User Manual

Instruction Application section

SUZHOU SHARE WELL AUTOMATION TECHNOLOGY CO.,LTD

Catalogue

PowerNexOS Command Overview

General Statements

Call	Call a function
Do..Loop	Loop statement that repeatedly executes between Do..Loop based on a condition
Else/ElseIf	Used with conditional statements; executes code between Else...EndIf if If is not satisfied
Exit	Forcefully exit a loop or function
For..To..Next	Repeatedly executes the code between For...Next a specified number of times
Function..Fend	Function definition
Global	Declare a global variable
GoSub..Return	Jump to a subroutine
GoTo	Jump to a labeled point
If..Then..Else..EndIf	Conditional statement; executes code between Then...EndIf if If is satisfied
Select..Case..Default..Send	Execute corresponding statements based on the Case results

Data Types

Boolean	Boolean type, True/False
Byte	Byte type, -128 to 127
Double	Double precision floating-point number
Int32	32-bit integer, -2147483648 to 2147483647
Integer	16-bit integer, -32768 to 32767
Long	Long integer, -2147483648 to 2147483647
Real	Single precision floating-point number
Short	Short integer, -32768 to 32767
String	String, up to 255 characters

UByte	Unsigned byte, 0 to 255
UInt32	Unsigned 32-bit integer, 0 to 4294967295
UShort	Unsigned short integer, 0 to 65535
Preserve	Define a global variable to retain memory during power loss; retains variable memory when stopping the program or rebooting the controller.

Motion Commands

!...!	Parallel processing of I/O input and output during motion
Motor	Enable or disable the entire system
ServoOff	Disable a single-axis servo
ServoOn	Enable a single-axis servo
Power	Set high or low power mode
SpeedFactor	Set/display global run speed, affecting Speed and SpeedS
Speed	Set/display PTP motion speed for commands such as Go, Jump, Pulse
Accel	Set/display acceleration and deceleration for PTP motion
Go	Perform PTP motion between the current position and a specified position
BGo	Execute offset PTP motion in the selected user coordinate system
TGo	Execute offset PTP motion in the selected tool coordinate system
SpeedS	Set/display CP motion speed for commands such as Move, Arc, Arc3, Jump3, Jump3CP
AccelS	Set/display acceleration and deceleration for linear interpolation motion
Move	Perform linear interpolation motion between the current position and a specified position
BMove	Execute offset linear interpolation motion in the selected user coordinate system
TMove	Execute offset linear interpolation motion in the selected tool coordinate system
Jump	PTP door-shaped motion
Arc	Perform circular interpolation motion in the XY plane
Arc3	Perform circular interpolation motion in three-dimensional space

CP	Set/display motion smoothing
CPZone	Set/display continuous motion smoothing parameters
Here	Teach/return the robot's current coordinates
Home	Return to the Home position
Local	Set/display the user coordinate system
TLSet	Set/display the tool coordinate system
Tool	Select/display the specified tool coordinate by number
Till	Determine conditions during motion and stop if conditions are met
Agl	Return the angle or position of a specified joint
AglToPls	Convert joint angles to pulses
Arch	Set/display the Arch parameter for Jump, Jump3, Jump3CP commands
Find	Set/display conditions for saving coordinates in action commands
FindPos	Return saved coordinates during the execution of action commands
InPos	Return whether the robot is in position
JA	Return robot coordinates based on specified joint angles
Joint	Display the robot's current coordinates in joint coordinate system
JS	Return whether the Sense input condition is satisfied
LimZ	Set/display the initial value of the third joint height (Z coordinate) for Jump commands
Pass	Perform PTP motion passing near a specified point without stopping
Pls	Return pulse values of each joint at the current position
PosFound	Return the execution status during the Find command
Pulse	Perform PTP motion to specified joint pulse positions or return points based on pulse values
RobLoad	Select/display the specified load by number
Sense	Set/display conditions for Jump, Jump3, Jump3CP to stop above the target coordinates
TillOn	Return the status of the Till command
WaitPos	Wait for the robot motion to stop
XY	Return position based on specified values
CX	Set/get the X coordinate of the position
CY	Set/get the Y coordinate of the position

CZ	Set/get the Z coordinate of the position
CU	Set/get the U coordinate of the position
CV	Set/get the V coordinate of the position
CW	Set/get the W coordinate of the position
CR	Set/get the R coordinate of the position
CS	Set/get the S coordinate of the position
CT	Set/get the T coordinate of the position
ColSup	Set/get the collision detection coefficient
ZeroPls	Set/get the robot's zero position
Hand	Set/get the current hand coordinate system
Elbow	Set/get the elbow posture
Wrist	Set/get the wrist posture
PDef	Determine if the point exists
PDel	Delete a point
PLabel	Set a label for point data
PLabel$	Get the point label
PNumber	Get the point number
PList	Get point data information
PLocal	Return the point in user coordinates
PDescription	Set a description for the point
PDescription$	Return the description of the point

I/O Commands

Wait	Delay command or wait for a condition to be met for a specified time
Mask	Bitwise AND operation on the value representing the Wait input condition, processed bit by bit
On	Set the DO to ON, or switch the DO ON for a specified time and then switch OFF
Off	Set the DO to OFF, or switch the DO ON for a specified time and then switch ON

Oport	Return the status of the specified DO
Sw	Return the status of the specified DI
SetSw	Set the specified DI to ON or OFF
In	Return the 8-bit DI status based on the 8421 code, 0 to 255
SetIn	Set the 8-bit DI status based on the 8421 code, 0 to 255
Out	Set/return the 8-bit DO status based on the 8421 code, 0 to 255
InW	Return the 16-bit DI status based on the 8421 code, 0 to 65535
SetInW	Set the 16-bit DI status based on the 8421 code, 0 to 65535
OutW	Set/return the 16-bit DO status based on the 8421 code, 0 to 65535
InBCD	Return the 8-bit DI status based on the 8421 code, 0 to 99
OpBCD	Set the 8-bit DO status based on the 8421 code, 0 to 99
InReal	Return the DI value as a 32-bit floating-point number
OutReal	Set/return the DI value as a 32-bit floating-point number
AIO_InW	Return the value of the analog input
AIO_OutW	Set/return the value of the analog output
MemOn	Set a bit in the user register to 1
MemOff	Set a bit in the user register to 0
MemSw	Return the value of a bit in the user register
MemIn	Return the value of the user register as 8 bits
MemOut	Set the value of the user register as 8 bits
MemInW	Return the value of the user register as 16 bits
MemOutW	Set the value of the user register as 16 bits
MemIn32	Return the value of the user register as 32 bits
MemOut32	Set the value of the user register as 32 bits
MemInReal	Return the value of the user register as a 32-bit floating-point number
MemOutReal	Set the value of the user register as a 32-bit floating-point number
SysMemOn	Set a bit in the system register to 1
SysMemOff	Set a bit in the system register to 0
SysMemSw	Set a bit in the system register to 1
SysMemIn	Return the value of the system register as 8 bits

SysMemOut	Set the value of the system register as 8 bits
SysMemInW	Return the value of the system register as 16 bits
SysMemOutW	Set the value of the system register as 16 bits
SysMemIn32	Return the value of the system register as 32 bits
SysMemOut32	Set the value of the system register as 32 bits
SysMemInReal	Return the value of the system register as a 32-bit floating-point number
SysMemOutReal	Set the value of the system register as a 32-bit floating-point number

Communication Commands

SetNet	Set TCP/IP port parameters
OpenNet	Open TCP/IP port
WaitNet	Wait for the TCP/IP port to establish a connection
ChkNet	Return the number of bytes in the network port's receive buffer
ClrNet	Clear the network port receive buffer
CloseNet	Close the TCP/IP port opened by OpenNet
SetCom	Set serial communication parameters
OpenCom	Open serial communication parameters
ChkCom	Return the number of bytes in the serial port's receive buffer
ClrCom	Clear the serial port receive buffer
CloseCom	Close the serial port opened by OpenCom
Read	Read a specified number of bytes from the communication port
ReadBin	Read binary data from the communication port
Write	Write a string to the communication port
WriteBin	Write binary data to the communication port
Print #	Output data to the specified communication port
Input #	Output data to the specified communication port
Line Input #	Read a line of data from the communication port

System Commands

Print	Print a string or data
Xqt	Start multithreading
Pause	Pause all threads
Halt	Pause a specified thread
Quit	Stop all or specified threads
Resume	Continue all or specified threads
Reset	Reset the controller to its initial state
MyTask	Return the current thread number
TaskDone	Confirm whether the thread has finished
TaskState	Return the status of a specified thread
TaskWait	Wait for a specified thread to finish
SyncLock	Use mutual exclusion locks to synchronize multiple threads
SyncUnLock	Unlock the signal locked by SyncLock
Signal	Send a signal to tasks executing the WaitSig command
WaitSig	Wait for a synchronization signal sent by the Signal command from other tasks
TW	Return the status of Wait, WaitNet, and WaitSig commands
ElapsedTime	Return the elapsed time in seconds since the timer started
ResetElapsedTime	Reset the ElapsedTime timer
Tmr	Return the elapsed time in seconds since the timer started
TmReset	Reset the Tmr timer
Eval	Execute statements in the command window

Palletizing Commands

Pallet	Define an array, obtain array positions, print the array
PalletClr	Clear the array

Conveyor Tracking Commands

SyStart	Start conveyor tracking function
SyEnd	End conveyor tracking function

SyReset	Clear current conveyor tracking queue data
SyGetPoint	Obtain position data for conveyor tracking
SyGetUserData	Get additional information about the current tracking target
SyGetPose	Get the current position of the tracking target on the conveyor
SyGetNum	Get the number of targets in the conveyor tracking queue
SyMove	Perform linear interpolation motion for conveyor tracking
SyArc	Perform circular interpolation motion for conveyor tracking
SyTrig	Record the specified conveyor encoder pulse count
SyPoint	Generate tracking points based on current conveyor tracking
SyAddVisTarget	Register visual coordinates into the conveyor tracking queue
SyRejectDis	Set/get the filtering distance for the conveyor tracking feature
SySave	Save modifications to the conveyor tracking feature within the program

Mathematical Operation Commands

+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation
Mod	Integer modulus
And	Bitwise AND
Or	Bitwise OR
Xor	Bitwise XOR
Not	Bitwise NOT
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to
=	Equal to, assignment
DegToRad	Convert degrees to radians
RadToDeg	Convert radians to degrees

Sin	Sine function
Cos	Cosine function
Tan	Tangent function
Asin	Arc sine function
Acos	Arc cosine function
Atan	Arc tangent function
Atan2	Arc tangent function (two arguments)
HexToFloat	Convert hexadecimal floating-point to single precision float
FloatToHex	Convert single precision float to hexadecimal floating-point
Abs	Absolute value
Sqr	Square root
Sgn	Return the sign of a value
LShift	Left shift
RShift	Right shift
BClr	Set the specified bit in the value to 0 and return that value
BSet	Set the specified bit in the value to 1 and return that value
BTst	Return the value of the specified bit in the number
Ubound	Get the length of the array

String Operation Commands

ParseStr	String parsing
+	String concatenation
<>	Not equal to
=	Equal to, assignment
Val	Convert string to numeric value
Str$	Convert numeric value to string
Len	Get the length of the string

Asc	Convert character to ASCII code
Chr$	Convert ASCII code to character
Left$	Extract specified characters from the left side of the string
Mid$	Extract specified range of characters or substring from the string
Right$	Extract specified characters from the right side of the string
LSet$	Get a substring of specified length from the left side
RSet$	Get a substring of specified length from the right side
Space$	Return a string of spaces of specified length
LCase$	Return string in lowercase
UCase$	Return string in uppercase
LTrim$	Remove spaces from the left side of the string and return
RTrim$	Remove spaces from the right side of the string and return
Trim$	Remove spaces from both sides of the string and return
InStr	Retrieve and return the position of a specified character in the string
Tab$	Return a string of specified number of tab characters
Hex$	Convert hexadecimal value to string and return

Point Operation Commands

+	Relative coordinate incremental motion
-	Relative coordinate negative incremental motion
/	Modify tool frame, modify user coordinate system
:	Absolute coordinate motion
=	Assignment
@	Convert to specified user coordinate system

X	X-direction component operation
Y	Y-direction component operation
Z	Z-direction component operation
U	U-direction component operation
V	V-direction component operation
W	W-direction component operation
RX	User coordinate system rotation component operation around the X-axis
RY	User coordinate system rotation component operation around the Y-axis
RZ	User coordinate system rotation component operation around the Z-axis
TLX	Tool coordinate system rotation component operation around the X-axis
TLY	Tool coordinate system rotation component operation around the Y-axis
TLZ	Tool coordinate system rotation component operation around the Z-axis
TLU	Tool coordinate system rotation component operation around the U-axis
TLV	Tool coordinate system rotation component operation around the V-axis
TLW	Tool coordinate system rotation component operation around the W-axis
SavePoints	Save point file
LoadPoints	Load point file
ClearPoints	Clear points

Text Symbol Declaration:

1. {parameter}: Curly braces indicate that the parameter within the braces is optional and not a mandatory syntax requirement.

2. [parameter]: Square brackets indicate that one of the parameters within the brackets must be filled in and is a mandatory syntax requirement.

General Statement Detailed Description

Call

[Function]

Invoke a function

[Syntax]

```
Call function_name
```

[Parameter Description]

function_name	The name of the function to be called

[Usage Example]

```
Function main
    Call Task2
    Task2
    Task2()
Fend
```

Do..Loop

[Function]

Repeatedly execute code between Do..Loop based on whether the condition is met.

[Syntax]

```
Do {While / Until condition_expression}
    ...
Loop
```

or

```
Do
    ...
Loop {While/Until condition_expression}
```

[Parameter Description]

While/Until Keywords that can be appended after Do or Loop to utilize the condition expression.

condition_expression Determines whether to exit the Do..Loop loop based on whether the condition expression is satisfied.

[Usage Example]

```
Do                    'Infinite loop
    Go P1
    Wait 1
    Go P2
    Wait 2
Loop

    Do Until Sw(1) <> 1 'Enter loop when DI1 is not equal to 1
    Go P1
    Wait 1
    Go P2
    Wait 2
Loop

Do                   'Infinite loop
    Go P1
    Wait 1
    Go P2
```

```
    Wait 2
Loop Until ChkNet(201) < 0 ' Exit loop when TCP/IP communication on port 201 is abnormal
```

Else/Elself

[Function]

Used in conjunction with conditional statements; executes the code within Else or Elself when the If condition is not satisfied.

[Syntax]

```
If condition_expression Then
    ...
Elself condition_expression Then
    ...
Else
    ...
Endlf
```

[Parameter Description]

condition_expression Determines which code within Else or Elself is executed based on whether the condition expression is satisfied.

[Usage Example]

```
If Sw(1) = 1 Then
    Print "DI1 is On"

Elself Sw(1) = 1  Then
    Print "DI1 is Off"

Else
    Print "DI1 is abnormal"
Endlf
```

Exit

[Function]

Used to forcefully exit a loop or a function.

[Syntax]

```
Exit [Do/For/Function]
```

[Parameter Description]

Exit Do Exits the Do..Loop loop. If there are nested Do..Loop structures, Exit Do will only exit one level of the loop.

Exit For Exits the For loop. If there are nested For structures, Exit For will only exit one level of the loop.

Exit Function Exits the Function. If there are nested Function calls, Exit Function will only exit to the immediate upper-level Function.

[Usage Example]

```
Do                  ' Infinite loop
    Go P1
    Go P2
    If Sw(1) = 1 Then ' Enter condition when DI1 equals 1
        Exit Do       ' Exit the Do loop
    EndIf
Loop

Integer var
For var =  1 To 1    ' Loop 5 times
    Go P1
    Go P2
    If Sw(1) =  1 Then ' Enter condition when DI1 equals 1
        Exit For      ' Exit the For loop
    EndIf
Next


Function main
    Exit Function     ' Exit the function
```

```
Fend
```

For..To..Next

[Function]

Repeatedly executes the code between For..Next a specified number of times.

[Syntax]

```
For variable  = start_value To end_value {Step increment_value }

    ...
Next
```

[Parameter Description]

variable A variable used for iteration in the For loop; this variable must be defined beforehand.

start_value The initial value of the variable at the start of the For loop.

end_value The final value of the variable at the end of the For loop.

increment_value The value added to the variable after each loop iteration; defaults to an increment of 1.

[Usage Example]

```
Integer var
For var = 1 To 5 Step 1  ' Loop 5 times
    Go P1
    Go P2
    Print var
Next
```

Function..Fend

[Function]

Global function definition

[Syntax]

```
Function function_name{(parameter  As data_type, parameter   data_type, ...)}{As  data_type}
    ...
Fend
```

[Parameter Description]

function_name The required name of the Function.

parameter list The formal parameters that need to be passed values or value operations in the Function, separated by commas for multiple variables.

ByRef Used when the argument is an array; allows modification of the parameter index value.

ByVal Used when the argument is an array; does not allow modification of the parameter index value.

parameter As data_type Required parameters. Please declare the data type of the parameters.

function As data_type Returns the value of the function. Please declare the data type of the variable that stores the return value of the function.

[Usage Example]

```
Function aa          'A regular function
    Print "123"
    Print "456"
    Print "789"
Fend

aa              ' Call the function

Function Trig_CCD(IO_Idx As Integer, Integer_Time As Double) 'Function with parameters
    Do
        On IO_Idx
        Wait Integer_Time
        Off IO_Idx
    Loop
Fend
```

```
Function Add(Num1 As Integer, Num2 As Integer) As Integer 'Function with parameters and return value
    Integer result
    result = Num1 + Num2
    Add = result
Fend
```

Global

[Function]

Global variable definition.

[Syntax]

```
Global data_type  variable_name
```

[Parameter Description]

variable_name The required name of the variable.

Notes:

1. Global variables can only be defined outside of functions, not within them.
2. Global variables can only be defined outside of functions, not within them.

[Usage Example]

```
Global String    PointList$(300,24)      ' 2D array of strings
Global Boolean   P_Mode                  'Boolean type
Global String    CCD_Dev_X$              'String type
Global Double    SafeHight               ' Double precision floating-point type
Global Integer   ABCD                    'Integer type
Global Integer   AABB(10)                '1D array of integers
```

GoSub..Return

[Function]

Transfers control to a subroutine at the label specified by GoSub, which returns control to the original position of GoSub via Return.

[Syntax]

```
GoSub [label]
...
[label]:
...
Return
```

[Parameter Description]

label	The required name of the label.

[Usage Example]

```
Function main
    GoSub checkin           ' Jump to the label
        On 1
        On 2
    Exit Function

Checkin:                    ' label
    If In(0) = 1 And In(1) = 1 Then
        On  1
    Else
        Off 1
    EndIf
    Return                  ' Return to the GoSub point
Fend
```

GoTo

[Function]

Jump to a label to continue program execution.

[Syntax]

```
GoTo [label]
...
[label]:
...
```

[Parameter Description]

label	The required name of the label.

Notes:

1. The difference from GoSub is that GoTo does not have a Return operation.

[Usage Example]

```
Function main
    If Sw(1) = Off Then
        GoTo mainAbort
    EndIf
    Print    "DI1 is On"

    Exit Function

    mainAbort:
    Print     "DI1 is Off"
Fend
```

If..Then..Else..EndIf

[Function]

Conditional statement that executes corresponding programs based on a condition expression.

[Syntax]

```
If condition_expression Then
    ...
ElseIf condition_expression Then
    ...
Else
    ...
EndIf
```

[Parameter Description]

condition_expression Executes the corresponding program based on whether the condition expression is true.

[Usage Example]

```
Function main
    String AA$
    Integer num

    Input #201, AA$

    Num = val(AA$)          'Convert the received content to a numerical value and assign it to num

    If num > 1 Then
        Print    "num is greater than 1"
    ElseIf num < 1 Then
        Print    "num is less than 1"
    Else
        Print    "num is equal to 1"

    EndIf
Fend
```

Select..Case..Default..Send

[Function]

Select statement that determines which program segment to execute based on the value.

[Syntax]

```
Select variable
Case value1
    ...
{Case value2
    ...}
Default
    ...
Send
```

[Parameter Description]

variable	The variable name used for condition evaluation.
value1、value2	Determines whether the variable equals these values, entering the corresponding program segment.
Default	Executes the Default program segment when none of the Cases are true.

[Usage Example]

```
Function main
    Integer I
    For I = 0 To 10
        Select I
            Case 0
                Print    "I = 1"
                Off 1; On 2; Jump P1
            Case 3
                Print    "I = 3"
                On 1; Off 2
                Jump P2;Move P3; On 3
            Case 7
                Print    "I = 3"
                On 4
            Default
                On 7
```

```
    Send
  Next
Fend
```

Data Type Details

Boolean

[Function]

Boolean type, 2 bytes.

[Syntax]

```
Boolean variable {(number of array elements)}
Global Boolean variable {(number of array elements)}
```

[Parameter Description]

variable	The required name of the variable.
Value Range	True/False

Notes:

1. Maximum number of local variables: 2,000.
2. Maximum number of global variables: 100,000.

[Usage Example]

```
Function main
  Boolean AA
  If AA = True Then
    Print   "AA is True "
  Elself AA = False Then
    Print   "AA is False "
  Endlf
Fend
```

Byte

[Function]

Byte type, 2 bytes.

[Syntax]

Byte variable{(number of array elements)}

Global Byte variable{(number of array elements)}

[Parameter Description]

variable	The required name of the variable.
Value Range	-128~127

Notes:

3. Maximum number of local variables: 2000

4. Maximum number of global variables: 100000

[Usage Example]

```
Function main
    Byte A1(10)          ' 1D array of Byte type
    Byte B1(10,10)       ' 2D array of Byte type
    Byte CC(5,5,5)       ' 3D array of Byte type
    Byte Test_ok

    Test_ok = 15

    Test_ok =  (test_ok And 8)

    If   test_ok <> 8 Then
        Print   " High bit is ON"
    Else
        Print   " High bit is OFF"
    EndIf
Fend
```

Double

[Function]

Double-precision floating-point number, 8 bytes.

[Syntax]

Double variable{(number of array elements)}

Global Double variable{(number of array elements)}

[Parameter Description]

variable	The required name of the variable.
Value Range	Can hold values up to 14 decimal places.

Notes:

5. Maximum number of local variables: 2000

6. Maximum number of global variables: 100000

[Usage Example]

```
Function main
    Double var1
    Double A1(10)           ' 1D array of Double type
    Double B1(5,5)          ' 2D array of Double type
    Double CC(5,5,5)        ' 3D array of Double type
    Double arrayvar(10)
    Integer i

    For  i = 1  To 5
        Print arrayvar(i)
    Next
Fend
```

Int32

[Function]

32-bit integer, 4 bytes.

[Syntax]

Int32 variable{(number of array elements)}

Global Int32 variable{(number of array elements)}

[Parameter Description]

variable	The required name of the variable.
Value Range	-2147483648~2147483647

Notes:

7. Maximum number of local variables: 2000

8. Maximum number of global variables: 100000

[Usage Example]

```
Function main
    Int32 A1(10)          ' 1D array of Int32 type
    Int32 B1(5,5)         ' 2D array of Int32 type
    Int32 CC(5,5,5)       ' 3D array of Int32 type
    Int32 var1,arrayvar(10)
Fend
```

Integer

[Function]

Integer type, 2 bytes.

[Syntax]

```
Integer variable{(number of array elements)}
Global Integer variable{(number of array elements)}
```

[Parameter Description]

variable	The required name of the variable.
Value Range	-32768~32767

Notes:

9. Maximum number of local variables: 2000

10. Maximum number of global variables: 100000

[Usage Example]

```
Function main
    Integer A1(10)          ' 1D array of Integer type
    Integer B1(5,5)         ' 2D array of Integer type
    Integer CC(5,5,5)       ' 3D array of Integer type
    Integer var1,arrayvar(10)
Fend
```

Long

[Function]

Long integer type, 4 bytes.

[Syntax]

Long variable{(number of array elements)}

Global Long variable{(number of array elements)}

[Parameter Description]

variable The required name of the variable.

Value Range -2147483648~2147483647

Notes:

11. Maximum number of local variables: 2000

12. Maximum number of global variables: 100000

[Usage Example]

```
Function main
    Long A1(10)          ' 1D array of Long type
    Long B1(5,5)         ' 2D array of Long type
    Long CC(5,5,5)       ' 3D array of Long type
    Long var1,arrayvar(10)
Fend
```

Real

[Function]

Single-precision floating-point number, 4 bytes.

[Syntax]

Real variable{(number of array elements)}

Global Real variable{(number of array elements)}

[Parameter Description]

variable	The required name of the variable.
Value Range	Up to six decimal places.

Notes:

13. Maximum number of local variables: 2000

14. Maximum number of global variables: 100000

[Usage Example]

```
Function main
    Real A1(10)          ' 1D array of Real type
    Real B1(5,5)         ' 2D array of Real type
    Real CC(5,5,5)       ' 3D array of Real type
    Real var1,arrayvar(10)
Fend
```

Short

[Function]

Short integer, 2 bytes.

[Syntax]

Short variable{(number of array elements)}

Global Short variable{(number of array elements)}

[Parameter Description]

variable The required name of the variable.

Value Range -32768~32767

Notes:

15. Maximum number of local variables: 2000

16. Maximum number of global variables: 100000

[Usage Example]

```
Function main
    Short A1(10)            '1D array of Short type
    Short B1(5,5)           '2D array of Short type
    Short CC(5,5,5)         '3D array of Short type
    Short var1,arrayvar(10)
Fend
```

String

[Function]

String type.

[Syntax]

```
String variable${(number of array elements)}
Global String variable${(number of array elements)}
```

[Parameter Description]

variable$	The required name of the variable.
Value Range	Maximum 255 characters.

Notes:

17. Maximum number of local variables: 200
18. Maximum number of global variables: 10000

[Usage Example]

```
Function main
    String A1$(10)              '1D array of String type
    String B1$(5,5)             '2D array of String type
    String CC$(1,2,3)           '3D array of String type
    String var1$,arrayvar$(10)
Fend
```

UByte

[Function]

Unsigned byte type, 2 bytes.

[Syntax]

UByte variable{(number of array elements)}

Global UByte variable{(number of array elements)}

[Parameter Description]

variable	The required name of the variable.
Value Range	0~255

Notes:

19. Maximum number of local variables: 2000

20. Maximum number of global variables: 100000

[Usage Example]

```
Function main
    UByte A1(10)                '1D array of UByte type
    UByte B1(5,5)               '2D array of UByte type
    UByte CC(1,2,3)             '3D array of UByte type
    UByte var1,arrayvar(10)
Fend
```

UInt32

[Function]

Unsigned 32-bit integer, 4 bytes.

[Syntax]

UInt32 variable{(number of array elements)}

Global UInt32 variable{(number of array elements)}

[Parameter Description]

variable	The required name of the variable.
Value Range	0~4294967295

Notes:

21. Maximum number of local variables: 2000

22. Maximum number of global variables: 100000

[Usage Example]

```
Function main
    UInt32 A1(10)                    ' 1D array of UInt32 type
    UInt32 B1(5,5)                   ' 2D array of UInt32 type
    UInt32 CC(1,2,3)                 ' 3D array of UInt32 type
    UInt32 var1,arrayvar(10)
Fend
```

UShort

[Function]

Unsigned short integer, 2 bytes.

[Syntax]

UShort variable{(number of array elements)}

Global UShort variable{(number of array elements)}

[Parameter Description]

variable The required name of the variable.

Value Range 0~65535

Notes:

23. Maximum number of local variables: 2000

24. Maximum number of global variables: 100000

[Usage Example]

```
Function main
    UShort A1(10)                    ' 1D array of UShort type
    UShort B1(5,5)                   ' 2D array of UShort type
    UShort CC(1,2,3)                 ' 3D array of UShort type
    UShort var1,arrayvar(10)
Fend
```

Preserve

[Function]

Define global variables to maintain their state during power loss, stopping the program or rebooting the controller while preserving variable memory.

[Syntax]

```
Global Preserve Integer variable{(number of array elements)}
```

[Parameter Description]

variable	The required name of the variable.
Value Range	-2147483648~2147483647

Notes:

25. Maximum number of local variables: 2000
26. Maximum number of global variables: 100000

[Usage Example]

```
Global Preserve Integer Hi
Global Preserve Integer PowerNex
```

Motion Command Detailed Description

!...!

[Function]

Parallel processing of input and output during motion.

[Syntax]

Motion Command ! Processing Statement!

[Parameter Description]

Motion Command Related motion commands such as Go, Move, Arc, Jump, etc.

Processing Statement As follows

1. D: Percentage processing statement.

2. On/Off、MemOff、Out、OutW、MemOut、Signal、Wait、Print、Print#, and other I/O commands. For details, please refer to the detailed explanation of I/O commands.

[Usage Example]

```
Function main
    Go P1 !D50; On 1!
    ' During PTP motion to point P1, when reaching 50% of the path, set DO1 to ON.

    Jump P2 !D30; On 1; D70; Off 1!
    ' During gate-type motion to point P2, when reaching 30% of the distance, set DO1 to ON, and OFF at 70%.

    Move P3 !D10; On 5; Wait 0.5; Off 5!
    ' During linear motion to point P3, when reaching 10% of the distance, set DO5 to ON, and OFF after 0.5 seconds.
    Fend
```

Motor

[Function]

Enable/disable the entire robot or return the robot's enable state.

[Syntax]

Motor {On/Off}

[Parameter Description]

On/Off On to enable the entire robot, Off to disable the entire robot. If omitted, it returns the robot's enable state.

[Usage Example]

```
Function main
    If Motor =  Off Then
        Motor On
    EndIf
Fend
```

[Return Description]

Return Data1 1: 0 / 1, where 0 means Off and 1 means On.

ServoOn 、ServoOff

[Function]

Enable/disable individual axes or return the enable state of individual axes.

[Syntax]

Enable/Disable:

```
ServoOn JointNumber1 {,JointNumber2...}}
ServoOff JointNumber1 {,JointNumber2...}}
```

Get Enable State:

```
ServoOn(JointNumber)
ServoOff(JointNumber)
```

[Parameter Description]

JointNumber Specify the joints to enable/disable using 1, 2, 3, 4, 5, 6, or to return the enable state of the joints.

[Usage Example]

```
Function main
    If ServoOff =  True Then     'Check the enable state of Joint 1
        ServoOn 1                'Enable Joint 1
    EndIf
Fend
```

[Return Description]

Return Data1 TRUE / FALSE, where TRUE means enabled and FALSE means disabled.

Power

[Function]

Switch between high and low power modes or return the current power mode.

[Syntax]

```
Power {High/Low}
```

[Parameter Description]

High/Low	High for high power, Low for low power. If omitted, it returns the current power state.

[Usage Example]

```
Function main
    If Power = Low Then     ' Check if currently in low power mode
        Power High          ' Switch to high power
    EndIf
Fend
```

[Return Description]

Return Data1	0 / 1, where 0 corresponds to low power mode and 1 corresponds to high power mode.

SpeedFactor

[Function]

Set the global motion speed or return the current global speed.

[Syntax]

SpeedFactor { Percentage }

[Parameter Description]

Percentage Set the current global speed as a percentage from 1% to 100%. If omitted, it returns the current global speed.

[Usage Example]

```
Function main
    Motor On
    Power High

    Speed 100       'PTP speed
    Accel 100,100
    SpeedS 2000     'Line speed
    AccelS 100,100

    SpeedFactor 80  ' Set global speed

    Print SpeedFactor   ' Print current global speed
Fend
```

[Return Description]

Return Data1 Percentage value from 1 to 100%, with the default data type being a floating-point number.

Speed

[Function]

Set the operation speed for PTP motion or return the current PTP motion speed.

[Syntax]

Speed {Percentage{, Rotation Speed Percentage, Approach Speed Percentage}}

[Parameter Description]

Percentage	Set the current PTP motion speed as a percentage from 1% to 100%.
Rotation Speed Percentage	Set the moving speed percentage for the Jump command, ranging from 1% to 100% (optional).
Approach Speed Percentage	Set the approach speed percentage for the Jump command, ranging from 1% to 100% (optional).

Notes: If this command is not used to define the PTP motion speed in the program, the default speed of 20% will be applied.

[Usage Example]

```
Function main
    Motor On
    Power High

    Speed 100        'PTP speed
    Accel 100,100
    SpeedS 2000      'Line speed
    AccelS 100,100

    SpeedFactor 80  'Set global speed

    Print SpeedFactor    ' Print current global speed
Fend
```

[Return Description]

Return Data1	Percentage value from 1 to 100%, with the default data type being a floating-point number.

Accel

[Function]

Set the acceleration and deceleration for PTP motion, or return the current acceleration and deceleration for PTP motion.

[Syntax]

Accel {Acceleration Percentage, Deceleration Percentage, {Rotation Acceleration Percentage, Rotation Deceleration Percentage, Approach Acceleration Percentage, Approach Deceleration Percentage}}

[Parameter Description]

Acceleration Percentage : Set the current PTP motion acceleration as a percentage from 1% to 100%.

Deceleration Percentage Set the current PTP motion deceleration as a percentage from 1% to 100%.

Rotation Acceleration Percentage Set the acceleration percentage for the Jump command's movement action, ranging from 1% to 100% (optional).

Rotation Deceleration Percentage Set the deceleration percentage for the Jump command's movement action, ranging from 1% to 100% (optional).

Approach Acceleration Percentage Set the acceleration percentage for the Jump command's approach action, ranging from 1% to 100% (optional).

Approach Deceleration Percentage Set the deceleration percentage for the Jump command's approach action, ranging from 1% to 100% (optional).

[Usage Example]

```
Function main
    Accel 100,100,50,50,50,50   'Set PTP motion acceleration and deceleration

    Print Accel(1)              ' Print PTP motion acceleration
    Print Accel(2)              ' Print PTP motion deceleration
    Print Accel(3)              ' Print Jump motion rotation acceleration
    Print Accel(4)              ' Print Jump motion rotation deceleration
    Print Accel(5)              ' Print Jump motion approach acceleration
    Print Accel(6)              ' Print Jump motion approach deceleration
Fend
```

[Return Description]

Return Data	Percentage value from 1 to 100%, with the default data type being a floating-point number.

Go

[Function]

Execute PTP motion from the current position to a specified position.

[Syntax]

Go Target Coordinate {CP}{Till/Find}{!...!}

[Parameter Description]

Target Coordinate	Specify the target position using point data.
CP	Set motion smoothing. Optional.
Till/Find	Till expression = On/Off. Find expression = On/Off. Optional. For details, please refer to the detailed explanation of Till/Find.
!...!	Parallel processing of I/O and other input/output during motion. Optional. For details, please refer to the detailed explanation of parallel processing.

[Usage Example]

```
Function main
    Go P1
    Go P2 CP

    Go P1+X(20)
    'Move to point P1 with a relative offset of 20mm in the X direction.

    Go P2:X(200)
    'Move to point P2 with the X coordinate changed to 200mm.

    Go P3 Till Sw(1) = On And Sw(5) = Off
    'During motion to point P3, stop when DI1 is On and DI5 is Off.

    Go P4 Till Sw(1) = On !D50; On 5!
    'During motion to point P4, stop when DI1 is On, and set DO5 to On when reaching 50% of the path.

    Go P5 /R        'Move to point P5 using right-hand coordinate system.
Fend
```

BGo

[Function]

Execute offset PTP motion in the selected user coordinate system.

[Syntax]

BGo Target Coordinate {CP}{Till/Find}{!...!}

[Parameter Description]

Target Coordinate	Specify the target position using point data.
CP	Set motion smoothing. Optional.
Till/Find	Till expression = On/Off. Find expression = On/Off. Optional. For details, please refer to the detailed explanation of Till/Find.
!...!	Parallel processing of I/O and other input/output during motion. Optional. For details, please refer to the detailed explanation of parallel processing.

[Usage Example]

```
Function main
    BGo XY(100, 0, 0, 0)            ' Move 100mm in the X direction in the selected user
                                    coordinate system.
    P1 = XY(300, 300, -20, 0)
    P2 = XY(300, 300, -20, 0) /L
    Local 1, XY(0, 0, 0, 45)

    Go P1
    Print Here
    'X:300.000 Y:300.000 Z:-20.000 U:0.000 V:0.000 W:0.000 /R /0
    BGo XY(0, 50, 0, 0)
    Print Here
    'X:300.000 Y:350.000 Z:-20.000 U:0.000 V:0.000 W:0.000 /R /0
    Go P2
    Print Here
    'X:300.000 Y:300.000 Z:-20.000 U:0.000 V:0.000 W:0.000 /L /0
    BGo XY(0, 50, 0, 0)
    Print Here
    'X:300.000 Y:350.000 Z:-20.000 U:0.000 V:0.000 W:0.000 /L /0
    BGo XY(0, 50, 0, 0) /L
    Print Here
    'X:264.645 Y:385.355 Z:-20.000 U:0.000 V:0.000 W:0.000 /L /0
Fend
```

TGo

[Function]

Execute offset PTP motion in the current tool coordinate system.

[Syntax]

TGo Target Coordinate {CP}{Till/Find}{!...!}

[Parameter Description]

Target Coordinate	Specify the target position using point data.
CP	Set motion smoothing. Optional.
Till/Find	Till expression = On/Off. Find expression = On/Off. Optional. For details, please refer to the detailed explanation of Till/Find.
!...!	Parallel processing of I/O and other input/output during motion. Optional. For details, please refer to the detailed explanation of parallel processing.

[Usage Example]

```
Function main
    TGo XY(100, 0, 0, 0)            ' Move 100mm in the X direction in the current tool
                                    coordinate system.

    Tool 0
    P1 = XY(300, 300, -20, 0)
    P2 = XY(300, 300, -20, 0) /L

    Go P1
    Print Here
    'X:300.000 Y:300.000 Z:-20.000 U:0.000 V:0.000 W:0.000 /R /0

    TGo XY(0, 0, -30, 0)
    Print Here
    'X:300.000 Y:350.000 Z:-50.000 U:0.000 V:0.000 W:0.000 /R /0

    Go P2
    Print Here
    'X:300.000 Y:300.000 Z:-20.000 U:0.000 V:0.000 W:0.000 /L /0

    TGo XY(0, 0, 30, 0)
    Print Here
    'X:300.000 Y:350.000 Z:-50.000 U:0.000 V:0.000 W:0.000 /L /0
Fend
```

Speeds

[Function]

Set the operation speed for linear interpolation motion, or return the current operation speed for linear interpolation motion.

[Syntax]

SpeedS {Speed{, Rotation Speed, Approach Speed}}

[Parameter Description]

Speed	Set the current linear interpolation motion speed in the range of 0.1 to 2000 mm/s.
Rotation Speed	Set the movement speed for the Jump3 command in the range of 0.1 to 2000 mm/s (optional).
Approach Speed	Set the approach speed for the Jump3 command in the range of 0.1 to 2000 mm/s (optional).

[Usage Example]

```
Function main
    Motor On
    Power High

    Speed 100        'PTP speed
    Accel 100,100
    SpeedS 2000      ' Line speed
    AccelS 100,100

    SpeedFactor 80   ' Global speed

    Print SpeedFactor    ' Print current global speed
Fend
```

[Return Description]

Return Data1	Percentage value from 1 to 100%, with the default data type being a floating-point number.

AccelS

[Function]

Set the acceleration and deceleration for linear interpolation motion, or return the current acceleration and deceleration for linear interpolation motion.

[Syntax]

AccelS { Acceleration Percentage, Acceleration Percentage, {Rotation Acceleration Percentage, Rotation Acceleration Percentage, Approach Acceleration Percentage, Approach Deceleration Percentage}}

[Parameter Description]

Acceleration Percentage Set the current linear interpolation motion acceleration as a percentage from 1% to 100%.

Acceleration Percentage Set the current linear interpolation motion deceleration as a percentage from 1% to 100%.

Rotation Acceleration Percentage Set the acceleration percentage for the Jump3 and Jum3CP commands' movement actions, ranging from 1% to 100% (optional).

Rotation Acceleration Percentage Set the deceleration percentage for the Jump3 and Jum3CP commands' movement actions, ranging from 1% to 100% (optional).

Approach Acceleration Percentage Set the acceleration percentage for the Jump3 and Jum3CP commands' approach actions, ranging from 1% to 100% (optional).

Approach Deceleration Percentage Set the deceleration percentage for the Jump3 and Jum3CP commands' approach actions, ranging from 1% to 100% (optional).

[Usage Example]

```
Function main
    Motor On
    Power High

    Speed 100       'PTP speed
    Accel 100,100
    SpeedS 2000     ' Line speed
    AccelS 100,100

    SpeedFactor 80  ' Global speed

    Print SpeedFactor   ' Print current global speed
```

```
Fend
```

Move

[Function]

Execute linear interpolation motion from the current position to a specified position.

[Syntax]

```
Move Target Coordinate {CP}{Till/Find}{!...!}
```

[Parameter Description]

Target Coordinate	Specify the target position using point data.
CP	Set motion smoothing. Optional.
Till/Find	Till expression = On/Off. Find expression = On/Off. Optional. For details, please refer to the detailed explanation of Till/Find.
!...!	Parallel processing of I/O and other input/output during motion. Optional. For details, please refer to the detailed explanation of parallel processing.

[Usage Example]

```
Function main
    Move P1          'Move in a straight line to point P1
    Move P2 CP       'Move in a straight line to point P2 with a smooth transition at P2

    Move P3 Till Sw(1) = On And Sw(5) = Off
    'During motion to point P3, stop when DI1 is On and DI5 is Off.

    Move P4 Till Sw(1) = On !D50; On 5!
    ' During motion to point P4, stop when DI1 is On, and set DO5 to On when reaching 50% of the path.
Fend
```

BMove

[Function]

Execute offset linear interpolation motion in the selected user coordinate system.

[Syntax]

BMove Target Coordinate {CP}{Till/Find}{!...!}

[Parameter Description]

Target Coordinate	Specify the target position using point data.
CP	Set motion smoothing. Optional.
Till/Find	Till expression = On/Off. Find expression = On/Off. Optional. For details, please refer to the detailed explanation of Till/Find.
!...!	Parallel processing of I/O and other input/output during motion. Optional. For details, please refer to the detailed explanation of parallel processing.

[Usage Example]

```
Function main
BMove XY(100 , 0 , 0 , 0)            ' Move 100mm in the X direction in the selected user
                                     coordinate system
    P1 = XY(300 , 300 , -20 , 0)
    P2 = XY(300 , 300 , -20 , 0) /L
    Local 1 , XY(0 , 0 , 0 , 45)

    Go P1
    Print Here
    'X:300.000 Y:300.000 Z:-20.000 U:0.000 V:0.000 W:0.000

    BMove XY(0 , 50 , 0 , 0)
    Print Here
    'X:300.000 Y:350.000 Z:-20.000 U:0.000 V:0.000 W:0.000

    Go P2
    Print Here
    'X:300.000 Y:300.000 Z:-20.000 U:0.000 V:0.000 W:0.000

    BMove XY(0 , 50 , 0 , 0)
    Print Here
    'X:300.000 Y:350.000 Z:-20.000 U:0.000 V:0.000 W:0.000

Fend
```

TMove

[Function]

Execute offset linear motion in the current tool coordinate system.

[Syntax]

TMove Target Coordinate {CP}{Till/Find}{!...!}

[Parameter Description]

Target Coordinate	Specify the target position using point data.
CP	Set motion smoothing. Optional.
Till/Find	Till expression = On/Off. Find expression = On/Off. Optional. For details, please refer to the detailed explanation of Till/Find.
!...!	Parallel processing of I/O and other input/output during motion. Optional. For details, please refer to the detailed explanation of parallel processing.

[Usage Example]

```
Function main
    TMove XY(100 , 0 , 0 , 0) ' Move 100mm in the X direction in the selected tool coordinate
                              system

    Tool 0          'Select Tool 0
    P1 = XY(300 , 300 , -20 , 0)
    P2 = XY(300 , 300 , -20 , 0) /L

    Go P1
    Print Here
    'X:300.000 Y:300.000 Z:-20.000 U:0.000 V:0.000 W:0.000

    TMove XY(0 , 0 , -30 , 0)
    Print Here
    'X:300.000 Y:350.000 Z:-50.000 U:0.000 V:0.000 W:0.000

    Go P2
    Print Here
    'X:300.000 Y:300.000 Z:-20.000 U:0.000 V:0.000 W:0.000

    TMove XY(0 , 0 , -30 , 0)
    Print Here
    'X:300.000 Y:350.000 Z:-50.000 U:0.000 V:0.000 W:0.000
```

```
Fend
```

Jump

[Function]

Perform door-shaped PTP motion from the current position to a specified position. First, it rises, then translates, and finally descends.

[Syntax]

```
Jump Target Coordinate {C Arch Number} {LimZCoordinate Value} {CP} {Till/Find/Sense} {!...!}
```

[Parameter Description]

Target Coordinate Specify the target position using point data.

C Arch Number Select different arches using C0 to C7. For details, please refer to the detailed explanation of Arch.

LimZ Coordinate Value Set the maximum allowable movement of the third joint during the Jump motion (limitation value). For details, please refer to the detailed explanation of LimZ. Optional.LimZ

CP Set motion smoothing. Optional.

Till/Find/Sense Till expression = On/Off. Find expression = On/Off. Sense expression = On/Off. Optional. For details, please refer to the detailed explanation of Till/Find/Sense.

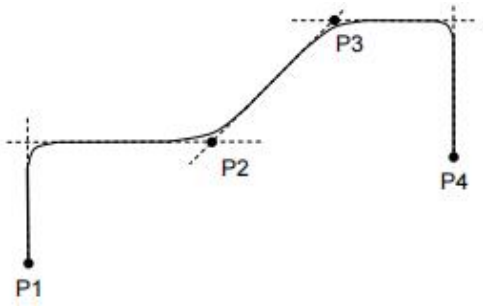
!...! Parallel processing of I/O and other input/output during motion. Optional. For details, please refer to the detailed explanation of parallel processing.

[Usage Example]

```
Function main
    Go P1
    Jump P2:Z(-50) C0 LimZ -50 CP
    Go P3:Z(0) CP
    Jump P4 C0 LimZ 0

    Jump P1
    Jump P2 CP
    Jump P3 Till Sw(1) = On And Sw(5) = Off
    'During motion to point P3, stop when DI1 is On and DI5 is Off.

    Jump P4 Till Sw(1) = On !D50; On 5!
    'During motion to point P4, stop when DI1 is On, and set DO5 to On when reaching 50% of the path.
```



```
    Jump P5 /R C1 LimZ-10 Sense Sw(2) = On
    'Select Arch1 for approach parameters, and during motion to point P5 in a right-hand coordinate system, the maximum Z-axis can only be -10mm.
    'Stop above the target point when DI2 is On, and set DO5 to On when reaching 50% of the path.
Fend
```

Arc、Arc3

[Function]

The Arc function is used to move the robotic arm from the current position to a specified position with arc interpolation in the XY plane.

The Arc3 function is used to move the robotic arm from the current position to a specified position with arc interpolation in three-dimensional space.

[Syntax]

Arc Intermediate Coordinate, Target Coordinate {CP}{Till/Find}{!...!}

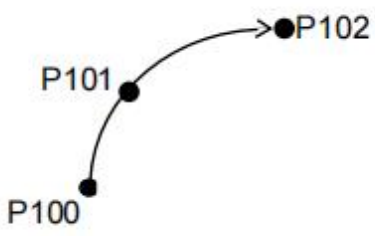
Arc3 Intermediate Coordinate, Target Coordinate {CP}{Till/Find}{!...!}

[Parameter Description]

Intermediate Coordinate	Specify the position that the specified arc will pass through using point data.
Target Coordinate	Specify the target position using point data.
CP	Set motion smoothing. Optional.
Till/Find	Till expression = On/Off. Find expression = On/Off. Optional. For details, please refer to the detailed explanation of Till/Find.
!...!	Parallel processing of I/O and other input/output during motion. Optional. For details, please refer to the detailed explanation of parallel processing.

[Usage Example]

```
Function main
    Go P100
    Arc P101 , P102
Fend
```



CP

[Function]

Set motion smoothing.

[Syntax]

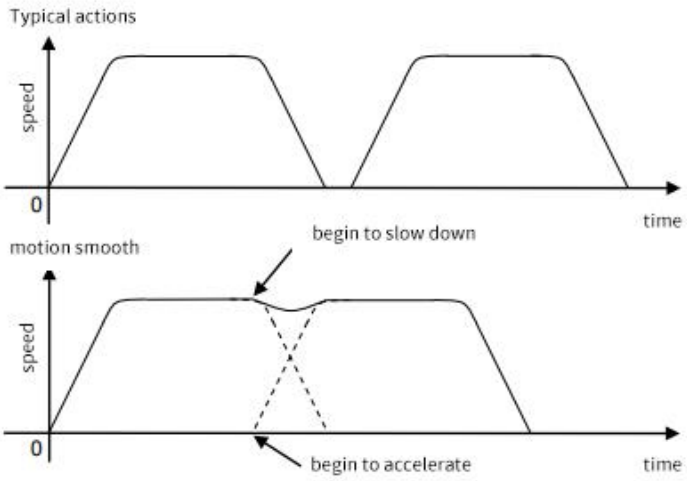
CP {On/Off}
Motion Command Target Coordinate CP

[Parameter Description]

On/Off On enables motion smoothing, Off disables it.

Motion Command Includes commands like Go, Move, Arc, Jump, etc.

[Usage Example]




```
Function main
    Go P1 CP
    Move P2 CP

    CP On          ' The motion commands following CP On will default to smooth transitions.
    Move P3
    Move P4
    CP Off

    If CP = Off Then
        CP On
    EndIf
Fend
```

CPZone

[Function]

Set the degree of motion smoothing.

[Syntax]

CPZone {On/Off} { PTP Motion Smoothing Ratio, Linear Interpolation Smoothing Ratio }

[Parameter Description]

On/Off	On enables smoothing ratio, Off disables the smoothing ratio, reverting to the system default ratio.
PTP Motion Smoothing Ratio	Set the blending degree of PTP motion smoothing in the range of 1 to 200 mm.
Linear Interpolation Smoothing Ratio	Set the blending degree of linear interpolation motion smoothing in the range of 1 to 200 mm.

[Usage Example]

```
Function main

    CPZone On , 50 , 50

Fend
```

Here

[Function]

Used to teach the robot to a point as a coordinate input or to retrieve the robot's current coordinates.

[Syntax]

Here {Point Number/Point Name}

[Parameter Description]

Point Number/Point Name Teach the current coordinates to the specified point.

[Usage Example]

```
Function main

    Here P1          'Teach the current coordinates to point P1
    Go Here + X(10)  'Offset 10mm in the X direction from the current position

Fend
```

Home

[Function]

Move the robot to the user-defined Home position.

[Syntax]

Home

[Parameter Description]

[Usage Example]

```
Function main

    Home        'Return to the Home position

Fend
```

Local

[Function]

Define and display user coordinate systems.

[Syntax]

One Point Teaching:Local Local User Coordinate Number, Point
Two Points Teaching: Local Local User Coordinate Number, Point1, Point2, {X/Y}
Three Points Teaching: Local Local User Coordinate Number , X-axis Point, Y-axis Point, {X/Y}
Four Points Teaching: Local Local User Coordinate Number, (Point1:Point2), (Point3:Point4), {L/R}

[Parameter Description]

Local User Coordinate Number Specify the user coordinate system to be defined using numbers from 1 to 64.

Origin、Point1... Specify the point data for the user coordinate using point variables.

L/R Align the user coordinate origin to the left (1) or right (2). Optional.

X/Y Define the line connecting the origin to the X-axis or Y-axis as the X-axis or Y-axis of the local coordinate system. Optional. Defaults to X.

[Usage Example]

```
Function main

    'Define the user coordinate system using the origin and the angle relative to the base coordinate system
    Local 1 , P1
    Local 2 , XY(100 , 200 , 300 , 60)

    Local 3 , P1 , P2 , X            'Two points teaching, specifying the X-axis with two points

    Local 4 , P1 , P2 , P3           'Three points teaching

    Local 5 , (P1:P11) , (P2:P12)    'Four points teaching

Fend
```

TLSet

[Function]

Define and display tool coordinate systems.

[Syntax]

TLSet {Tool Coordinate Number, {Tool Setting Data}}

[Parameter Description]

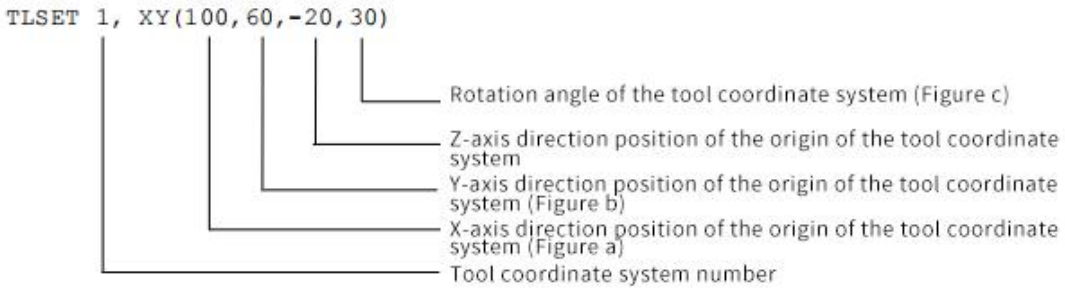
Tool Coordinate Number Specify the tool coordinate system to be defined using numbers from 1 to 64.

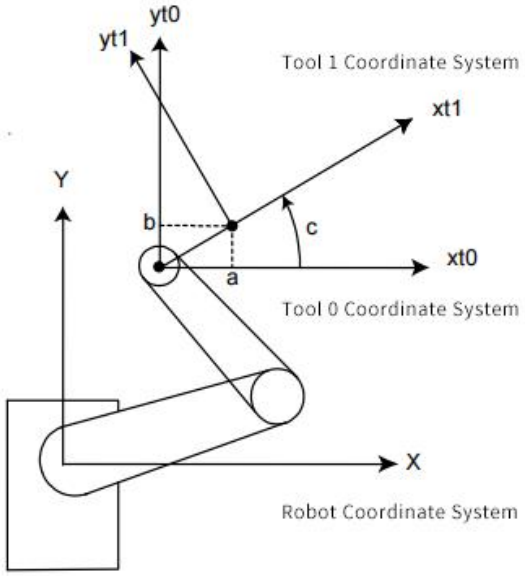
Tool Setting Data Set the origin and orientation of the tool coordinate system using point numbers, point names, or point variables.

Notes: If all parameters are omitted, all TLSet configurations will be displayed.

[Usage Example]

```
Function main
    TLSet 1,XY(50 , 100 , -20 , 30)
    TLSet 2 , P10 +X(20)
Fend
```





Tool

[Function]

Select a tool coordinate system or return the currently selected tool coordinate system.

[Syntax]

Tool {Tool Coordinate Number}

[Parameter Description]

Tool Coordinate Number Specify the selected tool coordinate system using numbers from 1 to 64.

[Usage Example]

```
Function main

    Tool 1                                  'Select Tool Coordinate System 1

    If Tool = 0 Then
        Tool 2
    Endlf

Fend
```

TCalib

[Function]

Define the tool coordinate system using the three-point method.

[Syntax]

TCalib Tool Coordinate Number, Tool Auxiliary Point1, Tool Auxiliary Point2, Tool Auxiliary Point3

[Parameter Description]

Tool Coordinate Number Specify the selected tool coordinate system using numbers from 1 to 64.

Tool Auxiliary Point1 Align the end center of the fixture mounted on the robot's lead screw to the teaching marker.

Tool Auxiliary Point2 Rotate the U-axis 60 to 120° clockwise or counterclockwise from Tool Auxiliary Point 1, and use the incremental motion function to realign the end center of the fixture to the teaching marker.

Tool Auxiliary Point3 Rotate the U-axis in the same direction again 60 to 120° from Tool Auxiliary Point 2, and use the incremental motion function to realign the end center of the fixture to the teaching marker.

[Usage Example]

```
Function main

    TCalib 1 , P0 , P1 , P2                'Use P0, P1, P2 as auxiliary points to teach Tool
                                           Coordinate System 1

Fend
```

Till

[Function]

Set the conditions for stopping motion commands such as Jump, Go, Move, etc.

[Syntax]

Till {condition_expression}

[Parameter Description]

condition_expression The condition under which the motion will stop. Various I/O instructions, operators, etc. can be used.

[Usage Example]

```
Function main
    Till Sw(1) = Off     'Set Till condition (DI1 is OFF)
        Go P1 Till       'Stop when the condition from the previous line is met

    Till Sw(1) = On And Sw(1) = On 'Set new Till condition
        Move P2 Till               'Stop when the condition from the previous line is met

        Move P5 Till Sw(10) = On   'Stop when the condition in this line is met
Fend
```

Agl

[Function]

Return the angle or position of a specified joint.

[Syntax]

Agl(JointNumber)

[Parameter Description]

JointNumber	Specify the joint number as an integer. For SCARA robots, the range is 1 to 4; for six-axis robots, it is 1 to 6. If there are extension axes, the numbering will increase according to the extension axes.

[Usage Example]

```
Function main
    Print Agl(1) , Agl(2)
Fend
```

AglToPls

[Function]

Convert the angles of the robot's joints to pulse values.

[Syntax]

AglToPls(Joint Position1, Joint Position2, Joint Position3, Joint Position4, {Joint Position5, Joint Position6}....)

[Parameter Description]

Joint Position	Specify the joint angles in order for each joint. The number of parameters can be increased as appropriate based on the model and any extension axes.
Command fit	When used as a parameter for Go, it executes absolute joint motion. When used as a parameter for Print, it outputs pulse values; other usages return spatial point data.

[Usage Example]

```
Function main

    P1 = AglToPls(0 , 0 , 0 , 90 , 0 , 0)
    Go P1

    Move AglToPls(0 , 0 , 0 , 90 , 0 , 0)

    Go AglToPls(0 , 0 , 0 , 90 , 0 , 0)

Fend
```

Arch

[Function]

Set/display the Arch parameters for the Jump, Jump3, and Jump3CP commands.

[Syntax]

```
Arch {Arch Number{, Transfer Distance, Approach Distance }}
Arch(Arch Number, Parameter Number)
```

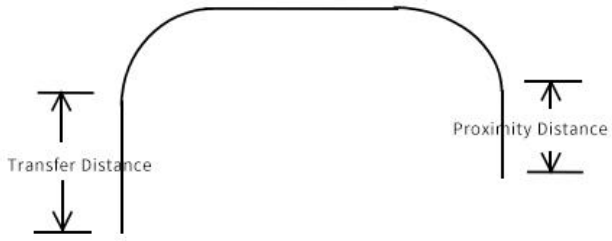
[Parameter Description]

Arch Number Specify the Arch number as an integer from 0 to 6. If not specified, the system uses a default value; see the table for details.

Transfer Distance The vertical distance from the starting point, in mm.

Approach Distance The vertical distance from the target point, in mm.

Parameter Number 1: Transfer Distance. 2: Approach Distance.



Default Values of the Arch Table

Arch Number	Transfer Distance	Proximity Distance
0	30	30
1	40	40
2	50	50
3	60	60
4	70	70
5	80	80
6	90	90

[Usage Example]

```
Function main

    Arch 0 , 20 , 20                'Set two distance parameters for Arch 0

    Jump P1 C0                      ' Enable Arch 0 in Jump

    If Arch(0,1) = 30 And Arch(0,2) = 30 Then
        'Check if the two distance parameters of Arch 0 meet the conditions
        Arch 0 , 25 , 25
    EndIf

Fend
```

Find

[Function]

Save coordinates based on the condition expression during motion.

[Syntax]

```
Find condition_expression
```

[Parameter Description]

condition_expression	The condition for saving coordinates during motion. Various I/O instructions, operators, etc. can be used.

[Usage Example]

```
Function main

    Go P10 Find Sw(5) = On            'Save coordinates when DI5 is set to On during motion

    If PosFound Then
        Go FindPos                    'Retrieve the coordinates saved by FindPos
    Else
        Print "DI5 was not set to ON during motion"
    EndIf

Fend
```

FindPos

[Function]

Return the coordinates saved by Find.

[Syntax]

FindPos
Go FindPos

[Parameter Description]

[Usage Example]

```
Function main

    Go P10 Find Sw(5) = On          ' Save coordinates when DI5 is set to On during motion

    If PosFound Then
        Go FindPos                  ' Retrieve the coordinates saved by FindPos
    Else
        Print "DI5 was not set to ON during motion"
    EndIf

Fend
```

InPos

[Function]

Return the robot's positioning status.

[Syntax]

InPos

[Parameter Description]

Notes:

1. Returns True when positioning is complete.
2. Returns True when positioning is complete.

[Usage Example]

```
Function main

    P0 = XY(0 , -100 , 0 , 0)
    P1 = XY(0 , 100 , 0 , 0)
    Xqt MonitorPosition
    Do
        Jump P0
        Wait 0.5
        Jump P1
        Wait 0.5
    Loop
Fend

Function MonitorPosition
    Boolean oldInPos , Pos

    Do
        Pos = InPos
        If Pos <> oldInPos Then
            Print "InPos = " , Pos
        EndIf

        oldInPos = Pos

    Loop
Fend
```

JA

[Function]

Return the robot's coordinates based on joint angles.

[Syntax]

JA(Joint Angle1, Joint Angle2, Joint Angle3, Joint Angle4, {Joint Angle5, Joint Angle6}....)

[Parameter Description]

Joint Angle Specify the joint angles in order for each joint. Units: rotary joints in degrees (deg), linear joints in millimeters (mm). The number of parameters can be increased as appropriate based on the model and any extension axes.

[Usage Example]

```
Function main
    P10 = JA(60 , 30 , -50 , 45)
    Go JA(135 , 90 , -50 ,90)
    P3 = JA(0 , 0 , 0 , 0 , 0 , 0)
Fend
```

Joint

[Function]

Used to display the current robot position in the joint coordinate system.

[Syntax]

Joint

[Parameter Description]

[Usage Example]

```
Function main
    Joint
    'Print Information]:Joint: 1:-11.880deg 2:30.432deg 3: -21.902mm 4:-18.552deg
```

```
Fend
```

JS

[Function]

Return whether the Sense condition during Jump, Jump3, or Jump3CP has been met.

[Syntax]

```
JS
```

[Parameter Description]

Notes:

1. Returns True if the Sense condition is met while stopping above the target point.
2. Returns False if the Jump action was completed normally and the target point was reached.

[Usage Example]

```
Function main
    Print searchSensor
Fend

Function searchSensor As Boolean
    Sense Sw(5) = On
    Jump P0
    Jump P1 Sense

    If JS = True Then
        Print "Sense condition met"
        searchSensor = True
    EndIf
Fend
```

LimZ

[Function]

Set/display the initial value of the Z-coordinate height of the third joint during Jump motion.

[Syntax]

```
LimZ { Z-coordinate value}
```

[Parameter Description]

Z-coordinate value Specify the coordinate value of the third joint within its range of motion.

[Usage Example]

```
Function main
    LimZ -10                    ' Set the default value of LimZ
    Jump P1                     ' During JUMP execution, maintain the Z-axis height at -10
    Jump P2 LimZ -20            ' During JUMP execution, maintain the Z-axis height at -20
    Jump P3                     ' During JUMP execution, maintain the Z-axis height at -10
    If LimZ > -10 Then          ' When the set value is greater than -10
        LimZ -30                ' Set the default value of LimZ to -30
    EndIf
Fend
```

Pass

[Function]

Perform PTP motion passing near specified points (without stopping).

[Syntax]

```
Pass Target Coordinate1{, Target Coordinate2, Target Coordinate3....}
```

[Parameter Description]

Target Coordinate Specify the position to move to using point numbers or point names.

Additional point parameters can be added as needed.

[Usage Example]

```
Function main
    Pass P1 , P2 , P3       ' Smoothly transition from P1 → P2 → P3
```

```
Fend
```

Pls

[Function]

Return the pulse value of the specified joint at the current position.

[Syntax]

```
Pls(JointNumber)
```

[Parameter Description]

JointNumber Specify the joint for which to obtain the pulse value using an integer from 1 to the number of joints.

[Usage Example]

```
Function main
    Real Axle_1 , Axle_2 , Axle_3 , Axle_4
    Axle_1 = pls(1)
    Axle_2 = pls(2)
    Axle_3 = pls(3)
    Axle_4 = pls(4)
    Print "Axis 1 Pulse: ",Axle_1
    Print "Axis 2 Pulse: ",Axle_2
    Print "Axis 3 Pulse: ",Axle_3
    Print "Axis 4 Pulse: ",Axle_4
Fend
```

PosFound

[Function]

Return the status executed by the Find command.

[Syntax]

PosFound

[Parameter Description]

Notes:

1. Returns True if Find is valid during motion; otherwise, returns False.

[Usage Example]

```
Function main

    Go P10 Find Sw(5) = On 'Save coordinates when DI5 is set to On during motion

    If PosFound Then
        Go FindPos          'Retrieve the coordinates saved by FindPos
    Else
        Print "DI5 was not set to ON during motion"
    EndIf
Fend
```

Pulse

[Function]

Move the robotic arm to the specified joint pulse values using PTP motion, or return the point corresponding to the pulse values.

[Syntax]

Pulse {Joint 1Pulse Value, Joint 2Pulse Value, Joint 3Pulse Value, Joint 4Pulse Value{, Joint 5Pulse Value, ...}}

Pulse(Joint 1Pulse Value, Joint 2Pulse Value, Joint 3Pulse Value, Joint 4Pulse Value{, Joint 5Pulse Value, ...})

[Parameter Description]

Joint Pulse Value Enter the pulse values for each joint in order. Additional parameters can be added based on the actual number of robot axes.

[Usage Example]

```
Function main
    Pulse 123456 , 222333 , 333444 , 111111
    'Move to the specified pulse position

    P1 = Pulse(123456 , 222333 , 333444 , 111111)
    'Save the position corresponding to the specified pulses into P1

    Pulse 'Print the current pulse values of all axes
Fend
```

RobLoad

[Function]

Select or return the specified load parameters.

[Syntax]

RobLoad {Load Number}

[Parameter Description]

Load Number Specify the load to select using an integer from 1 to 64.

[Usage Example]

```
Function main
    RobLoad 3
    Go P1

    If RobLoad = 0 Then
        RobLoad 1
    EndIf
Fend
```

Sense

[Function]

Set/display the conditions for Jump, Jump3, and Jump3CP to stop above the target point.

[Syntax]

Sense {condition_expression}

[Parameter Description]

condition_expression	The condition for stopping above the target point during motion. Various I/O instructions, operators, etc., can be used.

[Usage Example]

```
Function main
    Sense Sw(5) = Off
    Jump P1 C2 Sense
    'Stop above the target point when DI5 is Off

    If JS = True Then
        Print "The robot has stopped above the target point"
    EndIf
    On 1; Wait 0.2; Off 1
Fend
```

TillOn

[Function]

Return the status of Till.

[Syntax]

TillOn

[Parameter Description]

Notes:

1. Returns True if the Till condition is met during motion; otherwise, returns False.

[Usage Example]

```
Function main
    Go P1 Till Sw(1) = On

    If TillOn Then
        Print "Motion has stopped because DI1 is set to On"
    EndIf
Fend
```

WaitPos

[Function]

Wait for the robot to decelerate and stop after a smooth motion command before passing control to the subsequent program.

[Syntax]

WaitPos

[Parameter Description]

Notes:

1. Usually, when smooth motion is effective (CP On or motion command with CP), the action command will hand over control to the subsequent program as it begins to decelerate. If you do not want to hand over control early, you can insert WaitPos, which will ensure that it completes the deceleration before handing over.

[Usage Example]

```
Function main
    Off 1
    CP On
    Move P1
    Move P2
    WaitPos          ' Wait for the robot to decelerate
    On 1
    CP Off
Fend
```

XY

[Function]

Return the point based on the specified data.

[Syntax]

XY(X coordinate value, Y coordinate value, Z coordinate value, U coordinate value{, V coordinate value, W coordinate value})

[Parameter Description]

coordinate value	Fill in the coordinates of each axis in order. Additional V-axis and W-axis coordinate values can be added as appropriate based on the actual robot model. Units: mm.

[Usage Example]

```
Function main
    P10 = XY(60 , 30 , -50 , 45) + P20
    'Generate P10 point by adding specified offset to index P20 point data

    P10 = XY(60 , 30 , -50 , 45) /L
    'Create P10 point with the specified coordinates and tool coordinate system

    Go XY(10 , 20 , 30 , 40)
    'Arc move to the specified coordinate position

Fend
```

CX、CY、CZ、CU、CV、CW、CR、CS、CT

[Function]

Set (modify) the coordinates of a point.

[Syntax]

CX(Point) {= Coordinate Value}

CY(Point) {= Coordinate Value}

CZ(Point) {= Coordinate Value}

....

[Parameter Description]

Point Point number or point name.

Coordinate Value The coordinate value to be set, units: mm.

[Usage Example]

```
Function main
    Double New_Y

    CX(P1) = 123                ' Change the X coordinate of P1 to 123mm
    CU(P1) = 321                ' Change the U coordinate of P1 to 321deg

    If CY(P1) > 200 Then
        CY(P1) = 200            ' Force P1's Y coordinate to equal 200 if it exceeds 200
    EndIf

    P1 = Here                   ' Get the current position and assign it to P1
    Print CX(P1)                ' Print the current X coordinate
    New_Y = CY(P1)              ' Assign the current Y coordinate to New_Y variable
Fend
```

TargetOK

[Function]

Set (modify) collision detection coefficients.

[Syntax]

TargetOK(Point)

....

[Parameter Description]

Point Point number or point name.

[Usage Example]

```
Function main
    If TargetOK(P1) = False Then
        Print " This point cannot be reached"
    ElseIf TargetOK(P1) = True Then
        Go P1
    EndIf
Fend
```

ColSup

[Function]

Set (modify) the collision detection coefficient.

[Syntax]

ColSup On/Off [,level]

ColSup

....

[Parameter Description]

On/Off Whether to set the collision detection coefficient.

level The value of the collision detection coefficient to be set, valid range: 1 - 300.

[Usage Example]

```
Function main
    ColSup On,50        'Set the collision detection coefficient
    ColSup Off          ' Disable collision detection settings
    Print ColSup        'Print the current collision detection coefficient
Fend
```

ZeroPls

[Function]

Set (modify) the robot's zero position.

[Syntax]

ZeroPls Pulse1 , Pulse2 , Pulse3 [, Pulse4 , Pulse5 , Pulse6]

ZeroPls

....

[Parameter Description]

Pulse1 , Pulse2 , Pulse3 [, Pulse4 , Pulse5 , Pulse6]	The pulse values for each axis that require zero position settings.

[Usage Example]

```
Function main

    ZeroPls 0 , 0  , 0  , 0
    'Set the zero position of the XYZU axes to pulse value 0

    Print ZeroPls
    ' Print the current zero position

Fend
```

Hand

[Function]

Set (modify) the tool coordinate system.

[Syntax]

Hand [Point , HandCoordinateSystem]
Hand

....

[Parameter Description]

Point	Point number or point name.
HandCoordinateSystem	Modify the tool coordinate system of the point; "Lefty" for left-hand system, "Righty" for right-hand system. If left blank, return the tool coordinate system of the point.

[Usage Example]

```
Function main

    Print Hand(P0)          'Print the tool coordinate system of point P0

    Hand P0,Righty          'Set point P0 to right-hand coordinate system

    Print Hand              ' Print the current tool coordinate system

Fend
```

Elbow

[Function]

Set (modify) the elbow posture.

[Syntax]

Elbow [Point , Posture]
Elbow
....

[Parameter Description]

Point Point number or point name.

Posture The elbow posture of the point; "Above" for elbow facing up, "Below" for elbow facing down.

[Usage Example]

```
Function main
    Print Elbow(P0)          'Print the posture of point P0
    Elbow P0 , Above         'Set the posture of point P0, elbow facing up
    Print Elbow              ' Print the current posture
Fend
```

Wrist

[Function]

Set (modify) the wrist posture.

[Syntax]

```
Wrist  [Print , Posture]
Wrist
....
```

[Parameter Description]

Point	Point number or point name.
Posture	The wrist posture of the point; "Flip" for wrist flipped, "NoFlip" for wrist not flipped.

[Usage Example]

```
Function main
    Print Wrist(P0)                 'Print the wrist posture of point P0
    Wrist P0 , NoFlip               'Set the posture of point P0, wrist not flipped
    Print Wrist                     ' Print the current posture
Fend
```

PDef

[Function]

Return whether the point has been defined.

[Syntax]

PDef (Point Number)

PDef (P Point Number)

PDef (P (Point Number))

PDef (Label Name)

[Parameter Description]

Point Number	Indexed by the order number in the point file.
P Point Number	Indexed by the order number in the point file.
P(Point Number)	Indexed by the order number in the point file.
Label Name	After assigning a label variable, indexed by the order number in the point file based on the variable value.

[Usage Example]

```
Function main

    Print "0:" , PDef(0)                'Print whether point P0 is defined
    Print "P1:" , PDef(P1)              'Print whether point P1 is defined

    Print "P(1):" , PDef(P(1))          'Print whether point P1 is defined

    Integer Point_Num
    Point_Num =  0
    Print "Point_Num:" , PDef(Point_Num) 'Print whether the corresponding point is defined
Fend
```

[Return Description]

Return Data1 TRUE / FALSE, TRUE means the point is defined, FALSE means the point is not defined.

PDel

[Function]

Delete a point.

[Syntax]

PDel Point Number

PDel Starting Point Number , Ending Point Number

[Parameter Description]

Point Number	Indexed by the order number in the point file.
Starting Point Number	Start deleting points from the specified number.
Ending Point Number	End deleting points at the specified number.

[Usage Example]

```
Function main

    PDel 1                  ' Delete the information for point P1

    PDel 1 , 5              ' Delete the information for points P1 - P5

    SavePoints "robot1"     ' Save to the point table

Fend
```

PLabel

[Function]

Set a label for the specified point data.

[Syntax]

PLabel Point Number, New Label

[Parameter Description]

Point Number	Indexed by the order number in the point file.
New Label	Assign a new label to the specified point number (Chinese characters are not supported).

[Usage Example]

```
Function main

    PLabel 1 , "New_Point"      'Assign a new label to point P1

    Go P(New_Point)

    SavePoints "robot1"         ' Save to the point file

Fend
```

PLabel$

[Function]

Return the label of the specified point data.

[Syntax]

PLabel$ (Point Number)

PLabel$ (P Point Number)

PLabel$ (P(Point Number))

[Parameter Description]

Point Number	Indexed by the order number in the point file.
P Point Number	Indexed by the order number in the point file.
P(Point Number)	Indexed by the order number in the point file.

Notes:

1. If this instruction indexes an empty point or the indexed point does not have a label, no output will be produced.

[Usage Example]

```
Function main

    Print PLabel$(1)                'Print the label of point 1

    Print PLabel$(P1)               'Print the label of point 1

    Print PLabel$(P(1))             'Print the label of point 1

    If PLabel$(2) = "" Then         'Check if the label of point 2 is empty
        Print "There is no label for this point"
    EndIf
Fend
```

PNumber

[Function]

Return the point number for the specified label name.

[Syntax]

PNumber (Label Name)

PNumber ("Label Name")

[Parameter Description]

Label Name	One of the items in the point data, which is a label.

[Usage Example]

```
Function main

    Print PNumber(EA)           ' Print the point number with label EA
    Print PNumber("AA")         ' Print the point number with label AA

Fend
```

PList

[Function]

Display the specified point data.

[Syntax]

PList

PList Point Number

PList Starting Point Number , Ending Point Number

[Parameter Description]

Point Number	Indexed by the order number in the point file.
P Point Number	Indexed by the order number in the point file.
P(Point Number)	Indexed by the order number in the point file.

[Usage Example]

```
Function main

    PList               ' Print all point data

    PList 1             ' Print data for point 1

    PList 2 , 10        ' Print data from point 2 to point 10

Fend
```

PLocal

[Function]

Specify / Get the user coordinate information of a point.

[Syntax]

```
PLocal  (Point Number)
PLocal  (Point Number) = Value
```

[Parameter Description]

Point Number	Indexed by the order number in the point file.
Value	The specified user coordinate system.

[Usage Example]

```
Function main

    Print PLocal(1)          ' Print the user coordinate of point 1

    PLocal(1) = 1            ' Set the user coordinate of point 1

Fend
```

PDescription

[Function]

Set the description for the specified point.

[Syntax]

PDescription Point Number , New Description

[Parameter Description]

Point Number	Indexed by the order number in the point file.
New Description	The description data displayed in the point file.

[Usage Example]

```
Function main

    PDescription 1 , " New Description "        ' Change the description of point P1 to " New
Description "
    Print PDescription$(1)              ' Print the description of point P1

Fend
```

PDescription$

[Function]

Return the description of the specified point.

[Syntax]

PDescription$ (Point Number)

PDescription$ (P Point Number)

PDescription$ (P (Point Number))

PDescription$ (Label Name)

[Parameter Description]

Point Number	Indexed by the order number in the point file.
P Point Number	Indexed by the order number in the point file.
P(Point Number)	Indexed by the order number in the point file.
Label Name	After assigning a label variable, indexed by the order number in the point file based on the variable value.

[Usage Example]

```
Function main

    PDescription 1 , "New Description"  ' Change the description of point P1 to "New Description"

    Print PDescription$(1)          ' Print the description of point P1

    Print PDescription$(P1)         ' Print the description of point P1

    Print PDescription$(P(1))       ' Print the description of point P1

Fend
```

I/O Instruction Detailed Description

Wait

[Function]

Delay instruction or wait for a condition to be true for a specified time.

[Syntax]

```
Wait Time
Wait condition_expression
Wait condition_expression, Time
```

[Parameter Description]

Time	Specify the delay time with a number between 0 and 2147483. Unit: seconds. Minimum delay is 0.01 seconds.
condition_expression	The condition for stopping at the target point during motion. Various I/O instructions, operators, etc., can be used.

[Usage Example]

```
Function main

    ' Wait for input 0 to become On
    Wait Sw(0) = On

    'Wait for 60.5 seconds before continuing
    Wait 60.5

    'Wait for input 0 to become Off and input 1 to become On
    Wait Sw(0) = Off And Sw(1) = On

    ' Wait for register bit 0 to become On or register bit 1 to become On
    Wait MemSw(0) = On Or MemSw(1) = On

    ' Wait for 1 second, then set output 1 to On
    Wait 1; On 1

Fend
```

Mask

[Function]

Used to perform a bitwise AND operation on the values representing the Wait input conditions.

[Syntax]

```
Wait condition_expression Mask Max Bit  = Condition Return Value
```

[Parameter Description]

Wait	Delay instruction or wait for a condition to be true for a specified time.
condition_expression	The specified DIO bit or group, register address, or bit.
Mask	Used to perform a bitwise AND operation on the values representing the Wait input conditions.
Max Bit	The maximum value of the indexed bit.
Condition Return Value	The target value that satisfies the current condition expression.

[Usage Example]

```
Function main
    Do
        Wait InW(0) Mask 1 = 0
        'Wait for the return value of the first bit of the 16-bit DI group 0 to be 0 to meet the exit condition

        Wait InW(0) Mask 3 = 2
        'Wait for the return value of the first 2 bit of the 16-bit DI group 0 to be 2 to meet the exit condition

        Wait InW(0) Mask 7 = 4

        'Wait for the return value of the first 3 bits of the 16-bit DI group 0 to be 4 to meet the exit condition


        Wait InW(0) Mask 15 = 8

        'Wait for the return value of the first 4 bits of the 16-bit DI group 0 to be 8 to meet the exit condition
```

```
    Wait InW(0) Mask 31 = 16

    'Wait for the return value of the first 5 bits of the 16-bit DI group 0 to be 16 to meet the exit condition


    Wait InW(0) Mask 63 = 32
    ' Wait for the return value of the first 6 bits of the 16-bit DI group 0 to be 32 to meet the exit condition
    Loop
Fend
```

On

[Function]

Control DO to be On, or turn Off after a certain time

[Syntax]

On DO_Number/DO_Label {, Time}

[Parameter Description]

DO_Number/DO_Label	Specify the DO number as an integer or use the DO label.
Time	Specify the duration (in seconds) for which the DO remains On. It will automatically turn Off after this duration.

[Usage Example]

```
Function main

    On 1                ' Set DO1 to On
    On 3                ' Set DO3 to On

    On 5 , 1            ' Set DO5 to On, turn Off after 1 second

Fend
```

Off

[Function]

Control DO to be Off, or turn On after a certain time

[Syntax]

```
On DO_Number/DO_Label {, Time}
```

[Parameter Description]

DO_Number/DO_Label	Specify the DO number as an integer or use the DO label.
Time	Specify the duration (in seconds) for which the DO remains On. It will automatically turn Off after this duration.

[Usage Example]

```
Function main

    Off 1               ' Set DO1 to Off
    Off 3               ' Set DO3 to Off

    Off 5 , 1           ' Set DO5 to Off, turn On after 1 second

Fend
```

Oport

[Function]

Return the Status of Specified DO

[Syntax]

```
Oport(DO_Number)
```

[Parameter Description]

DO_Number Specify the DO whose status needs to be retrieved as an integer.

Notes:

1. Returns 1 if the DO is On.
2. Returns 0 if the DO is Off.

[Usage Example]

```
Function main

    If Oport(1) = 0 Then
            On 1
    EndIf

    Wait Oport(3)
Fend
```

Sw

[Function]

Return the Status of Specified DI

[Syntax]

```
Sw(DI_Number)
```

[Parameter Description]

DI_Number Specify the DI whose status needs to be retrieved as an integer.

Notes:

1. Returns 1 if the DO is On.
2. Returns 0 if the DO is Off.

[Usage Example]

```
Function main

    If Sw(1) = 0 Then
        SetSw 1 ,  On
    EndIf

    Wait Sw(1)
Fend
```

SetSw

[Function]

Set Specified DI to On or Off

[Syntax]

```
SetSw DI_Number, State
```

[Parameter Description]

DI_Number Specify the DI to control as an integer.

State Specify the state, 0 for Off, 1 for On. You can also use "Off" or "On" as inputs.

[Usage Example]

```
Function main

    SetSw 1 , On
    SetSw 4 , Off


Fend
```

In

[Function]

Return the Status of 8-bit DI Based on 8421, 0~255

[Syntax]

```
In(DI_Group_Number)
```

[Parameter Description]

DI_Group_Number Specify the group of DIs whose status needs to be read as an integer.

Notes:

1. Specify the group of DIs whose status needs to be read as an integer.
2. The principle diagram of 8421 is as follows:

Return Value	7	6	5	4	3	2	1	0
1	Off	Off	Off	Off	Off	Off	Off	On
5	Off	Off	Off	Off	Off	On	Off	On
15	Off	Off	Off	Off	On	On	On	On
255	On	On	On	On	On	On	On	On

[Usage Example]

```
Function main

    Print In(0)

    Integer A1
    A1 =  In(1)

    Select A1
    Case 0
        Go P1

    Case 8
        Go P2

    Default
        Go P3

    Send
Fend
```

SetIn

[Function]

Set the Status of 8-bit DI Based on 8421, 0~255

[Syntax]

```
SetIn DI_GROUP_NUMBER, Status_Value
```

[Parameter Description]

DI_GROUP_NUMBER Specify the group of DIs whose status needs to be set as an integer.

Status_Value Set the status of the group of DIs according to the 8421 representation.

Notes:

1. When DI_Group_Number is 0, it corresponds to DI0~DI7. When 1, it corresponds to DI8~DI15, and so on.
2. The principle diagram of 8421 is as follows:

Return Value	7	6	5	4	3	2	1	0
1	Off	Off	Off	Off	Off	Off	Off	On
5	Off	Off	Off	Off	Off	On	Off	On
15	Off	Off	Off	Off	On	On	On	On
255	On	On	On	On	On	On	On	On

[Usage Example]

```
Function main
    SetIn 0 ,85              'Set DI0, DI2, DI4, DI6 to On
Fend
```

Out

[Function]

Set the Status of 8-bit DO Based on 8421, 0~255

[Syntax]

```
Out DO_GROUP_NUMBER, Status_Value
Out(DO_GROUP_NUMBER)
```

[Parameter Description]

DO_Group_Number Specify the group of DOs whose status needs to be set or retrieved as an integer.

Status_Value Set the status of the group of DOs according to the 8421 representation.

Notes:

1. When DO_Group_Number is 0, it corresponds to DO0~DO7. When 1, it corresponds to DO8~DO15, and so on.

2. The principle diagram of 8421 is as follows:

Return Value	7	6	5	4	3	2	1	0
1	Off	Off	Off	Off	Off	Off	Off	On
5	Off	Off	Off	Off	Off	On	Off	On
15	Off	Off	Off	Off	On	On	On	On
255	On	On	On	On	On	On	On	On

[Usage Example]

```
Function main
    Out 0 ,85                   'Set DO0, DO2, DO4, DO6 to On

    If Out(0) = 0 Then          ' Read the status of DO0 to DO7.
        Out 0 ,85
    EndIf
Fend
```

InW

[Function]

Return the Status of 16-bit DI Based on 8421, 0~65535

[Syntax]

InW(DI_GROUP_NUMBER)

[Parameter Description]

DI_GROUP_NUMBER Specify the group of DIs whose status needs to be read as an integer.

Notes:

1. When DI_Group_Number is 0, it returns the status of DI0~DI15. When 1, it returns the status of DI16~DI31, and so on.

2. The principle diagram of 8421 is as follows:

Return Value	7	6	5	4	3	2	1	0
1	Off	Off	Off	Off	Off	Off	Off	On
5	Off	Off	Off	Off	Off	On	Off	On
15	Off	Off	Off	Off	On	On	On	On
255	On	On	On	On	On	On	On	On

[Usage Example]

```
Function main

    Print InW(0)

    Integer A1
    A1 =  InW(1)

    Select A1
    Case 0
        Go P1

    Case 8
        Go P2

    Default
        Go P3

    Send
Fend
```

SetInW

[Function]

Set the Status of 16-bit DI Based on 8421, 0~65535

[Syntax]

```
SetInW DI_GROUP_NUMBER , Status_Value
```

[Parameter Description]

DI_GROUP_NUMBER Specify the group of DIs whose status needs to be set as an integer.

Status_Value Set the status of the group of DIs according to the 8421 representation.

Notes:

1. When DI_GROUP_NUMBER is 0, it corresponds to DI0~DI15. When 1, it corresponds to DI16~DI31, and so on.

2. The principle diagram of 8421 is as follows:

Return Value	7	6	5	4	3	2	1	0
1	Off	Off	Off	Off	Off	Off	Off	On
5	Off	Off	Off	Off	Off	On	Off	On
15	Off	Off	Off	Off	On	On	On	On
255	On	On	On	On	On	On	On	On

[Usage Example]

```
Function main
    SetInW 0 ,85            'Set DI0, DI2, DI4, DI6 to On
Fend
```

OutW

[Function]

Set/Return the Status of 16-bit DO Based on 8421, 0~65535

[Syntax]

```
OutW DO_GROUP_NUMBER, Status_Value
OutW(DO_GROUP_NUMBER)
```

[Parameter Description]

DO_Group_Number Specify the group of DOs whose status needs to be set or retrieved as an integer.

Status_Value Set the status of the group of DOs according to the 8421 representation.

Notes:

1. When DO_GROUP_NUMBER is 0, it corresponds to DO0~DO15. When 1, it corresponds to DO16~DO31, and so on.

2. The principle diagram of 8421 is as follows:

Return Value	7	6	5	4	3	2	1	0
1	Off	Off	Off	Off	Off	Off	Off	On
5	Off	Off	Off	Off	Off	On	Off	On
15	Off	Off	Off	Off	On	On	On	On
255	On	On	On	On	On	On	On	On

[Usage Example]

```
Function main
    OutW 0 ,85                  'Set DO0, DO2, DO4, DO6 to On

    If OutW(0) = 0 Then         ' Read the status of DO0~DO15
        OutW 0 ,85
    EndIf
Fend
```

InBCD

[Function]

Return the Status of 8-bit DI Based on 8421, 0~99

[Syntax]

InBCD(DI_GROUP_NUMBER)

[Parameter Description]

DI_GROUP_NUMBER Specify the group of DIs whose status needs to be read as an integer.

Notes:

1. When DI_GROUP_NUMBER is 0, it returns the status of DI0~DI7. When 1, it returns the status of DI8~DI15, and so on.
2. The difference between this function and In is the range of values. In has a range of 0~255.
3. The principle diagram of 8421 is as follows:

Return Value	7	6	5	4	3	2	1	0
1	Off	Off	Off	Off	Off	Off	Off	On
5	Off	Off	Off	Off	Off	On	Off	On
15	Off	Off	Off	Off	On	On	On	On
255	On	On	On	On	On	On	On	On

[Usage Example]

```
Function main

    Print InBCD(0)

    Integer A1
    A1 =  InBCD(1)

    Select A1
    Case 0
        Go P1

    Case 8
        Go P2

    Default
        Go P3

    Send
Fend
```

OpBCD

[Function]

Set the Status of 8-bit DO Based on 8421, 0~99

[Syntax]

```
OpBCD DO_GROUP_NUMBER, Status_Value
```

[Parameter Description]

DO_GROUP_NUMBER Specify the group of DOs whose status needs to be set or retrieved as an integer.

Status_Value Set the status of the group of DOs according to the 8421 representation.

Notes:

1. When DO_GROUP_NUMBER is 0, it corresponds to DO0~DO7. When 1, it corresponds to DO8~DO15, and so on.
2. The difference between this function and Out is the range of values. Out has a range of 0~255.
3. The principle diagram of 8421 is as follows:

Return Value	7	6	5	4	3	2	1	0
1	Off	Off	Off	Off	Off	Off	Off	On
5	Off	Off	Off	Off	Off	On	Off	On
15	Off	Off	Off	Off	On	On	On	On
255	On	On	On	On	On	On	On	On

[Usage Example]

```
Function main
    OpBCD 0 ,85                    'Set DO0, DO2, DO4, DO6 to On
Fend
```

InReal

[Function]

Return DI Status as a 32-bit Floating Point Number

[Syntax]

```
InReal(DI_GROUP_NUMBER)
```

[Parameter Description]

DI_GROUP_NUMBER Specify the group of DOs whose status needs to be set or retrieved as an integer.

Notes:

1. When DI_GROUP_NUMBER is 0, it corresponds to DI0~DI31. When 1, it corresponds to DI32~DI63, and so on.

[Usage Example]

```
Function main
    Real realVal

    realVal =  InReal(32)
Fend
```

OutReal

[Function]

Set/Return DO Status as a 32-bit Floating Point Number

[Syntax]

```
OutReal DO_GROUP_NUMBER, Status_Value
OutReal(DO_GROUP_NUMBER)
```

[Parameter Description]

DO_Group_Number	Specify the group of DOs whose status needs to be set or retrieved as an integer.
Status_Value	Set the status of the group of DOs according to the 32-bit floating point representation.

Notes:

1. When DO_GROUP_NUMBER is 0, it corresponds to DO0~DO31. When 1, it corresponds to DO32~DO63, and so on.

[Usage Example]

```
Function main
    OutReal 32 , 2.543
Fend
```

AIO_InW

[Function]

Read the Value of Analog Input (AI)

[Syntax]

```
AIO_InW(AI_NUMBER)
```

[Parameter Description]

AI_NUMBER	Specify the Analog Input to read its value as an integer.

[Usage Example]

```
Function main
    If AIO_InW(1) > 123456 Then
        Print " Analog value greater than 123456"
    EndIf
Fend
```

AIO_OutW

[Function]

Set/Read the Value of Analog Output (AO)

[Syntax]

```
AIO_OutW AI_NUMBER, Analog_Value
AIO_OutW(AI_NUMBER)
```

[Parameter Description]

AI_NUMBER	Specify the Analog Output to set or read its value as an integer.
Analog_Value	Set the value for the specified Analog Output.

[Usage Example]

```
Function main
    If AIO_OutW(1) > 123456 Then
        AIO_OutW 1 , 0          ' Set AO1 to zero if the analog value is greater than 123456
    EndIf
Fend
```

MemOn

[Function]

Set a specific bit in the user register to 1

[Syntax]

```
MemOn bit_position
```

[Parameter Description]

bit_position Specify which bit to set to 1.

Notes:

1. A user register is 16 bits, so bits 0~15 correspond to the first register, bits 16~31 correspond to the second register, and so on.

[Usage Example]

```
Function main
    MemOn 3         ' Set the 3rd bit of the first register to 1
    MemOn 16        ' Set the 0th bit of the second register to 1
Fend
```

MemOff

[Function]

Set a specific bit in the user register to 0

[Syntax]

```
MemOff bit_position
```

[Parameter Description]

bit_position Specify which bit to set to 0.

Notes:

1. A user register is 16 bits, so bits 0~15 correspond to the first register, bits 16~31 correspond to the second register, and so on.

[Usage Example]

```
Function main
    MemOff 3            ' Set the 3rd bit of the first register to 0
    MemOff 16           ' Set the 0th bit of the second register to 0
Fend
```

MemSw

[Function]

Get the value of a specific bit in the user register

[Syntax]

MemSw(bit_position)

[Parameter Description]

bit_position Specify which bit to read.

Notes:

1. A user register is 16 bits, so bits 0~15 correspond to the first register, bits 16~31 correspond to the second register, and so on.

[Usage Example]

```
Function main

    If MemSw(16) = 1 Then               ' Get the value of the 0th bit of the second register
        Go P1
    Else
        Go P2
    EndIf

Fend
```

MemIn

[Function]

Get the value of 8 bits from the user register

[Syntax]

MemIn(bit_group_number, {Signed/Unsigned})

[Parameter Description]

bit_group_number Specify which group of bits to read.

Signed/Unsigned Read as a signed or unsigned value. The default is unsigned.

Notes:

1. A user register is 16 bits, so groups 0~1 correspond to the low and high bytes of the first register, groups 2~3 correspond to the low and high bytes of the second register, and so on.

[Usage Example]

```
Function main

    Print MemIn(2)
    ' Get the low byte of the first register

    Print MemIn(1 , Signal)
    ' Get the high byte of the first register as a signed value

Fend
```

MemOut

[Function]

Set the Value of 8 Bits in the User Register

[Syntax]

```
MemOut bit_group_number, value
```

[Parameter Description]

bit_group_number Specify which group of bits to read.

value The value to be assigned to the register.

Notes:

1. A user register is 16 bits, so groups 0~1 correspond to the low and high bytes of the first register, groups 2~3 correspond to the low and high bytes of the second register, and so on.

[Usage Example]

```
Function main
    MemOut 1 , 123
    'Set the high byte of the first register to 123
Fend
```

MemInW

[Function]

Get the Value of 16 Bits in the User Register

[Syntax]

MemInW(register_number, {Signed/Unsigned})

[Parameter Description]

register_number Specify which register to read.

Signed/Unsigned Read as a signed or unsigned value. The default is unsigned.

Notes:

1. A user register is 16 bits, so 0 refers to the first register, 1 refers to the second register, and so on.

[Usage Example]

```
Function main

    Print MemInW(1)
    ' Get the value of the first register

    Print MemInW(2 , Signal)
    ' Get the high byte of the second register as a signed value

Fend
```

MemOutW

[Function]

Set the Value of 16 Bits in the User Register

[Syntax]

```
MemOutW register_number, value
```

[Parameter Description]

register_number Specify which register to read.

value The value to be assigned to the register.

Notes:

1. A user register is 16 bits, so 0 refers to the first register, 1 refers to the second register, and so on.

[Usage Example]

```
Function main
    MemOutW 5 , 1234
    ' Set the value of the fifth register to 1234
Fend
```

MemIn32

[Function]

Get the Value of 32 Bits in the User Register

[Syntax]

MemIn32(register_group, {Signed/Unsigned})

[Parameter Description]

register_group	Specify which group of registers to read.
value	The value to be assigned to the register.

Notes:

1. A user register is 16 bits, so 0 refers to the combined 32-bit value of the first and second registers.

1 refers to the combined 32-bit value of the third and fourth registers, and so on.

[Usage Example]

```
Function main

    Print MemIn32(0)
    ' Get the combined 32-bit value of the first and second registers

    Print MemIn32(1 , Signal)
    ' Get the combined 32-bit value of the third and fourth registers as a signed value

Fend
```

MemOut32

[Function]

Set the Value of 32 Bits in the User Register

[Syntax]

```
MemOut32 register_group, value
```

[Parameter Description]

register_group	Specify which group of registers to set.
value	The value to be assigned to the registers.

Notes:

1. A user register is 16 bits, so 0 refers to the combined 32-bit value of the first and second registers.

1 refers to the combined 32-bit value of the third and fourth registers, and so on.

[Usage Example]

```
Function main
    MemOut32 1 , 1234
    ' Set the combined 32-bit value of the second and third registers to 1234
Fend
```

MemInReal

[Function]

Get the Value of 32 Bits in the User Register as a 32-bit Floating Point Number

[Syntax]

```
MemInReal(register_group)
```

[Parameter Description]

register_group	Specify which group of registers to set.

Notes:

1. A user register is 16 bits, so 0 refers to the combined 32-bit value of the first and second registers.

1 refers to the combined 32-bit value of the third and fourth registers, and so on.

[Usage Example]

```
Function main

    Print MemInReal(0)
    ' Get the combined 32-bit value of the first and second registers as a 32-bit floating point number

Fend
```

MemOutReal

[Function]

Set the Value of 32 Bits in the User Register as a 32-bit Floating Point Number

[Syntax]

```
MemOutReal register_group, value
```

[Parameter Description]

register_group	Specify which group of registers to set.
value	The value to be assigned to the registers.

Notes:

1. A user register is 16 bits, so 0 refers to the combined 32-bit value of the first and second registers.

1 refers to the combined 32-bit value of the third and fourth registers, and so on.

[Usage Example]

```
Function main
    MemOutReal 1 , 123.321
    ' Set the combined 32-bit value of the second and third registers to 123.321
Fend
```

SysMemOn

[Function]

Set a Specific Bit in the System Register to 1

[Syntax]

```
SysMemOn bit_group_number
```

[Parameter Description]

bit_group_number	Specify which bit to set to 1.

Notes:

1. A system register is 16 bits, so 0 to 15 bits is the 0th register. 16 to 31 bits is the first register, and so on.

[Usage Example]

```
Function main
    SysMemOn 0          ' Set the 1st bit of the first system register to On
    SysMemOn 15         ' Set the 16st bit of the first system register to On
    SysMemOn 16         ' Set the 1st bit of the second system register to On
Fend
```

SysMemOff

[Function]

Set a Specific Bit in the System Register to 0

[Syntax]

```
SysMemOff bit_group_number
```

[Parameter Description]

bit_group_number Specify which bit to set to 0.

Notes:

1. A system register is 16 bits, so bits 0~15 correspond to the first register, bits 16~31 correspond to the second register, and so on.

[Usage Example]

```
Function main
    SysMemOff 0             'Set the 1st bit of the first system register to Off
    SysMemOff 15            'Set the 16th bit of the first system register to Off
    SysMemOff 16            'Set the 1st bit of the second system register to Off
Fend
```

SysMemSw

[Function]

Get the Value of a Specific Bit in the System Register

[Syntax]

```
SysMemSw bit_group_number
```

[Parameter Description]

bit_group_number	Specify which group of bits to read.

Notes:

1. A system register is 16 bits, so bits 0~15 correspond to the first register, bits 16~31 correspond to the second register, and so on.

[Usage Example]

```
Function main

    If SysMemSw(0) = On Then                'Condition is true when the 1st bit of the first
                                            system register is On
        Go P1

    ElseIf SysMemSw(15) = On Then           ' Condition is true when the 16th bit of the first
                                            system register is On
        Go P2
    EndIf

Fend
```

SysMemIn

[Function]

Get the Value of 8 Bits in the System Register

[Syntax]

SysMemIn(bit_group_number, {Signed/Unsigned})

[Parameter Description]

bit_group_number	Specify which group of bits to read.
Signed/Unsigned	Read as a signed or unsigned value. The default is unsigned.

Notes:

A system register is 16 bits, so groups 0~1 correspond to the low and high bytes of the first register, groups 2~3 correspond to the low and high bytes of the second register, and so on.

[Usage Example]

```
Function main

    Print SysMemIn(0)
    'Print the low byte of the first register

    Print SysMemIn(1 , Unsigned)
    ' Print the high byte of the first register as an unsigned value

Fend
```

SysMemOut

[Function]

Set the Value of 8 Bits i

n the System Register

[Syntax]

SysMemOut bit_group_number, value

[Parameter Description]

bit_group_number	Specify which group of bits to read.
value	The value to be assigned to the registers.

Notes:

A system register is 16 bits, so groups 0~1 correspond to the low and high bytes of the first register, groups 2~3 correspond to the low and high bytes of the second register, and so on.

[Usage Example]

```
Function main
    SysMemOut 1 , 123
    ' Set the high byte of the first register to 123
Fend
```

SysMemInW

[Function]

Get the Value of 16 Bits in the System Register

[Syntax]

SysMemInW register_number, {Signed/Unsigned}])

[Parameter Description]

register_number Specify which register to read.

Signed/Unsigned Read as a signed or unsigned value. The default is unsigned.

Notes:

1. A system register is 16 bits, so 0 refers to the first register, 1 refers to the second register, and so on.

[Usage Example]

```
Function main

    Print SysMemInW(0,Unsigned)
    ' Print the first system register in unsigned integer format

    Print SysMemInW(1 , Signal)
    ' Print the second system register as a signed value

Fend
```

SysMemOutW

[Function]

Set the 16-bit value of a system register.

[Syntax]

```
SysMemOutW register_number, value
```

[Parameter Description]

register_number	Specify which register to read.
value	The value to be assigned to the registers.

Notes:

1. A system register is 16 bits, so 0 is the 0th register. 1 is the 1st register, and so on.

[Usage Example]

```
Function main
    SysMemOutW 0 , 123
    'Write the value 123 to the address of the 0th register
Fend
```

SysMemIn32

[Function]

Get the 32-bit value of a system register.

[Syntax]

SysMemIn32 (register_group, {Signed/Unsigned})

[Parameter Description]

register_group	Specify which group of registers to set.
Signed/Unsigned	Read as a signed or unsigned value. The default is unsigned.

Notes:

A system register is 16 bits, so 0 is the merged 32-bit value of the 0th and 1st registers. 1 is the merged 32-bit value of the 2nd and 3rd registers, and so on.

[Usage Example]

```
Function main

    Print SysMemIn32(0)
    ' Get the merged 32-bit value of the 0th and 1st registers

    Print SysMemIn32(1 , Unsigned)
    'Get the merged 32-bit value of the 2nd and 3rd registers as an unsigned value

Fend
```

SysMemOut32

[Function]

Set the 32-bit value of a system register.

[Syntax]

SysMemOut32register_group, value

[Parameter Description]

register_group	Specify which group of registers to set.
value	The value to be assigned to the registers.

Notes:

A system register is 16 bits, so 0 is the merged 32-bit value of the 0th and 1st registers. 1 is the merged 32-bit value of the 2nd and 3rd registers, and so on.

[Usage Example]

```
Function main
    SysMemOut32 1 , 1234
    'Set the merged 32-bit value of the 2nd and 3rd system registers to 1234
Fend
```

SysMemInReal

[Function]

Get the 32-bit value of a system register in the form of a 32-bit floating-point number.

[Syntax]

SysMemInReal (register_group)

[Parameter Description]

register_group Specify which group of registers to set.

Notes:

A system register is 16 bits, so 0 is the merged 32-bit value of the 0th and 1st registers. 1 is the merged 32-bit value of the 2nd and 3rd registers, and so on.

[Usage Example]

```
Function main

    Print SysMemInReal(0)
    'Get the merged 32-bit value of the 0th and 1st registers as a 32-bit floating-point number

Fend
```

SysMemOutReal

[Function]

Set the 32-bit value in the system register in the form of a 32-bit floating-point number.

[Syntax]

```
SysMemOutReal register_group, value
```

[Parameter Description]

register_group	Specify which group of registers to set.
value	The value to be assigned to the registers.

Notes:

A system register is 16 bits; thus, 0 is the 32-bit value formed by combining the 0th and 1st registers.

1 is the 32-bit value formed by combining the 2nd and 3rd registers, and so on.

[Usage Example]

```
Function main
    SysMemOutReal 1 , 123.321
    'Set the combined 32-bit value of the 2nd and 3rd registers to 123.321
Fend
```

Detailed Explanation of Communication Instructions

SetNet

[Function]

Set the communication parameters for TCP/IP.

[Syntax]

SetNet #Port_Number, IP_Address, Port_Number{, End_Character, Flow_Control, Timeout, Communication_Protocol}

[Parameter Description]

#Port_Number	Specifies the port number whose communication parameters need to be modified. Range: #201~216
IP_ADDRESS	The IP address used for TCP/IP communication. For example: 192.168.1.1.
Port_Number	The port number used for TCP/IP communication. For example: 8090. Range: 0~65535
End_Character	The end character for the communication data. Can be set to CR, LF, CRLF (Carriage Return, Line Feed, Carriage Return Line Feed). Optional.
Flow_Control	Refers to software flow control. Set to NONE. Optional.
Timeout	Sets the maximum duration for sending and receiving, in seconds. 0 means no timeout. Optional.
Communication_Protocol	Currently only supports TCP protocol. Optional.

[Usage Example]

```
Function main
    ReConnect:
    CloseNet #201
    SetNet #201 , "192.168.1.220" , 9000 ,CRLF
    OpenNet #201 As Client
    WaitNet #201, 5
    If ChkNet(201) =  -1 Then
        Print "Visual TCP communication error, port is open but communication is not established."
        Wait 1
        GoTo ReConnect
    Elself ChkNet(201) = -2 Then
        Print "Visual TCP communication error, another task is using the port."
        Wait 1
        GoTo ReConnect
    Elself ChkNet(201) = -3 Then
```

```
        Print "Visual TCP communication error, port is not open."
        Wait 1
        GoTo ReConnect
    EndIf
Fend
```

OpenNet

[Function]

Used to initiate TCP/IP communication or obtain the thread number of OpenNet.

[Syntax]

```
OpenNet #PortNumbe As [Client/Server]
OpenNet(PortNumbe)
```

[Parameter Description]

#PortNumbe Specifies the port number for which communication parameters need to be modified.

Range: #201~216

Client/Server Client refers to the client. Server refers to the server.

[Usage Example]

```
Function main
    ReConnect:
    CloseNet #201
    SetNet #201 , "192.168.1.220" , 9000 ,CRLF
    OpenNet #201 As Client
    WaitNet #201, 5
    If ChkNet(201) =  -1 Then
        Print "Visual TCP communication error, the port is open but communication has not been
    established."
        Wait 1
        GoTo ReConnect
    ElseIf ChkNet(201) = -2 Then
        Print "Visual TCP communication error, another task is using the port."
        Wait 1
        GoTo ReConnect
    ElseIf ChkNet(201) = -3 Then
        Print "Visual TCP communication error, the port is not open."
        Wait 1
        GoTo ReConnect
    EndIf
Fend
```

WaitNet

[Function]

Used to wait for the establishment of a TCP/IP communication port connection.

[Syntax]

```
WaitNet #PortNumbe{, Timeout}
```

[Parameter Description]

#PortNumbe Specifies the port number for which communication parameters need to be modified.

Range: #201~216

Timeout Sets the waiting time in seconds. If omitted, it will wait indefinitely.

[Usage Example]

```
Function main
    ReConnect:
    CloseNet #201
    SetNet #201 , "192.168.1.220" , 9000 ,CRLF
    OpenNet #201 As Client
    WaitNet #201, 5
    If ChkNet(201) =  -1 Then
        Print "Visual TCP communication error, the port is open but communication has not been
    established."
        Wait 1
        GoTo ReConnect
    Elself ChkNet(201) = -2 Then
        Print "Visual TCP communication error, another task is using the port."
        Wait 1
        GoTo ReConnect
    Elself ChkNet(201) = -3 Then
        Print "Visual TCP communication error, the port is not open."
        Wait 1
        GoTo ReConnect
    Endlf
Fend
```

ChkNet

[Function]

Used to return the number of bytes in the communication port's receive buffer.

[Syntax]

ChkNet(PortNumbe)

[Parameter Description]

PortNumbe Specifies the port number for which communication parameters need to be modified.

Range: 201~216

Notes:

1. Returns the byte count as an integer. If no data exists, it returns a negative value indicating the communication port status. -1 means the port is open but has no communication connection. -2 means the communication port is being used by another thread. -3 means the communication port is not open.

[Usage Example]

```
Function main
    ReConnect:
    CloseNet #201
    SetNet #201 , "192.168.1.220" , 9000 ,CRLF
    OpenNet #201 As Client
    WaitNet #201, 5
    If ChkNet(201) =  -1 Then
        Print "Visual TCP communication error, the port is open but communication has not been
    established."
        Wait 1
        GoTo ReConnect
    ElseIf ChkNet(201) = -2 Then
        Print "Visual TCP communication error, another task is using the port."
        Wait 1
        GoTo ReConnect
    ElseIf ChkNet(201) = -3 Then
        Print "Visual TCP communication error, the port is not open."
        Wait 1
        GoTo ReConnect
    EndIf
Fend
```

CloseNet

[Function]

Used to close the TCP/IP communication port opened by OpenNet.

[Syntax]

CloseNet [#PortNumbe/All]

[Parameter Description]

#PortNumbe Specifies the port number for which communication parameters need to be modified.

Range: #201~216。All closes all communication ports.

[Usage Example]

```
Function main
    ReConnect:
    CloseNet #201
    SetNet #201 , "192.168.1.220" , 9000 ,CRLF
    OpenNet #201 As Client
    WaitNet #201, 5
    If ChkNet(201) =  -1 Then
        Print "Visual TCP communication error, the port is open but communication has not been
    established."
        Wait 1
        GoTo ReConnect
    Elself ChkNet(201) = -2 Then
        Print "Visual TCP communication error, another task is using the port."
        Wait 1
        GoTo ReConnect
    Elself ChkNet(201) = -3 Then
        Print "Visual TCP communication error, the port is not open."
        Wait 1
        GoTo ReConnect
    Endlf
Fend
```

SetCom

[Function]

Used to set RS232/485 serial communication parameters.

[Syntax]

SetCom #PortNumbe{, BaudRate, DataBitLength, StopBitLength, Parity, EndCharacter, H/WFlowControl, S/WFlowControl, Timeout}

[Parameter Description]

#PortNumbe	Specifies the port number for which communication parameters need to be modified. Range: #1~16
BaudRate	Can be specified as 4800, 9600, 14400, 19200, 38400, 56000, 57600, 115200, 128000, 230400, 256000. Default is 19200 if omitted.
DataBitLength	Can be specified as 7 or 8 for the data bit length of each character. Can be omitted.
StopBitLength	Can be specified as 1 or 2 for the stop bit length of each character. Can be omitted.
EndCharacter	The ending character of the communication data. Can be set to CR, LF, CRLF (Carriage Return, Line Feed, Carriage Return Line Feed). Can be omitted.
H/WFlowControl,	Specify RTS if hardware flow control is enabled; specify NONE if disabled. Can be omitted.
S/WFlowControl	Specify XON if software flow control is enabled; specify NONE if disabled. Can be omitted.
Timeout	Sets the maximum time for sending and receiving in seconds. 0 means no timeout. Can be omitted.

[Usage Example]

```
Function main
    Integer AA
    ReConnect:
    SetCom #2,38400,8,2,N,CRLF,NONE,NONE,0
    OpenCom #2
    Do
        Print #2 ,"OK"
        Input #2,AA
        Print AA
    Loop
```

```
Fend
```

OpenCom

[Function]

Used to open RS232/485 serial communication, or to obtain the thread number of the RS232/485 serial port.

[Syntax]

```
OpenCom #PortNumbe

OpenCom(PortNumbe)
```

[Parameter Description]

#PortNumbe Specifies the port number for which communication parameters need to be modified. Range: #1~16

[Usage Example]

```
Function main
    Integer AA
    ReConnect:
    SetCom #2,38400,8,2,N,CRLF,NONE,NONE,0
    OpenCom #2
    Do
        Print #2 ,"OK"
        Input #2,AA
        Print AA
    Loop

Fend
```

ChkCom

[Function]

Used to return the number of bytes in the receive buffer of the serial communication port.

[Syntax]

```
ChkCom(PortNumbe)
```

[Parameter Description]

PortNumbe Specifies the port number for which communication parameters need to be modified. Range: 1~16

Notes:

1. Returns the number of bytes as an integer. If no data exists, it returns a negative value indicating the status of the communication port. -2 means another thread is using this communication port. -3 means the communication port is not opened.

[Usage Example]

```
Function main
    Integer numChars
    numChars = ChkCom(1)
Fend
```

ClrCom

[Function]

Used to return the number of bytes in the receive buffer of the serial communication port.

[Syntax]

```
ClrCom #PortNumbe
```

[Parameter Description]

#PortNumbe Specifies the port number for which communication parameters need to be modified. Range: #1~16

[Usage Example]

```
Function main
    ClrCom #1
Fend
```

CloseCom

[Function]

Used to close the serial communication port opened by OpenCom.

[Syntax]

```
CloseCom [#PortNumbe/All]
```

[Parameter Description]

#PortNumbe Specifies the port number for which communication parameters need to be modified. Range: #1~16。All means to close all.

[Usage Example]

```
Function main
    CloseCom #1
Fend
```

Read

[Function]

Used to read a specified number of characters from the communication port, excluding the terminator.

[Syntax]

```
Read #PortNumbe, stringVar$, charCount
```

[Parameter Description]

#PortNumbe Specifies the port number for which communication parameters need to be modified.

TCP/IP range: #201~216. Serial port range: #1~16.

stringVar$ A string variable used to receive characters from the port.

charCount Number of characters to read.

[Usage Example]

```
Function main
    String strBin$
```

```
    Read #201,strBin$,4          'Read four characters from port #201
Fend
```

ReadBin

[Function]

Used to read binary data from a TCP/IP communication port.

[Syntax]

```
ReadBin #PortNumbe, variable_name
ReadBin #PortNumbe, array_variable_name(), byte_count
```

[Parameter Description]

#PortNumbe	Specifies the port number that needs to have its communication parameters modified. TCP/IP range: #201~216. Serial port range: #1~16.
variable_name	A variable used to receive binary data from the port.
array_variable_name()	An array variable used to receive binary data from the port.
byte_count	The number of bytes to read.

[Usage Example]

```
Function main
    Integer num
    ReadBin #201,num
    Integer arr(5)
    ReadBin #201,arr(),5
Fend
```

Write

[Function]

Writes a string to the communication port, without an end character.

[Syntax]

```
Write #PortNumbe, string
```

[Parameter Description]

#PortNumbe	Specifies the port number that needs to have its communication parameters modified. TCP/IP range: #201~216. Serial port range: #1~16.
string	The string to be written to the communication port.

[Usage Example]

```
Function main
    Write #201 , "abcd"
Fend
```

WriteBin

[Function]

Writes binary data to the communication port, without an end character.

[Syntax]

```
WriteBin #PortNumbe, data
WriteBin #PortNumbe, array_variable_name(), byte_count
```

[Parameter Description]

#PortNumbe	Specifies the port number that needs to have its communication parameters modified. TCP/IP range: #201~216. Serial port range: #1~16.
variable_name	A variable used to receive binary data from the port.
array_variable_name()	An array variable used to receive binary data from the port.
byte_count	The number of bytes to read.

[Usage Example]

```
Function main
    WriteBin #201 , 123456
    Integer arr(5)
    arr(0) = 22
    arr(1) = 43
    arr(2) = 67
    arr(3) = 45
    arr(4) = 33
    WriteBin #201 , arr() , 5
Fend
```

Print

[Function]

Outputs data or strings to the specified communication port, including an end character.

[Syntax]

```
Print #PortNumbe, data1{, data2...}
```

[Parameter Description]

#PortNumbe Specifies the port number that needs to have its communication parameters modified. TCP/IP range: #201~216. Serial port range: #1~16.

data Can be variable, numeric, string, etc., to output the desired content. The amount of data can be increased as needed.

[Usage Example]

```
Function main
    String CCD_X$,CCD_Y$,CCD_U$
    CloseNet #201
    SetNet #201 , "192.168.1.220" , 9000 ,CRLF
    OpenNet #201 As Client
    WaitNet #201, 5
    Print #201 , "EXW,1"
    Input #201 , CCD_X$,CCD_Y$,CCD_U$
    Print CCD_X$,CCD_Y$,CCD_U$
Fend
```

Input

[Function]

Used to read data or strings from the communication port, including an end character.

[Syntax]

```
Input #PortNumbe, variable1{, variable2...}
```

[Parameter Description]

#PortNumbe Specifies the port number that needs to have its communication parameters modified. TCP/IP range: #201~216. Serial port range: #1~16.

variable The variable name used to receive data or strings. Additional variables can be added according to the length of the data.

Notes:

1. Input # defaults to separating strings by “,” (comma) or “ ” (space). For other delimiters, please use the token string command.

[Usage Example]

```
Function main
    String CCD_X$,CCD_Y$,CCD_U$
    CloseNet #201
    SetNet #201 , "192.168.1.220" , 9000 ,CRLF
    OpenNet #201 As Client
    WaitNet #201, 5
    Print #201 , "EXW,1"
    Input #201 , CCD_X$,CCD_Y$,CCD_U$
    Print CCD_X$,CCD_Y$,CCD_U$
Fend
```

Line Input

[Function]

Used to read a line of data or a string from the communication port, including the end delimiter.

[Syntax]

```
Line Input #PortNumbe, Variable
```

[Parameter Description]

#PortNumbe Specifies the port number that needs to have its communication parameters modified. TCP/IP range: #201~216. Serial port range: #1~16.

Variable The name of the variable used to receive the data or string.

[Usage Example]

```
Function main
    String CCD_TxT$
    CloseNet #201
    SetNet #201 , "192.168.1.220" , 9000 ,CRLF
    OpenNet #201 As Client
    WaitNet #201, 5
    Print #201 , "EXW,1"
    Line Input #201 , CCD_TxT$
    Print CCD_TxT$
Fend
```

System Command Detailed Description

Print

[Function]

Print command, used to print strings, numbers, and variables.

[Syntax]

Print Content1{, Content2.....}

[Parameter Description]

Content The string, number, or variable to be printed. Additional variables can be added as needed.

[Usage Example]

```
Function main
    ReConnect:
    CloseNet #201
    SetNet #201 , "192.168.1.220" , 9000 ,CRLF
    OpenNet #201 As Client
    WaitNet #201, 5
    If ChkNet(201) =  -1 Then
        Print "Visual TCP communication error, the port is open but communication has not been
    established."
        Wait 1
        GoTo ReConnect
    Elself ChkNet(201) = -2 Then
        Print "Visual TCP communication error, another task is using the port."
        Wait 1
        GoTo ReConnect
    Elself ChkNet(201) = -3 Then
        Print "Visual TCP communication error, the port is not open."
        Wait 1
        GoTo ReConnect
    Endlf

    Print ChkNet(201)

Fend
```

Xqt

[Function]

Start multithreading.

[Syntax]

Xqt {TaskNumber, } FunctionName(Parameter1...) {, Normal/NoPause/NoEmgAbort}

[Parameter Description]

TaskNumber	An integer that specifies the multithreading number. It can be omitted.
FunctionName	The name of the function to execute in multithreading. If the function has parameters, parentheses must be included with the parameters filled in; if not, only the function name should be provided.
Normal	Standard multithreading. If all three options are omitted, the default is Normal.
NoPause	Multithreading that will not pause under conditions of Pause, external pause signals, or when the safety door is open.
NoEmgAbort	Multithreading that can continue processing even during emergency stops or errors.

Notes:

1.Front-end multithreading, Task Numbers 1-16. Back-end multithreading, Task Numbers 66-80.

[Usage Example]

```
Function main
    Xqt A1
Fend

Function A1
    Print "Runing....."
Fend
```

Pause

[Function]

Used to pause all pauseable multithreading.

[Syntax]

Pause

[Parameter Description]

Notes:

1.Front-end multithreading, Task Numbers 1-16. Back-end multithreading, Task Numbers 66-80.

[Usage Example]

```
Function main
    Xqt A1
    Wait 1
    Pauser              'Pause the program
Fend

Function A1
    Print "Runing....."
Fend
```

Halt

[Function]

Used to pause the specified executing multithreading

[Syntax]

Halt [FunctionName/TaskNumber]

[Parameter Description]

FunctionName The name of the executing multithreading function.

TaskNumber The task number of the executing multithreading.

Notes:

1.Front-end multithreading, Task Numbers 1-16. Back-end multithreading, Task Numbers 66-80.

[Usage Example]

```
Function main
    Xqt A1
    Wait 1
    Halt A1          'Pause A1 thread
Fend

Function A1
    Do
        Print "Runing....."
    Loop
Fend
```

Quit

[Function]

Stop all or specified threads

[Syntax]

Quit [FunctionName/TaskNumber/All]

[Parameter Description]

FunctionName The name of the executing multithreading function.。

TaskNumber The task number of the executing multithreading.

All Stop all threads.

Notes:

1.Front-end multithreading, Task Numbers 1-16. Back-end multithreading, Task Numbers 66-80.

[Usage Example]

```
Function main
    Xqt A1
    Wait 1
    Quit A1          'Stop A1 thread
Fend

Function A1
    Do
        Print "Runing....."
    Loop
Fend
```

StartMain

[Function]

Used to enable the main task.

[Syntax]

StartMain [MainTaskName]

[Parameter Description]

MainTaskName main - main63

[Usage Example]

```
Function bgmain              'Background task
    If MemIn W(0) = 1 Then
        StartMain main
        Wait MemIn W(0) <> 1      'Wait for the signal to disappear to prevent repeated triggering
    EndIf
Fend
```

Resume

[Function]

Used to continue running the multithreading that was paused by the Halt command.

[Syntax]

Resume [FunctionName/TaskNumber/All]

[Parameter Description]

FunctionName The name of the executing multithreading function.

TaskNumber The task number of the executing multithreading.

All Continue all threads.

Notes:

1. Front-end multithreading, Task Numbers 1-16. Back-end multithreading, Task Numbers 66-80.

[Usage Example]

```
Function main
    Xqt A1
    Wait 1
    Halt A1          'Pause A1 thread
    Wait Sw(1) = 1
    Resume A1        'Continue running A1 thread
Fend

Function A1
    Do
        Print "Runing....."
    Loop
Fend
```

Reset

[Function]

Used to reset the controller's abnormal alarm status.

[Syntax]

```
Resume [Error]
```

[Parameter Description]

Error **Indicates that all errors will be cleared by the instruction.**

Notes:

1. If the Error parameter is specified when invoking the instruction, all alarms will be cleared; if the Error parameter is not specified, only the emergency stop status will be cleared.

[Usage Example]

```
Function main
    Resume          'Clear emergency stop status
Fend

Function A1
    Resume Error    'Clear abnormal alarm status
Fend
```

MyTask

[Function]

Used to return the task number of the current thread.

[Syntax]

MyTask

[Parameter Description]

Notes:

1.Front-end multithreading, Task Numbers 1-16. Back-end multithreading, Task Numbers 66-80.

[Usage Example]

```
Function main
    Xqt 2 , A1
    Xqt 3 , A1
    Xqt 4 , A1
Fend

Function A1
    On MyTask       'Set the IO port corresponding to the current task number to On
    Wait 1
    Off MyTask      'Set the IO port corresponding to the current task number to Off
Fend
```

TaskDone

[Function]

Confirm if the thread has finished.

[Syntax]

TaskDone(FunctionName/TaskNumber)

[Parameter Description]

FunctionName Multi-threaded function name.

TaskNumber Multi-threaded task number.

Notes:

1. Front-end multithreading, Task Numbers 1-16. Back-end multithreading, Task Numbers 66-80.

2. Returns True if the thread has finished. Otherwise, returns False.

[Usage Example]

```
Function main
    Xqt A1
    Do
        Print "A1"
    Loop Until TaskDone(A1)
Fend

Function A1
    Go P1
Fend
```

TaskState

[Function]

Return the current state of the thread.

[Syntax]

TaskState(FunctionName/TaskNumber)

[Parameter Description]

FunctionName Multi-threaded function name.

TaskNumber Multi-threaded task number.

Notes:

1. Front-end multithreading, Task Numbers 1-16. Back-end multithreading, Task Numbers 66-80.

   0 means not executed. 4 means standby. 5 means starting. 6 means running. 7 means error state. 8 means paused. 9 means recovering.10 means stopping.

[Usage Example]

```
Function main
    If TaskState(A1) = 0 Then
        Xqt 2 , A1
    EndIf
Fend

Function A1
    Go P1
Fend
```

TaskWait

[Function]

Used to wait for the specified thread to finish.

[Syntax]

TaskWait(FunctionName/TaskNumber)

[Parameter Description]

FunctionName Multi-threaded function name.

TaskNumber Multi-threaded task number.

Notes:

1. Front-end multithreading, Task Numbers 1-16. Back-end multithreading, Task Numbers 66-80.

[Usage Example]

```
Function main
    Xqt 2 , A1
    TaskWait A1
Fend

Function A1
    Go P1
Fend
```

SyncLock

[Function]

Used for mutual exclusion locking to synchronize multiple threads.

[Syntax]

SyncLock SignalNumber{, Timeout}

[Parameter Description]

SignalNumber Specify the signal number to be received using an expression or value. Range 0-63.

Timeout Specify the time to wait before locking using an expression or value. Optional.

[Usage Example]

'The example below demonstrates using SyncLock and SyncUnlock to ensure that only one task writes information to the log file at a time.

```
Function Main
    Xqt Func1
    Xqt Func2
Fend
Function Func1
    Long count
    Do
    Wait 0.5
        count = count+1
        LogMsg("Msg from Func1,"+ Str$(count))
    Loop
Fend

Function Func2
    Long count
    Do
    Wait 0.5
        count = count+1
        LogMsg("Msg from FuncmsgSCloseCom2,"+ Str$(count))
    Loop
Fend

Function LogMsg(msg$ As String)
    SyncLock 1
    Print msg$
    Wait 1
```

```
    SyncUnlock 1
Fend
```

SyncUnLock

[Function]

Used to release the SyncLock lock with the specified signal number.

[Syntax]

SyncUnLock SignalNumber

[Parameter Description]

SignalNumber Specify the signal number to be received using an expression or value. Range 0-63.

[Usage Example]

The following example shows the use of SyncLock and SyncUnlock to ensure that only one task writes information to the log file once.

```
Function Main
    Xqt Func1
    Xqt Func2
Fend
Function Func1
    Long count
    Do
    Wait 0.5
        count = count+1
        LogMsg("Msg from Func1,"+ Str$(count))
    Loop
Fend

Function Func2
    Long count
    Do
    Wait 0.5
        count = count+1
        LogMsg("Msg from FuncmsgSCloseCom2,"+ Str$(count))
    Loop
Fend

Function LogMsg(msg$ As String)
    SyncLock 1
    Print msg$
    Wait 1
    SyncUnlock 1
Fend
```

Signal

[Function]

Used to send a signal to the task executing the WaitSig command.

[Syntax]

```
Signal SignalNumber
```

[Parameter Description]

SignalNumber	Specify the signal number to be received using an expression or value. Range 0-63.

[Usage Example]

```
Function main
    Xqt 2 , A1
    Signal 1
Fend

Function A1
    WaitSig 1
Fend
```

WaitSig

[Function]

Used to wait for the synchronization signal sent by other tasks' Signal command.

[Syntax]

```
WaitSig SignalNumber{, Timeout}
```

[Parameter Description]

SignalNumber Specify the signal number to be received using an expression or value. Range 0-63.

Timeout Specify the maximum waiting time using an expression or value. This parameter is optional.

[Usage Example]

```
Function main
    Xqt 2 , A1
    Signal 1
Fend

Function A1
    WaitSig 1
Fend
```

TW

[Function]

Used to return the status of the Wait, WaitNet, WaitSig commands.

[Syntax]

TW

[Parameter Description]

Notes:

1. Returns False if the conditions of Wait, WaitNet, and WaitSig are satisfied within the specified time. Returns True on timeout.

[Usage Example]

```
Function main
    Wait Sw(0) = On , 5
    If TW = True Then
        'Wait for 5 seconds before DI0 changes to ON state
        Print "DI0 is not set to On"
    EndIf
Fend
```

ElapsedTime

[Function]

Returns the time elapsed in seconds since the timer started counting for the calculation cycle, excluding program pause time.

[Syntax]

ElapsedTime

[Parameter Description]

Notes:

1. The timer measurement range is 0 to 1.7E+31. The minimum unit is 0.001 seconds.

[Usage Example]

```
Function Main
    Integer i
    ResetElapsedTime                    'Reset the timer used for calculating cycle time
    For i = 1 To 10
        GoSub Cycle
        Wait 0.1
    Next
    Cycle:
    Print "Count: ",i,ElapsedTime / 10  'Calculate and print the cycle time
    If i < 10 Then
        Return
    EndIf
Fend
```

ResetElapsedTime

[Function]

Used to reset the timer for the calculation beat time utilized by ElapsedTime.

[Syntax]

ResetElapsedTime

[Parameter Description]

[Usage Example]

```
Function Main
    Integer i
    ResetElapsedTime                    'Reset the timer used for calculating cycle time
    For i = 1 To 10
        GoSub Cycle
        Wait 0.1
    Next
    Cycle:
    Print "Count: ",i,ElapsedTime / 10  'Calculate and print the cycle time
    If i < 10 Then
        Return
    EndIf
Fend
```

Tmr

[Function]

Returns the elapsed time in seconds since the timer started, including any program pause duration.

[Syntax]

Tmr(timer_ID)

[Parameter Description]

timer_ID Specifies which timer's elapsed time to return as an expression or numeric value.

Range: 0~63.

[Usage Example]

```
Function Main
    Integer i
    TmReset 0                       'Reset the timer used for calculating cycle time
    For i = 1 To 10
        GoSub Cycle
        Wait 0.1
    Next
    Cycle:
    Print "Count: ",i,Tmr(0) / 10   'Calculate and print the cycle time
    If i < 10 Then
        Return
    EndIf
Fend
```

TmReset

[Function]

Used to reset the Tmr timer.

[Syntax]

```
TmReset timer_ID
```

[Parameter Description]

timer_ID Specifies which timer’s elapsed time to return as an expression or numeric value.

Range: 0~63.

[Usage Example]

```
Function Main
    Integer i
    TmReset 0                       'Reset the timer used for calculating cycle time
    For i = 1 To 10
        GoSub Cycle
        Wait 0.1
    Next
    Cycle:
    Print "Count: ",i,Tmr(0) / 10   'Calculate and print the cycle time
    If i < 10 Then
        Return
    EndIf
Fend
```

Eval

[Function]

Used to execute statements in the command window.

[Syntax]

```
Eval command
```

[Parameter Description]

command Specifies the command to be executed as a string.

return value Returns the error code resulting from the command execution. Returns 0 when the command completes successfully.

[Usage Example]

```
Function Main

Integer errCMD
String CMD$
OpenCom #1

Do
    Line Input #1, CMD$
    errCMD = EVal(CMD$)
    Print #1, errCMD
Loop

Fend
```

Stacking Statement Detailed Description

Pallet

[Function]

Used to define/display pallets.

[Syntax]

Define Pallet:

```
Pallet {OutSide,} pallet_ID, position 1, position 2, position 3{, position 4,} count1, count2
```

Print a single pallet:

```
Pallet pallet_ID
```

Print all pallets:

```
Pallet
```

Retrieve pallet coordinates:

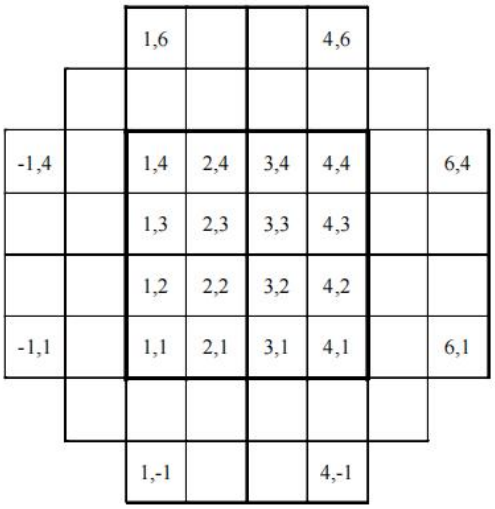
```
Pallet(pallet_ID, pallet_position_number)
Pallet(pallet_ID, row_coordinate, column_coordinate)
```

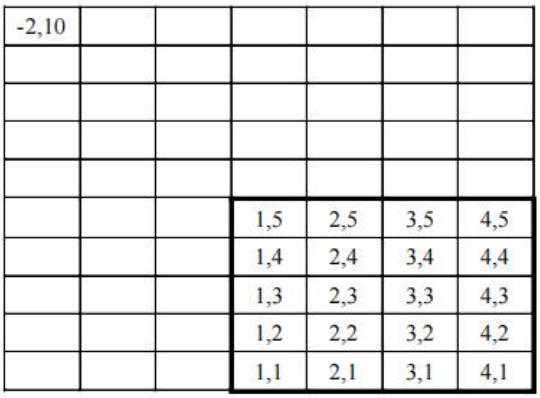
[Parameter Description]

OutSide	Generates additional pallets outside the specified row and column range. Optional.
pallet_ID	Specifies the pallet number using an expression or numeric value. Range: 0~15.
position 1~3	Defines the pallet using the standard three-point method.
position 4	Can further enhance the precision of pallet coordinates using a four-point method.
count1	Specifies the number of coordinates between position 1 and position 2 as an integer. Range: 1~32767.
count2	Specifies the number of coordinates between position 1 and position 3 as an integer. Range: 1~32767.
row_coordinate	Specifies the row number of the pallet as an integer.
column_coordinate	Specifies the column number of the pallet as an integer.

Notes:

1. The effect of OutSide is shown in the figure below: It expands the pallet coordinates based on the original pallet.





[Usage Example]

```
Function main
    Integer var

    'Define a 5*3 pallet
    Pallet 1 , P1 , P2 , P3 , 5 , 3

    'Print pallet number 2
    Pallet 2

    'Print all pallets
    Pallet

    'For loop to traverse pallet coordinates
    For var = 1 To 15
        Go Pallet(1 , var)
    Next

    Go Pallet(1,2,3) 'Move to position in pallet 1 at row 2, column 3
Fend
```

PalletClr

[Function]

Used to clear the set Pallet.

[Syntax]

```
PalletClr pallet_ID
```

[Parameter Description]

pallet_ID Specifies the pallet ID to clear using an expression or numeric value. Range: 0~15.

[Usage Example]

```
Function main
    Integer var

    'Define a 5*3 pallet
    Pallet 1 , P1 , P2 , P3 , 5 , 3

    'Clear pallet number 1
    PalletClr 1

Fend
```

Conveyor Belt Tracking Statement Detailed Description

SyStart

[Function]

Start the conveyor belt tracking function.

[Syntax]

SyStart conveyor_belt_ID

[Parameter Description]

conveyor_belt_ID	Specifies the conveyor belt ID using an expression or numeric value. Range: 1~4.

Notes:

1. This command is used when the robot needs to start tracking materials on the conveyor belt.

[Usage Example]

```
Recrt:
    SyStart Cvt_One 'Start conveyor belt tracking


If SyGetPose(Cvt_One) > 700 Then
    Print "Insufficient material processing distance"
    SyEnd Cvt_One 'Stop conveyor belt tracking
    GoTo Recry
EndIf

SyStart Cvt_One    'Start conveyor belt tracking to proceed with production
.
.
.
.
SyEnd Cvt_One      'End conveyor belt tracking
```

SyEnd

[Function]

End the conveyor belt tracking function.

[Syntax]

SyEnd conveyor_belt_ID

[Parameter Description]

conveyor_belt_ID Specifies the conveyor belt ID using an expression or numeric value. Range: 1~4.

Notes:

1. This command is used when the robot has completed tracking on the conveyor belt and no longer needs to track.

[Usage Example]

```
Recrt:
    SyStart Cvt_One 'Start conveyor belt tracking

If SyGetPose(Cvt_One) > 700 Then
    Print "Insufficient material processing distance"
    SyEnd Cvt_One 'Stop conveyor belt tracking
    GoTo Recry
EndIf

SyStart Cvt_One    'Start conveyor belt tracking to proceed with production
.
.
.
.
SyEnd Cvt_One      'End conveyor belt tracking
```

SyReset

[Function]

Clear the current conveyor belt tracking target queue and reset conveyor belt tracking.

[Syntax]

SyReset conveyor_belt_ID

[Parameter Description]

conveyor_belt_ID Specifies the conveyor belt ID using an expression or numeric value. Range: 1~4.

[Usage Example]

```
SyReset Cvt_One       'Initialize conveyor belt tracking
Recrt:
    SyStart Cvt_One 'Start conveyor belt tracking

If SyGetPose(Cvt_One) > 700 Then
    Print "Insufficient material processing distance"
    SyEnd Cvt_One 'Stop conveyor belt tracking
    GoTo Recry
EndIf

SyStart Cvt_One    'Start conveyor belt tracking to proceed with production
.
.
.
.
SyEnd Cvt_One      'End conveyor belt tracking
```

SyGetPoint

[Function]

Retrieve point information from the specified conveyor belt target queue.

[Syntax]

SyGetPoint(conveyor_belt_ID)

[Parameter Description]

conveyor_belt_ID Specifies the conveyor belt ID using an expression or numeric value. Range: 1~4.

[Usage Example]

```
SyMove SyGetPoint(Cvt_One):X(X1) :Y(Y1) :Z(Z1) :U(RZ1) CP

SyMove SyGetPoint(Cvt_One):X(X1) :Y(Y1) :Z(Z1) :U(RZ1)

On 0
Wait 1

Off 0
Wait 1
```

SyGetUserData

[Function]

Retrieve additional information about a specific point in the target queue of the conveyor belt.

[Syntax]

SyGetUserData(conveyor_belt_ID)

[Parameter Description]

conveyor_belt_ID Specifies the conveyor belt ID using an expression or numeric value. Range: 1~4.

Notes:

1. This information is added by the SyAddVisTarget command.

[Usage Example]

```
Print SyGetUserData(1)
```

SyGetPose

[Function]

Get the current position of a target on the specified conveyor belt. Unit: mm.

[Syntax]

SyGetPose(conveyor_belt_ID)

[Parameter Description]

conveyor_belt_ID Specifies the conveyor belt ID using an expression or numeric value. Range: 1~4.

[Usage Example]

```
Recrt:
    SyStart Cvt_One 'Start conveyor belt tracking

If SyGetPose(Cvt_One) > 700 Then
    Print "Insufficient material processing distance"
    SyEnd Cvt_One 'Stop conveyor belt tracking
    GoTo Recry
EndIf

SyStart Cvt_One    'Start conveyor belt tracking to proceed with production
.
.
.
.
SyEnd Cvt_One      'End conveyor belt tracking
```

SyGetNum

[Function]

Get the number of targets in the queue of the specified conveyor belt.

[Syntax]

SyGetNum(conveyor_belt_ID)

[Parameter Description]

conveyor_belt_ID Specifies the conveyor belt ID using an expression or numeric value. Range: 1~4.

[Usage Example]

```
If SyGetNum(1)  > 1 Then
    Print "There are targets in the tracking queue"
EndIf
```

SyMove

[Function]

Conveyor belt tracking linear interpolation motion.

[Syntax]

SyMove Target Coordinate {CP}{Till/Find}{!...!}

[Parameter Description]

Target Coordinate	Specify the target position using point data.
CP	Set motion smoothing. Optional.
Till/Find	Till expression = On/Off. Find expression = On/Off. Optional. For details, please refer to the detailed explanation of Till/Find.
!...!	Parallel processing of I/O and other input/output during motion. Optional. For details, please refer to the detailed explanation of parallel processing.

[Usage Example]

```
SyMove SyGetPoint(Cvt_One):X(X1) :Y(Y1) :Z(Z1) :U(RZ1) CP

SyMove SyGetPoint(Cvt_One):X(X1) :Y(Y1) :Z(Z1) :U(RZ1)

On 0
Wait 1

Off 0
Wait 1
```

SyArc

[Function]

Conveyor belt tracking circular arc interpolation motion.

[Syntax]

SyArc Waypoint_Coordinate, Target Coordinate {CP}{Till/Find}{!...!}

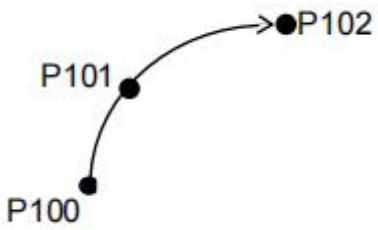
[Parameter Description]

Waypoint_Coordinate	Specifies the position that the circular arc will pass through, using point data obtained from conveyor belt tracking instructions.
Target Coordinate	Specify the target position using point data.
CP	Set motion smoothing. Optional.
Till/Find	Till expression = On/Off. Find expression = On/Off. Optional. For details, please refer to the detailed explanation of Till/Find.
!...!	Parallel processing of I/O and other input/output during motion. Optional. For details, please refer to the detailed explanation of parallel processing.

[Usage Example]

```
On 0
Wait 1
SyArc SyGetPoint(Cvt_One):X(X1) :Y(Y1) :Z(Z1) :U(RZ1)

On 0
Wait 1
```



SyTrig

[Function]

Record the current encoder pulse value of the conveyor belt.

[Syntax]

SyTrig conveyor_belt_ID

[Parameter Description]

conveyor_belt_ID Specifies the conveyor belt ID using an expression or numeric value. Range: 1~4.

Notes:

1. The subsequent SyAddVisTarget will use the encoder pulse value recorded by this command.

[Usage Example]

```
SyTrig 1
```

SyPoint

[Function]

Convert regular coordinates or points to conveyor belt tracking coordinates or points.

[Syntax]

SyPoint(conveyor_belt_ID, Coordinate_X, Coordinate_Y, Coordinate_U)
SyPoint(conveyor_belt_ID, Position)

[Parameter Description]

conveyor_belt_ID Specifies the conveyor belt ID using an expression or numeric value.

Range: 1~4.

Coordinate Specifies the coordinates in mm that need to be converted to conveyor belt tracking coordinates.

position Specifies the point to be converted to conveyor belt tracking coordinates using an expression, point name, or point number.

[Usage Example]

```
SyAddVisTarget 1 , SyPoint(1 , 10 , 20 , 30)
SyMove SyPoint(1 , 5 , 5 , 5)
```

SyAddVisTarget

[Function]

Register visual inspection results into the conveyor belt tracking queue.

[Syntax]

SyAddVisTarget conveyor_belt_ID, position {, Additional_Information}

[Parameter Description]

conveyor_belt_ID	Specifies the conveyor belt ID using an expression or numeric value. Range: 1~4.
position	Specifies the point to be registered using an expression, point name, or point number.
Additional_Information	Specifies additional information as an integer.

[Usage Example]

```
Do
    Input #TCP_Id , temp1$(0) , temp1$(1) , temp1$(2) , temp1$(3)
    Print temp1$(0) ,",", temp1$(1) ,",", temp1$(2) ,",", temp1$(3)
    CCD_X$ = temp1$(1)
    CCD_Y$ = temp1$(2)
    CCD_U$ = temp1$(3)
    If Val(temp1$(0)) = 1 Then
        Print "Register a point"
        SyAddVisTarget Cvt_One , SyPoint(Cvt_One , Val(CCD_X$) , Val(CCD_Y$) , Val(CCD_U$)
    EndIf
Loop Until ChkNet(TCP_Id) < 0
```

SyRejectDis

[Function]

Set/get the rejection distance for the conveyor belt tracking function.

[Syntax]

```
SyRejectDis conveyor_belt_ID , Distance
SyRejectDis (conveyor_belt_ID)
```

[Parameter Description]

conveyor_belt_ID Specifies the conveyor belt ID using an expression or numeric value. Range: 1~4.

Distance Specifies the distance in millimeters to differentiate material spacing.

Notes:

1. 1. Do not use this function during the tracking process.

[Usage Example]

```
Recrt:
    SyRejectDis Cvt_One , 20
    SySave

    SyRejectDis Cvt_One

    SyStart Cvt_One 'Start conveyor belt tracking

If SyGetPose(Cvt_One) > 700 Then
    Print "Insufficient material processing distance”
    SyEnd Cvt_One 'Stop conveyor belt tracking
    GoTo Recry
EndIf

SyStart Cvt_One    'Start conveyor belt tracking to proceed with production
.
.
.
.
```

```
SyEnd Cvt_One      'End conveyor belt tracking
```

SySave

[Function]

Save modifications made to the conveyor belt tracking function in the program.

[Syntax]

```
SySave
```

[Parameter Description]

Notes:

1. 1. All parameter modifications to the conveyor belt tracking function in the program need to be saved using this command.

[Usage Example]

```
Recrt:
    SyRejectDis Cvt_One , 20
    SySave

    SyRejectDis Cvt_One

    SyStart Cvt_One 'Start conveyor belt tracking

If SyGetPose(Cvt_One) > 700 Then
    Print "Insufficient material processing distance”
    SyEnd Cvt_One 'Stop conveyor belt tracking
    GoTo Recry
EndIf

SyStart Cvt_One    'Start conveyor belt tracking to proceed with production
.
.
.
.
SyEnd Cvt_One      'End conveyor belt tracking
```

Mathematical Operations Command Detailed Description

+、-、*、/、**、Mod、And、Or、Xor、Not、>、>=、< 、<=、<>、=

[Function]

Addition, Subtraction, Multiplication, Division, Exponentiation, Modulus, AND, OR, XOR, NOT, Greater Than, Greater Than or Equal To, Less Than, Less Than or Equal To, Not Equal To, Equal To

[Syntax]

Operators	Format Example	Description
+	A+B	Addition
-	A-B	Subtraction
*	A*B	Multiplication
/	A/B	Division
**	A**B	Exponentiation
=	A=B	A equals B
>	A>B	A is greater than B
<	A<B	A is less than B
>=	A>=B	A is greater than or equal to B
<=	A<=B	A is less than or equal to B
<>	A<>B	A is not equal to B
And	A And B	Logical AND
Mod	A Mod B	Modulus of integers
Not	Not A	NOT
Or	A Or B	Logical OR
Xor	A Xor B	Logical XOR

[Parameter Description]

[Usage Example]

DegToRad

[Function]

Convert degrees to radians.

[Syntax]

DegToRad(Angle)

[Parameter Description]

Angle	Specifies the angle to be converted to radians as a numeric value. Unit: deg.

[Usage Example]

```
Function main
    Double rad_num , cos_num
    rad_num =  DegToRad(30)
    cos_num =  Cos(rad_num)
Fend
```

RadToDeg

[Function]

Convert radians to degrees.

[Syntax]

RadToDeg(Radian)

[Parameter Description]

Radian	Specifies the radian value to be converted to degrees as a numeric value.

[Usage Example]

```
Function main
    Double deg_num
    deg_num =  RadToDeg(1.21)
Fend
```

Sin、Cos、Tan、Asin、Acos、Atan、Atan2

[Function]

Sine, Cosine, Tangent, Inverse Sine, Inverse Cosine, Inverse Tangent, Inverse Tangent2

[Syntax]

```
Sin(Radian)
Cos(Radian)
TaVn(Radian)
Asin(Value)
Acos(Value)
Atan(Value)
Atan2(y, x)
```

[Parameter Description]

Radian	Specifies the radian value to be used in the calculations as an integer or floating-point number.
Value	Specifies the numeric value to be used in the calculations as an integer or floating-point number.

[Usage Example]

```
Function main
    Double num

    num = Sin(DegToRad(30))
    num = Cos(DegToRad(30))
    num = Tan(DegToRad(30))

    num = Asin(0.5)
    num = Acos(0.5)
    num = Atan(0.5)
    num = Atan2(1,2)
Fend
```

HexToFloat

[Function]

Convert hexadecimal to single-precision floating-point number. Compliant with IEEE754.

[Syntax]

HexToFloat(Value)

[Parameter Description]

Value Can be specified as a decimal integer or hexadecimal number (&HValue).

[Usage Example]

```
Function main
    Print HexToFloat(&H40490FDA)          'print3.1415926
Fend
```

FloatToHex

[Function]

Convert single-precision floating-point number to hexadecimal. Compliant with IEEE754.

[Syntax]

FloatToHex(Value)

[Parameter Description]

Value Can be specified as a decimal integer or single-precision floating-point number.

[Usage Example]

```
Function main
    Print Hex$(FloatToHex(3.1415926))        'print 40490FDA
Fend
```

Abs

[Function]

Get the absolute value.

[Syntax]

Abs(Value)

[Parameter Description]

Value	Can be specified as a decimal integer or floating-point number.

[Usage Example]

```
Function main
    Abs(-1.234)
Fend
```

Sqr

[Function]

Calculate the square root.

[Syntax]

Sqr(Value)

[Parameter Description]

Value	Can be specified as a decimal integer or floating-point number.

[Usage Example]

```
Function main
    Sqr(4)
Fend
```

Sgn

[Function]

Return the sign of a numeric value.

[Syntax]

```
Sgn(Value)
```

[Parameter Description]

Value	Can be specified as a decimal integer or floating-point number.

[Usage Example]

```
Function main
    Sgn(4)
Fend
```

LShift

[Function]

Left Shift.

[Syntax]

```
LShift(Value, Bit_Count)
```

[Parameter Description]

Value	Can be specified as a decimal integer or floating-point number.
Bit_Count	Specifies the number of bits to shift left as an integer.

[Usage Example]

```
Function main
    LShift(4433 , 3)
Fend
```

RShift

[Function]

Right Shift.

[Syntax]

```
RShift(Value, Bit_Count)
```

[Parameter Description]

Value	Can be specified as a decimal integer or floating-point number.
Bit_Count	Specifies the number of bits to shift left as an integer.

[Usage Example]

```
Function main
    RShift(4433 , 3)
Fend
```

BClr

[Function]

Set the specified Bit of a numeric value to 0 and return that value.

[Syntax]

```
BClr(Value, Bit_Count)
```

[Parameter Description]

Value	Can be specified as a decimal integer or floating-point number.
Bit_Count	Specifies which bit to clear as an integer.

[Usage Example]

```
Function main
    BClr(123 , 2)
Fend
```

BSet

[Function]

Set the specified Bit of a numeric value to 1 and return that value.

[Syntax]

```
BSet(Value, Bit_Count)
```

[Parameter Description]

Value	Can be specified as a decimal integer or floating-point number.
Bit_Count	Specifies which bit to clear as an integer.

[Usage Example]

```
Function main
    BSet(123 , 2)
Fend
```

BTst

[Function]

Return the value of the specified Bit of a numeric value.

[Syntax]

```
BTst(Value, Bit_Count)
```

[Parameter Description]

Value	Can be specified as a decimal integer or floating-point number.
Bit_Count	Specifies which bit to clear as an integer.

[Usage Example]

```
Function main
    BTst(123 , 2)
Fend
```

Ubound

[Function]

Return the length of an array.

[Syntax]

Ubound(Array_Name {, Dimension })

[Parameter Description]

Array_Name	Can be specified as a decimal integer or floating-point number.
Dimension	1 for one-dimensional, 2 for two-dimensional, 3 for three-dimensional. Can be omitted; default is one-dimensional.

[Usage Example]

```
Function main
    Integer i , a(10)
    For i = 0 To UBound(a)
        a(i) = i
    Next
Fend
```

String Operation Command Detailed Description

ParseStr

[Function]

String Splitting.

[Syntax]

```
ParseStr String, String_Array$(), Delimiter$
Array_Length = ParseStr(String, String_Array$(), Delimiter$)
```

[Parameter Description]

String	The string to be split.
String_Array$()	The array to receive the substrings split by the delimiter.
Delimiter$	The delimiter used to split the string.
Array_Length	The length of the resulting array after the string has been split.

[Usage Example]

```
Function main
    String myStr$ , strArr$(0)
    Integer i
    myStr$ = "1$2$3$4"
    ParseStr myStr$ , strArr$() , "$"
    For i = 0 To UBound(strArr$())
        Print "Array Content = " , strArr$(i)
    Next
Fend
```

+

[Function]

String Concatenation.

[Syntax]

String + String

[Parameter Description]

String The strings to be concatenated.

[Usage Example]

```
Function main
    String myStr$
    myStr$ = "Pro" + "Easy"
Fend
```

<>、=

[Function]

Not Equal To, Equal To, Assignment.

[Syntax]

String <> String

String = String

stringVar = String

[Parameter Description]

String The string to be used in the operation.

[Usage Example]

```
Function main
    String myStr$
    myStr$ = "stop"
    IF myStr$ = "Run" Then
        Print "Run"
    ElseIF myStr$ <> "Run" Then
        Print "stop"
    EndIF
Fend
```

Val

[Function]

Convert String to Numeric Value.

[Syntax]

Val(String)

[Parameter Description]

String	The string to be converted.

[Usage Example]

```
Function main
    String myStr$
    myStr$ = "3.1415926"

    Double num
    num = Val(myStr$)
Fend
```

Str$

[Function]

Convert Numeric Value to String.

[Syntax]

Str$(Value)

[Parameter Description]

Value	The numeric value to be converted.

[Usage Example]

```
Function main
    String myStr$
    Double num
    num    = Val(myStr$)
    myStr$ = Str$(num)
Fend
```

Len

[Function]

Get String Length.

[Syntax]

Len(String)

[Parameter Description]

String The string to be converted.

[Usage Example]

```
Function main
    String myStr$
    Print Len(myStr$)
Fend
```

Asc

[Function]

Convert String to ASCII Code

[Syntax]

Asc(String)

[Parameter Description]

String The string to be converted.

[Usage Example]

```
Function Main
    Integer AA1 , BB1 , CC1
    AA1 = Asc("a")
    BB1 = Asc("b")
    CC1 = Asc("c")
    Print "The ASCII Value of AA1 is",AA1
    Print "The ASCII Value of BB1 is",BB1
    Print "The ASCII Value of CC1 is",CC1
Fend
```

Chr$

[Function]

Convert ASCII Code to String.

[Syntax]

Chr$(ASCIIcode)

[Parameter Description]

ASCII code	1 to 255 Specifies the ASCII code

[Usage Example]z

```
Function Main
    String Test$
    Test$ = Chr$(&H41) + Chr$(&H42) + Chr$(&H43)
    Print "The value of Test= " , Test$
Fend
```

Left$

[Function]

Extract a specified substring from the left side of a string.

[Syntax]

Left$(String, count)

[Parameter Description]

String	The string from which to extract.
count	The number of characters to extract.

[Usage Example]

```
Function Main
    Print Left$("ABCDEFG" , 2)
    '>>> AB
    Print Left$("ABCDEFG" , 3)
    '>>> ABC
Fend
```

Mid$

[Function]

Extract a specified substring from a specified range in a string.

[Syntax]

```
Mid$(String, Start_Position {, count})
```

[Parameter Description]

String	The string from which to extract.
Start_Position	The starting position for the substring extraction.
count	The number of characters to extract. If omitted, extraction continues to the end of the string.

[Usage Example]

```
Function main
    Print Mid$("123456" , 3 , 3)
Fend
```

Right$

[Function]

Extract a specified substring from the right side of a string.

[Syntax]

```
Right$(String, count)
```

[Parameter Description]

String	The string from which to extract.
count	The number of characters to extract.

[Usage Example]

```
Function Main
    Print Right$("ABCDEFG" , 2)
    '>>> FG
    Print Right$("ABC" , 3)
    '>>> ABC
```

```
Fend
```

LSet$

[Function]

Extract a specified length of substring from the left side of a string, padding with spaces if the string length is insufficient.

[Syntax]

```
LSet$(String, count)
```

[Parameter Description]

String	The string from which to extract.
count	The number of characters to extract.

[Usage Example]

```
Function Main
    String temp$
    temp$ = "123"
    temp$ = LSet$(temp$ , 10)        'temp$ = "123"
Fend
```

RSet$

[Function]

Extract a specified length of substring from the right side of a string, padding with spaces if the string length is insufficient.

[Syntax]

```
RSet$(String, count)
```

[Parameter Description]

String	The string from which to extract.
count	The number of characters to extract.

[Usage Example]

```
Function Main
    String temp$
    temp$ = "123"
    temp$ = RSet$(temp$ , 10)        'temp$ = "123"
Fend
```

Space$

[Function]

Return a string consisting of a specified number of spaces.

[Syntax]

```
Space$(count)
```

[Parameter Description]

count The number of space characters.

[Usage Example]

```
Function Main
    Print "XYZ" + Space$(1) + "ABC"
    '>>>XYZ ABC
    Print Space$(3) + "ABC"
    '>>>ABC
Fend
```

LCase$

[Function]

Return a string of lowercase letters.

[Syntax]

```
LCase$(String)
```

[Parameter Description]

String The string to be converted.

[Usage Example]

```
Function Main
    String Test$
    Test$ = "Data"
    Test$ = LCase$(Test$)          'Test$ = "Data"
Fend
```

UCase$

[Function]

Return a string of uppercase letters.

[Syntax]

UCase$(String)

[Parameter Description]

String The string to be converted.

[Usage Example]

```
Function Main
    String Test$
    Test$ = "Data"
    Test$ = UCase$(Test$)          'Test$ = "Data"
Fend
```

LTrim$

[Function]

Remove leading spaces from a string and return the result.

[Syntax]

LTrim$(String)

[Parameter Description]

String The string to be converted.

[Usage Example]

```
Function Main
    String Test$
    Test$ = "   Data   "
    Test$ = LTrim$(Test$)          'Test$ = "Data   "
Fend
```

RTrim$

[Function]

Remove trailing spaces from a string and return the result.

[Syntax]

RTrim$(String)

[Parameter Description]

String The string to be converted.

[Usage Example]

```
Function Main
    String Test$
    Test$ = "   Data   "
    Test$ = RTrim$(Test$)           'Test$ = "   Data"
Fend
```

Trim$

[Function]

Remove spaces from both sides of a string and return the result.

[Syntax]

Trim$(String)

[Parameter Description]

String The string to be converted.

[Usage Example]

```
Function Main
    String Test$
    Test$ = "   Data   "
    Test$ = Trim$(Test$)            'Test$ = "Data"
Fend
```

InStr

[Function]

Retrieve and return the position of a specified character in a string.

[Syntax]

```
InStr(String, Search_String)
```

[Parameter Description]

String The string to be searched.

Search_String The substring to search for.

[Usage Example]

```
Function main
    Print InStr("ABCDEF" , "C")
Fend
```

Tab$

[Function]

Return a string consisting of a specified number of tab characters.

[Syntax]

```
Tab$(count)
```

[Parameter Description]

count The specified number of tab characters.

[Usage Example]

```
Function Main
    Integer i
    Print "X" , Tab$(1) , "Y"
    For i = 1 To 10
        Print x(i) , Tab$(1) , y(i)
    Next i
Fend
```

Hex$

[Function]

Convert a hexadecimal number to a string.

[Syntax]

Hex$(Value)

[Parameter Description]

Value The numeric value to be converted to a string.

[Usage Example]

```
Function Main
    Integer stat
    stat = 10
    Print Hex$(stat)
    '>>>A
    Print  Hex$(255)
    '>>>FF
Fend
```

Detailed Description of Positioning Operation Instructions

+、-

[Function]

Relative Coordinate Incremental Motion.

[Syntax]

Motion Command Position + Component

Motion Command Position - Component

[Parameter Description]

Component Directional component.

[Usage Example]

```
Function main
    Go P1 + X(10)
    Move P2 - U(5)
Fend
```

/

[Function]

Modify Tool Coordinate System, Modify User Coordinate System.

[Syntax]

Motion Command /User_Coordinate_System_ID

Motion Command /[R/L]

[Parameter Description]

User_Coordinate_System_ID	Specifies the user coordinate system ID to switch to. Range: 1~64
R/L	Specifies the tool coordinate system to switch, R for right-hand, L for left-hand.

[Usage Example]

```
Function main
    Go P1 + X(10) /R        'Move to X+10mm at point P1 using right-hand coordinate system
    Go P1 + X(10) /3        'Move to X+10mm at point P1 using user coordinate system 3
Fend
```

:

[Function]

Absolute Coordinate Motion.

[Syntax]

Motion_Command Position : Component

[Parameter Description]

Component	Directional component.

[Usage Example]

```
Function main
    Go P1 :X(10)     'Move to point P1 with X=10mm
Fend
```

=

[Function]

Assignment.

[Syntax]

Position = Command

[Parameter Description]

[Usage Example]

```
Function main
    P1 = XY(10 , 20 , 30 ,40)
Fend
```

@

[Function]

Switch to Specified User Coordinate System.

[Syntax]

@User_Coordinate_System_ID

[Parameter Description]

User_Coordinate_System_ID Specifies the user coordinate system ID to switch to. Range: 1~64

[Usage Example]

```
Function main
    GO XY(10 , 20 , 30 ,40)  @2
Fend
```

X、Y、Z、U、V、W

[Function]

Directional component.

[Syntax]

+/-/:Directional component.

[Parameter Description]

[Usage Example]

```
Function main
    Go P1 + X(10)
    Move P2 - U(5)
    Go P1 +X(5) /R            'Move to X+5mm at point P1 using right-hand coordinate system
    Go P1 +X(5) /3            'Move to X+5mm at point P1 using user coordinate system 3
    Go P1 :X(10)              'Move to point P1 with X=10mm
Fend
```

RX、RY、RZ

[Function]

Specify the rotation components of the user coordinate system around the X, Y, and Z axes.

[Syntax]

+/-[RX/RY/RZ]

[Parameter Description]

[Usage Example]

```
Function main
      Go P1 + X(10) + RX(10)
    'The user coordinate at point P1 rotates 10°  around the X-axis and moves to X+10mm at point P1
Fend
```

TLX、TLY、TLZ、TLU、TLV、TLW

[Function]

Specify the rotation components of the tool coordinate system around the X, Y, Z, U, V, and W axes.

[Syntax]

+/-[TLX/TLY/TLZ/TLU/TLV/TLW]

[Parameter Description]

[Usage Example]

```
Function main
      Go P1 + X(10) + TLX(10)
    'The tool coordinate at point P1 rotates 10°  around the X-axis and moves to X+10mm at point P1
Fend
```

SavePoints

[Function]

Save the positioning data in the current program memory to a position file. A new file will be created if it does not exist.

[Syntax]

SavePoints FileName

[Parameter Description]

FileName The name of the position file as a string.

[Usage Example]

```
Function main
    P1 = XY(10 , 20 , 30 , 40)
    P2 = XY(40 , 30 , 20 , 10)
    SavePoints "ABCD" 'Save P1 and P2 to the ABCD position file
Fend
```

LoadPoints

[Function]

Load position file.

[Syntax]

LoadPoints FileName{, Merge}

[Parameter Description]

FileName	The name of the position file as a string.
Merge	An optional setting to prevent clearing current points before reading new points. If set, new points will be added to the existing points. If points to be added already exist in the file, they will be overwritten. This parameter is optional.

[Usage Example]

```
Function main

    'Load common points into the current robot
    LoadPoints "R1Common.pts"

    'Merge points from component model 1
    LoadPoints "R1Model.pts" , Merge

    'Load point file for robot 2
    LoadPoints "R2Model.pts"

Fend
```

ClearPoints

[Function]

Clear the positioning data stored in program memory.

[Syntax]

ClearPoints

[Parameter Description]

[Usage Example]

```
Function main
    P1 = XY(10 , 20 , 30 , 40)
    P2 = XY(40 , 30 , 20 , 10)
    ClearPoints      'The previously defined points P1 and P2 will no longer exist
Fend
```