 位置(1~8) NN: Bit 數量(1~64)
DATA_Bit HHHHHHHHBb	HHHHHHHH: ID B: Byte 位置(1~8) b: Bit 位置(1~8)

由於 ID 是使用 16 進制表示，位置與數量則是使用 10 進制表示，請參考下方的使用方式。

範例

未採用變數地址的表示法

```
short f
GetData(f, “CAN Device”, DATA, 4e55108, 1)
GetData(f, “CAN Device”, DATA, 4e65108, 1)
```

採用變數地址的表示法

```
short f
unsigned int address = 0x4e55108
GetData(f, “CAN Device”, DATA, address, 1)
address = address + 0x10000//   == 0x4e65108
GetData(f, “CAN Device”, DATA, address, 1)
```

注意事項：

1. 將變數宣告為 Unsigned int，並使用 16 進制來表示地址。
2. 因為 Unsigned int 的長度為 4 bytes，而且 Bb 與 NN 分別都需要 1 byte 的空間。所以當使用變數作為地址參數時，讀寫 DATA_Bit 地址類型時，地址格式將變動為 HHHHHHBb (最大 ID 只能到 0xffffff)；讀寫 DATA 地址類型時，地址格式將變動為 HHHHBbNN (最大 ID 只能到 0xffff)。