Chapter 25　Macro User's Manual

This document is the user's manual of macro module, which describes syntax, usage, and programming methods of macro commands.

The document includes the following chapters:

Macro Description
Macro Usage Description
Value operation and data transform function
Use macro command to control external device
Macro Commands and PLC Communication (Local Bit, Local Word)
Macro Operation Instruction
Notes about Using Macro
Compiling Error Message
Source Code Examples

● Macro Description

1. Constants and Variables

a. Constants

(1) Decimal constant

(2) Hexadecimal constant

(3) ASCII code (character constant)

(4) Boolean: True (not zero), False (zero)

b. Variables

(1) Naming rules

A variable must start with an alphabet and no longer than 32 characters.

(2) Variable types

char.	Character (8 bit) variable
bool	Boolean(1 bit) variable
short	Short integer(16 bit) variable
int	Double word (32 bit)variable
float	Floating point (32 bit) variable

c. Operator

(1) Assignment operator

Assignment operator：=

(2) Arithmetic operators

Addition	：+
Subtraction	：-
Multiplication	：*
Division	：/
Modulo Division	：%

(3) Comparison operators

Less than	：<
Less than or equal	：<=
Greater than	：>
Greater than or equal	：>=
Equal	：==
Not equal	：<>

(4) Logic operators:

Conditional AND	：and

Conditional OR	：	or
Exclusive OR	：	xor
Boolean NOT	：	not

(5) Bitwise and shift operators:

(a) Shift operators

Left shift	：	<<
Right shift	：	>>

(b) Bitwise operators

Bitwise AND	：	&
Bitwise OR	：	|
Bitwise XOR	：	^
Bitwise complement	：	~

Caution, if you would like to operate the value for Logic operators, please use Bitwise command.

2. Priority of operators

The process order of many operators within an expression is called the priority of operators.

a. Priority of the same kind of operator (From left to right, from up to low)

Arithmetic operator: ^ → (* , /) → (mod) → (+ , -)

Shift operator: From left to right within the expression

Comparison operator: From left to right within the expression

Logic operator: Not → And → Or → Xor,

b. Arithmetic operator is prior to Bitwise operator

Bitwise operator is prior to Comparison operator

Logic operator is prior to Assignment operator

3. Array

Only support fixed length, 1-D array which is:

1-D array: Array_Name [Array_Size]

The array size can be integer which from 0 to 4294967295

Minimum of array index = 0

Maximum of array index = Array size – 1

Example : Array[MAX] MAX = 100

Minimum of array index = 0

Maximum of array index = 99 (100 – 1)

4. Expression
  a. Operation object
    (1) Constants
    (2) Variables
    (3) Array
    (4) Function
  b. Components of expression
   An expression is combined operation objects with operators by following specific rules.

5. Statement
  a. Definition statement
    (1) type name: Define the type of name
            Example: int a，Define variable a as an integer
    (2) type name[constant]:   Define the type of array name
            Example: int a[10]，Define variable a as a 1-D array of size 10

    Assignment statement
    The form is ： Variable = Expression
     Example: a = 2

   b. Logic statement and branches
    (1) One-line format

```
if Condition then
        [Statements]
end if
```

Example:

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

(2) Block format

```
if Condition then
        [Statements]
[else [if Condition – n then
                [Else_If_Statements] ….
[else
        [Else_Statements]]
] ]
end if
```

Example:

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

Syntax description

Condition	Necessary. This is a control statement. It will be FALSE when the value of condition is 0; and will be TRUE when the value of condition is 1.
Statements	It is optional in block format statement but necessary in one-line format without ELSE. The statement will be executed when the condition is TRUE.
Condition – n	Optional. See Condition.
Else_If_Statements	Optional in one-line or multi-line format statement. The **else if** statement will be executed when the relative Condition – n is TRUE.
Else_Statements	Optional. The **else** statement will be executed when Condition and Condition—n are both FALSE.

c. Looping control

(1) for–next Statement

Use this for fixed execution counts. To means increase by step while down means decrease by step.

```
for Counter = Start to end [step Step]
    [Statements]
next [Counter]

for Counter = Start down end [step Step]
    [Statements]
next [Counter]
```

Example:

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

Syntax description

Counter	Necessary. The counter of looping control. It can be integer or character.
Start	Necessary. The initial value of Counter.
End	Necessary. The end value of Counter.
Step	Optional. The increment/decrement step of Counter. It can be integer and can be omitted when value is 1.
Statements	Optional. Statement block between For and Next which will be executed fixed counts.

(2) while – wend statement

Loop controlled by Condition. When Condition is TRUE, the statements will be executed repetitively until the condition turns to FALSE.

```
while Condition
    [statements]
wend
```

Example:

```
while a == 2
    b = b + 1
    GetData(a, “Local HMI”, LB, 5, 1)
wend
```

Syntax description

Condition	Necessary. Logic expression which control the execution of statements.
Statements	Optional. Statement block. The statement will be executed when the condition is TRUE.

(3) break

Used in looping control or select statement. It skips immediately to the end of the statement.

(4) continue

Used in looping control statement. It quits the current iteration of a loop and starts the next one.

(5) return

To stop executing the current method.

Reserved keywords:

The following keywords are reserved for Macro which can not be used in function name, array name, or variable name.

+ , - ,* , / ,^, mod, >= ,>, < ,<=, <> , == , And, Or, Xor, Not, <<, >>, = , & , |, ^,~,If ,Then, Else, EndIf, Select ,Case ,For, To, Down Step, Next, while, wend break, continue, return.

- Macro usage description

1. Local variables and global variables
   a. Local variables: Its value remains valid only within a specific statement.
   b. Global variables: Its value always remains valid after declaration.

When local variable and global variable have the same declaration name, only the local variable will be valid.

2. Variable and constant initialization
   a. Variable initialization
      (1) Initialize a value of variable in the declaration statement directly.
          Example: int h = 9
      (2) Use assignment operator to initialize a value after declaration.
          Example: temp = 9
      (3) Array initialization
          Format: int g[10] = { 1,2,3, , 10 }

   The initial values are written within the {} and divided by comma (,). These values are assigned orderly from left to right starting from array index=0.

   b. Constans.
      Macro supports:
      (1) Decimal integer constant
      (2) Hexadecimal integer constant: start with 0x
      (3) Character constant,
      (4) Boolean constant: True / False,

3. Boolean variables and Boolean expressions
   a. Boolean variables:
      True or False. Not zero value means TRUE while zero value means FALSE.
   b. Boolean expressions:
      The value of Boolean expression is not zero mean TRUE.
      The value of Boolean expression is zero mean FALSE.

4. Declaration statement

a. Declaration outside a function is a global variable declaration.

b. Declaration inside a function is local variable declaration. This declaration must at the very beginning of a statement within a function. Other statements before declaration statements will cause compiler error.

For example：

```
macro_command main( )
char i

i = 9//Assign statement within declaration area causes compiler error
int g[10]

for g[2] = 0 to 2
g[3] = 4
next g[2]
end macro_command
```

5. Function call and passing parameters

a. Function define

The format of function statement is:

```
sub Type FunName(Type1 var1, Type2 var2, …, TypeN varN)
        ……..
        return ret
end sub
```

Type is the return datae type，FunName is function name，Type1~TypeN is the parameter variable type，var1~ varN is parameter, ret is the return data

For example：

```
 sub int func(int i)
        int h

                h = i + 10
        …..
        return h
```

end sub

b. Function call

A function must be defined before its execution. Otherwise, a compiler error 'Function not defined' will occur.

For example：

```
macro_command main()
    int   i
    i = Func(i) // call an undefined function causes compiler error
end macro_command
```

c. Passing parameters

(1) Passing by value through local variable.

(2) Through the same global variables

6. Main Function

Macro must has one and only one main function which is the execution start point of Macro. The format is:

```
macro_command Function_name()


end macro_command
```

- **Value operation and data transform function**

(1) Value operation

■ **SQRT**

Radical operation

Syntax SQRT(source, result)

SQRT according to the radical operation of source which is storage in result, source can be constants or variables, result must be variables.

For example:

```
macro_command main()

float source, result

SQRT(15, result)

GetData(source, "Local HMI", LW, 0, 1)//    source == 9.0

SQRT(source, result)//    result == 3.0

SetData(result, "Local HMI", LW, 0, 1)

end macro_command
```

■ **SIN**

Sine operation

Syntax SIN(source, result)

SIN according to the sine operation of source which is storage in result, source is angle value and can be constants or variables, result must be variables.

For example:

```
macro_command main()

float source, result

SIN(90, result)//    result == 1

GetData(source, "Local HMI", LW, 0, 1)
```

```
SIN(source, result)
//    source == 30 => result == 0.5

SetData(result, "Local HMI", LW, 0, 1)

end macro_command
```

■ **COS**

Cosine operation

Syntax   COS(source, result)

COS according to the cosine operation of source which is storage in result, source is angle value and can be constants or variables, result must be variables.

For example:

```
macro_command main()

float source, result

COS(90, result)//    result == 0

GetData(source, "Local HMI", LW, 0, 1)

COS(source, result)
//    source == 60 => result == 0.5

SetData(result, "Local HMI", LW, 0, 1)

  end macro_command
```

■ **TAN**

Tangent operation

Syntax   TAN(source, result)

TAN according to the tangent operation of source which is storage in result, source is angle value and can be constants or variables, result must be variables.

For example:

```
macro_command main()
```

```
float source, result

TAN(45, result)//    result == 1

GetData(source, "Local HMI", LW, 0, 1)

TAN(source, result)
//    source == 60 => result == 1.732

SetData(result, "Local HMI", LW, 0, 1)

  end macro_command
```

- **COT**

  Cotangent operation

  Syntax COT(source, result)

  COT according to the cotangent operation of source which is storage in result, source is angle value and can be constants or variables, result must be variables.

For example:

```
macro_command main()

float source, result

COT(45, result)//    result == 1

GetData(source, "Local HMI", LW, 0, 1)

COT(source, result)
//    source == 60 => result == 0.5774

SetData(result, "Local HMI", LW, 0, 1)

  end macro_command
```

- **SEC**

  Secant operation

Syntax SEC(source, result)

SEC according to the secant operation of source which is storage in result, source is angle value and can be constants or variables, result must be variables.

For example:

```
macro_command main()

float source, result

SEC(45, result)//    result == 1.414

GetData(source, "Local HMI", LW, 0, 1)

SEC(source, result)
//    source == 60 => result == 2

SetData(result, "Local HMI", LW, 0, 1)

  end macro_command
```

■ **CSC**

Cosecant operation

Syntax SEC(source, result)

CSC according to the cosecant operation of source which is storage in result, source is angle value and can be constants or variables, result must be variables.

For example:

```
macro_command main()

float source, result

CSC(45, result)//    result == 1.414

GetData(source, "Local HMI", LW, 0, 1)

CSC(source, result)
//    source == 30 => result == 2
```

```
SetData(result, "Local HMI", LW, 0, 1)

  end macro_command
```

■ **ASIN**

The inverse function of sine operation

Syntax ASIN(source, result)

ASIN according to the inverse function of sine operation of source which is storage in result, source can be constants or variables, result is angle value and must be variables.

For example:

```
macro_command main()

float source, result

ASIN(0.8660, result)//    result == 60

GetData(source, "Local HMI", LW, 0, 1)

ASIN(source, result)
//    source == 0.5 => result == 30

SetData(result, "Local HMI", LW, 0, 1)

  end macro_command
```

■ **ACOS**

The inverse function of cosine operation

Syntax ACOS(source, result)

ACOS according to the inverse function of cosine operation of source which is storage in result, source can be constants or variables, result is angle value and must be variables.

For example:

```
macro_command main()

float source, result
```

```
ACOS(0.8660, result)//    result == 30

GetData(source, "Local HMI", LW, 0, 1)

ACOS(source, result)
//    source == 0.5 => result == 60

SetData(result, "Local HMI", LW, 0, 1)

  end macro_command
```

- **ATAN**

  The inverse function of tangent operation

  Syntax   ATAN(source, result)

  ATAN according to the inverse function of tangent operation of source which is storage in result, source can be constants or variables, result is angle value and must be variables.

For example:

```
macro_command main()

float source, result

ATAN(1, result)//    result == 45

GetData(source, "Local HMI", LW, 0, 1)

ATAN(source, result)
//    source == 1.732 => result == 60

SetData(result, "Local HMI", LW, 0, 1)

  end macro_command
```

- **RAND**

  A random value

  Syntax   RAND(result)

When use RAND(), RAND will to appear a random value, then storage the value in result and it must be variables.

For example:

```
macro_command main()

short random

RAND(random)

SetData(random, "Local HMI", LW, 120, 1)

end macro_command
```

(2) Value transformation

■ **BIN2BCD**

Transform decimal system value into BCD value

Syntax BIN2BCD(source, result)

BIN2BCD according to the decimal system to transform into BCD of source which is storage in result, source can be constants or variables, result must be variables.

For example:

```
macro_command main()

short source, result

BIN2BCD(1234, result)//    result == 0x1234

GetData(source, "Local HMI", LW, 4, 1)

BIN2BCD(source, result)
//    source == 5678 => result == 0x5678

SetData(result, "Local HMI", LW, 6, 1)

end macro_command
```

- **BCD2BIN**

  Transform BCD value into decimal system value

  Syntax   BCD2BIN(source, result)

  BCD2BIN according to the BCD to transform into decimal system of source which is storage in result, source can be constants or variables, result must be variables.

For example:

```
macro_command main()

short source, result

BCD2BIN(0x1234, result)//    result == 1234

GetData(source, "Local HMI", LW, 4, 1)

BCD2BIN(source, result)
//    source == 0x5678 => result == 5678

SetData(result, "Local HMI", LW, 6, 1)

  end macro_command
```

- **DEC2ASCII**

  Transform decimal system value into ASCII

  Syntax   DEC2ASCII(source, result[start], no)

  DEC2ASCII according to the decimal system to transform into ASCII of source which is sequence storage in result[], no is to be transformed character. The first character storage in result[start], the second character storage in result[start + 1], the last character storage in result[start + (no – 1)].

  Source of no can be constants or variables, result must be matrix type variables.

For example:

```
macro_command main()

short source
```

```
char result[4]

GetData(source, "Local HMI", LW, 30, 1)

DEC2ASCII(source, result[0], 4)//    no. of ASCII == 4
//    source == 5678 =>
//    result[0] == '5', result[1] == '6', result[2] == '7', result[3] == '8'

SetData(result[0], "Local HMI", LW, 40, 4)//    write 4 bytes == 2 words

end macro_command
```

For variables type of result is char (size is byte), when to execute above the example, LW storage contents as follow：

```
[LW40] == 0x3635
[LW41] == 0x3837
```

When to change variables type of result into short (size is word)

```
macro_command main()

short source
short result[4]

GetData(source, "Local HMI", LW, 30, 1)

DEC2ASCII(source, result[0], 4)//    no. of ASCII == 4
//    source == 5678 =>
//    result[0] == '5', result[1] == '6', result[2] == '7', result[3] == '8'

SetData(result[0], "Local HMI", LW, 40, 4) //    write 4 words

end macro_command
```

When to execute above the example, LW storage contents as follow：

```
[LW40] == 0x0035     (==‘5’)
[LW41] == 0x0036     (==‘6’)
[LW42] == 0x0037     (==‘7’)
[LW43] == 0x0038     (==‘8’)
```

When to change variables type of result into int (size is double words)

```
macro_command main()

short source
int result[4]

GetData(source, "Local HMI", LW, 30, 1)

DEC2ASCII(source, result[0], 4)//   no. of ASCII == 4
//   source == 5678 =>
//   result[0] == '5', result[1] == '6', result[2] == '7', result[3] == '8'

SetData(result[0], "Local HMI", LW, 40, 4) //   write 4 double words

end macro_command
```

When to execute above the example, LW storage contents as follow：

```
[LW40] == 0x0035     (==‘5’)
[LW41] == 0x0000
[LW42] == 0x0036     (==‘6’)
[LW43] == 0x0000
[LW44] == 0x0037     (==‘7’)
[LW45] == 0x0000
[LW46] == 0x0038     (==‘8’)
[LW47] == 0x0000
```

■ **HEX2ASCII**

Transform hexadecimal system value into ASCII

Syntax   HEX2ASCII(source, result[start], no)

HEX2ASCII according to the hexadecimal system to transform into ASCII of source which is sequence storage in result[], no is to be transformed character. The first character storage in result[start], the second character storage in result[start + 1], the last character storage in result[start + (no – 1)].
Source of no can be constants or variables, result must be matrix type variables.

For example:

```
macro_command main()

short source
char result[4]

GetData(source, "Local HMI", LW, 30, 1)

HEX2ASCII(source, result[0], 4)//    no. of ASCII == 4
//    source == 0x5678
//    result[0] == '5', result[1] == '6', result[2] == '7', result[3] == '8'

SetData(result[0], "Local HMI", LW, 40, 4)//    write 4 bytes == 2 words

end macro_command
```

For variables type of result is char (size is byte), when to execute above the example, LW storage contents as follow：

[LW40] == 0x3635
[LW41] == 0x3837

■ **ASCII2DEC**

Transform ASCII into decimal system value

Syntax   ASCII2DEC(source[start], result, no)

ASCII2DEC according to source[] value that ASCII to transform into decimal system of source which is storage in result, the first character storage in source[start], the second character storage in source[start +

1], the last character storage in source[start + (no – 1)], no is to be transformed character.

Source must be matrix type variables, result must be variables and no can be constants or variables.

For example:

```
macro_command main()

char source[4]
short result

GetData(source[0], "Local HMI", LW, 80, 4)

ASCII2DEC(source[0], result, 4)
//    source[0] = '5', source[1] = '6', source[2] = '7', source[3] = '8'    =>
//    result == 5678

SetData(result, "Local HMI", LW, 90, 1)

end macro_command
```

When to execute as follow example, result are equal to 5678

```
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 == 5678

SetData(result, "Local HMI", LW, 90, 1)

end macro_command
```

■ **ASCII2HEX**

Transform ASCII into hexadecimal system value

Syntax ASCII2HEX(source[start], result, no)

ASCII2HEX according to source[] value that ASCII to transform into hexadecimal system of source which is storage in result, the first character storage in source[start], the second character storage in source[start + 1], the last character storage in source[start + (no – 1)], no is to be transformed character.

Source must be matrix type variables, result must be variables and no can be constants or variables.

For example:

```
macro_command main()

char source[4]
short result

GetData(source[0], "Local HMI", LW, 80, 4)

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

SetData(result, "Local HMI", LW, 90, 1)

end macro_command
```

(3) Value manipulation

■ **FILL**

Input specific value into variables

Syntax FILL(source[start], sign, no)

FILL according to variables of source[start] to source[start + (no – 1)]sequence setting to sign. Source must be variables, sign can be constants or variables, no is to be set sum of variables and can be constants or variables.

For example:

```
macro_command main()
```

```
char result[4]
char sign

GetData(sign, "Local HMI", LW, 110, 1)

FILL(result[0], 0x31, 2)
//    result[0] == 0x31, result[1] == 0x31

FILL(result[0], sign, 4)
//    result[0] == result[1] == result[2] == result[3] == sign

SetData(result[0], "Local HMI", LW, 115, 4)//    write 4 bytes == 2 
words

  end macro_command
```

Above the example, variables type of result is char (size is byte), if sign is 0x35, then to execute MACRO, LW storage contents as follow：

[LW115] == 0x3535
[LW116] == 0x3535

When to change variables type of result into short (size is word)：

```
macro_command main()

short result[4]
char sign

GetData(sign, "Local HMI", LW, 110, 1)


FILL(result[0], sign, 4)
//    result[0] == result[1] == result[2] == result[3] == sign

SetData(result[0], "Local HMI", LW, 115, 4)//    write 4 words
```

```
end macro_command
```

Then to execute MACRO(if sign is 0x35)，LW storage contents as follow：

```
[LW115] == 0x0035
[LW116] == 0x0035
[LW117] == 0x0035
[LW118] == 0x0035
```

■ **SWAPB**

To exchange the data of high bye with low byte

Syntax SWAPB(source, result)

SWAPB according to that exchange high byte with low byte of source which is storage in result. Source can be constants or variables, result must be variables.

For example:

```
macro_command main()

short source, result

SWAPB(0x5678, result)//    result == 0x7856

GetData(source, "Local HMI", LW, 125, 1)

SWAPB(source, result)
//    source == 0x1234 => result == 0x3412

SetData(source, "Local HMI", LW, 125, 1)

end macro_command
```

■ **SWAPW**

To exchange the data of high word with low word

Syntax SWAPW(source, result)

SWAPW according to that exchange high word with low word of source which is storage in result. Source can be constants or variables, result must be variables.

For example:

```
macro_command main()

int source, result

GetData(source, "Local HMI", LW, 130, 1)

SWAPW(source, result)
//    source == 0x12345678 => result == 0x56781234

SetData(result, "Local HMI", LW, 130, 1)

end macro_command
```

■ **LOBYTE**

Read value’s low byte

Syntax LOBYTE(source, result)

LOBYTE according to that read value’s low byte of source which is storage in result. Source can be constants or variables, result must be variables.

For example:

```
macro_command main()

short source, result

LOBYTE(0x1234, result)//    result == 0x34


GetData(source, "Local HMI", LW, 140, 1)

LOBYTE(source, result)
//    source == 0x1234 => result == 0x34

SetData(result, "Local HMI", LW, 140, 1)
```

```
end macro_command
```

■ **HIBYTE**

Read value’s high byte

Syntax HIBYTE(source, result)

HIBYTE according to that read value’s high byte of source which is storage in result. Source can be constants or variables, result must be variables.

For example:

```
macro_command main()

short source, result

HIBYTE(0x1234, result)//    result == 0x12

GetData(source, "Local HMI", LW, 140, 1)

HIBYTE(source, result)
//    source == 0x1234 => result == 0x12

SetData(result, "Local HMI", LW, 140, 1)

end macro_command
```

■ **LOWORD**

Read value’s low word

Syntax LOWORD(source, result)

LOWORD according to that read value’s low word of source which is storage in result. Source can be constants or variables, result must be variables.

For example:

```
macro_command main()

int source, result
```

```
LOWORD(0x12345678, result)//    result == 0x5678

GetData(source, "Local HMI", LW, 140, 1)

LOWORD (source, result)
//    source == 0x12345678 => result == 0x5678

SetData(result, "Local HMI", LW, 140, 1)

end macro_command
```

■ **HIWORD**

Read value's high word

Syntax   HIWORD(source, result)

HIWORD according to that read value's high word of source which is storage in result. Source can be constants or variables, result must be variables.

For example:

```
macro_command main()

int source, result

HIWORD(0x12345678, result)//    result == 0x1234

GetData(source, "Local HMI", LW, 140, 1)

HIWORD (source, result)
//    source == 0x12345678 => result == 0x1234

SetData(result, "Local HMI", LW, 140, 1)

end macro_command
```

(4) Bit manipulation

■ **GETBIT**

Read value that appointed bit position state

Syntax GETBIT(source, result, bit_pos)

GETBIT according to that bit_pos to appoint bit position state of source which is storage in result, result's value is 0 or 1. Source of bit_pos can be constants or variables, result must be variables.

For example:

```
macro_command main()

int source, result
short bit_pos

GetData(source, "Local HMI", LW, 180, 1)
GetData(bit_pos, "Local HMI", LW, 182, 1)


GETBIT(source, result, bit_pos)
//    source == 4, bit_pos == 0 => result = 0
//    source == 4, bit_pos == 1 => result = 0
//    source == 4, bit_pos == 2 => result = 1


SetData(result, "Local HMI", LW, 183, 1)

end macro_command
```

■ **SETBITON**

Set bit position state to 1

Syntax SETBITON(source, result, bit_pos)

SETBITON according to content of source can change the bit_pos state to 1, then storage in result. Source of bit_pos can be constants or variables, result must be variables.

For example:

```
macro_command main()

int source, result
short bit_pos
```

```
GetData(source, "Local HMI", LW, 180, 1)
GetData(bit_pos, "Local HMI", LW, 182, 1)

SETBITON(source, result, bit_pos)
//    source == 4, bit_pos = 1 => result == 6


SetData(result, "Local HMI", LW, 180, 1)

end macro_command
```

■ **SETBITOFF**

Set bit position state to 0

Syntax   SETBITOFF(source, result, bit_pos)

SETBITOFF according to content of source can change the bit_pos state to 0, then storage in result. Source of bit_pos can be constants or variables, result must be variables.

For example:

```
macro_command main()

int source, result
short bit_pos

GetData(source, "Local HMI", LW, 180, 1)
GetData(bit_pos, "Local HMI", LW, 182, 1)

SETBITOFF(source, result, bit_pos)
//    source == 6, bit_pos = 1 => result == 4


SetData(result, "Local HMI", LW, 180, 1)

  end macro_command
```

■ **INVBIT**

Set bit position state to inverse

Syntax INVBIT(source, result, bit_pos)

INVBIT according to content of source can change the bit_pos state to inverse, then storage in result. Source of bit_pos can be constants or variables, result must be variables.

For example:

```
macro_command main()

int source, result
short bit_pos

GetData(source, "Local HMI", LW, 180, 1)
GetData(bit_pos, "Local HMI", LW, 182, 1)

INVBIT(source, result, bit_pos)
//    source == 6, bit_pos = 1 => result == 4
//    source == 4, bit_pos = 1 => result == 6

SetData(result, "Local HMI", LW, 180, 1)

  end macro_command
```

(5) Communication

■ **DELAY**

Delay setting time, then to execute the order.

Syntax DELAY(time)

When use DELAY function, HMI will delay setting time, then to execute MACRO, the unit of time is ms and can be constants or variables.

For example:

```
macro_command main()

char s[100]
int a = 5000//    ms
s[0] = 'a'
s[1] = 'b'
s[2] = 'c'
```

```
s[2] = 'd'

DELAY(a)
DELAY(4000)//    delay 4000ms

SetData(s[0], "Local HMI", LW, 0, 3)

end macro_command
```

■ **ADDSUM**

Use addition to figure out checksum

Syntax	ADDSUM(source[start], result, no)
	ADDSUM according to add up variables of source[start] to source[start + (no – 1)], then storage in result. Source and result must be variables, no is sum of variables and can be constants or variables.

For example:

```
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)

end macro_command
```

■ **XORSUM**

Use XOR to figure out checksum

Syntax	XORSUM(source[start], result, data_count)
	XORSUM according to value of source[start] to source[start + (data_count – 1)], use XOR to figure out checksum, then storage in

result. Source and result must be variables, data_count is sum of variables and can be constants or variables.

For example:

```
macro_command main()
char data[5]
short checksum

data[0] = 0x1
data[1] = 0x2
data[2] = 0x3
data[3] = 0x4
data[4] = 0x5

XORSUM(data[0], checksum, 5)

end macro_command
```

- **CRC**

16 bit CRC operation

Syntax	CRC(source[start], result, data_count)
	CRC according to value of source[start] to source[start + (data_count – 1)] to figure out 16 bit CRC, then storage in result. Source and result must be variables, data_count is sum of count and can be constants or variables.

For example:

```
macro_command main()
char data[5]
short 16bit_CRC

data[0] = 0x1
data[1] = 0x2
data[2] = 0x3
data[3] = 0x4
data[4] = 0x5
```

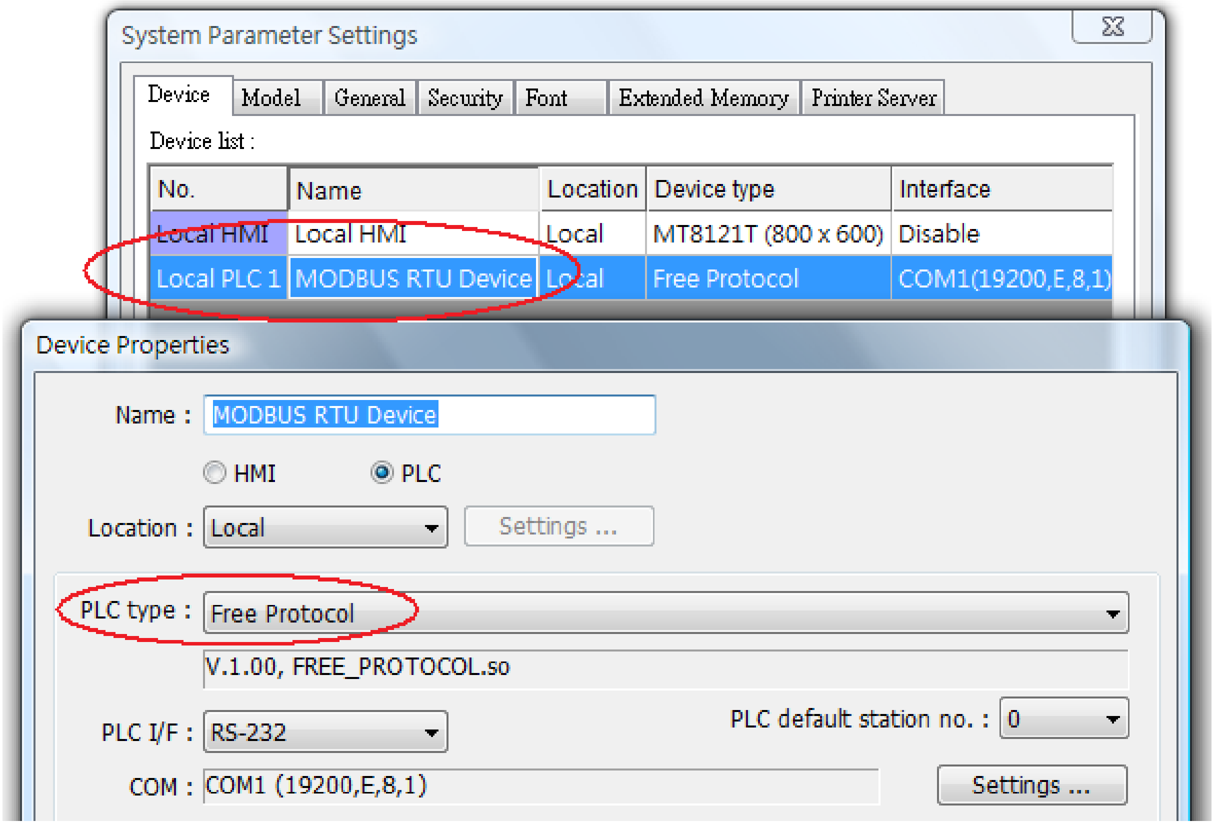
```
CRC(data[0], 16bit_CRC, 5)

end macro_command
```

■ **OUTPORT**

From communication port (COM port or Ethernet) to output data

Syntax OUTPORT(source[start], device_name, data_count)

OUTPORT according to communication port defined by device_name, source[start] to source[start + (data_count – 1)] of value sequence output, source must be variables and data_count must be constants. device_name must be Free Protocol, as follow:



No.	Name	Location	Device type	Interface
Local HMI	Local HMI	Local	MT8121T (800 x 600)	Disable
Local PLC 1	MODBUS RTU Device	Local	Free Protocol	COM1(19200,E,8,1)

Name : MODBUS RTU Device

○ HMI [✓ ● PLC]

Location : Local

PLC type : Free Protocol

V.1.00, FREE_PROTOCOL.so

PLC I/F : RS-232

PLC default station no. : 0

COM : COM1 (19200,E,8,1)

For example:

If ”MODBUS RTU Device” has be defined in device table, and set COM 1 for communication port, as follow example will show that how to use OUTPORT function to setting the MODBUS RTU device state.

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

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

FILL(command[0], 0, 32)//    init
FILL(response[0], 0, 32)

command[0] = 0x1//    station no
command[1] = 0x5//    write signle coil

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

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

CRC(command[0], checksum, 6)

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

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

//    return_value == 0 -> error
SetData(return_value, "Local HMI", LW, 0, 1)
SetData(response[0], "Local HMI", LW, 10, 8)//    send response to LW

end macro_command
```

■ INPORT

From communication port (COM port or Ethernet) to read data

Syntax INPORT(source[start], device_name, read_count, return_value)

INPORT according to communication port defined by device_name to read data, then storage in source[].

INPORT requests quantity of read data is connect with variables type of source and read_count, for example

```
char data[10]
INPORT(data[0], “device”, 10, return_value)
```

Foe variable type of data is char, the length of char is 1 byte, INPORT() read 10 * 1= 10 byte data.

If variables type of data is short, as follow：

```
short data[10]
INPORT(data[0], “device”, 10, return_value)
```

For the length of short is 2 bytes, INPORT() read 10 * 2= 30 byte data.

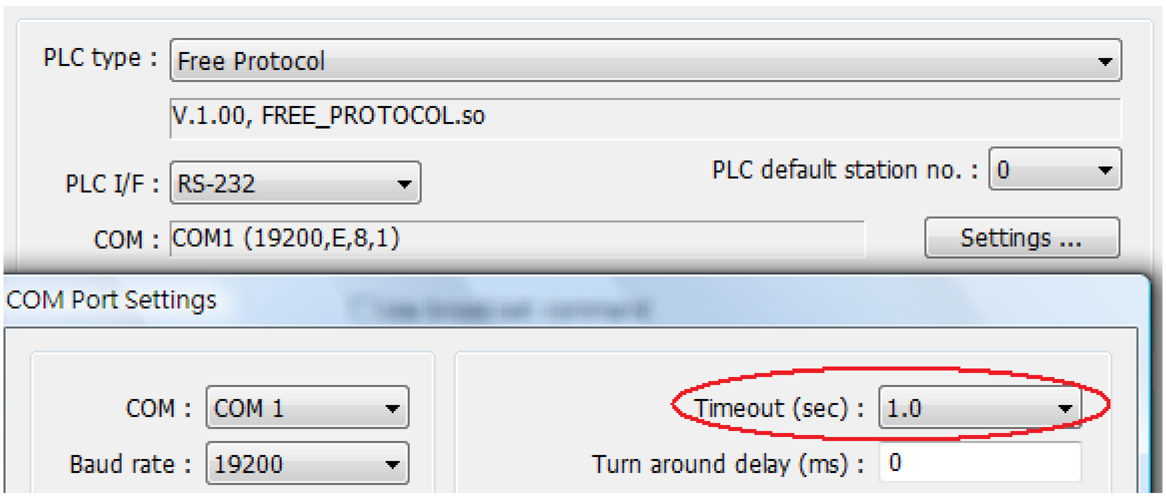
device_name must be Free Protocol.

When to finish INPORT() function, quantity of read data storage in return_value, unit is byte.

[Execute fail]

If return_value is 0, HMI unable to read data that defined by read_count after timeout.

Timeout of INPORT() function must defined by device table, as follow:


PLC type : Free Protocol
V.1.00, FREE_PROTOCOL.so
PLC I/F : RS-232
PLC default station no. : 0
COM : COM1 (19200,E,8,1)
Settings ...

COM Port Settings
COM : COM 1
Baud rate : 19200
Timeout (sec) : 1.0
Turn around delay (ms) : 0

For example:

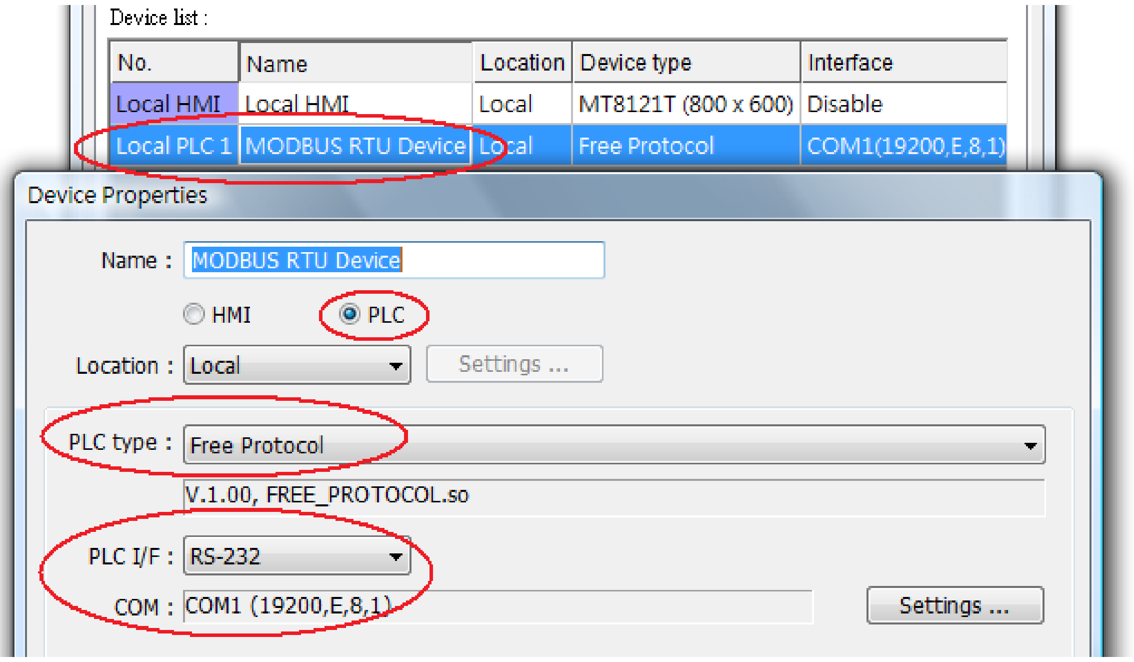
If ”MODBUS RTU Device” has be defined in device table, and set COM 1 for communication port, as follow will show that how to use OUTPORT function to read the MODBUS RTU device state.

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

command[0] = 0x1//    station no
command[1] = 0x3//    read holding registers

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

read_no = 2//    read 4x1_1, 4x1_2
HIBYTE(read_no, command[4])
LOBYTE(read_no, command[5])
```

```
CRC(command[0], checksum, 6)

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

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

//    return_value == 0 -> error
SetData(return_value, "Local HMI", LW, 0, 1)
SetData(response[0], "Local HMI", LW, 10, 9)//    send response to LW

if return_value > 0 then
    read_data[0] = response[4] + (response[3] << 8)//    4x1_1
    read_data[1] = response[6] + (response[5] << 8)//    4x1_2

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

    end macro_command
```

- Use macro command to control external device

When HMI is not support user's device, user can use macro command "OUTPORT" and "INPORT" to control those devices.
First, user has to new a device "Free Protocol" in system parameter, for example, this device is using COM1, and communicating parameter is 19200, E, 8, 1 and is named "MODBUS RTU Device" as below illustration.



Device list :

No.	Name	Location	Device type	Interface
Local HMI	Local HMI	Local	MT8121T (800 x 600)	Disable
Local PLC 1	MODBUS RTU Device	Local	Free Protocol	COM1(19200,E,8,1)

Device Properties

Name : MODBUS RTU Device

HMI [✓ PLC]

Location : Local

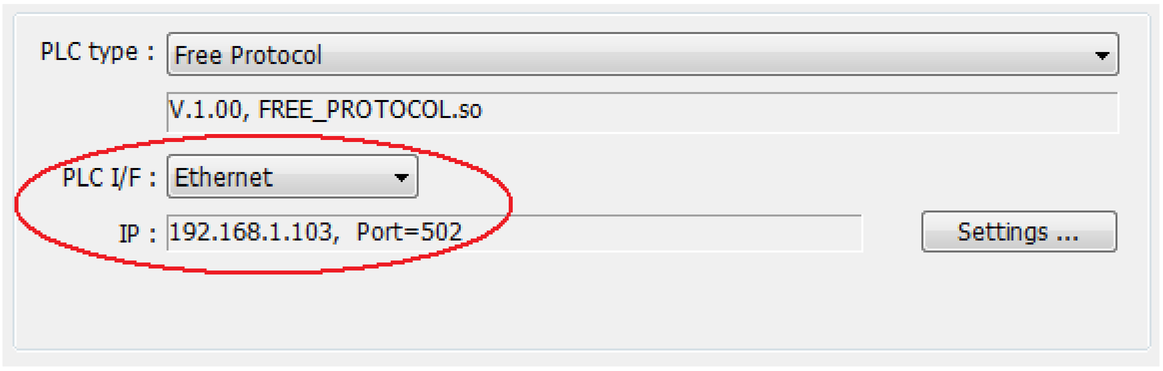
PLC type : Free Protocol

V.1.00, FREE_PROTOCOL.so

PLC I/F : RS-232

COM : COM1 (19200,E,8,1)

The device can be used Ethernet interface, for example, to use MODBUS TCP/IP, at this time, the PLC I/F is Ethernet and setting IP address and port no. as below illustration.



PLC type : Free Protocol

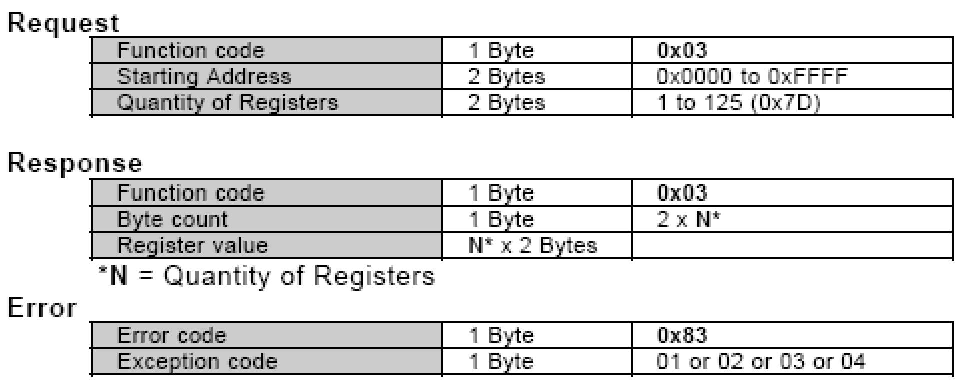
V.1.00, FREE_PROTOCOL.so

PLC I/F : Ethernet

IP : 192.168.1.103, Port=502

To read 4x_1, 4x_2 value, first user has to use OUTPORT command to ask read command from device, the syntax is
OUTPORT(command[start], device_name, cmd_count)

If this device is used MODBUS RTU protocol, user has to refer content of MODBUS RTU protocol for writing the command in the macro.
We use ”Reading Holding Registers (0x03)” command in MODBUS RTU protocol to read value of 4x_1, 4x_2.
Below illustration is part of the content of protocol (No show the station no.- byte 0 and CRC- latest two bytes here).



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

According to the protocol, the content of command as following, total is 8 bytes：

command[0] : Station no.	(BYTE 0)
command[1] : Command	(BYTE 1)
command[2] : Address high byte	(BYTE 2)
command[3] : Address low byte	(BYTE 3)
command[4] : Read value high byte	(BYTE 4)
command[5] : Read value low byte	(BYTE 5)
command[6] : 16-bit CRC low byte	(BYTE 6)
command[7] : 16-bit CRC high byte	(BYTE 7)

The content of Macro as following,

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

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

command[0] = 0x1//    station no
command[1] = 0x3//    read holding registers
```

```
address = 0//    start from 4x_1 and read two words, the initial address of 4x_1 is 0
HIBYTE(address, command[2])
LOBYTE(address, command[3])

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

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

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

Use OUPORT to send the command

```
OUTPORT(command[0], "MODBUS RTU Device", 8)//    send command
```

Use INPORT command to read the response of device. According to the content of protocol, total 9 bytes.

command[0] : Station no.	(BYTE 0)
command[1] : Command	(BYTE 1)
command[2] : Read byte number of value	(BYTE 2)
command[3] : 4x_1 high byte	(BYTE 3)
command[4] : 4x_1 low byte	(BYTE 4)
command[5] : 4x_2 high byte	(BYTE 5)
command[6] : 4x_2 low byte	(BYTE 6)
command[7] : 16-bit CRC high byte	(BYTE 7)
command[8] : 16-bit CRC low byte	(BYTE 8)

The content of INPORT command as following,

```
INPORT(response[0], "MODBUS RTU Device", 9, return_value)//    read response
```

return_value is for record the byte number of INPORT, if return_value is 0, that means false to read.

According to the content of protocol, response[1] is equal to 0x3, that means the correct response of device. If response is correctly, after got the value of 4x_1 and 4x_2, the result will set and display on LW100 and LW101.

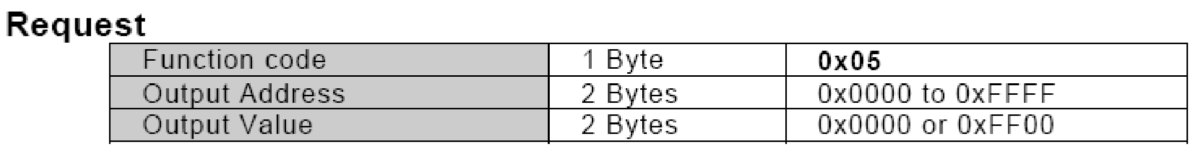
```
if (return_value >0 and response[1] == 0x3) then
    read_data[0] = response[4] + (response[3] << 8)//    4x_1
    read_data[1] = response[6] + (response[5] << 8)//    4x_2

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

The whole procedure as following,

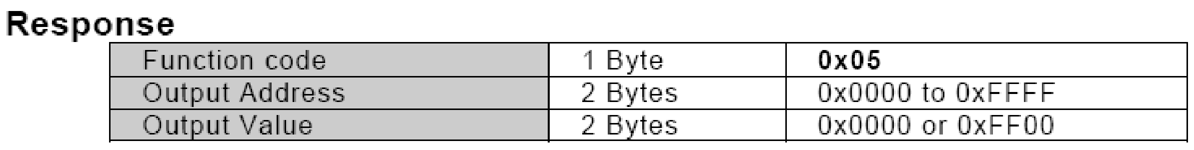
```
//    Read Holding Registers
macro_command main()

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

FILL(command[0], 0, 32)//    init
FILL(response[0], 0, 32)

command[0] = 0x1//    station no
command[1] = 0x3//    read holding registers

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

read_no = 2//    read 4x_1, 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])
```

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

SetData(return_value, "Local HMI", LW, 0, 1)//    return_value == 0 -> error
SetData(response[0], "Local HMI", LW, 10, 9)//    send response to LW

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
```

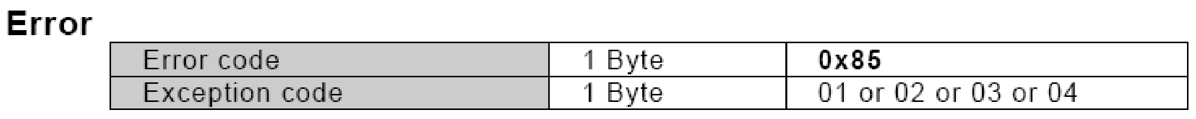
Below example is to describe how to set ON or OFF in 0x_1, the illustration is a part of content of protocol (No show the station no.- byte 0 and CRC- latest two bytes here). This is use command of ”Write Single Coil(0x5)” :


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

First, to use OUTPORT command to send a request to the device, according to the command of protocol, the content of command[] as following,

```
char command[32]

FILL(command[0], 0, 32)//    init
```

```
command[0] = 0x1//    station no
command[1] = 0x5//    write signle coil

address = 0//    Set address 0, first bit (0x_1)
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 command
```

After sending command, use INPORT to read the result of setting, according to the content of protocol, total has to read 8 bytes.

```
INPORT(response[0], "MODBUS RTU Device", 8, return_value)//    read response
```

The whole procedure as following,

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

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

FILL(command[0], 0, 32)//    init
FILL(response[0], 0, 32)
```

```
command[0] = 0x1//    station no
command[1] = 0x5//    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 command
INPORT(response[0], "MODBUS RTU Device", 8, return_value)//    read response

SetData(return_value, "Local HMI", LW, 0, 1)//    return_value == 0 -> error
SetData(response[0], "Local HMI", LW, 10, 8)//    send response to LW

end macro_command
```

● Macro command and PLC communication (LocalBit, LocalWord):

Usage：Communicate with PLC through a function library
In the command program, Macro can communicate with data in the PLC. The function GetData(…) can receive data from the PLC through EasyView. The function SetData(…) can set data to the PLC through EasyView. The Macro command handles the communication details.

```
1. GetData(  Supported data types: DestData,
             char*    szPLCName，
             char*    szDeviceType，
             int      nAddress,
             int      nDataCount)
```

Description

Get data from PLC

Parameters：

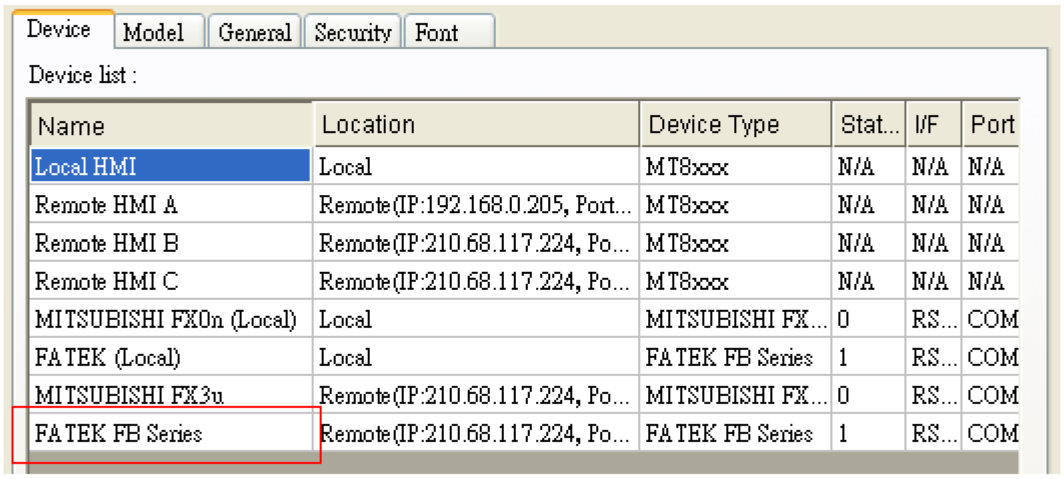
DestData	The address of data to get
szPLCName	PLC name
szDeviceType	PLCtype and encoding method of PLC address
nAddress	The address of PLC
nDataCount	Number of data

Return value:

None

szPLCName

Set the PLC operation object, identify the plc name by use the double quotation marks, those name have been defined in the Device List of System rameter.Example:”FATEK FB Series”，See the following graph, If use the name never been defined in the Device List,will will cause compiler error.
The name of HMI is fixed as “Local HMI”.



Name	Location	Device Type	Stat...	I/F	Port
Local HMI	Local	MT8xxx	N/A	N/A	N/A
Remote HMI A	Remote(IP:192.168.0.205, Port...	MT8xxx	N/A	N/A	N/A
Remote HMI B	Remote(IP:210.68.117.224, Po...	MT8xxx	N/A	N/A	N/A
Remote HMI C	Remote(IP:210.68.117.224, Po...	MT8xxx	N/A	N/A	N/A
MITSUBISHI FX0n (Local)	Local	MITSUBISHI FX...	0	RS...	COM
FATEK (Local)	Local	FATEK FB Series	1	RS...	COM
MITSUBISHI FX3u	Remote(IP:210.68.117.224, Po...	MITSUBISHI FX...	0	RS...	COM
FATEK FB Series	Remote(IP:210.68.117.224, Po...	FATEK FB Series	1	RS...	COM

strDeviceType Format

AAA_BBB

AAA is the register name in the PLC. Example: LB or LW, BBB means the format data (BIN or BCD).

For example: if strDeviceType is LB_BIN，It means the register is LB and the format is BIN.

If use BIN format, "_BIN" can be ignored，Example: LW_BIN is equal to LW，They both mean the register is LW and the format is BIN.

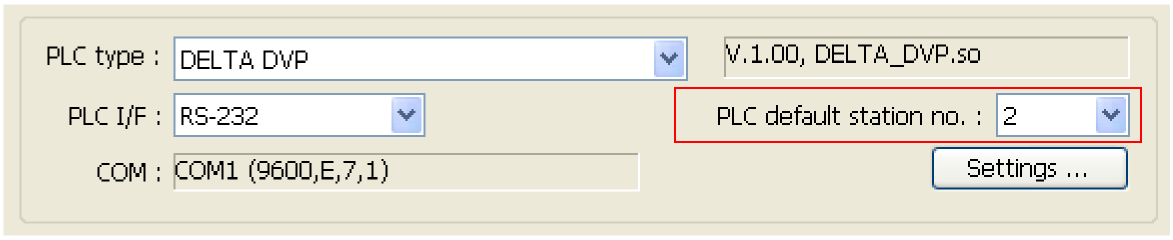
NAddress format

N#AAAAA

N means the station number of PLC，range is from 0 to 255. If use the default station number in system parameter, 'N#' can be canceled,；AAAAA is the address of PLC register.

For example: if strAddress is 2#10，It means the station number of plc is 2,the address of plc register is 10. So the function GetData(a, "DELTA DVP", M, 2#10, 1) means that read the data in the address M10 of "DELTA DVP" No.2 PLC.

If strAddress is，'N#' is canceled，Now It will use the default station number of system parameter. See the following graph，now the default number is 2。


PLC type : DELTA DVP

V.1.00, DELTA_DVP.so

PLC I/F : RS-232

PLC default station no. : 2

COM : COM1 (9600,E,7,1)

For example:

bool a
bool b[30]
short c
short d[50]
int e
int f[10]
double g[10]

// read the state of LB2，and save in variable a
GetData(a, “Local HMI”, LB, 2, 1)

// Read the total 30 states of LB0~LB29, and save in variables b[0]~b[29]
GetData(b[0], “Local HMI”, LB, 0, 30)

// Read one word data from LW2，and save in variable c
GetData(c, “Local HMI”, LW, 2, 1)

// Read total 50 word datas from LW0~LW49, and save in variables d[0]~d[49].
GetData(d[0], “Local HMI”, LW, 0, 50)

// Read one double word from LW6 ,and save in variable e
// note：the type of e is int
GetData(e, “Local HMI”, LW, 6, 1)

// read total 20 word data, and save in variables f[0]~f[9]
// note：the type of f[10] is int
// f[0] save the data of LW0~LW1，f[1] save the data of LW2~LW3，the rest may be deduced by analogy, GetData (f[0], “Local HMI”, LW, 0, 10)

// read one float data from LW2, size is double word，and save in the variable f

```
GetData(f, “Local HMI”, LW, 2, 1)
```

2. SetData (Supported data types: DestData，
char* szPLCName，
char* szDeviceType，
int nAddress,
int nDataCount)

Description

Send data to PLC, data can be inputted by filling a dialog.

Parameters：

DestData	The address of data to set
szPLCName	PLC Name
szDeviceType	PLCtype and encoding method of PLC address
nAddress	The address of PLC
nDataCount	Number of data

Return value

None

For example:

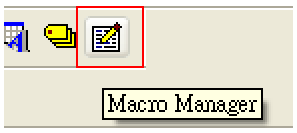
```
int i
bool a = True
bool b[30]
short c = False
short d[50]
int e = 5
int f[10]

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

//     set the state of LB2
SetData(a, “Local HMI”, LB, 2, 1)

//   set states of LB0~ LB29
SetData(b[0], “Local HMI”, LB, 0, 30)

//   set the data of LW2
SetData(c, “Local HMI”, LW, 2, 1)

//   set datas of LW0~LW49
SetData(d[0], “Local HMI”, LW, 0, 50)

//   set the data of LW6~LW7
//   note：the type of e is int
SetData(e, “Local HMI”, LW, 6, 1)

//   set datas of LW0~LW19
SetData(f[0], “Local HMI”, LW, 0, 10)
```

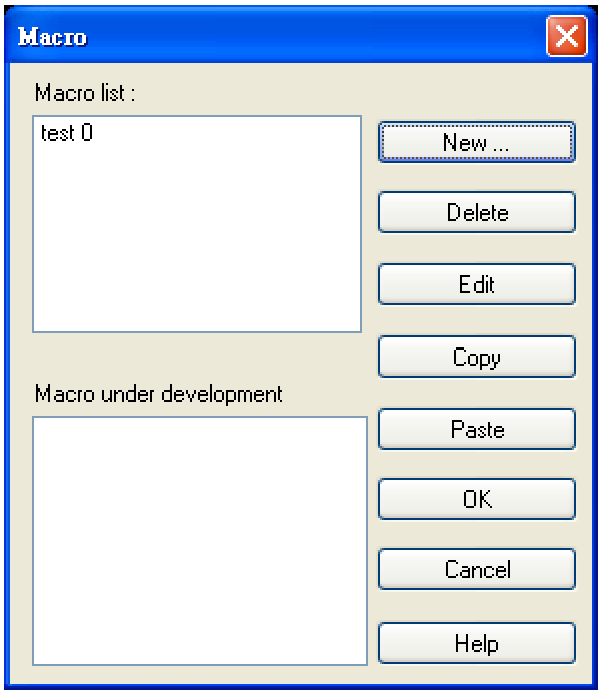
- Macro operation manual

1. Macro programming can be divided into three steps:

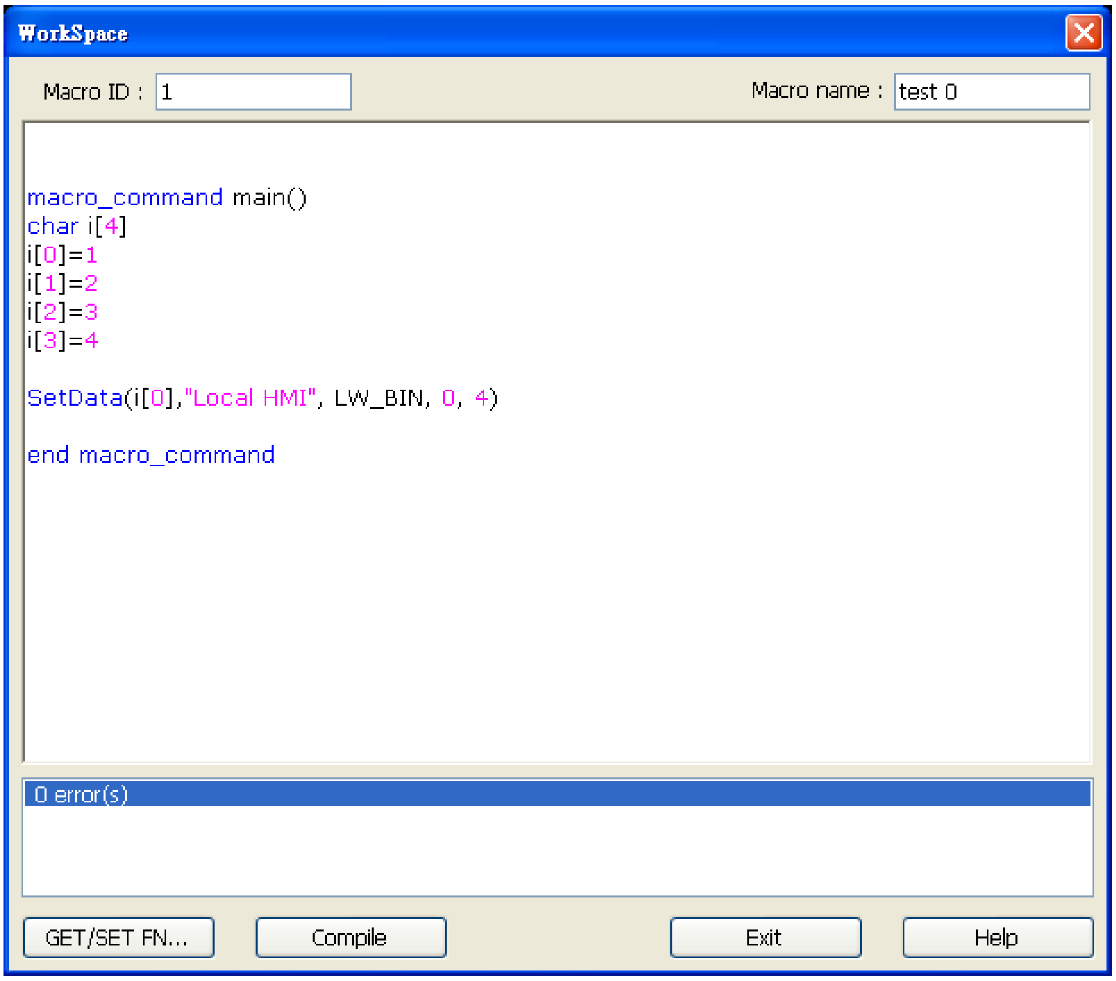
Step 1：click the first icon in the Macro tool box of EasyBuilder 8000



Step 2： Each Macro can be copied, deleted or edited in MacroControlDlg dialog. The source code of Macro can be edited by opening MacroWorkSpaceDlg dialog.



Step 3: Editing the source code of the Macro. Make sure the name and number of the Macro are correct. Compile the Macro and fix the error message.



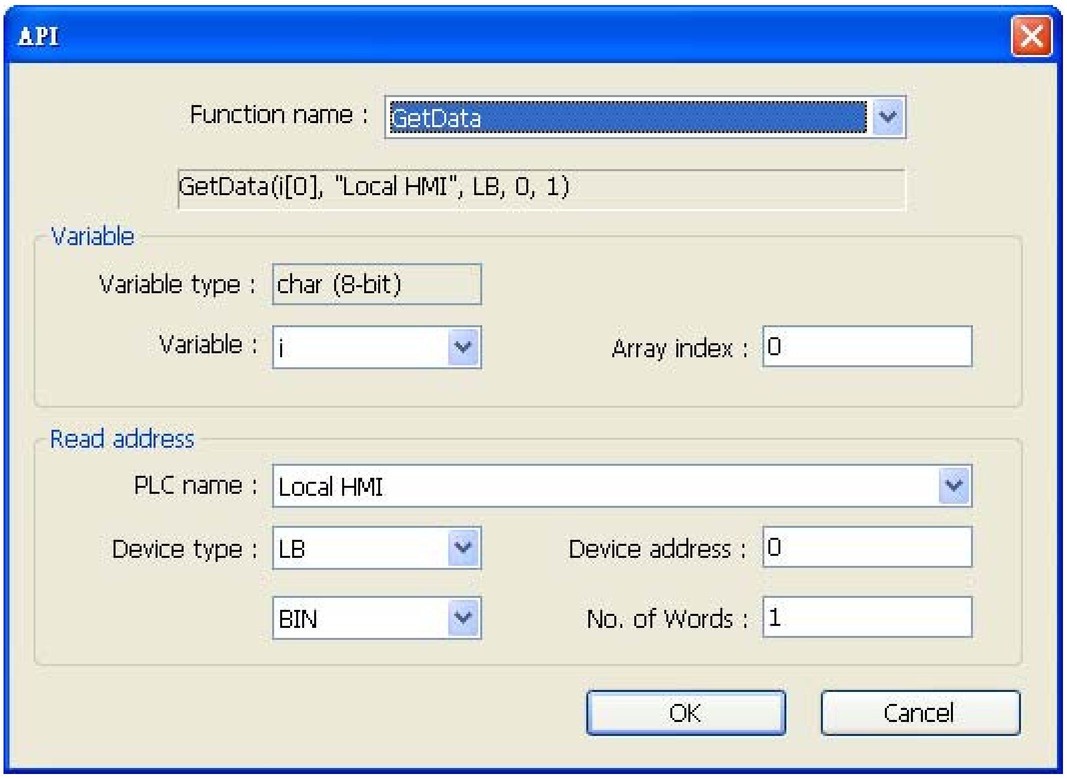
```
macro_command main()
char i[4]
i[0]=1
i[1]=2
i[2]=3
i[3]=4

SetData(i[0],"Local HMI", LW_BIN, 0, 4)

end macro_command
```

```
0 error(s)
```

2. Editing the communication source code of Macro:

a. Input

Step 1：Enter the keyword “Insert” in the proper position. Or by moving the cursor to the proper position and push [PLC API] button.It will appear a dialogue as follows.



Function name : GetData

GetData(i[0], "Local HMI", LB, 0, 1)

Variable

Variable type : char (8-bit)

Variable : i

Array index : 0

Read address

PLC name : Local HMI

Device type : LB

Device address : 0

BIN

No. of Words : 1

Step 2：Select functions and parameters of the library in Library Editing Dialog. Push button “OK” to enter this sub-function; push button “Cancel” to abort this sub-function.

b. Edit

Move the cursor onto the modifying position to modify it.

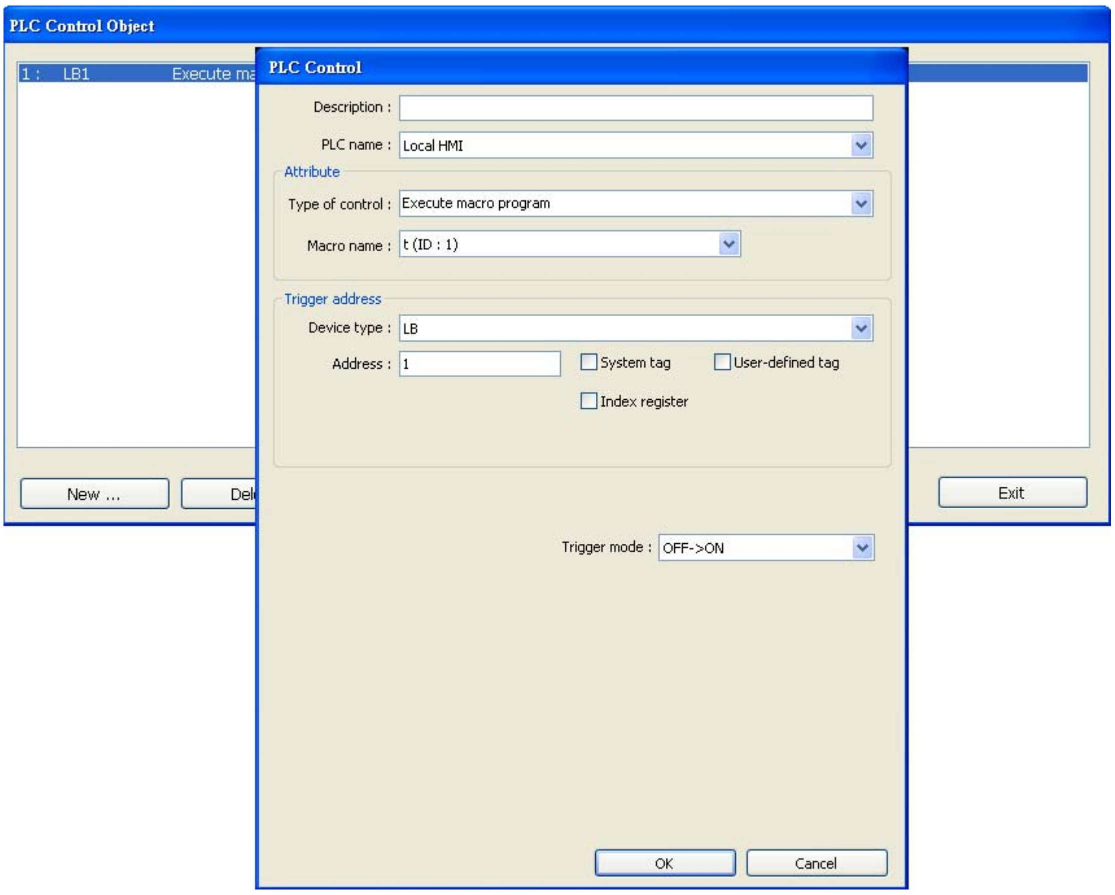
c. Delete

Highlight the selected function and push the button “Delete” to delete it.

3. Trigger condition of Macro

The objects of Set Bit, Toggle Switch, Function Key and PLC Control can be used to trig Macro, The following text show how to trigger macro by use PLC Control object.

Step 1： Select control type to “Execute Macro Program” in the object property dialog of PlcControl.



Description :
PLC name : Local HMI
Type of control : Execute macro program
Macro name : t (ID : 1)
Device type : LB
Address : 1
Trigger mode : OFF->ON

Step 2：Select a Macro name and define a trigger condition in the object property dialog PlcControl(Now it is LB1).

- Some notes about using Macro

1. Limitation of storage space of Macro

   The size of a Macro in a xob file is limited by the storage capacity. The maximum storage space of local variables in a Macro is 4K bytes. So the define range of different variable types are limited as following:

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

2. Limitation of maximum lines of Macro to execute

   There are at most 255 Macros in a xob file.

3. Macro operation may cause deadlock of the MT8000.

   When there is a infinite loop in a Macro without communicating with PLC

   When the size of array exceeds the storage space in a Macro.

4. The Limitation of communication speed of Macro

   The execution of Macro may be slow down when communicating with PLC.

   This is caused by the data transferring time. Avoid too many complicated action in the Macro.

- Compiler error message

1. Error message format:

error C# : error description
# is the error number.

Example: error C37 : undeclared identifier : i

When there are compile errors, the error description can be referenced by the compile error message number.

2：Error description

(C1) syntax error：’identifier’

There are many possibilities to cause compiler error.

For example:

```
macro_command main()
char i, 123xyz    //this is an unsupported variable name ,”Error message: “Syntax
error: 123xyz”
end macro_command
```

(C2) ‘identifier’ used without having been initialized

Macro just support static array, must define the size of an array during declaration.

For example：

```
macro_command main()
char i
int g[i]        //    i used without having been initialized
end macro_command
```

(C3) redefinition error : ‘identifier’

The name of variable and function within its scope must be unique.

For example：

```
macro_command main()
int g[10]，g      //error
end macro_command
```

(C4) function name error : ‘identifier’
   reserved keywords and constant can not be the name of a function

For example：

```
sub int if()      // error
```

(C5) parentheses have not come in pairs
   Statement missing “(“ or “)”

For example：

```
macro_command main    )//    missing C
```

(C6) illegal expression without matching ‘if’
   Missing expression in If statement

(C7) illegal expression (no ‘then’) without matching ‘if’
   Missing “Then” in If statement

(C8) illegal expression (no ‘end if’)
   Missing “EndIf”

(C9) illegal ‘end if’ without matching ‘if’
Unfinished “If’ statement before “End If”

(C10) illegal ‘else’
  The format of “If” statement is:
  If [logic expression] Then

[Else [If [logic expression] Then]]

EndIf

Any format other than this format will cause compile error.

(C11) ‘case’ expression not constant
There should be constant behind “Case”

(C12) ‘select’ statement contains no ‘case’
Missing “Case” behind “Select”

(C13) illegal expression without matching ‘select case’
Missing “expression” behind “Select Case”

(C14) ‘select’ statement contains no ‘end select’
"Missing “End Select” statement

(C15) illegal ‘case’
Illegal “Case” statement"

(C16) illegal expression (no 'select') without matching ‘end select’
The format of “Select Case” statement is:
Select Case [expression]
Case [constant]

Case [constant]

Case [constant]

Case Else

End Select

Any format other than this format will cause compile error.

(C17) illegal expression (no 'for') without matching ‘next’
“For” statement error: missing “For” before “Next”

(C18) illegal variable type (not interger or char)
Should be integer of char variable

(C19) variable type error
Missing assign statement

(C20) must be key word ‘to’ or ‘down’
Missing keyword “to” or “down”
(C21) illegal expression (no 'next')

The format of “For” statement is:
For [variable] = [initial value] To [end value] [Step]

Next [variable]
Any format other than this format will cause compile error.

(C22) ‘wend’ statement contains no ‘while’
“While” statement error: missing “While” before “Wend”

(C23) illegal expression without matching ‘wend’

The format of “While” statement is:
While [logic expression]

Wend
Any format other than this format will cause compile error.

(C24) syntax error : ‘break’
“Break” statement can only be used in “For”, “While”, or “Select Case” statement
“Break” statement takes one line of Macro.

(C25) syntax error : ‘continue’
“Continue” statement can only be used in “For” statement, or “While” statement
“Continue” statement takes one line of Macro.

(C26) syntax error
expression is error.
(C27) syntax error
The mismatch of operation object in expression cause compile error.

For example：

```
macro_command main( )
int a, b

for a = 0 to 2
        b = 4 + xyz //illegal operation object
next a
end macro_command
```

(C28) must be ‘macro_command’
There must be ’macro_command’

(C29) must be key word ‘Sub’

The format of function declaration is:

```
sub [data type] function_name(…)
………..
end sub
```

For example::

```
sub int pow(int exp)
…….
end sub
```

Any format other than this format will cause compile error.

(C30) number of parameters is incorrect
Mismatch of the number of parameters

(C31) parameter type is incorrect

Mismatch of data type of parameter

(C32) variable is incorrect
The parameters of a function must be equivalent to the arguments passing to a function to avoid compile error.

(C33) function name : undeclared function
Undefined function

(C34) expected constant expression
Illegal member of array

(C35) invalid array declaration
Illegal definition of array

(C36) array index error
Illegal index of array

(C37) undeclared identifier : i ‘identifier’
Any variable or function should be declared before use.

(C38) PLC encoding method is not supported
The parameter of GetData(…) , SetData(…) should be legal PLC address.

(C39) ‘idenifier’ must be integer, char or constant
The format of array is:
Declaration: array_name[constant] (constant is the size of the array)
Usage: array_name[integer, character or constant]
Any format other than this format will cause compile error.

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

For example :

```
Macro_Command main(  )
int a, b
For a = 0 To 2
```

```
    b = 4 + a
int h , k
//declaration statement position error//
Next a
End Macro_Command
```

(C41) float variables cannot be contained in shift calculation
Floating point can not bitwise shift

(C42) function must return a value
Missing function return value

(C43) function should not return a value
Function can not return a value

(C44) float variables cannot be contained in calculation
Illegal Float data type in expression

(C45) PLC address error
Error PLC address

(C46) array size overflow (max. 4k)
Stack can not exceed 4k bytes

(C47) macro command entry function is not only one
Only one main entrance in the Macro is allowed

(C48) macro command entry function must be only one

The only one main entrance of Macro is:

```
Macro_Command function_name( )

End Macro_Command
```

(C49) a extended addresse's station no. must be between 0 and 255

For example:

SetData(bits[0] , “PLC 1”, LB , 300#123, 100)

300#123 中的 300 means the station no is 300，but the maximum is 255

(C50) a invalid PLC name

PLC name is not included in the Device List of system paramter

For example:

SetData(bits[0] , “PLC 1”, LB , 300#123, 100)

There is no “PLC 1” in Device List.

(C51) macro command do not control a remote device

Macro just can control local machine

For example

SetData(bits[0] , “PLC 1”, LB , 300#123, 100)

“PLC 1“ is connected with the remote device ,so it is can not work.

- Example source code

1:”For” statement and other expressions (arithmetic, bitwise shift, logic and comparison)

```
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”, 3x, 0,9)
        GetData(b[0],”Device 1”, 3x, 11,10)
end If
next i
end macro_command
```

2: while, if, break

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

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

```
wend
end macro_command
```

3: Global variables and function call

```
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
```

4. ”If” statement

```
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”, 3x, 0, 1)
end if
```

```
if k[0] == 0 then
        SetData(k[1], “Device 1”, 3x, 0, 1)
else
                SetData(k[2], “Device 1”, 3x, 0, 1)
end if

if k[0] == 0 then
        SetData(k[1], “Device 1”, 3x, 1, 1)
else if k[2] == 1 then
        SetData(k[3], “Device 1”, 3x, 2, 1)
end If

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

5. while statement

```
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”, 3x, 2, 1)
end macro_command
```

6. break、continue statement

```
    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”, 3x, 2, 1)
        break
        end if
    wend
    end macro_command
```

7. array statement

```
    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”, 3x, 0, 13)
end macro_command
```